aws

Guida per gli sviluppatori

Amazon DynamoDB

Versione API 2012-08-10

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

Amazon DynamoDB: Guida per gli sviluppatori

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

I marchi e l'immagine commerciale di Amazon non possono essere utilizzati in relazione a prodotti o servizi che non siano di Amazon, in una qualsiasi modalità che possa causare confusione tra i clienti o in una qualsiasi modalità che denigri o discrediti Amazon. Tutti gli altri marchi non di proprietà di Amazon sono di proprietà dei rispettivi proprietari, che possono o meno essere affiliati, collegati o sponsorizzati da Amazon.

Table of Contents

Che cos'è Amazon DynamoDB? 1
Caratteristiche 2
Serverless 2
NoSQL 2
Servizio completamente gestito 2
Prestazioni con risposte in pochi millisecondi su qualsiasi scala 3
Casi d’uso 3
Funzionalità 4
Replica multi-attiva con tabelle globali 4
Transazioni ACID 4
Change Data Capture 5
Indici secondari 5
Integrazioni dei servizi 5
Integrazioni serverless 5
Importazione ed esportazione di dati in Amazon S3 6
Integrazione Zero-ETL 6
Caching 6
Sicurezza 6
Resilienza 7
Tabelle globali 8
Backup e ripristino continui point-in-time 8
Backup e ripristino on demand 8
Accesso a DynamoDB 8
Prezzi 9
Nozioni di base 9
Nozioni di base su DynamoDB 11
Risorse per gli utenti alle prime armi 12
Best practice aggiuntive di Amazon DynamoDB per gli utenti alle prime armi 13
Risorse AWS CLI 13
Risorse di programmazione 14
Accesso a DynamoDB 14
Utilizzo della console 15
Utilizzando il AWS CLI 15
Utilizzo dell'API 18

Utilizzo di NoSQL Workbench 18
Intervalli di indirizzi IP 19
Endpoint dual-stack per Internet Protocol versione 6 () IPv6 20
Prerequisiti 20
Configurazione di DynamoDB 21
Configurazione di DynamoDB (servizio Web) 21
Configurazione di DynamoDB locale (versione scaricabile) 25
Fase 1: creazione di una tabella 51
Fase 2: scrittura dei dati 104
Fase 3: lettura dei dati 136
Fase 4: aggiornamento dei dati 163
Fase 5: esecuzione di query sui dati 196
Passaggio 6: pulizia (facoltativo) 234
Fasi successive 253
Console-to-Code 253
Come funziona 254
Vantaggi dell'utilizzo Console-to-Code con DynamoDB 254
Casi d'uso di esempio 255
Nozioni di base 255
Come funziona 256
Cheat sheet 256
Configurazione iniziale 256
SDK o CLI 257
Operazioni di base 257
Regole di denominazione 258
Nozioni di base sulle quote del servizio 259
Ulteriori informazioni 261
Componenti principali 261
Tabelle, elementi e attributi 262
Chiave primaria 265
Indici secondari 267
DynamoDB Streams 270
API DynamoDB 271
Piano di controllo 272
Piano dati 272
DynamoDB Streams 274

Transazioni ... 275
Regole di denominazione e tipi di dati supportati ... 275
Regole di denominazione ... 276
Tipi di dati ... 277
Descrittori del tipo di dati ... 282
Classi di tabella DynamoDB ... 282
Partizioni e distribuzione dei dati in DynamoDB ... 283
Distribuzione dei dati: chiave di partizione ... 284
Distribuzione dei dati: chiave di partizione e chiave di ordinamento ... 285
Come passare da SQL a NoSQL ... 288
Relazionale o NoSQL? ... 289
Accesso e autenticazione ... 292
Creazione di una tabella ... 295
Recupero di informazioni su una tabella ... 297
Scrittura dei dati in una tabella ... 299
Lettura dei dati di una tabella ... 303
Gestione degli indici ... 311
Modifica dei dati in una tabella ... 317
Eliminazione dei dati da una tabella ... 321
Rimozione di una tabella ... 323
Risorse e strumenti di apprendimento di Amazon DynamoDB ... 324
Strumenti per la codifica e la visualizzazione ... 324
Linee guida prescrittive ... 325
Knowledge Center ... 326
Post di blog, repository e guide ... 327
Modellazione dei dati e modelli di progettazione ... 327
Corsi di formazione ... 328
Letture e scritture ... 329
Coerenza di lettura di DynamoDB ... 329
Letture a coerenza finale ... 329
Operazioni a elevata consistenza di lettura ... 330
Consistenza di lettura per le tabelle globali ... 330
Operazioni di lettura e scrittura ... 330
Utilizzo per operazioni di lettura ... 331
Utilizzo per operazioni di scrittura ... 332
Capacità di throughput di DynamoDB ... 335

Modalità on demand ... 335
Modalità provisioning ... 336
Modalità di capacità on demand di DynamoDB ... 336
Unità di richiesta di lettura e unità di richiesta di scrittura ... 338
Throughput e proprietà di dimensionamento ... 338
Throughput massimo di DynamoDB per le tabelle on demand ... 339
Modalità di capacità assegnata ... 341
Unità di capacità di lettura e scrittura ... 343
Scelta delle impostazioni di velocità di trasmissione effettiva iniziali ... 343
Scalabilità automatica di DynamoDB ... 344
Gestione della capacità effettiva di trasmissione con il dimensionamento automatico ... 345
Capacità riservata ... 377
Throughput a caldo ... 378
Check your table's warm throughput ... 379
Aumento del throughput a caldo di una tabella ... 382
Creazione di una tabella con throughput caldo più elevato ... 389
Scenari di throughput a caldo ... 400
Capacità di espansione e capacità adattiva ... 402
Capacità di ottimizzazione ... 402
Capacità adattiva ... 402
Passaggio tra modalità di capacità ... 404
Dalla modalità con provisioning alla modalità on demand ... 404
Dalla modalità on demand alla modalità con provisioning ... 406
Programmazione con DynamoDB ... 408
Panoramica del supporto degli SDK AWS per DynamoDB ... 408
Supporto SDK per endpoint basati su account AWS ... 410
Interfacce programmatiche che funzionano con DynamoDB ... 411
Interfacce di programmazione di livello superiore ... 418
Esecuzione di esempi di codice ... 481
API di basso livello ... 488
Programmazione con Python ... 494
Informazioni su Boto ... 495
Documentazione Boto ... 495
Livelli del client e delle risorse ... 496
Utilizzo di batch_writer ... 499
Ulteriori esempi di codice ... 500

Sicurezza delle sessioni e dei thread ... 500
Config ... 501
Gestione degli errori ... 506
Registrazione ... 508
Hook degli eventi ... 509
Impaginazione e impaginatore ... 511
Waiter ... 513
Programmazione con JavaScript ... 513
Informazioni su AWS SDK per JavaScript ... 514
AWS SDK per JavaScript V3 ... 514
Documentazione per JavaScript ... 514
Livelli di astrazione ... 515
Funzione di utilità marshall ... 517
Lettura degli elementi ... 518
Scritture condizionali ... 519
Impaginazione ... 520
Config ... 523
Waiter ... 526
Gestione degli errori ... 526
Registrazione ... 528
Considerazioni ... 529
Programmazione con AWS SDK for Java 2.x ... 530
Informazioni su AWS SDK for Java 2.x ... 531
Nozioni di base ... 532
Documentazione SDK per Java 2.x ... 541
Interfacce supportate ... 542
Ulteriori esempi di codice ... 556
Programmazione sincrona e asincrona ... 557
Client HTTP ... 557
Config ... 559
Gestione degli errori ... 566
AWS ID della richiesta ... 567
Registrazione dei log ... 567
Paginazione ... 570
Annotazioni sulle classi di dati ... 572
Gestione degli errori ... 572

Componenti degli errori ... 572
Errori transazionali ... 573
Messaggi e codici di errore ... 573
Gestione degli errori nell'applicazione ... 579
Ripetizione dei tentativi in caso di errore e backoff esponenziale ... 579
Operazioni in batch e gestione degli errori ... 580
Utilizzo di AWS SDK ... 581
Utilizzo di DynamoDB ... 583
Utilizzo delle tabelle ... 583
Operazioni di base sulle tabelle ... 584
Considerazioni sulla scelta di una classe di tabella in DynamoDB ... 593
Etichette e cartellini ... 594
Utilizzo delle tabelle globali ... 600
Come funziona ... 601
Tutorial: creazione di tabelle globali ... 610
Sicurezza per tabelle globali ... 648
Fatturazione per le tabelle globali ... 662
Versioni delle tabelle globali ... 665
Best practice per le tabelle globali ... 677
Utilizzo di elementi ... 679
Dimensioni e formati degli elementi ... 680
Lettura di un elemento ... 681
Scrittura di un elemento ... 682
Valori restituiti ... 685
Operazioni batch ... 686
Contatori atomici ... 689
Scritture condizionali ... 689
Utilizzo di espressioni ... 696
Time to Live (TTL) ... 737
Esecuzione di query ... 768
Scansione di tabelle ... 777
Linguaggio di query PartiQL ... 786
Utilizzo degli elementi: Java ... 835
Uso di elementi: .NET ... 867
Utilizzo degli indici ... 900
Indici secondari globali ... 905

Indici secondari locali ... 1005
Utilizzo delle transazioni ... 1059
Come funziona ... 1060
Utilizzo di IAM con le transazioni ... 1070
Codice di esempio ... 1073
Utilizzo di flussi ... 1077
Opzioni ... 1078
Utilizzo di Kinesis Data Streams ... 1080
Utilizzo di DynamoDB Streams ... 1098
Accelerazione in memoria con DAX ... 1182
Casi d'uso per DAX ... 1183
Note per l'utilizzo di DAX ... 1184
Come funziona ... 1185
Come sono elaborate le richieste da DAX ... 1187
Cache degli elementi ... 1188
Cache delle query ... 1189
Componenti del cluster DAX ... 1190
Nodi ... 1191
Cluster ... 1191
Regioni zone di disponibilità ... 1193
Gruppi di parametri ... 1193
Gruppi di sicurezza ... 1193
ARN del cluster ... 1194
Endpoint del cluster ... 1194
Endpoint dei nodi ... 1194
Gruppi di sottoreti ... 1195
Eventi ... 1195
Finestra di manutenzione ... 1195
Creazione di un cluster DAX ... 1197
Creazione di un ruolo di servizio IAM per DAX per l'accesso a DynamoDB ... 1197
DAX e IPv6 ... 1199
Utilizzando il AWS CLI ... 1202
Utilizzo della console ... 1208
Modelli di consistenza ... 1213
Consistenza tra i nodi del cluster DAX ... 1213
Comportamento della cache di elementi DAX ... 1214

Comportamento della cache di query DAX ... 1217
Letture fortemente consistenti e transazionali ... 1218
Caching negativo ... 1219
Strategie di scrittura ... 1219
Sviluppo con il client DAX ... 1223
Tutorial: Esecuzione di un'applicazione di esempio ... 1223
Modifica di un'applicazione esistente per l'utilizzo con DAX ... 1289
Gestione dei cluster DAX ... 1290
Autorizzazioni IAM per la gestione di un cluster DAX ... 1291
Dimensionamento di un cluster DAX ... 1294
Personalizzazione delle impostazioni del cluster DAX ... 1295
Configurazione delle impostazioni TTL ... 1296
Supporto per l'assegnazione di tag per DAX ... 1298
AWS CloudTrail integrazione ... 1299
Eliminazione di un cluster DAX ... 1299
Monitoraggio di DynamoDB Accelerator ... 1300
Strumenti di monitoraggio per DAX ... 1300
Monitoraggio con CloudWatch ... 1302
Registrazione delle operazioni di DAX con AWS CloudTrail ... 1328
Istanze espandibili T3/T2 DAX ... 1329
Famiglia di istanze T2 DAX ... 1329
Famiglia di istanze T3 DAX ... 1329
Controllo degli accessi DAX ... 1330
Ruolo di servizio IAM per DAX ... 1331
Policy IAM per consentire l'accesso del cluster DAX ... 1333
Caso di studio: accesso a DynamoDB e DAX ... 1334
Accesso a DynamoDB, ma senza accesso con DAX ... 1336
Accesso a DynamoDB e a DAX ... 1338
Accesso a DynamoDB tramite DAX, ma senza accesso diretto a DynamoDB ... 1344
Crittografia DAX dei dati inattivi ... 1347
Abilitazione della crittografia dei dati inattivi mediante la Console di gestione AWS ... 1349
Crittografia DAX in transito ... 1350
Utilizzo di ruoli collegati ai servizi per DAX ... 1351
Autorizzazioni del ruolo collegato ai servizi per DAX ... 1352
Creazione di un ruolo collegato ai servizi per DAX ... 1353
Modifica di un ruolo collegato ai servizi per DAX ... 1353

Eliminazione di un ruolo collegato ai servizi per DAX 1354
Accesso a DAX su più account AWS 1355
Configurazione di IAM 1356
Configurazione VPC 1359
Modifica del client DAX per consentire l'accesso multi-account 1361
Guida alle dimensioni del cluster DAX 1366
Panoramica 1366
Stima del traffico. 1367
Test di caricamento 1368
Modellazione dei dati 1370
Utilizzo di Raccolte di elementi 1371
Accelerare le query organizzando i dati con le raccolte di elementi 1373
Fondamenti della modellazione dei dati 1373
Progettazione tabella singola 1374
Progettazione tabelle multiple 1376
Elementi costitutivi della modellazione dei dati 1378
Chiave di ordinamento composita 1378
Multilocazione 1380
Indice Sparse 1381
Time to Live 1382
Archiviazione Time to Live 1383
Partizionamento verticale 1384
Partizionamento di scrittura 1387
Pacchetti di progettazione dello schema di modellazione dei dati 1388
Prerequisiti 1389
Social network 1390
Profilo di gioco 1399
Sistema di gestione dei reclami 1408
Pagamenti ricorrenti 1427
Aggiornamenti sullo stato del dispositivo 1432
Negozio online 1446
Modellazione relazionale 1470
Modelli di database relazionali tradizionali 1471
In che modo DynamoDB elimina la necessità di operazioni JOIN 1473
In che modo le transazioni DynamoDB eliminano il sovraccarico del processo di scrittura .. 1474
Fase iniziale 1475

Esempio 1477
Migrazione a DynamoDB 1482
Motivi per migrare 1482
Considerazioni sulle tempistiche di migrazione 1483
Come funziona 1486
Strumenti di migrazione 1487
Scelta di una strategia di migrazione 1487
Migrazione offline 1490
Migrazione ibrida 1492
Online: migrazione di ogni tabella 1:1 1494
Online: migrazione con una tabella di staging personalizzata 1495
NoSQL Workbench 1498
Scarica 1499
Installa 1500
Data modeler 1508
Creazione di un nuovo modello 1508
Importazione di un modello esistente 1515
Esportazione di un modello 1518
Modifica di un modello esistente 1520
Visualizzatore dati 1524
Aggiunta di dati di esempio 1524
Importazione da CSV 1527
Facet 1528
Visualizzazione aggregata 1531
Commit di un modello di dati 1532
Operation Builder 1535
Connessione a set di dati 1536
Creazione di operazioni 1537
Clonazione di tabelle 1549
Esportazione in CSV 1551
Modelli di dati di esempio 1551
Modello di dati del dipendente 1552
Modello di dati del forum di discussione 1552
Modello di dati della libreria musicale 1553
Modello di dati della stazione sciistica 1553
Modello di dati delle offerte per carta di credito 1554

Modello di dati dei segnalibri ... 1554
Cronologia delle versioni ... 1555
Backup e ripristino ... 1562
Backup point-in-time ... 1564
Prima di iniziare ... 1564
Abilitazione del recupero point-in-time ... 1565
Backup on demand ... 1569
Come funziona ... 1570
Backup di una tabella ... 1573
Ripristino di una tabella ... 1576
Eliminazione di un backup di tabella ... 1582
Uso di IAM ... 1583
Fatturazione per i backup ... 1591
Come funziona ... 1591
Esempio di fatturazione dei backup DynamoDB ... 1592
Ripristini ... 1597
Ripristino di una tabella tramite ripristino point-in-time (PITR) ... 1597
Ripristino point-in-time (PITR) di una tabella DynamoDB ... 1599
Utilizzo di AWS backup ... 1605
Come funziona ... 1606
Creazione di backup ... 1609
Copia di backup ... 1611
Ripristino di una tabella ... 1612
Eliminazione di backup ... 1613
Backup on demand gestiti da AWS Backup e backup DynamoDB ... 1614
Esempi di codice ... 1616
Nozioni di base ... 1629
Hello DynamoDB ... 1630
Informazioni di base ... 1640
Azioni ... 1796
Scenari ... 2204
Accelerazione delle letture con DAX ... 2207
Scenari avanzati basati sugli indici secondari globali ... 2215
Costruisci un'app per inviare dati a una tabella DynamoDB ... 2218
Confronto di più valori con un unico attributo ... 2220
Aggiornamento condizionale del TTL di un elemento ... 2242

Connessione a un'istanza locale 2249
Conteggio degli operatori di espressione 2250
Creazione di una REST API per monitorare i dati COVID-19 2283
Creazione di un'applicazione di messaggistica 2284
Creazione di un'applicazione serverless per gestire foto 2285
Creazione di una tabella con un indice secondario globale 2289
Creazione di una tabella con throughput a caldo abilitato 2293
Creazione di un'applicazione Web per tracciare i dati DynamoDB 2303
Creazione di un'applicazione di chat websocket 2305
Creazione di un elemento con un TTL 2306
Creazione e gestione di tabelle globali MRSC 2312
Creazione e gestione di tabelle globali che descrivono la MREC 2332
Eliminare i dati utilizzando PartiQL DELETE 2342
Rilevamento dei DPI nelle immagini 2348
Inserimento di dati con istruzioni PartiQL INSERT 2349
Richiamo a una funzione Lambda da un browser 2354
Gestione degli indici secondari globali 2354
Gestione di policy basate su risorse 2357
Monitoraggio delle prestazioni di DynamoDB 2360
Esecuzione di operazioni query avanzate 2360
Esecuzione di operazioni di tipo elenco 2395
Esecuzione di operazioni di mappatura 2428
Esecuzione di operazioni su set 2452
Esecuzione di una query su una tabella mediante batch di istruzioni PartiQL 2482
Esecuzione di una query mediante PartiQL 2545
Esecuzione di query su una tabella utilizzando un indice secondario globale 2599
Esecuzione di query su una tabella utilizzando una condizione begins_with 2608
Esecuzione di query su una tabella utilizzando un intervallo di date 2614
Esecuzione di query su una tabella con un'espressione di filtro complessa 2623
Esecuzione di query su una tabella con un'espressione di filtro dinamica 2631
Esecuzione di query su una tabella con un'espressione di filtro e un limite 2640
Esecuzione di query su una tabella con attributi annidati 2645
Esecuzione di query su una tabella con impaginazione 2654
Esecuzione di query su una tabella con operazioni a elevata consistenza di lettura 2665
Esecuzione di query sui dati utilizzando istruzioni PartiQL SELECT 2669
Esecuzione di query per elementi TTL 2675

Esecuzione di query utilizzando modelli di data e ora ... 2679
Salvataggio di EXIF e altre informazioni sull'immagine ... 2694
Configurazione del controllo degli accessi basato su attributi ... 2695
Funzionamento dell'ordine di aggiornamento delle espressioni ... 2698
Aggiornamento dell'impostazione di throughput a caldo di una tabella ... 2729
Aggiornamento del TTL di un elemento ... 2735
Aggiornamento dei dati con PartiQL UPDATE ... 2739
Utilizzo di un'API Gateway per richiamare una funzione Lambda ... 2745
Utilizzo di Step Functions per richiamare le funzioni Lambda ... 2747
Utilizzo di un modello di documento ... 2748
Utilizzo di un modello di persistenza degli oggetti di alto livello ... 2764
Utilizzo di operazioni di contatori atomici ... 2774
Utilizzo di operazioni condizionali ... 2797
Utilizzo dei nomi degli attributi di espressione ... 2827
Utilizzo degli eventi pianificati per richiamare una funzione Lambda ... 2854
Utilizzo di indici secondari locali ... 2856
Lavora con Streams e Time-to-Live ... 2859
Utilizzo di tabelle globali e replica multi-Regione con coerenza possibile (MREC) ... 2863
Utilizzo dell'assegnazione di tag per le risorse ... 2866
Utilizzo della crittografia delle tabelle ... 2869
Esempi serverless ... 2871
Richiamare una funzione Lambda da un trigger DynamoDB ... 2871
Segnalazione di errori di elementi batch per funzioni Lambda con un trigger DynamoDB ... 2880
AWS contributi della comunità ... 2892
Creare e testare un'applicazione serverless ... 2892
Sicurezza ... 2895
AWS politiche gestite ... 2896
AWS politiche gestite ... 2896
AWS policy gestita: Access_v2 AmazonDynamo DBFull ... 2897
AmazonDynamoDBReadOnlyAccess ... 2897
DynamoDB aggiorna le policy gestite AWS ... 2898
Policy basate sulle risorse ... 2900
Crea tabella ... 2901
Collegamento di una policy basata su risorse ... 2908
Collegamento di una policy a un flusso ... 2912
Rimozione di una policy basata su risorse ... 2915

Accesso multi-account ... 2916
Blocco dell'accesso pubblico ... 2918
Operazioni API ... 2921
Autorizzazione IAM ... 2929
Esempi ... 2929
Considerazioni ... 2937
Best practice ... 2938
Controllo dell'accesso basato sugli attributi ... 2939
Qual è il vantaggio di utilizzare l’ABAC? ... 2941
Chiavi di condizione ... 2942
Considerazioni ... 2942
Abilitazione dell’ABAC in DynamoDB ... 2943
Utilizzo dell’ABAC ... 2946
Casi d'uso di esempio ... 2947
Risoluzione dei problemi ... 2961
Protezione dei dati ... 2962
Crittografia dei dati a riposo ... 2963
Protezione delle connessioni DynamoDB ... 2990
IAM ... 2992
Identity and Access Management ... 2992
Utilizzo delle condizioni ... 3024
Convalida della conformità ... 3049
Resilienza ... 3050
Sicurezza dell’infrastruttura ... 3051
Utilizzo di endpoint VPC ... 3051
AWS PrivateLink per DynamoDB ... 3061
Tipi di endpoint Amazon VPC ... 3062
Considerazioni sull'utilizzo AWS PrivateLink per Amazon DynamoDB ... 3063
Creazione di un endpoint Amazon VPC ... 3064
Accesso agli endpoint di interfaccia Amazon DynamoDB ... 3064
Accesso alle tabelle DynamoDB e alle operazioni API di controllo dagli endpoint di interfaccia DynamoDB ... 3064
Aggiornamento di una configurazione DNS locale ... 3066
Creazione di una policy sugli endpoint Amazon VPC ... 3069
Utilizzo degli endpoint DynamoDB con accesso privato a Console di gestione AWS ... 3070
AWS PrivateLink per DynamoDB Streams ... 3070

AWS PrivateLink per DAX ... 3076
Analisi della configurazione e delle vulnerabilità ... 3079
Best practice di sicurezza ... 3080
Best practice relative alla sicurezza preventiva ... 3080
Best practice relative alla sicurezza di rilevamento ... 3083
Monitoraggio e registrazione ... 3087
Piano di monitoraggio ... 3087
Baseline delle prestazioni ... 3087
Servizi integrati ... 3088
Strumenti di monitoraggio automatici ... 3088
Monitoraggio di parametri ... 3089
Come si utilizzano i parametri di DynamoDB? ... 3089
Visualizzazione dei parametri nella console CloudWatch ... 3091
Visualizzazione delle metriche nella AWS CLI ... 3091
Metriche e dimensioni ... 3092
Creazione di CloudWatch allarmi in DynamoDB ... 3124
Registrazione di log delle operazioni ... 3128
Informazioni su DynamoDB in CloudTrail ... 3129
Informazioni sulle voci dei file di registro di DynamoDB ... 3133
Contributor Insights ... 3152
Come funziona ... 3152
Nozioni di base ... 3163
Uso di IAM ... 3173
Best practice ... 3180
Progettazione NoSQL ... 3180
NoSQL e RDBMS ... 3181
Due concetti chiave ... 3181
Approccio generale ... 3182
NoSQL Workbench ... 3183
DynamoDB Well-Architected Lens ... 3184
Ottimizzazione dei costi ... 3184
Esecuzione della valutazione di Amazon DynamoDB Well-Architected Lens ... 3235
I pilastri di Amazon DynamoDB Well-Architected Lens ... 3235
Progettazione delle chiavi di partizione ... 3238
Distribuzione dei carichi di lavoro ... 3239
Partizionamento di scrittura ... 3240

Caricamento dei dati in modo efficiente ... 3242
Progettazione della chiave di ordinamento ... 3243
Controllo della versione ... 3244
Indici secondari ... 3245
Linee guida generali ... 3246
Indici di tipo sparse ... 3249
Aggregazione ... 3251
Sovraccarico di GSI ... 3253
Partizionamento GSI ... 3254
Creazione di una replica ... 3259
Elementi di grandi dimensioni ... 3260
Compression ... 3261
Partizionamento verticale ... 3261
Uso di Amazon S3 ... 3262
Dati di serie temporali ... 3262
Modello di progetto per i dati di serie temporali ... 3263
Esempi di tabella di serie temporali ... 3263
Many-to-many relazioni ... 3264
Elenchi di adiacenza ... 3265
Grafici materializzati ... 3266
Esecuzione di query e scansione ... 3271
Prestazioni delle scansioni ... 3271
Come evitare i picchi ... 3272
Scansioni parallele ... 3275
Progettazione di tabelle ... 3276
Utilizzo delle tabelle globali ... 3276
Aspetti chiave ... 3277
Aspetti chiave della MREC ... 3278
Aspetti chiave della MRSC ... 3279
Casi d’uso ... 3281
Modalità di scrittura ... 3282
Strategie di instradamento in DynamoDB ... 3289
Processi di evacuazione ... 3295
Pianificazione della capacità effettiva di trasmissione ... 3298
Lista di controllo per la preparazione ... 3300
Conclusioni e risorse ... 3309

Piano di controllo ... 3310
Operazioni sui dati in blocco ... 3311
Aggiornamento batch condizionale ... 3311
Operazioni in blocco efficienti ... 3316
Implementazione del controllo delle versioni ... 3318
Quando usare questo modello ... 3318
Progettazione dei modelli ... 3319
Utilizzo del modello ... 3320
Report di fatturazione e utilizzo ... 3322
Capacità di throughput ... 3326
Streams ... 3330
Storage ... 3331
Backup e ripristino ... 3332
Trasferimento dati ... 3336
CloudWatch ... 3336
DAX ... 3338
Migrazione di una tabella DynamoDB da un account a un altro ... 3339
Migrazione di una tabella con AWS Backup per il backup e il ripristino tra account ... 3340
Migrazione di una tabella utilizzando l'esportazione in S3 e l'importazione da S3 ... 3342
Prontuario DAX ... 3345
Valutazione dell'idoneità di DAX ... 3345
Configurazione del client DAX ... 3348
Configurazione del cluster DAX ... 3349
Dimensionamento del cluster DAX ... 3356
Implementazione di un cluster ... 3363
Operazioni sui cluster ... 3364
Monitoraggio di DAX ... 3367
Utilizzo di DynamoDB con altri servizi AWS ... 3370
Integrazione con Amazon Cognito ... 3370
Integrazione con Amazon Redshift ... 3373
Considerazioni sull'integrazione multi-account con CMK ... 3373
Integrazione Zero-ETL con Amazon Redshift ... 3377
Caricamento di dati da DynamoDB in Amazon Redshift con COPY ... 3386
Integrazione con Amazon EMR ... 3387
Panoramica ... 3388
Tutorial: Utilizzo di Amazon DynamoDB e Apache Hive ... 3389

Creazione di una tabella esterna in Hive ... 3398
Elaborazione delle istruzioni HiveQL ... 3402
Esecuzioni di query sui dati in DynamoDB ... 3403
Copia di dati da e verso Amazon DynamoDB ... 3405
Ottimizzazione prestazioni ... 3419
Integrazione con S3 ... 3425
Importazione da Amazon S3 ... 3426
Esportazione in Amazon S3 ... 3446
Integrazione con Amazon SageMaker Lakehouse ... 3469
Integrazione Zero-ETL con Amazon SageMaker Lakehouse ... 3469
Integrazione con il Servizio OpenSearch di Amazon ... 3471
Come funziona ... 3471
Creazione di un'integrazione ... 3472
Passaggi successivi ... 3473
Gestione delle modifiche sostanziali ... 3473
Integrazione Zero-ETL con il Servizio OpenSearch ... 3477
Integrazione con Amazon EventBridge ... 3479
Come funziona ... 3480
Creazione di un'integrazione tramite la console ... 3481
Passaggi successivi ... 3484
Integrazione con Amazon MSK ... 3485
Come funziona ... 3485
Esempio di integrazione ... 3486
Passaggi successivi ... 3493
Best practice per l'integrazione ... 3493
Creazione di uno snapshot ... 3493
Change Data Capture ... 3494
Utilizzo dell'IA generativa con DynamoDB ... 3495
Casi d'uso di IA generativa per DynamoDB ... 3495
Blog di IA generativa per DynamoDB ... 3496
Sfruttare l'integrazione zero-ETL di DynamoDB con Service OpenSearch ... 3496
Quote e vincoli ... 3498
Esecuzione di attività di gestione delle quote ... 3498
Accesso alle quote di DynamoDB ... 3498
Visualizzazione delle quote correnti nella console ... 3499
Visualizzazione delle quote correnti tramite la AWS CLI ... 3499

Richiesta di aumento delle quote ... 3501
Quote ... 3502
Throughput di lettura/scrittura ... 3503
Capacità prenotata ... 377
Tabelle ... 3506
Tabelle globali ... 3506
Indici secondari ... 3508
Attributi degli indici secondari proiettati ... 3508
DynamoDB Streams ... 3508
Importazione da Amazon S3 ... 3509
Esportazione delle tabelle in Amazon S3 ... 3509
Backup e ripristino ... 3509
Contributor Insights ... 3509
Vincoli ... 3509
Modalità di capacità in lettura/scrittura ... 3510
Indici secondari ... 3508
Chiavi di partizione e chiavi di ordinamento ... 3511
Regole di denominazione ... 3512
Tipi di dati ... 3513
Item ... 3514
Attributes ... 3514
Parametri di espressione ... 3515
Transazioni di DynamoDB ... 3516
DynamoDB Streams ... 3516
DynamoDB Accelerator (DAX) ... 3517
Vincoli specifici delle API ... 3517
Crittografia a riposo per DynamoDB ... 3520
Riferimento API ... 3522
Risoluzione dei problemi ... 3523
Errori interni del server ... 3523
Indagine degli errori interni del server ... 3524
Ridurre al minimo l'impatto degli errori interni del server ... 3524
Miglioramento della consapevolezza operativa ... 3525
Latenza ... 3529
Throttling ... 3532
Diagnosi della limitazione (della larghezza di banda della rete) ... 3535

Guida alla risoluzione 3542
Limitazione (della larghezza di banda della rete) e contropressione di scrittura sui GSI 3571
Metriche di limitazione (della larghezza di banda della rete) di CloudWatch 3573
Appendice 3577
Risoluzione dei problemi di creazione della SSL/TLS connessione con DynamoDB 3577
Test dell'applicazione o del servizio 3577
Test del browser del client 3578
Aggiornamento del client dell'applicazione software 3578
Aggiornamento del browser del client 3579
Aggiornamento manuale del bundle di certificati 3579
Tabelle e dati di esempio da utilizzare in DynamoDB 3580
File di dati di esempio 3581
Creazione di tabelle di esempio e caricamento di dati 3594
Creazione di tabelle di esempio e caricamento di dati - Java 3594
Creazione di tabelle di esempio e caricamento di dati - .NET 3604
Applicazione di esempio che utilizza AWS SDK for Python (Boto3) 3616
Fase 1: distribuzione e test in locale 3617
Fase 2: esame del modello di dati e dei dettagli di implementazione 3622
Fase 3: distribuzione in produzione 3632
Fase 4: Eliminazione delle risorse 3641
Parole riservate in DynamoDB 3642
AWS Esempi di SDK for Java 1.x 3655
DAX e Java SDK v1 3655
Modifica dell'applicazione SDK per Java 1.x esistente per l'utilizzo di DAX 3668
Esecuzione di query su indici secondari globali con SDK per Java 1.x 3673
AWS Esempi di SDK for Go 1.x 3677
Go e DAX 3677
AWS SDK per esempi di Node.js 2.x 3679
Node.js e DAX 3680
Cronologia dei documenti 3690
Aggiornamenti precedenti 3726
Funzionalità legacy 3762
Tabelle globali versione 2017.11.29 (Legacy) 3762
Come funziona 3763
Best practice e requisiti 3768
Creazione di una tabella globale 3772

Monitoraggio delle tabelle globali 3777
Uso di IAM con le tabelle globali 3779
Versione precedente dell'API DynamoDB di basso livello (2011-12-05) 3782
BatchGetItem 3783
BatchWriteItem 3791
CreateTable 3798
DeleteItem 3807
DeleteTable 3814
DescribeTables 3818
GetItem 3822
ListTables 3826
PutItem 3829
Query 3837
Scan 3853
UpdateItem 3871
UpdateTable 3881
Parametri condizionali legacy di DynamoDB 3886
AttributesToGet 3888
AttributeUpdates 3889
ConditionalOperator 3892
Expected (Atteso) 3893
KeyConditions 3898
QueryFilter 3902
ScanFilter 3904
Scrittura di condizioni con parametri legacy 3906
..... mmmcmxv

Che cos'è Amazon DynamoDB?

Amazon DynamoDB è un database NoSQL distribuito senza server, completamente gestito, con prestazioni a una cifra in millisecondi su qualsiasi scala.

DynamoDB risponde alle esigenze di superare le complessità di scalabilità e operative dei database relazionali. È progettato e ottimizzato appositamente per carichi di lavoro operativi che richiedono prestazioni costanti su qualsiasi scala. Ad esempio, DynamoDB offre prestazioni costanti con risposte in pochi millisecondi per un caso d'uso di carrello degli acquisti, indipendentemente dal fatto che gli utenti siano 10 oppure 100 milioni. Lanciato nel 2012, DynamoDB continua ad aiutare ad abbandonare i database relazionali, riducendo al contempo i costi e migliorando le prestazioni su larga scala.

I clienti di tutte le dimensioni, settori e aree geografiche utilizzano DynamoDB per creare applicazioni moderne e serverless che possono iniziare in piccolo e scalare a livello globale. DynamoDB è scalabile per supportare tabelle di qualsiasi dimensione, garantendo al contempo prestazioni costanti con risposte in pochi millisecondi e alta disponibilità.

Per eventi come Amazon Prime Day, DynamoDB supporta più proprietà e sistemi Amazon ad alto traffico, tra cui Alexa, i siti Amazon.com e tutti i centri logistici Amazon. Per tali eventi, APIs DynamoDB ha gestito trilioni di chiamate dalle proprietà e dai sistemi Amazon. DynamoDB serve continuamente centinaia di clienti con tabelle che hanno un traffico di picco di oltre mezzo milione di richieste al secondo. Serve inoltre centinaia di clienti per cui le dimensioni delle tabelle superano i 200 TB ed elabora oltre un miliardo di richieste all'ora.

Argomenti

- Caratteristiche di DynamoDB
- Casi d'uso di DynamoDB
- Funzionalità di DynamoDB
- Integrazioni dei servizi
- Sicurezza
- Resilienza
- Accesso a DynamoDB
- Prezzi di DynamoDB
- Nozioni di base su DynamoDB

Caratteristiche di DynamoDB

Serverless

Con DynamoDB non è necessario effettuare il provisioning di alcun server o applicare patch, gestire, installare, mantenere o utilizzare alcun software. DynamoDB offre una manutenzione senza tempi di inattività. Non ha versioni (principale, secondaria o patch) e non ci sono finestre di manutenzione.

La modalità di capacità on demand di DynamoDB offre pay-as-you-go prezzi per le richieste di lettura e scrittura, in modo da pagare solo per ciò che si utilizza. Con la modalità on demand, DynamoDB aumenta o riduce verticalmente in modo istantaneo le tabelle per adattarle alla capacità e mantiene le prestazioni senza alcuna amministrazione. Inoltre, può essere ridotto verticalmente fino a zero, in modo da non dover pagare per il throughput quando la tabella non ha traffico e non ci sono avvii a freddo.

NoSQL

In quanto database NoSQL, DynamoDB è progettato appositamente per offrire prestazioni, scalabilità, gestibilità e flessibilità migliorate rispetto ai database relazionali tradizionali. Per supportare un'ampia varietà di casi d'uso, DynamoDB supporta sia modelli chiave-valore che modelli di dati documentali.

A differenza dei database relazionali, DynamoDB non supporta un operatore JOIN. Si consiglia di denormalizzare il modello di dati per ridurre i round trip del database e la potenza di elaborazione necessaria per rispondere alle query. In quanto database NoSQL, DynamoDB offre una elevata consistenza di lettura e transazioni ACID per creare applicazioni a livello delle grandi imprese.

Servizio completamente gestito

Essendo un servizio di database completamente gestito, DynamoDB si occupa della gestione indifferenziata di un database, in modo da poter concentrarsi sulla creazione di valore per i clienti. Gestisce l'installazione, le configurazioni, la manutenzione, l'alta disponibilità, il provisioning dell'hardware, la sicurezza, i backup, il monitoraggio e altro ancora. In questo modo, quando si crea una tabella DynamoDB, questa è immediatamente pronta per i carichi di lavoro di produzione. DynamoDB migliora costantemente la sua disponibilità, affidabilità, prestazioni, sicurezza e funzionalità senza richiedere aggiornamenti o tempi di inattività.

Prestazioni con risposte in pochi millisecondi su qualsiasi scala

DynamoDB è stato progettato appositamente per migliorare le prestazioni e la scalabilità dei database relazionali per offrire prestazioni con risposte in pochi millisecondi su qualsiasi scala. Per raggiungere questa scalabilità e prestazioni, DynamoDB è ottimizzato per carichi di lavoro ad alte prestazioni e APIs offre soluzioni che incoraggiano un utilizzo efficiente del database. Omette funzionalità inefficienti e non performanti su larga scala, ad esempio le operazioni JOIN. DynamoDB offre prestazioni costanti con risposte in pochi millisecondi per le proprie applicazioni, indipendentemente dal fatto gli utenti siano 100 oppure 100 milioni.

Casi d'uso di DynamoDB

I clienti di tutte le dimensioni, settori e aree geografiche utilizzano DynamoDB per creare applicazioni moderne e serverless che possono iniziare in piccolo e scalare a livello globale. DynamoDB è ideale per i casi d'uso che richiedono prestazioni costanti su qualsiasi scala con un sovraccarico operativo minimo o nullo. L'elenco seguente presenta alcuni casi d'uso in cui è possibile utilizzare DynamoDB:

- Applicazioni per servizi finanziari: si consideri il caso di una società di servizi finanziari che sviluppa applicazioni, come il trading e l'instradamento in tempo reale, la gestione dei prestiti, la generazione di token e i log delle transazioni. Con le tabelle globali DynamoDB, le tue applicazioni possono rispondere agli eventi e servire il traffico da te Regioni AWS scelto con prestazioni di lettura e scrittura locali veloci.

  DynamoDB è adatto per applicazioni con i requisiti di disponibilità più rigorosi. Elimina l'onere operativo derivante dal dimensionamento manuale delle istanze per aumentare l'archiviazione o il throughput, il controllo delle versioni e la gestione delle licenze.

  È possibile utilizzare le transazioni DynamoDB per ottenere atomicità, coerenza, isolamento e durabilità (ACID) in una o più tabelle con un'unica richiesta. Le transazioni ACID si adattano a carichi di lavoro che includono l'elaborazione di transazioni finanziarie o l'evasione degli ordini. DynamoDB adatta istantaneamente i carichi di lavoro, man mano che aumentano o diminuiscono, consentendo di scalare in modo efficiente il database in base alle condizioni di mercato, come le ore di attività del trading.
- Applicazioni di videogiochi: le aziende di videogiochi possono utilizzare DynamoDB per tutte le parti delle piattaforme di gioco, ad esempio lo stato del gioco, i dati dei giocatori, la cronologia delle sessioni e le classifiche. DynamoDB è la scelta ideale per la sua scalabilità, le prestazioni costanti e la facilità delle operazioni garantite dalla sua architettura serverless. DynamoDB è perfetto per le

architetture che possono essere aumentate in orizzontale necessarie per supportare videogiochi di successo, poiché dimensiona rapidamente il throughput del gioco sia in entrata che in uscita (dimensionamento fino a zero senza avvio a freddo). Questa scalabilità ottimizza l'efficienza dell'architettura, che si tratti di aumento orizzontale per i picchi di traffico o di dimensionamento quando l'utilizzo del gioco è basso.

- Applicazioni di streaming: le aziende di media e intrattenimento utilizzano DynamoDB come indice di metadati per contenuti, servizio di gestione dei contenuti o per fornire statistiche sportive quasi in tempo reale. Utilizzano DynamoDB anche per gestire la lista di controllo degli utenti e i servizi di creazione di segnalibri ed elaborare miliardi di eventi giornalieri dei clienti per generare suggerimenti. Questi clienti traggono vantaggio dalla scalabilità, dalle prestazioni e dalla resilienza di DynamoDB. DynamoDB si adatta alle variazioni dei carichi di lavoro man mano che aumentano o diminuiscono, rendendo possibili casi d'uso di streaming multimediale in grado di supportare qualsiasi livello di domanda.

Per ulteriori informazioni su come i clienti di diversi settori utilizzano DynamoDB, consulta Amazon DynamoDB Customers e This is My Architecture.

Funzionalità di DynamoDB

Replica multi-attiva con tabelle globali

Le tabelle globali forniscono la replica multiattiva dei dati su tutti gli utenti prescelti Regioni AWS con una disponibilità del 99,999%. Le tabelle globali forniscono una soluzione completamente gestita per l'implementazione di un database multi-Regione e multi-attivo, senza dover creare e mantenere la propria soluzione di replica. Con le tabelle globali, puoi specificare Regioni AWS dove desideri che le tabelle siano disponibili. DynamoDB replica le modifiche continue ai dati su tutte queste tabelle.

Le applicazioni distribuite a livello globale possono accedere ai dati localmente nelle Regioni selezionate per ottenere prestazioni di lettura e scrittura con risposte in pochi millisecondi. Poiché le tabelle globali sono multi-attive, non è necessaria una tabella principale. Ciò significa che non si verificano failover complicati o ritardati o tempi di inattività del database durante il failover di un'applicazione tra Regioni.

Transazioni ACID

DynamoDB è progettato per carichi di lavoro mission critical. Include il supporto per le transazioni ACID per applicazioni che richiedono una logica aziendale complessa. DynamoDB fornisce supporto

nativo lato server per le transazioni, semplificando l'esperienza degli sviluppatori nell'apportare modifiche coordinate a più elementi all'interno e tra all-or-nothing le tabelle.

Change Data Capture per architetture basate sugli eventi

DynamoDB supporta lo streaming dei record di Change Data Capture (CDC) a livello di elemento quasi in tempo reale. DynamoDB offre due modelli di streaming per il CDC: flussi DynamoDB e flusso di dati Kinesis per DynamoDB. Ogni volta che un'applicazione crea, aggiorna o elimina elementi in una tabella, i flussi registrano una sequenza in ordine cronologico di ogni modifica a livello di elemento quasi in tempo reale. Ciò rende i flussi DynamoDB ideali per le applicazioni con architettura basata sugli eventi che utilizzano le modifiche e agiscono di conseguenza.

Indici secondari

DynamoDB offre la possibilità di creare indici secondari globali e locali, che consentono di eseguire query sui dati della tabella utilizzando una chiave alternativa. Con questi indici secondari è possibile accedere ai dati con attributi diversi dalla chiave primaria, garantendo la massima flessibilità di accesso ai dati.

Integrazioni dei servizi

DynamoDB si integra ampiamente con Servizi AWS diversi sistemi per aiutarti a ottenere più valore dai tuoi dati, eliminare il sollevamento indifferenziato di carichi di lavoro e gestire i carichi di lavoro su larga scala. Alcuni esempi sono: AWS CloudFormation, Amazon CloudWatch, Amazon S3, AWS Identity and Access Management (IAM) e. AWS Auto Scaling Le sezioni seguenti descrivono alcune delle integrazioni di servizi che è possibile eseguire utilizzando DynamoDB:

Integrazioni serverless

Per creare applicazioni end-to-end serverless, DynamoDB si integra nativamente con una serie di applicazioni serverless. Servizi AWS Ad esempio, è possibile integrare DynamoDB con AWS Lambda per creare trigger, vale a dire pezzi di codice che rispondono automaticamente agli eventi nei flussi DynamoDB. Con i trigger è possibile creare applicazioni basate sugli eventi che rispondono alle modifiche di dati nelle tabelle DynamoDB. Per l'ottimizzazione dei costi, è possibile filtrare gli eventi elaborati da Lambda da un flusso DynamoDB.

L'elenco seguente presenta alcuni esempi di integrazioni serverless con DynamoDB:

- AWS AppSyncper creare GraphQL APIs
- Amazon API Gateway per la creazione di REST APIs
- Lambda per il calcolo serverless
- Flusso di dati Amazon Kinesis per Change Data Capture (CDC)

Importazione ed esportazione di dati in Amazon S3

L'integrazione di DynamoDB con Amazon S3 consente di esportare facilmente i dati in un bucket Amazon S3 per l'analisi e il machine learning. DynamoDB supporta le esportazioni di tabelle complete e le esportazioni incrementali per esportare dati modificati, aggiornati o eliminati in un periodo di tempo specificato. È anche possibile importare dati da Amazon S3 in una nuova tabella DynamoDB.

Integrazione Zero-ETL

DynamoDB supporta l'integrazione zero-ETL con Amazon Redshift e l'utilizzo di una pipeline di inserimento con Amazon OpenSearch DynamoDB. Queste integrazioni consentono di eseguire analisi complesse e utilizzare funzionalità di ricerca avanzate sui dati delle tabelle DynamoDB. Ad esempio, è possibile eseguire ricerche di testo completo e vettoriali e ricerche semantiche sui dati DynamoDB. Le integrazioni Zero-ETL non hanno alcun impatto sui carichi di lavoro di produzione in esecuzione su DynamoDB.

Caching

DynamoDB Accelerator (DAX) è un servizio di caching completamente gestito e a disponibilità elevata creato per DynamoDB. Migliora fino a 10 volte le prestazioni anche in presenza di milioni di richieste al secondo, consentendo di ottenere risposte in pochi microsecondi invece che in millisecondi. DAX fa tutto il lavoro necessario per aggiungere l'accelerazione in memoria alle tabelle DynamoDB, senza dover gestire l'invalidamento della cache, il popolamento dei dati o la gestione dei cluster.

Sicurezza

DynamoDB utilizza IAM per controllare in modo sicuro l'accesso alle risorse DynamoDB. Con IAM è possibile gestire a livello centrale le autorizzazioni che controllano quali utenti DynamoDB possono accedere alle risorse. Utilizzi IAM per controllare chi è autenticato (ha effettuato l'accesso)

e autorizzato (dispone delle autorizzazioni) a utilizzare le risorse. Poiché DynamoDB utilizza IAM, non sono necessari nomi utente o password per accedere a DynamoDB. Senza le complicate policy di rotazione delle password da gestire il profilo di sicurezza risulta più semplice. Con IAM è anche possibile abilitare un controllo granulare degli accessi per fornire l'autorizzazione a livello di attributo. È anche possibile definire policy basate su risorse con il supporto per il Sistema di analisi degli accessi IAM e Blocco dell'accesso pubblico (BPA) per semplificare la gestione delle policy.

Per impostazione predefinita, DynamoDB crittografa tutti i dati a riposo dei clienti. La crittografia dei dati a riposo migliora la sicurezza dei dati utilizzando le chiavi di crittografia archiviate in AWS Key Management Service (AWS KMS). La crittografia dei dati inattivi consente di creare applicazioni ad alto livello di sicurezza che rispettano rigorosi requisiti normativi e di conformità per la crittografia. Quando si accede a una tabella crittografata, DynamoDB decrittografa i dati della tabella in modo trasparente. Non è necessario modificare codice o applicazioni per utilizzare o gestire le tabelle crittografate. DynamoDB continua a offrire la consueta latenza di pochi millisecondi e tutte le query di DynamoDB funzionano senza problemi sui dati crittografati.

È possibile specificare se DynamoDB deve utilizzare Chiave di proprietà di AWS una (tipo di crittografia predefinito) Chiave gestita da AWS o una chiave gestita dal cliente per crittografare i dati degli utenti. La crittografia predefinita che utilizza chiavi KMS di proprietà di AWS è disponibile senza costi aggiuntivi. Per la crittografia lato client è possibile utilizzare l'SDK AWS di crittografia del database.

DynamoDB aderisce inoltre a diversi standard di conformità, tra cui HIPAA, PCI DSS e GDPR, che consentono di soddisfare i requisiti normativi.

Resilienza

Per impostazione predefinita, DynamoDB replica automaticamente i dati su tre zone di disponibilità per fornire un'elevata durabilità e un accordo sul livello di servizio (SLA) con disponibilità del 99,99%. DynamoDB offre anche funzionalità aggiuntive per aiutare a raggiungere gli obiettivi di continuità aziendale e disaster recovery.

DynamoDB include le seguenti funzionalità per supportare la resilienza dei dati e le esigenze di backup:

Funzionalità

- Tabelle globali
- Backup e ripristino continui point-in-time

- Backup e ripristino on demand

Tabelle globali

Le tabelle globali DynamoDB consentono uno SLA con disponibilità del 99,999% e una resilienza multi-Regione. Ciò consente di creare applicazioni resilienti e ottimizzarle per l'Obiettivo del tempo di ripristino (RTO) e l'Obiettivo del punto di ripristino (RPO) più bassi. Le tabelle globali si integrano anche con AWS Fault Injection Service (AWS FIS) per eseguire esperimenti di iniezione di guasti sui carichi di lavoro delle tabelle globali. Ad esempio, la sospensione della replica delle tabelle globali su qualsiasi tabella di replica.

Backup e ripristino continui point-in-time

I backup continui offrono una granularità al secondo e la possibilità di avviare un ripristino. point-in-time Con point-in-time il ripristino, è possibile ripristinare una tabella in qualsiasi momento, fino a un secondo negli ultimi 35 giorni. È possibile impostare il periodo di recupero su qualsiasi valore compreso tra 1 e 35 giorni.

I backup continui e l'avvio di un point-in-time ripristino non utilizzano la capacità assegnata. Inoltre, non hanno alcun impatto sulle prestazioni o sulla disponibilità delle applicazioni.

Backup e ripristino on demand

Backup e ripristino on demand consentono backup completi di una tabella per la conservazione e l'archiviazione a lungo termine per esigenze di conformità normativa. I backup non influiscono sulle prestazioni della tabella ed è possibile eseguire il backup di tabelle di qualsiasi dimensione. Grazie AWS Backup all'integrazione, puoi pianificare, copiare, AWS Backup etichettare e gestire automaticamente il ciclo di vita dei backup on-demand di DynamoDB. In questo modo è possibile copiare i backup su richiesta tra account e regioni e trasferire i backup più vecchi alla conservazione a freddo per ottimizzare i costi. AWS Backup

Accesso a DynamoDB

È possibile lavorare con DynamoDB utilizzando DynamoDBNoSQL Workbench per DynamoDB, Console di gestione AWSAWS Command Line Interfacethe o DynamoDB. APIs

Per ulteriori informazioni, consulta Accesso a DynamoDB.

Prezzi di DynamoDB

DynamoDB addebita la lettura, la scrittura e l'archiviazione dei dati nelle tabelle, oltre a qualsiasi funzionalità opzionale che si sceglie di abilitare. DynamoDB ha due modalità di capacità con le rispettive opzioni di fatturazione per le letture e le scritture sulle tabelle: on demand e con provisioning.

DynamoDB è incluso anche nel piano sempre gratuito, che offre 25 GB di archiviazione. Il piano sempre gratuito include anche 25 unità di capacità di scrittura e 25 unità di capacità di lettura (WCU, RCU) allocate, sufficienti per gestire 200 milioni di richieste al mese.

Per ulteriori informazioni, consulta Prezzi di Amazon DynamoDB.

Nozioni di base su DynamoDB

In caso di utilizzo di DynamoDB per la prima volta, si consiglia di iniziare con la lettura degli argomenti seguenti:

- Nozioni di base su DynamoDB: illustra il processo di configurazione di DynamoDB, creazione di tabelle di esempio e caricamento dei dati. Questo argomento fornisce anche informazioni sull'esecuzione di alcune operazioni di base del Console di gestione AWS database utilizzando NoSQL AWS CLI Workbench e DynamoDB. APIs
- DynamoDB core components: descrive i concetti di base di DynamoDB.
- Best practice per la progettazione e l'architettura con DynamoDB : fornisce consigli sulla progettazione NoSQL, su DynamoDB Well-Architected Lens, sulla progettazione di tabelle e su diverse altre funzionalità di DynamoDB. Queste best practice aiutano a massimizzare le prestazioni e ridurre al minimo i costi di throughput durante l'utilizzo di DynamoDB.

Ti consigliamo inoltre di consultare i seguenti tutorial che presentano end-to-end procedure complete per familiarizzare con DynamoDB. È possibile completare questi tutorial utilizzando la funzionalità del piano sempre gratuito.

- Creazione ed esecuzione di Query di una tabella NoSQL con Amazon DynamoDB
- Creazione di un'applicazione utilizzando un datastore chiave-valore NoSQL

Per informazioni su risorse, strumenti e strategie per la migrazione a DynamoDB, consulta l'argomento relativo. Per leggere i blog e i white paper più recenti, consulta Amazon DynamoDB resources.

Nozioni di base su DynamoDB

Le seguenti sezioni forniscono informazioni sulla connessione alle tabelle DynamoDB e la loro creazione e gestione.

Prima di iniziare, dovresti apprendere i concetti di base di Amazon DynamoDB. Sono disponibili una rapida panoramica in Che cos'è Amazon DynamoDB? e un approfondimento in Componenti principali di Amazon DynamoDB. Quindi, continua con Prerequisiti.

> **Note**
>
> Quando ti registri AWS, puoi iniziare a usare DynamoDB utilizzando AWS il piano gratuito. Se non sono ancora stati utilizzati tutti i vantaggi previsti dal piano gratuito per Amazon DynamoDB, non verrà addebitato alcun costo per gli esempi in questa sezione. In caso contrario, verranno addebitati i costi di utilizzo standard di DynamoDB dal momento della creazione fino all'eliminazione.
>
> Se non si desidera sottoscrivere un account per un piano gratuito, è possibile configurare DynamoDB locale (versione scaricabile) sul computer. La versione scaricabile consente di sviluppare e testare applicazioni localmente senza registrare un AWS account o accedere al servizio web DynamoDB.

Argomenti

- Risorse Amazon DynamoDB per gli utenti alle prime armi
- Accesso a DynamoDB
- Prerequisiti
- Configurazione di DynamoDB
- Fase 1: creare una tabella DynamoDB
- Fase 2: scrivere dati in una tabella DynamoDB
- Fase 3: leggere i dati da una tabella DynamoDB
- Fase 4: aggiornare i dati in una tabella
- Fase 5: eseguire query sui dati in una tabella DynamoDB
- Fase 6: (facoltativo) eliminare la tabella DynamoDB per eseguire la pulizia delle risorse

- Conoscenze aggiuntive su DynamoDB
- Genera codice di infrastruttura per Amazon DynamoDB utilizzando Console-to-Code

Risorse Amazon DynamoDB per gli utenti alle prime armi

Si consiglia agli utenti alle prime armi di iniziare a leggere le seguenti sezioni e di farvi riferimento ogni volta che è necessario.

- Punti salienti del servizio e prezzi – Nella pagina sui dettagli del prodotto sono disponibili la panoramica, i casi d'uso comuni, i punti salienti del servizio e i prezzi di DynamoDB.
- DynamoDB resources: la sezione delle risorse su DynamoDB contiene video, tutorial e i prontuari che introducono al servizio, ai concetti di modellazione dei dati e alle caratteristiche e funzionalità principali.
- Nozioni di base: la sezione Nozioni di base su DynamoDB include informazioni sulla configurazione di DynamoDB, sulla creazione di tabelle di esempio e sul caricamento dei dati. Fornisce inoltre informazioni sull'esecuzione di operazioni di base del database utilizzando la Console di gestione AWS, la AWS CLI, NoSQL Workbench e le API DynamoDB.
- Corso di base su DynamoDB: un corso digitale gratuito che insegna i fondamenti di DynamoDB, tra cui la progettazione di tabelle, i tipi di dati e le operazioni di base. Questo corso è ideale per gli sviluppatori che non conoscono i database DynamoDB e NoSQL.
- DynamoDB Nuggets: una raccolta di videotutorial brevi e mirati che spiegano i concetti e le funzionalità chiave di DynamoDB. Questi brevi video trattano argomenti come la modellazione dei dati, le strategie di partizionamento, i modelli di accesso e le best practice in un formato di facile comprensione.
- Repository di esempi di codice per DynamoDB: fornisce esempi pratici di codice per DynamoDB in vari linguaggi di programmazione.
- Formazione gratuita su DynamoDB: AWSoffre corsi di formazione digitali gratuiti che coprono concetti, funzionalità e best practice di DynamoDB. La serie di corsi approfonditi su DynamoDB aiuta ad apprendere a progettare modelli di dati efficienti, implementare chiavi di partizione efficaci e utilizzare gli indici secondari in modo efficace.
- NoSQL Workbench per DynamoDB: NoSQL Workbench è uno strumento visivo unificato che offre funzionalità di modellazione e visualizzazione dei dati, nonché di sviluppo di query. Consente di progettare, eseguire query e gestire tabelle DynamoDB, rendendolo particolarmente utile per imparare i concetti di modellazione dei dati e testare i modelli di accesso.

- Modelli di progettazione DynamoDB: Best practice ed esempi di modellazione dei dati che dimostrano le best practice per diversi casi d'uso. Include esempi pratici ed esempi di codice per l'implementazione di soluzioni efficaci.
- Tutorial pratici: tutorial dettagliati nella Console di gestione AWS che guidano l'utente nelle attività più comuni di DynamoDB, dalla creazione di tabelle all'implementazione di query complesse.
- Migrazione a DynamoDB: Migrazione a DynamoDB fornisce una panoramica del processo, degli strumenti e delle strategie per la migrazione di un database a DynamoDB.
- AWSWell-Architected Lens per DynamoDB: the section called "DynamoDB Well-Architected Lens" offre best practice architetturali per progettare e gestire applicazioni affidabili, sicure, efficienti e convenienti utilizzando DynamoDB. Include principi di progettazione, modelli di architettura e linee guida operative in linea con i sei pilastri del Well-Architected Framework.

Best practice aggiuntive di Amazon DynamoDB per gli utenti alle prime armi

Al termine delle sezioni precedenti, leggi queste sezioni:

- Capacità di throughput di DynamoDB

  Fornisce una panoramica delle due modalità di throughput disponibili per DynamoDB e considerazioni sulla selezione della modalità di capacità appropriata per l'applicazione. La modalità on demand è l'opzione di throughput predefinita e consigliata per la maggior parte dei carichi di lavoro DynamoDB.
- Best practice

  Identificazione e risoluzione di problemi per massimizzare le prestazioni e ridurre al minimo i costi durante l'utilizzo di DynamoDB.

Risorse AWS CLI

Se si desidera utilizzare l'Interfaccia a riga di comando AWS (AWS CLI), è possibile utilizzare questi documenti per iniziare:

- Documentazione di AWS CLI

  In questa sezione vengono fornite informazioni sul download della AWS CLI, come fare in modo che la AWS CLI lavori sul tuo sistema e fornendo le tue credenziali AWS.
- Documentazione della AWS CLI per DynamoDB

Questo documento distinto riguarda tutti i comandi della AWS CLI per DynamoDB, inclusi la sintassi e gli esempi.

Risorse di programmazione

È possibile scrivere programmi applicativi per utilizzare l'API DynamoDB con una serie di linguaggi di programmazione comuni. Ecco alcune risorse:

- Strumenti per Amazon Web Services

  AWS offre una serie di kit di sviluppo software (SDK, software development kits) che supportano DynamoDB. È possibile scrivere codice per DynamoDB utilizzando Java, .NET, PHP, Ruby e altri linguaggi. Gli SDK possono semplificare enormemente lo sviluppo delle applicazioni formattando le richieste a DynamoDB, analizzando le risposte e offrendo logica di ripetizione dei tentativi e gestione degli errori.
- Documentazione di riferimento dell'API DynamoDB

  Se non si intendono utilizzare gli SDK AWS, è possibile interagire direttamente con DynamoDB utilizzando l'API pertinente. Questo documento riguarda tutte le operazioni API DynamoDB, inclusi la sintassi e gli esempi. In questa sezione puoi trovare suggerimenti per risolvere i problemi e informazioni su come creare e autenticare richieste e gestire risposte.

Accesso a DynamoDB

Puoi accedere ad Amazon DynamoDB utilizzando, Console di gestione AWS the AWS CLI() o AWS Command Line Interface l'API DynamoDB.

Argomenti

- Utilizzo della console
- Utilizzando il AWS CLI
- Utilizzo dell'API
- Utilizzo di NoSQL Workbench per DynamoDB
- Intervalli di indirizzi IP
- Endpoint dual-stack per Internet Protocol versione 6 () IPv6

Utilizzo della console

Puoi accedere a Amazon DynamoDB da casahttps://console.aws.amazon.com/dynamodb/. Console di gestione AWS

Ecco alcune delle azioni che è possibile eseguire nella console DynamoDB:

- Gestione di tabelle: creazione, aggiornamento ed eliminazione di tabelle. Il calcolatore di capacità può essere utile per stimare i requisiti di capacità.
- Interazione con i dati: visualizzazione, aggiunta, aggiornamento ed eliminazione degli elementi presenti nelle tabelle. Gestione delle impostazioni del Time to Live (TTL).
- Monitoraggio e analisi: visualizzazione di dashboard, monitoraggio e configurazione di allarmi, nonché analisi di metriche e avvisi per le tabelle DynamoDB.
- Ottimizzazione ed estensione: gestione di indici secondari, flussi, trigger, capacità riservata e altre funzionalità avanzate per migliorare l'utilizzo di DynamoDB.

La console DynamoDB fornisce un'interfaccia completa per la gestione delle risorse DynamoDB. Si consiglia di accedere alla console e interagire con essa per saperne di più.

Utilizzando il AWS CLI

È possibile utilizzare il AWS Command Line Interface (AWS CLI) per controllare più AWS servizi dalla riga di comando e automatizzarli tramite script. È possibile utilizzarlo AWS CLI per operazioni ad hoc, come la creazione di una tabella. È possibile utilizzarla anche per incorporare operazioni Amazon DynamoDB all'interno di script di utilità.

Prima di poterlo utilizzare AWS CLI con DynamoDB, è necessario ottenere un ID della chiave di accesso e una chiave di accesso segreta. Per ulteriori informazioni, consulta Concessione dell'accesso programmatico .

Per un elenco completo di tutti i comandi disponibili per DynamoDB in, consulta AWS CLIAWS CLI il riferimento ai comandi.

Scaricamento e configurazione di AWS CLI

AWS CLI È disponibile all'indirizzo http://aws.amazon.com/cli. Viene eseguita su Windows, macOS o Linux. Dopo averlo scaricato AWS CLI, segui questi passaggi per installarlo e configurarlo:

1. Passa alla Guida per l'utente di AWS Command Line Interface.
2. Segui le istruzioni riportate in Installazione della AWS CLI e Configurazione della AWS CLI.

Utilizzo di AWS CLI con DynamoDB

Il formato della riga di comando è costituito dal nome dell'operazione DynamoDB seguito dai parametri per l'operazione. AWS CLI Supporta una sintassi abbreviata per i valori dei parametri, oltre a JSON.

Ad esempio, il seguente comando crea una tabella denominata Music. La chiave di partizione è Artist e la chiave di ordinamento è. SongTitle Per facilitare la leggibilità, in questa sezione i comandi lunghi sono suddivisi su righe separate.

```
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
```

I seguenti comandi aggiungono nuovi item alla tabella. In questi esempi viene utilizzata una combinazione di sintassi abbreviata e 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
```

Sulla riga di comando, può essere difficile comporre un JSON valido. Tuttavia, AWS CLI può leggere i file JSON. Considera ad esempio il seguente frammento di codice JSON, che viene memorizzato in un file denominato key-conditions.json:

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

È ora possibile emettere una richiesta `Query` utilizzando l' AWS CLI. In questo esempio il contenuto del file key-conditions.json viene utilizzato per il parametro `--key-conditions`:

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

Utilizzo di AWS CLI con DynamoDB local

AWS CLI Possono anche interagire con DynamoDB locale (versione scaricabile) in esecuzione sul computer. Per abilitare questo comportamento, aggiungi il seguente parametro a ciascun comando:

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

L'esempio seguente utilizza AWS CLI per elencare le tabelle in un database locale.

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

Se DynamoDB sta usando un numero di porta diverso da quello predefinito (8000), modificare di conseguenza il valore di `--endpoint-url`.

> **Note**
>
> Non AWS CLI possono utilizzare la versione locale di DynamoDB (versione scaricabile) come endpoint predefinito. Di conseguenza devi specificare `--endpoint-url` per ogni comando.

Utilizzo dell'API

Puoi usare Console di gestione AWS and the AWS Command Line Interface per lavorare in modo interattivo con Amazon DynamoDB. Tuttavia, per ottenere il massimo da DynamoDB, è possibile scrivere codice applicativo utilizzando. AWS SDKs

AWS SDKs Forniscono un ampio supporto per DynamoDB in JavaJavaScript , nelbrowser, .NET, Node.js,PHP,Python, Ruby, C++, Go, Android e iOS.

Prima di poterlo utilizzare AWS SDKs con DynamoDB, è necessario ottenere AWS un ID della chiave di accesso e una chiave di accesso segreta. Per ulteriori informazioni, consulta Configurazione di DynamoDB (servizio Web) .

Per una panoramica di alto livello sulla programmazione di applicazioni DynamoDB con, vedere. AWS SDKs Programmazione con DynamoDB e SDK AWS

Utilizzo di NoSQL Workbench per DynamoDB

È possibile accedere a DynamoDB anche scaricando e utilizzando NoSQL Workbench per DynamoDB.

NoSQL Workbench per Amazon DynamoDB è un'applicazione GUI lato client multipiattaforma per operazioni e sviluppo di database moderni. È disponibile per Windows, macOS e Linux. NoSQL Workbench è uno strumento visuale di sviluppo che offre funzionalità di modellazione, visualizzazione dei dati, nonché funzionalità di sviluppo di query che consentono di progettare, creare, eseguire query e gestire le tabelle DynamoDB. NoSQL Workbench ora include la versione locale di DynamoDB come parte opzionale del processo di installazione, il che semplifica la modellazione dei dati nella versione locale di DynamoDB. Per ulteriori informazioni sulla versione locale di DynamoDB e i relativi requisiti, consulta Configurazione di DynamoDB locale (versione scaricabile) .

> **Note**
>
> Il NoSQL Workbench per DynamoDB attualmente non AWS supporta gli accessi configurati con l'autenticazione a due fattori (2FA).

Modellazione dei dati

Grazie a NoSQL Workbench per DynamoDB, è possibile creare nuovi modelli di dati o progettare modelli in base ai modelli di dati esistenti che soddisfino i pattern di accesso ai dati delle applicazioni. Puoi anche importare ed esportare il modello di dati progettato alla fine del processo. Per ulteriori informazioni, consulta Creazione di modelli di dati con NoSQL Workbench.

Visualizzazione dei dati

Il visualizzatore del modello di dati fornisce un canvas in cui è possibile mappare query e visualizzare i pattern di accesso (facet) dell'applicazione senza dover scrivere codice. Ogni facet corrisponde a un pattern di accesso differente in DynamoDB. Puoi generare automaticamente dati di esempio da utilizzare nel modello di dati. Per ulteriori informazioni, consulta Visualizzazione dei modelli di accesso ai dati.

Creazione di operazioni

NoSQL Workbench offre una ricca intergaffia utente grafica per query di sviluppo e test. È possibile utilizzare Operation builder per visualizzare, esplorare ed eseguire query sui set di dati in tempo reale. È possibile anche utilizzare l'Operation builder strutturato per creare ed eseguire operazioni di data plane. Supporta le espressioni si proiezione e di condizione e consente di generare codice di esempio in più lingue. Per ulteriori informazioni, consulta Esplorazione di set di dati e creazione di operazioni con NoSQL Workbench.

Intervalli di indirizzi IP

Amazon Web Services (AWS) pubblica i propri intervalli di indirizzi IP correnti in formato JSON. Per vedere gli intervalli correnti, scarica ip-ranges.json. Per ulteriori informazioni, consulta Intervalli di indirizzi IP di AWS nella Riferimenti generali di AWS.

Per trovare gli intervalli di indirizzi IP che puoi usare per accedere alle tabelle e agli indici di DynamoDB, cerca nel file ip-ranges.json la seguente stringa: `"service": "DYNAMODB"`.

> **Note**
>
> Gli intervalli di indirizzi IP non si applicano a DynamoDB Streams o DynamoDB Accelerator (DAX).

Endpoint dual-stack per Internet Protocol versione 6 () IPv6

DynamoDB offre endpoint dual-stack compatibili con entrambi e. IPv4 IPv6 Le convenzioni di denominazione degli endpoint sono:

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

Per un elenco completo degli endpoint DynamoDB e della disponibilità regionale, consulta l'argomento Endpoint e quote di Amazon DynamoDB nella guida di riferimento generale.AWS

Per ulteriori informazioni sulla configurazione della AWS CLI per l'utilizzo di endpoint dual-stack, consulta la sezione Impostare l'uso degli endpoint dual-stack per tutti i servizi nella guida all'interfaccia a riga di comando. AWS AWS

Per ulteriori informazioni sulla configurazione dei client SDK per l'utilizzo di endpoint dual-stack, consulta l'argomento sugli endpoint Dual-stack e FIPS nella guida e agli strumenti.AWS SDKs

Prima di utilizzare DynamoDB IPv6 con, è necessario aggiornare il ruolo utente IAM o le policy basate sulle risorse utilizzate per il filtraggio degli indirizzi IP per includere gli intervalli di indirizzi. IPv6 Le politiche di filtraggio degli indirizzi IP che non tengono conto dell'indirizzo possono causare problemi di IPv6 accesso. Per ulteriori informazioni, vedere la sezione Operatori delle condizioni dell'indirizzo IP nella guida AWS Identity and Access Management.

Prerequisiti

Prima di iniziare con il tutorial per Amazon DynamoDB, consulta i modi in cui è possibile accedere a DynamoDB in Accesso a DynamoDB. Quindi, configura DynamoDB tramite il servizio web o la

versione scaricata in locale in Configurazione di DynamoDB . Dopodiché, continua con Fase 1: creare una tabella DynamoDB.

> **Note**
>
> - Se prevedi di interagire con DynamoDB solo tramite Console di gestione AWS, non hai bisogno AWS di una chiave di accesso. Completa la procedura descritta nella sezione Registrazione a AWS, quindi continua con. Fase 1: creare una tabella DynamoDB
> - Se non vuoi sottoscrivere un account per un piano gratuito, puoi configurare la versione locale di DynamoDB (versione scaricabile). Poi prosegui con Fase 1: creare una tabella DynamoDB.
> - Esistono differenze quando si lavora con i comandi CLI nei terminali su Linux e Windows. La seguente guida presenta i comandi formattati per i terminali Linux (incluso macOS) e i comandi formattati per Windows CMD. Scegli il comando più adatto all'applicazione terminale che intendi usare.

Configurazione di DynamoDB

Oltre al servizio web Amazon DynamoDB AWS , fornisce una versione scaricabile di DynamoDB che puoi eseguire sul tuo computer. La versione scaricabile è utile per sviluppare e testare il codice. La versione scaricabile consente di scrivere e testare applicazioni in locale, senza effettuare l'accesso al servizio Web DynamoDB.

Gli argomenti contenuti in questa sezione descrivono come configurare DynamoDB (versione scaricabile) e il servizio Web DynamoDB.

Argomenti

- Configurazione di DynamoDB (servizio Web)
- Configurazione di DynamoDB locale (versione scaricabile)

Configurazione di DynamoDB (servizio Web)

Per utilizzare il servizio Web Amazon DynamoDB:

1. Registrazione per AWS

2. Ottieni una chiave di AWS accesso (utilizzata per accedere a DynamoDB a livello di codice).

> **Note**
>
> Se prevedi di interagire con DynamoDB solo tramite Console di gestione AWS, non hai bisogno di AWS una chiave di accesso e puoi passare direttamente a. Utilizzo della console

3. Configura le credenziali (utilizzata per accedere a DynamoDB in modo programmatico).

Iscrizione a AWS

Per utilizzare il servizio DynamoDB, è necessario disporre di un account. AWS Se non disponi di un account, ti verrà chiesto di crearne uno durante la registrazione. Non ti viene addebitato alcun costo per AWS i servizi a cui ti iscrivi a meno che non li utilizzi.

Per iscriverti a AWS

1. Apri la https://portal.aws.amazon.com/billing/registrazione.
2. Segui le istruzioni online.

   Nel corso della procedura di registrazione riceverai una telefonata o un messaggio di testo e ti verrà chiesto di inserire un codice di verifica attraverso la tastiera del telefono.

   Quando ti iscrivi a un Account AWS, Utente root dell'account AWSviene creato un. L'utente root dispone dell'accesso a tutte le risorse e tutti i Servizi AWS nell'account. Come best practice di sicurezza, assegna l'accesso amministrativo a un utente e utilizza solo l'utente root per eseguire le attività che richiedono l'accesso di un utente root.

Concessione dell'accesso programmatico

Prima di poter accedere a DynamoDB a livello di codice o tramite AWS CLI(), è necessario disporre AWS Command Line Interface dell'accesso programmatico. Non è necessario un accesso programmatico se si intende utilizzare solo la console DynamoDB.

Gli utenti hanno bisogno dell'accesso programmatico se vogliono interagire con l'esterno di. AWS Console di gestione AWS Il modo per concedere l'accesso programmatico dipende dal tipo di utente che accede. AWS

Per fornire agli utenti l'accesso programmatico, scegli una delle seguenti opzioni.

Quale utente necessita dell'accesso programmatico?	Per	Come
IAM	(Consigliato) Utilizza le credenziali della console come credenziali temporanee per firmare le richieste programmatiche a,, o. AWS CLI AWS SDKs AWS APIs	Segui le istruzioni per l'interfaccia che desideri utilizzare. • Per la AWS CLI, consulta Login for AWS local development nella Guida per l'AWS Command Line Interface utente. • Per AWS SDKs, consulta Login for AWS local development nella AWS SDKs and Tools Reference Guide.
Identità della forza lavoro (Utenti gestiti nel centro identità IAM)	Utilizza credenziali temporanee per firmare le richieste programmatiche a AWS CLI, AWS SDKs, o. AWS APIs	Segui le istruzioni per l'interfaccia che desideri utilizzare. • Per la AWS CLI, vedere Configurazione dell'uso AWS IAM Identity Center nella AWS CLI Guida per l'utente.AWS Command Line Interface • Per AWS SDKs gli strumenti e AWS APIs, consulta l'autenticazione di IAM Identity Center nella Guida di riferimento AWS SDKs and Tools.
IAM	Utilizza credenziali temporanee per firmare le richieste	Seguendo le istruzioni riportate in Utilizzo delle credenziali temporanee con le

Quale utente necessita dell'accesso programmatico?	Per	Come
	programmatiche a AWS CLI, AWS SDKs, o. AWS APIs	AWS risorse nella Guida per l'utente IAM.
IAM	(Non consigliato) Utilizza credenziali a lungo termine per firmare richieste programmatiche a AWS CLI,, AWS SDKs o. AWS APIs	Segui le istruzioni per l'interfaccia che desideri utilizzare. • Per la AWS CLI, consulta Autenticazione tramite credenziali utente IAM nella Guida per l'utente.AWS Command Line Interface • Per gli strumenti AWS SDKs e gli strumenti, consulta Autenticazione tramite credenziali a lungo termine nella Guida di riferimento agli strumenti e agli AWS SDKs strumenti. • Per AWS APIs, consulta la sezione Gestione delle chiavi di accesso per gli utenti IAM nella Guida per l'utente IAM.

Configurazione delle credenziali

Prima di poter accedere a DynamoDB a livello di codice o tramite, è necessario configurare AWS CLI le credenziali per abilitare l'autorizzazione per le applicazioni.

Esistono vari modi per eseguire questa operazione. Ad esempio, puoi creare manualmente il file delle credenziali per archiviare l'ID chiave di accesso e la chiave di accesso segreta. È inoltre possibile utilizzare il AWS CLI comando per creare automaticamente il file`aws configure`. In alternativa, puoi utilizzare le variabili di ambiente. Per ulteriori informazioni sulla configurazione delle credenziali, consulta la guida per sviluppatori SDK specifica per la programmazione. AWS

Per installare e configurare il, consulta. AWS CLIUtilizzando il AWS CLI

Integrazione con altri servizi DynamoDB

Puoi integrare DynamoDB con molti altri servizi. AWS Per ulteriori informazioni, consulta gli argomenti seguenti:

- Utilizzo di DynamoDB con altri servizi AWS
- CloudFormation per DynamoDB
- Utilizzo di AWS Backup con Dynamo DB
- AWS Identity and Access Management (IAM) e DynamoDB
- Utilizzo di AWS Lambda con Amazon DynamoDB

Configurazione di DynamoDB locale (versione scaricabile)

Con la versione scaricabile di Amazon DynamoDB è possibile sviluppare e testare applicazioni senza effettuare l'accesso al servizio Web DynamoDB. Il database, invece, è autonomo sul tuo computer. Una volta pronti a implementare l'applicazione in produzione, rimuovere l'endpoint locale nel codice e puntare al servizio Web DynamoDB.

Questa versione locale ti aiuta a risparmiare sui costi di throughput, storage dei dati e trasferimento dei dati. Inoltre, lo sviluppo dell'applicazione non richiede una connessione a Internet.

La versione locale di DynamoDB è disponibile solo per il download (richiede JRE), come una dipendenza Apache Maven o come immagine Docker.

Se si preferisce utilizzare il servizio Web Amazon DynamoDB, consulta Configurazione di DynamoDB (servizio Web) .

Argomenti

- Distribuzione di DynamoDB in locale sul computer
- Note per l'utilizzo locale di DynamoDB
- Cronologia delle versioni di DynamoDB Local
- Telemetria in DynamoDB locale

Distribuzione di DynamoDB in locale sul computer

> **Note**
>
> - DynamoDB locale è disponibile in tre versioni: v3.x (Corrente), v2.x (Legacy) e v1.x (Obsoleta).
> - Si consiglia DynamoDB v3.x per test e sviluppo in locale.
> - La migrazione della versione di DynamoDB locale da V2.x a V3.x richiede l'aggiornamento delle istruzioni di importazione da `com.amazonaws.services.dynamodbv2` a `software.amazon.dynamodb` e l'aggiornamento delle dipendenze Maven per gli utenti Maven.
> - Se stai migrando un'applicazione che utilizza l'SDK per Java v1.x all'SDK per Java 2.x, segui la procedura per SDK AWS per Java 2.x.

Download di DynamoDB locale

Completare la procedura riportata di seguito per configurare ed eseguire DynamoDB sul proprio computer.

Come configurare DynamoDB sul computer

1. Scarica DynamoDB locale gratuitamente da una delle seguenti posizioni.

Collegamenti per il download	checksum
.tar.gz | .zip	.tar.gz.sha256 | .zip.sha256

> **Important**
>
> Per eseguire DynamoDB v2.6.0 sul computer è necessaria la versione 17.x o successiva dell'ambiente di runtime Java (JRE). L'applicazione non verrà eseguita nelle versioni di JRE precedenti.

2. Dopo aver scaricato l'archivio, estrai i contenuti e copia la directory estratta in una ubicazione a scelta.

3. Per avviare DynamoDB sul computer, aprire una finestra del prompt dei comandi, spostarsi nella directory in cui è stato estratto `DynamoDBLocal.jar`, quindi immettere il comando seguente.

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

> **Note**
>
> Se usi Windows PowerShell, assicurati di includere il nome del parametro o l'intero nome e valore in questo modo:
> `java -D"java.library.path=./DynamoDBLocal_lib" -jar DynamoDBLocal.jar`
> DynamoDB esegue le richieste in arrivo fino al suo arresto. Per arrestare DynamoDB, digitare Ctrl+C al prompt dei comandi.
> DynamoDB utilizza la porta 8000 per impostazione predefinita. Se la porta 8000 non è disponibile, il comando genera un'eccezione. Per un elenco completo delle opzioni di runtime di DynamoDB, incluso `-port`, immettere il comando seguente.
> `java -Djava.library.path=./DynamoDBLocal_lib -jar DynamoDBLocal.jar -help`

4. Prima di poter accedere a AWS Command Line Interface in modo programmatico o tramite (AWS CLI), è necessario configurare le credenziali per abilitare l'autorizzazione per le applicazioni. Perché possa funzionare, la versione scaricabile di DynamoDB richiede le credenziali, come mostrato nell'esempio seguente.

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

   È possibile utilizzare il comando `aws configure` di AWS CLI per impostare le credenziali. Per ulteriori informazioni, consulta Utilizzando il AWS CLI.

5. Inizia a scrivere le applicazioni. Per accedere a DynamoDB in esecuzione localmente con, utilizzare AWS CLI il parametro. `--endpoint-url` Ad esempio, per elencare le tabelle DynamoDB utilizzare il seguente comando:

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

Esecuzione di DynamoDB locale come immagine Docker

La versione scaricabile di Amazon DynamoDB è disponibile come parte dell'immagine Docker. Per ulteriori informazioni, consulta dynamodb-local. Per visualizzare la versione corrente di DynamoDB locale, immetti il seguente comando:

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

Per un esempio di utilizzo di DynamoDB local come parte di un'applicazione REST basata su AWS SAM(), consulta Applicazione SAM DynamoDB per AWS Serverless Application Model la gestione degli ordini. Questa applicazione di esempio mostra come utilizzare DynamoDB locale per i test.

Se desideri eseguire un'applicazione multi-container che utilizza anche il container DynamoDB Local, utilizza Docker Compose per definire ed eseguire tutti i servizi nell'applicazione, incluso DynamoDB Local.

Per installare ed eseguire DynamoDB in locale con Docker Compose:

1. Scarica e installa Docker Desktop.
2. Copiare il codice seguente in un file e salvarlo con nome `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
   ```

   Se vuoi che l'applicazione e DynamoDB locale si trovino in container separati, utilizza il file yaml seguente.

   ```
   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
```

Lo script docker-compose.yml crea un container `app-node` e un container `dynamodb-local`. Lo script esegue un comando nel container `app-node` che utilizza la AWS CLI per connettersi al container `dynamodb-local` e descrive i limiti di account e tabelle.

Per utilizzarlo con la propria immagine dell'applicazione, sostituire il valore `image` nell'esempio seguente con quello dell'applicazione:

```
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**
>
> Gli script YAML richiedono di specificare una chiave di AWS accesso e una chiave AWS segreta, ma non devono essere AWS chiavi valide per accedere a DynamoDB locale.

3. Emettere il seguente comando dalla riga di comando:

```
docker-compose up
```

Esecuzione di DynamoDB locale come dipendenza Apache Maven

> **Note**
>
> Se stai migrando un'applicazione che utilizza l'SDK per Java v1.x all'SDK per Java 2.x, segui la procedura per SDK AWS per Java 2.x.

Completare la seguente procedura per utilizzare Amazon DynamoDB nell'applicazione come dipendenza.

Come distribuire DynamoDB come repository Apache Maven

1. Scarica e installa Apache Maven. Per ulteriori informazioni, consulta le pagine relative al download di Apache Maven e all'installazione di Apache Maven.
2. Aggiungi il repository Maven DynamoDB al file POM (Project Object Model) dell'applicazione:

```
<!--Dependency:-->
<dependencies>
   <dependency>
```

```
      <groupId>com.amazonaws</groupId>
      <artifactId>DynamoDBLocal</artifactId>
      <version>3.1.0</version>
   </dependency>
</dependencies>
```

Modello di esempio da utilizzare con Spring Boot 3 Spring Framework 6: and/or

```
<?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**
>
> Puoi anche utilizzare l'URL di Maven Central Repository.

Esegui DynamoDB in locale in AWS CloudShell

AWS CloudShell è una shell preautenticata basata su browser che puoi avviare direttamente da. Console di gestione AWSÈ possibile accedere AWS CloudShell da diversi modi Console di gestione AWS . Per ulteriori informazioni, consulta Guida introduttiva AWS CloudShell.

Segui questi passaggi per eseguire DynamoDB localmente ovunque AWS CloudShell nel. Console di gestione AWS

Per eseguire DynamoDB in locale nel AWS CloudShellConsole di gestione AWS

1. Avvia AWS CloudShell dall'interfaccia della console, scegli una shell disponibile Regione AWS e passa alla shell preferita, come Bash o la shell PowerShell Z.
2. Per sceglierne una Regione AWS, vai al menu Seleziona una regione e seleziona una supportata Regione AWS. (Le Regioni disponibili sono evidenziate.)
3. Da Console di gestione AWS, avvia AWS CloudShell scegliendo una delle seguenti opzioni:
   a. Nella barra di navigazione seleziona l'icona AWS CloudShell.
   b. Nella casella Cerca, inserisci la parola CloudShell, quindi scegli CloudShell.
   c. Nel widget Visitato di recente, scegli CloudShell.
   d. Dalla barra degli strumenti della console, scegli CloudShell.
4. Per eseguire DynamoDB AWS CloudShell in locale puoi usare l'alias. `dynamodb-local` È possibile specificare opzioni aggiuntive della riga di comando per modificare le impostazioni di DynamoDB locale. Consulta the section called "Note per l'utilizzo" per conoscere le opzioni disponibili.

   > **Note**
   >
   > Per eseguire DynamoDB in locale in background, esegui DynamoDB local utilizzando:. AWS CloudShell `dynamodb-local &`

5. Per accedere a DynamoDB AWS CloudShell in esecuzione localmente con, utilizzare AWS CLI il parametro. `--endpoint-url` Ad esempio, per elencare le tabelle DynamoDB utilizza il seguente comando:

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

Per un esempio di progetto che mostra diversi approcci per configurare e utilizzare DynamoDB locale, tra cui il download di file JAR, l'esecuzione come immagine Docker e l'utilizzo come dipendenza Maven, consulta DynamoDB Local Sample Java Project.

Note per l'utilizzo locale di DynamoDB

A eccezione dell'endpoint, le applicazioni eseguite con la versione scaricabile di Amazon DynamoDB dovrebbero funzionare anche con il servizio Web DynamoDB. Tuttavia, quando si utilizza DynamoDB in locale, è necessario tenere presente quanto riportato di seguito:

- Se si utilizza l'*-sharedDb*opzione, DynamoDB crea un singolo file di database denominato .db. shared-local-instance Tutti i programmi che si connettono a DynamoDB hanno accesso a questo file. Se elimini il file, tutti i dati in esso archiviati andranno persi.
- Se si omette`-sharedDb`, il file di database viene denominato myaccesskeyid_region.db, con l'ID della chiave di AWS accesso e la AWS regione così come appaiono nella configurazione dell'applicazione. Se elimini il file, tutti i dati in esso archiviati andranno persi.
- Se si utilizza l'opzione `-inMemory`, DynamoDB non scrive alcun file di database. Tutti i dati verranno invece scritti in memoria e non verranno salvati all'arresto di DynamoDB.
- Se si utilizza l'opzione `-inMemory`, è richiesta anche l'opzione `-sharedDb`.
- Se si utilizza l'opzione `-optimizeDbBeforeStartup`, è necessario specificare anche il parametro `-dbPath`, in modo che DynamoDB sia in grado di individuare il file di database.
- AWS SDKs Per DynamoDB è necessario che la configurazione dell'applicazione specifichi un valore della chiave di accesso e AWS un valore Region. A meno che non stia utilizzando l'opzione `-sharedDb` o `-inMemory`, DynamoDB usa questi valori per denominare il file di database locale. Questi valori non devono essere necessariamente validi per essere eseguiti AWS localmente. Tuttavia, potresti trovare utile utilizzare valori validi in modo da poter eseguire il codice nel cloud successivamente, modificando solo l'endpoint in uso.
- La versione locale di DynamoDB restituisce sempre un valore null per `billingModeSummary.`
- DynamoDB local `AWS_ACCESS_KEY_ID` può contenere solo lettere (A–Z, a–z) e numeri (0-9).
- DynamoDB local non Point-in-time supporta il ripristino (PITR).

Argomenti

- Opzioni della riga di comando
- Impostazione dell'endpoint locale
- Differenze tra DynamoDB scaricabile e il servizio Web DynamoDB

Opzioni della riga di comando

Con la versione scaricabile di DynamoDB è possibile utilizzare le opzioni della riga di comando seguenti:

- `-corsvalue`— Abilita il supporto per la condivisione di risorse tra origini diverse (CORS) per. JavaScript È necessario fornire un elenco di domini specifici "consentiti" separati da virgole. L'impostazione predefinita di `-cors` è un asterisco (*), che permette l'accesso pubblico.
- `-dbPath value`: la directory in cui DynamoDB scrive il file di database. Se non specifichi questa opzione, il file verrà scritto nella directory corrente. Non puoi specificare sia `-dbPath` che `-inMemory` contemporaneamente.
- `-delayTransientStatuses`: fa sì che DynamoDB introduca ritardi per determinate operazioni. DynamoDB (versione scaricabile) può eseguire alcune attività quasi istantaneamente, come le operazioni su tabelle e indici. create/update/delete Tuttavia, il servizio DynamoDB richiede più tempo per queste attività. L'impostazione di questo parametro consente a DynamoDB in esecuzione sul computer di simulare più da vicino il comportamento del servizio Web DynamoDB. Attualmente, questo parametro causa ritardi solo per i Global Secondary Indexes in stato CREATING o DELETING.
- `-help`: stampa un riepilogo di utilizzo e le opzioni.
- `-inMemory`: DynamoDB viene eseguito in memoria invece di utilizzare un file di database. Quando si arresta DynamoDB, i dati non verranno salvati. Non puoi specificare sia `-dbPath` che `-inMemory` contemporaneamente.
- `-optimizeDbBeforeStartup`: ottimizza le tabelle di database sottostanti prima di avviare DynamoDB sul computer. Inoltre, quando utilizzi questo parametro, è necessario specificare `-dbPath`.
- `-port value`: il numero di porta utilizzato da DynamoDB per comunicare con l'applicazione. Se non viene specificata questa opzione, la porta predefinita è `8000`.

> **Note**
>
> DynamoDB utilizza la porta 8000 per impostazione predefinita. Se la porta 8000 non è disponibile, il comando genera un'eccezione. Puoi utilizzare l'opzione `-port` per specificare un numero di porta diverso. Per un elenco completo delle opzioni di runtime di DynamoDB, tra cui `-port`, digitare il comando seguente:
> `java -Djava.library.path=./DynamoDBLocal_lib -jar DynamoDBLocal.jar -help`

- `-sharedDb`: se si specifica `-sharedDb`, DynamoDB utilizza un singolo file di database anziché file separati per ogni credenziale e regione.
- `-disableTelemetry`: quando specificato, DynamoDB locale non invierà alcuna telemetria.
- `-version`: stampa la versione di DynamoDB locale.

Impostazione dell'endpoint locale

Per impostazione predefinita, gli strumenti AWS SDKs and utilizzano gli endpoint per il servizio Web Amazon DynamoDB. Per utilizzare gli strumenti SDKs e con la versione scaricabile di DynamoDB, è necessario specificare l'endpoint locale:

`http://localhost:8000`

AWS Command Line Interface

Puoi usare il AWS Command Line Interface (AWS CLI) per interagire con DynamoDB scaricabile.

Per accedere a DynamoDB eseguito in locale, utilizza il parametro `--endpoint-url`. Di seguito è riportato un esempio di utilizzo di AWS CLI per elencare le tabelle in DynamoDB sul computer.

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

> **Note**
>
> Non AWS CLI possono utilizzare la versione scaricabile di DynamoDB come endpoint predefinito. Pertanto, è necessario specificare con ogni comando. `--endpoint-url` AWS CLI

AWS SDKs

Il modo in cui viene specificato un endpoint dipende dal linguaggio di programmazione e dall'SDK AWS in uso. Nelle seguenti sezioni viene descritto come procedere:

- Java: Configurazione dell'endpoint e della regione AWS(DynamoDB local supporta l'SDK AWS per Java V1 e V2)
- CodeSamples.Java. RegionAndEndpoint .NET: Configurazione dell'endpoint e della regione AWS

Differenze tra DynamoDB scaricabile e il servizio Web DynamoDB

La versione scaricabile di DynamoDB è destinata solo a scopi di sviluppo e test. Per contro, il servizio Web DynamoDB è un servizio gestito con caratteristiche di scalabilità, disponibilità e durabilità che lo rendono ideale per l'uso in produzione.

Le differenze tra la versione scaricabile di DynamoDB e il servizio Web sono le seguenti:

- Regioni AWS e distinct non Account AWS sono supportati a livello di client.
- Le impostazioni di velocità effettiva assegnata vengono ignorate nella versione scaricabile di DynamoDB, sebbene siano richieste dall'operazione `CreateTable`. In `CreateTable`, puoi specificare i numeri che preferisci di throughput di lettura e scrittura assegnato, sebbene questi numeri non vengano utilizzati. Puoi chiamare `UpdateTable` tutte le volte al giorno che desideri. Tuttavia, le modifiche ai valori di throughput assegnato vengono ignorate.
- Le operazioni `Scan` vengono eseguite in sequenza. Le scansioni in parallelo non sono supportate. I parametri `Segment` e `TotalSegments` dell'operazione `Scan` vengono ignorati.
- La velocità delle operazioni di lettura e scrittura sui dati della tabella è limitata solo dalla velocità del computer. Le operazioni `CreateTable`, `UpdateTable` e `DeleteTable` hanno luogo immediatamente e lo stato della tabella è sempre ACTIVE. Le operazioni `UpdateTable` che modificano solo le impostazioni di throughput di cui è stato effettuato il provisioning sulle tabelle o sui Global Secondary Indexes si verificano immediatamente. Se un'operazione `UpdateTable` crea o elimina alcuni indici secondari globali, tali indici passeranno attraverso gli stati normali (come CREATING e DELETING, rispettivamente) prima di raggiungere lo stato ACTIVE. La tabella rimane in stato ACTIVE (Attivo) durante questa fase.
- Le operazioni di lettura hanno consistenza finale. Tuttavia, a causa della velocità dell’esecuzione di DynamoDB locale sul computer, la maggior parte delle letture risulta a elevata consistenza.
- I parametri di raccolta di item e le dimensioni di raccolta di item non vengono monitorati. Nelle risposte delle operazioni, vengono restituiti valori null anziché parametri di raccolta di item.
- In DynamoDB, il limite di dati restituiti per ogni set di risultati è 1 MB. Sia il servizio Web DynamoDB che la versione scaricabile impongono questo limite. Tuttavia, durante l'esecuzione di una query su un indice, il servizio DynamoDB calcola solo la dimensione della chiave e degli attributi proiettati. Per contro, la versione scaricabile di DynamoDB calcola la dimensione dell'intero elemento.
- Se si utilizzano i flussi DynamoDB, la velocità di creazione degli shard potrebbe variare. Nel servizio Web DynamoDB, il comportamento della creazione di shard è in parte influenzato dall'attività di partizionamento della tabella. Quando si esegue DynamoDB in locale, non si verifica

un partizionamento di tabella. Nei due casi, gli shard sono temporanei, per cui l'applicazione non deve dipendere dal comportamento degli shard.

- `TransactionConflictExceptions`non vengono generati da DynamoDB scaricabile per transazioni. APIs Ti consigliamo di utilizzare un framework fittizio Java per simulare `TransactionConflictExceptions` nel gestore DynamoDB per testare la modalità di risposta dell'applicazione alle transazioni in conflitto.
- Nel servizio web DynamoDB, indipendentemente dal fatto che l'accesso avvenga tramite la console o tramite, i nomi delle tabelle fanno distinzione AWS CLI tra maiuscole e minuscole. Una tabella denominata `Authors` e una denominata `authors` possono esistere entrambe come tabelle separate. Nella versione scaricabile, i nomi di tabella rispettano la distinzione tra maiuscole e minuscole e il tentativo di creare queste due tabelle genera un errore.
- L'assegnazione di tag non è supportata nella versione scaricabile di DynamoDB.
- La versione scaricabile di DynamoDB ignora il parametro Limit in. ExecuteStatement

Cronologia delle versioni di DynamoDB Local

Nella tabella seguente sono descritte le modifiche importanti apportate a ogni nuova versione di DynamoDB local.

Versione	Modifica	Descrizione	Data
3.1.0	Miglioramento delle prestazioni per le query PartiQL, inclusa la dipendenza Joda-time	• Aggiornamento della versione dell'SDK per Java all'ultima versione pubblica da 2.25.50 a 2.33.0 • Inclusa la dipendenza Joda-time all'interno del file Pom.xml • Miglioramento delle prestazioni per le query PartiQL	14 settembre 2025

Versione	Modifica	Descrizione	Data
		• Aggiornamento delle dipendenze per risolvere più problemi di vulnerabilità CVE	
3.0.0	Migrazione da AWS SDK Java V1 a V2	• Migrazione da SDK Java V1 a AWS V2 • Struttura del pacchetto aggiornata da `com.amazonaws.services.dynamodbv2` a `software.amazon.dynamodb.services` • Dipendenze SDK Java V1 rimosse AWS	17 luglio 2025

Versione	Modifica	Descrizione	Data
2.6.0	Supporta la tabella ARN come nome della tabella in DynamoDB APIs Correzione delle prestazioni e aggiornamenti di sicurezza	• Aggiunto il supporto per l'utilizzo della tabella ARN come nome di tabella in diversi DynamoDB APIs • Correzione di bug per `CreateStreamTable` su macchine ad alte prestazioni, come Mac M3 • Aggiornamento delle dipendenze per risolvere i problemi di vulnerabilità (CVE-2022-49043,CVE-2024-56732, CVE-2020-29582, CVE-2025-21502, CVE-2024-50602, CVE-2025-24970, CVE-2025-25193)	13 marzo 2025

Versione	Modifica	Descrizione	Data
2.5.4	Aggiornamento a dipendenze Jetty	• Aggiornamento da Jetty 12.0.8 a Jetty 12.0.14 (Risolve CVE-2024-6763, CVE-2024-8184, CVE-2024-47535) Mitigazione per (CVE-2024-21634)	12 dicembre 2024
2.5.3	Aggiornamento delle dipendenze Jackson alla versione 2.17.x in Log4j Core (Risolve CVE-2022-1471)	• Aggiornamento delle dipendenze Jackson alla versione 2.17.x in Log4j Core (Risolve CVE-2022-1471) per risolvere una vulnerabilità di sicurezza critica nella libreria SnakeYAML, che è una dipendenza transitiva	6 novembre 2024

Versione	Modifica	Descrizione	Data
2.5.2	Correzione di bug per il flusso di lavoro di aggiornamento delle tabelle	• Correzione di bug per il flusso di lavoro quando la tabella di aggiornamento tenta di aggiornare la tabella con Modalità di fatturazione on demand a Con provisioning con un indice secondario globale (GSI)	20 giugno 2024
2.5.1	Patch per i bug introdotti nella funzionalità `OndemandThroughPut`	• Correzione di alcuni bug relativi a `OndemandThroughPut`	5 giugno 2024
2.5.0	Support per un throughput massimo configurabile per tabelle on demand, `ReturnValuesOnConditionCheckFailure`, `BatchExecuteStatement` e `ExecuteTransaction Request`	• Aggiunta di telemetria alla modalità con embedding • Correzione della traduzione per SDKv2 `ConditionalCheckException`	28 maggio 2024

Versione	Modifica	Descrizione	Data
2.4.0	Supporto per `ReturnValuesOnConditionCheckFailure` – Modalità con embedding	• Correzione della modalità con embedding per `TrimmedDataAccessException` per il funzionamento su più flussi • Correzione della traduzione delle eccezioni SDKv2 in modalità incorporata	17 aprile 2024
2.3.0	Aggiornamento di Jetty e JDK	• Aggiornamento a Jetty 12.0.2 • Aggiornamento a JDK 17 • Aggiornamento alla versione 4.10.1 ANTLR4	14 marzo 2024

Versione	Modifica	Descrizione	Data
2.2.0	Aggiunta del supporto per la protezione dall'eliminazione delle tabelle e per il parametro `ReturnValuesOnConditionCheckFailure`	• Aggiunta del supporto per la protezione da eliminazione delle tabelle • È stato aggiunto il supporto per ReturnValuesOnConditionCheckFailure • Aggiunta del supporto per il flag -version	14 dicembre 2023
2.1.0	Support per progetti SQLLite Native Libraries for Maven e aggiunta di telemetria	• Aggiunta di telemetria a DynamoDB locale • Copia dinamicamente le librerie native per i progetti Maven SQLLite • Libreria io.github.ganadist.sqlite4java rimossa dalla dipendenza Maven • Aggiornamento a 32.1.1-jre GoogleGuava	23 ottobre 2023

Versione	Modifica	Descrizione	Data
2.0.0	Migrazione da javax a jakarta namespace e Support JDK11	• Migrazione da javax a jakarta namespace e supporto JDK11 • Correzione della gestione dell'acce sso non valido e della chiave segreta allo startup del server • Correzione delle vulnerabilità identificate da Maven aggiornando le dipendenze	5 luglio 2023
1.25.1	Aggiornamento delle dipendenze Jackson alla versione 2.17.x in Log4j Core (Risolve CVE-2022-1471)	Aggiornamento delle dipendenze Jackson alla versione 2.17.x in Log4j Core (Risolve CVE-2022-1471) per risolvere una vulnerabilità di sicurezza critica nella libreria SnakeYAML, che è una dipendenza transitiva	6 novembre 2024

Versione	Modifica	Descrizione	Data
1.25.0	Aggiunta del supporto per la protezione dall'eliminazione delle tabelle e per il parametro `ReturnValuesOnConditionCheckFailure`	• Aggiunta del supporto per la protezione da eliminazione delle tabelle • Aggiunto supporto per ReturnValuesOnConditionCheckFailure • Aggiunta del supporto per il flag -version	18 dicembre 2023
1.24.0	Support per progetti SQLLite Native Libraries for Maven e aggiunta di telemetria	• Aggiunta di telemetria a DynamoDB locale • Copia dinamicamente le librerie native per i progetti Maven SQLLite • Libreria io.github.ganadist.sqlite4java rimossa dalla dipendenza Maven • Aggiornamento a 32.1.1-jre GoogleGuava	23 ottobre 2023

Versione	Modifica	Descrizione	Data
1.23.0	Gestione dell'accesso non valido e chiave segreta all'avvio del server	• Correzione della gestione dell'acce sso non valido e della chiave segreta allo startup del server • Correzione delle vulnerabilità identificate da Maven aggiornando le dipendenze	28 giugno 2023
1.22.0	Supporto di Limit Operation per PartiQL	• Ottimizza la clausola IN per PartiQL • Supporto per Limit Operation • Supporto M1 per progetti Maven	8 giugno 2023
1.21.0	Supporto per 100 azioni per transazione	• Azioni per transazio ne incrementate da 25 a 100 • Aggiornamento dell'immagine docker Open JDK a 11 • Correzione della parità per l'eccezione generata quando si duplicano elementi in BatchExec uteStatement	26 gennaio 2023

Versione	Modifica	Descrizione	Data
1.20.0	Aggiunto il supporto per M1 Mac	• Aggiunto il supporto per M1 Mac • Aggiornamento della dipendenza Jetty a 9.4.48.v2 0220622	12 settembre 2022
1.19.0	Aggiornato il parser PartiQL	Aggiornato il parser PartiQL e altre librerie correlate	27 luglio 2022
1.18.0	Aggiornati log4j-core e Jackson-core	Aggiornato log4j-core a 2.17.1 e Jackson-c ore 2.10.x a 2.12.0	10 gennaio 2022
1.17.2	Aggiornato log4j-core	Aggiornata dipendenz a log4j-core alla versione 2.16	16 gennaio 2021
1.17.1	Aggiornato log4j-core	Aggiornata dipendenz a log4j-core per correggere l'exploit zero-day per impedire l'esecuzione di codice in modalità remota - Log4Shel	10 gennaio 2021
1.17.0	Javascript Web Shell obsoleta	• Aggiornata la dipendenza dell' AWS SDK a AWS SDK for Java 1.12.x • Javascript Web Shell obsoleta	8 gennaio 2021

Telemetria in DynamoDB locale

In AWS, sviluppiamo e lanciamo servizi basati su ciò che apprendiamo dalle interazioni con i clienti e utilizziamo il feedback dei clienti per iterare sui nostri prodotti. La telemetria offre informazioni aggiuntive che ci consentono di comprendere meglio le necessità dei clienti, diagnosticare i problemi e fornire funzionalità per migliorare l'esperienza cliente.

DynamoDB locale raccoglie dati di telemetria, come metriche di utilizzo generiche, informazioni sui sistemi e sull'ambiente ed errori. Per informazioni dettagliate sui tipi di telemetria raccolti, consulta. Tipo di informazioni da raccogliere

DynamoDB locale non raccoglie informazioni personali, come nomi utente o indirizzi e-mail. Inoltre, non estrae informazioni sensibili a livello di progetto.

In qualità di cliente, puoi controllare se la telemetria è attivata e puoi modificare le impostazioni in qualsiasi momento. Se la telemetria resta attiva, DynamoDB locale invia i dati di telemetria in background senza richiedere alcuna interazione aggiuntiva al cliente.

Disattivazione della telemetria utilizzando le opzioni della riga di comando

Puoi disattivare la telemetria utilizzando le opzioni della riga di comando all'avvio di DynamoDB locale utilizzando l'opzione `-disableTelemetry`. Per ulteriori informazioni, consulta Opzioni della riga di comando .

Disattivazione della telemetria per una singola sessione

Nei sistemi operativi macOS e Linux, puoi disattivare la telemetria per una singola sessione. Per disattivare la telemetria per la sessione corrente, esegui il comando seguente per impostare la variabile di ambiente `DDB_LOCAL_TELEMETRY` su `false`. Ripeti il comando per ogni nuovo terminale o sessione.

```
export DDB_LOCAL_TELEMETRY=0
```

Disattivazione della telemetria per il tuo profilo in tutte le sessioni

Esegui i seguenti comandi per disattivare la telemetria per tutte le sessioni quando esegui DynamoDB locale sul tuo sistema operativo.

Per disattivare la telemetria in Linux

1. Esegui:

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

2. Esegui:

```
source ~/.profile
```

Per disattivare la telemetria in Linux

1. Esegui:

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

2. Esegui:

```
source ~/.profile
```

Per disattivare la telemetria in Linux

1. Esegui:

```
setx DDB_LOCAL_TELEMETRY 0
```

2. Esegui:

```
refreshenv
```

Disattivazione della telemetria con DynamoDB locale con embedding in progetti Maven

È possibile disattivare la telemetria utilizzando DynamoDB locale con embedding nei progetti Maven.

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

// AWS SDK v2
```

```
DynamoDbClient ddbClientSDKv2Local =
 DynamoDBEmbedded.create(disableTelemetry).dynamoDbClient();
```

Tipo di informazioni da raccogliere

- Informazioni sull'utilizzo: la telemetria generica come l'avvio/arresto del server e l'API o l'operazione chiamate.
- Informazioni sul sistema e sull'ambiente: la versione Java, il sistema operativo (Windows, Linux o macOS), l'ambiente in cui viene eseguito DynamoDB locale (ad esempio, JAR indipendente, container Docker o come dipendenza Maven) e i valori hash degli attributi di utilizzo.

Ulteriori informazioni

I dati di telemetria raccolti da DynamoDB locale rispettano le policy sulla privacy dei dati di AWS. Per ulteriori informazioni, consulta gli argomenti seguenti:

- Termini del servizio AWS
- Domande frequenti sulla privacy dei dati

Fase 1: creare una tabella DynamoDB

In questa fase, viene creata una tabella `Music` in Amazon DynamoDB. La tabella presenta i seguenti dettagli:

- Chiave di partizione: `Artist`
- Chiave di ordinamento: `SongTitle`

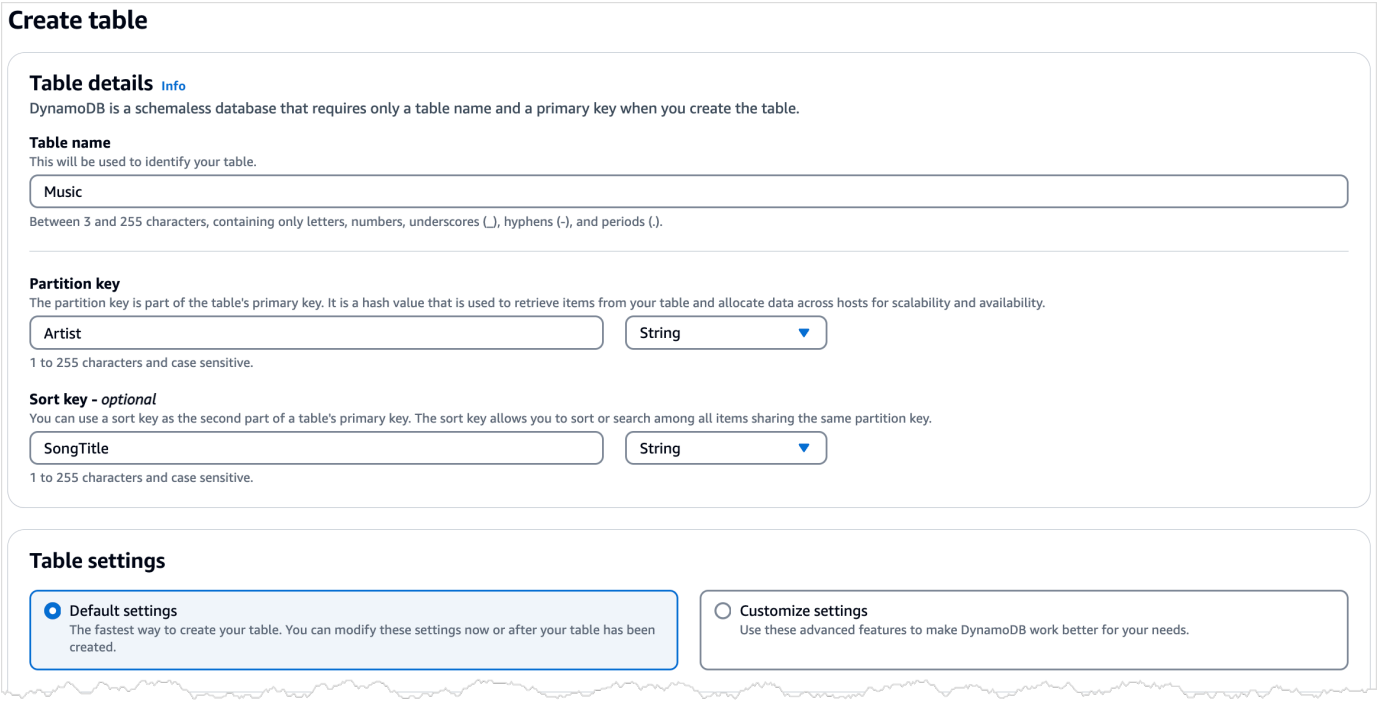
Per ulteriori informazioni sulle operazioni delle tabelle, consulta Utilizzo di tabelle e dati in DynamoDB.

> **Note**
>
> Prima di iniziare, accertati di aver seguito i passaggi in Prerequisiti.

Console di gestione AWS

Per creare una nuova tabella `Music` utilizzando la console DynamoDB:

1. Accedi Console di gestione AWS e apri la console DynamoDB all'indirizzo. https://console.aws.amazon.com/dynamodb/
2. Nel riquadro di navigazione a sinistra, selezionare Tables (Tabelle).
3. Scegliere Create table (Crea tabella).
4. Inserisci i Dettagli tabella come segue:
   a. Nel campo Table name (Nome tabella) immetti **`Music`**.
   b. In Partition key (Chiave di partizione), inserisci **`Artist`**.
   c. Per Chiave di ordinamento, immetti **`SongTitle`**.
5. Per Impostazioni tabella, mantieni la selezione predefinita di Impostazioni predefinite.
6. Per creare la tabella seleziona Crea tabella.



7. Una volta che la tabella è nello stato `ACTIVE`, si consiglia di abilitare Backup point-in-time per DynamoDB sulla tabella eseguendo la procedura seguente:
   a. Seleziona il nome della tabella per aprirla.
   b. Seleziona Backup.
   c. Scegli Modifica nella sezione di Point-in-time ripristino (PITR).
   d. Nella pagina Modifica impostazioni point-in-time di ripristino, scegli Attiva il point-in-time ripristino.
   e. Scegli Save changes (Salva modifiche).

AWS CLI

L' AWS CLI esempio seguente crea una nuova `Music` tabella utilizzando`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
```

Windows CMD

```
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
```

Utilizzando `create-table` viene restituito il seguente risultato di esempio.

```
{
    "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"
        }
    }
}
}
```

Da notare che il valore del campo `TableStatus` è impostato su `CREATING`.

Per verificare se DynamoDB ha terminato la creazione della tabella `Music`, utilizza il comando `describe-table`.

Linux

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

Windows CMD

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

Questo comando restituisce il seguente risultato. Quando DynamoDB termina la creazione della tabella, il valore del campo `TableStatus` è impostato su `ACTIVE`.

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

Una volta che la tabella è in stato `ACTIVE`, la best practice è abilitare Backup point-in-time per DynamoDB sulla tabella eseguendo il comando seguente:

Linux

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

Windows CMD

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

Questo comando restituisce il seguente risultato.

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

> **Note**
>
> L'attivazione di backup continui con point-in-time ripristino comporta implicazioni in termini di costi. Per ulteriori informazioni sui prezzi, consulta Prezzi di Amazon DynamoDB.

AWS SDK

Gli esempi di codice seguenti mostrano come creare una tabella DynamoDB utilizzando un SDK AWS .

.NET

SDK per .NET (v4)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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;
        }
    }
```

- Per i dettagli sull'API, consulta la CreateTablesezione AWS SDK per .NET API Reference.

Bash

AWS CLI con lo script Bash

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
}
```

Le funzioni di utilità utilizzate in questo esempio.

```
##############################################################################
# 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
}
```

- Per i dettagli sull'API, consulta CreateTable AWS CLICommand Reference.

C++

SDK per C++

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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);
}
```

Codice che attende che la tabella diventi attiva.

```
//! 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;
}
```

- Per i dettagli sull'API, consulta la CreateTablesezione AWS SDK per C++ API Reference.

CLI

AWS CLI

Esempio 1: per creare una tabella con tag

Nell'esempio seguente `create-table` utilizza gli attributi e lo schema di chiavi specificati per creare una tabella denominata `MusicCollection`. Questa tabella utilizza la velocità effettiva assegnata ed è crittografata a riposo utilizzando la CMK di AWS proprietà predefinita. Il comando applica inoltre un tag alla tabella, con una chiave `Owner` e un valore `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
```

Output:

```
{
    "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"
    }
}
```

Per ulteriori informazioni, consulta Operazioni di base per le tabelle nella Guida per gli sviluppatori di Amazon DynamoDB.

Esempio 2: per creare una tabella in modalità on demand

Nell'esempio seguente viene creata una tabella denominata `MusicCollection` utilizzando la modalità on demand, anziché la modalità basata su throughput allocato. Ciò risulta utile per le tabelle con carichi di lavoro imprevedibili.

```
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
```

Output:

```
{
    "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"
        }
    }
}
```

Per ulteriori informazioni, consulta Operazioni di base per le tabelle nella Guida per gli sviluppatori di Amazon DynamoDB.

Esempio 3: per creare una tabella e crittografarla con una CMK gestita dal cliente

Nell'esempio seguente viene creata una tabella denominata `MusicCollection` e la crittografa utilizzando una gestita dal cliente (CMK).

```
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
```

Output:

```
{
    "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"
        }
    }
}
```

Per ulteriori informazioni, consulta Operazioni di base per le tabelle nella Guida per gli sviluppatori di Amazon DynamoDB.

Esempio 4: per creare una tabella con un indice secondario locale

Nell'esempio seguente vengono utilizzati gli attributi e lo schema di chiavi specificati per creare una tabella denominata `MusicCollection` con un indice secondario locale denominato `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\"]
                }
            }
        ]"
```

Output:

```
{
    "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"
            }
        ]
    }
}
```

Per ulteriori informazioni, consulta Operazioni di base per le tabelle nella Guida per gli sviluppatori di Amazon DynamoDB.

Esempio 5: per creare una tabella con un indice secondario globale

Nell'esempio seguente viene creata una tabella denominata `GameScores` con un indice secondario globale denominato `GameTitleIndex`. La tabella di base ha una chiave di partizione di `UserId` e una chiave di ordinamento di `GameTitle`, permettendo di trovare il miglior punteggio di un singolo utente per un gioco specifico in modo efficiente, mentre il GSI ha una chiave di partizione di `GameTitle` e una chiave di ordinamento di `TopScore`,

permettendo di trovare rapidamente il punteggio più alto complessivo per un determinato gioco.

```
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
                }
            }
        ]"
```

Output:

```
{
    "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"
            }
        ]
    }
}
```

Per ulteriori informazioni, consulta Operazioni di base per le tabelle nella Guida per gli sviluppatori di Amazon DynamoDB.

Esempio 6: per creare una tabella con più indici secondari globali contemporaneamente

Nell'esempio seguente viene creata una tabella denominata `GameScores` con due indici secondari globali. Gli schemi dei GSI vengono passati tramite un file, anziché nella riga di comando.

```
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
```

Contenuto di `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
        }
    }
]
```

Output:

```
{
    "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"
            }
        ]
    }
}
```

Per ulteriori informazioni, consulta Operazioni di base per le tabelle nella Guida per gli sviluppatori di Amazon DynamoDB.

Esempio 7: per creare una tabella con i flussi abilitati

Nell'esempio seguente viene creata una tabella denominata `GameScores` con i flussi DynamoDB abilitati. Le immagini vecchie e nuove di ciascun elemento verranno scritte nel flusso.

```
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
```

Output:

```
{
    "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"
    }
}
```

Per ulteriori informazioni, consulta Operazioni di base per le tabelle nella Guida per gli sviluppatori di Amazon DynamoDB.

Esempio 8: per creare una tabella con il flusso KEYS_ONLY abilitato

Nell'esempio seguente viene creata una tabella denominata `GameScores` con i flussi DynamoDB abilitati. Solo gli attributi chiave dell'elemento modificato vengono scritti nel flusso.

```
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
```

Output:

```
{
    "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
    }
}
```

Per ulteriori informazioni, consulta Acquisizione dei dati di modifica per DynamoDB Streams nella Guida per gli sviluppatori di Amazon DynamoDB.

Esempio 9: per creare una tabella con la classe Standard-Accesso infrequente

Nell'esempio seguente viene creata una tabella denominata `GameScores` e viene assegnata la classe di tabella Standard-Accesso Infrequente (DynamoDB Standard-IA). Questa classe di tabella è ottimizzata a livello di archiviazione in quanto l'archiviazione rappresenta il costo principale.

```
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
```

Output:

```
{
    "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
    }
}
```

Per ulteriori informazioni, consulta Classi di tabelle nella Guida per gli sviluppatori di Amazon DynamoDB.

Esempio 10: per creare una tabella con l'opzione Protezione da eliminazione abilitata

Nell'esempio seguente viene creata una tabella denominata `GameScores` e viene abilitata la protezione da eliminazione.

```
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
```

Output:

```
{
    "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
    }
}
```

Per ulteriori informazioni, consulta Uso della protezione da eliminazione nella Guida per gli sviluppatori di Amazon DynamoDB.

- Per i dettagli sull'API, consulta Command CreateTableReference AWS CLI .

Go

SDK per Go V2

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
}
```

- Per i dettagli sull'API, consulta la CreateTablesezione AWS SDK per Go API Reference.

Java

SDK per Java 2.x

> **Note**
>
> C'è di più su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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 "";
    }
}
```

- Per i dettagli sull'API, CreateTableconsulta AWS SDK for Java 2.x API Reference.

JavaScript

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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;
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK per JavaScript.
- Per i dettagli sull'API, CreateTableconsulta AWS SDK per JavaScript API Reference.

SDK per JavaScript (v2)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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);
  }
});
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK per JavaScript.
- Per i dettagli sull'API, CreateTableconsulta AWS SDK per JavaScript API Reference.

Kotlin

SDK per Kotlin

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
    }
}
```

- Per i dettagli sull'API, CreateTableconsulta AWS SDK for Kotlin API reference.

PHP

SDK per PHP

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Creare una tabella .

```
        $tableName = "ddb_demo_table_$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],
        ]);
    }
```

- Per i dettagli sull'API, CreateTableconsulta AWS SDK per PHP API Reference.

PowerShell

Strumenti per PowerShell V4

Esempio 1: Questo esempio crea una tabella denominata Thread con una chiave primaria composta da 'ForumName' (hash del tipo di chiave) e 'Subject' (intervallo dei tipi di chiave). Lo schema utilizzato per costruire la tabella può essere inserito in ogni cmdlet come mostrato o specificato mediante il parametro -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
```

Output:

```
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  : {}
```

Esempio 2: Questo esempio crea una tabella denominata Thread con una chiave primaria composta da 'ForumName' (hash del tipo di chiave) e 'Subject' (intervallo dei tipi di chiave). Viene inoltre definito un indice secondario locale. La chiave dell'indice secondario locale verrà impostata automaticamente dalla chiave hash primaria sulla tabella ()ForumName. Lo schema utilizzato per costruire la tabella può essere inserito in ogni cmdlet come mostrato o specificato mediante il parametro -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
```

Output:

```
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}
```

Esempio 3: Questo esempio mostra come utilizzare una singola pipeline per creare una tabella denominata Thread con una chiave primaria composta da " ForumName (hash del tipo di chiave) e 'Subject' (intervallo dei tipi di chiave) e un indice secondario locale. Add- DDBKey Schema e Add- DDBIndex Schema creano automaticamente un nuovo TableSchema oggetto se uno non viene fornito dalla pipeline o dal parametro -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
```

Output:

```
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}
```

- Per i dettagli sull'API, vedere CreateTablein AWS Strumenti per PowerShell Cmdlet Reference (V4).

Strumenti per V5 PowerShell

Esempio 1: Questo esempio crea una tabella denominata Thread con una chiave primaria composta da 'ForumName' (hash del tipo di chiave) e 'Subject' (intervallo dei tipi di chiave). Lo schema utilizzato per costruire la tabella può essere inserito in ogni cmdlet come mostrato o specificato mediante il parametro -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
```

Output:

```
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  : {}
```

Esempio 2: Questo esempio crea una tabella denominata Thread con una chiave primaria composta da 'ForumName' (hash del tipo di chiave) e 'Subject' (intervallo dei tipi di chiave). Viene inoltre definito un indice secondario locale. La chiave dell'indice secondario locale verrà impostata automaticamente dalla chiave hash primaria sulla tabella ()ForumName. Lo schema utilizzato per costruire la tabella può essere inserito in ogni cmdlet come mostrato o specificato mediante il parametro -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
```

Output:

```
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}
```

Esempio 3: Questo esempio mostra come utilizzare una singola pipeline per creare una tabella denominata Thread con una chiave primaria composta da " ForumName (hash del tipo di chiave) e 'Subject' (intervallo dei tipi di chiave) e un indice secondario locale. Add- DDBKey Schema e Add- DDBIndex Schema creano automaticamente un nuovo TableSchema oggetto se uno non viene fornito dalla pipeline o dal parametro -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
```

Output:

```
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}
```

- Per i dettagli sull'API, vedere CreateTablein AWS Strumenti per PowerShell Cmdlet Reference (V5).

Python

SDK per Python (Boto3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Crea una tabella per la memorizzazione dei dati del filmato.

```
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
```

- Per i dettagli sull'API, consulta CreateTable AWSSDK for Python (Boto3) API Reference.

Ruby

SDK per Ruby

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
```

- Per i dettagli sull'API, CreateTableconsulta AWS SDK per Ruby API Reference.

Rust

SDK per Rust

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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))
        }
    }
}
```

- Per i dettagli sulle API, consulta la CreateTableguida di riferimento all'API AWS SDK for Rust.

SAP ABAP

SDK per SAP ABAP

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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.
```

- Per i dettagli sulle API, CreateTableconsulta AWS SDK for SAP ABAP API reference.

Swift

SDK per Swift

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
        }
    }
```

- Per i dettagli sull'API, consulta la CreateTableguida di riferimento all'API AWS SDK for Swift.

Per ulteriori esempi di DynamoDB, consulta Esempi di codice per l'utilizzo di DynamoDB AWS SDKs.

Dopo aver creato la nuova tabella, procedere a Fase 2: scrivere dati in una tabella DynamoDB.

Fase 2: scrivere dati in una tabella DynamoDB

In questa fase, inserire diverse voci nella tabella `Music` creata in Fase 1: creare una tabella DynamoDB.

Per ulteriori informazioni sulle operazioni di scrittura, consulta Scrittura di un elemento.

Console di gestione AWS

Seguire questi passaggi per scrivere i dati nella tabella `Music` utilizzando la console DynamoDB.

1. Apri la console DynamoDB all'indirizzo. https://console.aws.amazon.com/dynamodb/
2. Nel riquadro di navigazione a sinistra, selezionare Tables (Tabelle).
3. Nella pagina Tabelle, seleziona la tabella Musica.
4. Scegli Explore table items (Esplora elementi della tabella).
5. Nella sezione Elementi restituiti, seleziona Crea elemento.
6. Nella pagina Crea elemento, procedi come segue per aggiungere elementi alla tabella:
    a. Scegli Add new attribute (Aggiungi nuovo attributo) e quindi scegli Number (Numero).
    b. Per Nome attributo inserisci **`Awards`**.

c. Ripeti questo processo per creare un **`AlbumTitle`** di tipo String (Stringa).

d. Inserisci i valori seguenti per il tuo elemento:

i. In Artist, digita **`No One You Know`**.

ii. In SongTitle, immettere **`Call Me Today`**.

iii. In AlbumTitle, immettere **`Somewhat Famous`**.

iv. Per Awards (Premi), inserisci **`1`**.

7. Scegli Create Item (Crea elemento).

8. Ripeti questo processo e crea un'altra voce con i seguenti valori:

a. In Artist, digita **`Acme Band`**.

b. Per SongTitleentrare. **`Happy Day`**

c. In AlbumTitle, immettere **`Songs About Life`**.

d. Per Awards (Premi), inserisci **`10`**.

9. Fatelo ancora una volta per creare un altro elemento con lo stesso Artista come il passaggio precedente, ma valori diversi per gli altri attributi:

a. In Artist, digita **`Acme Band`**.

b. Per SongTitleentrare**`PartiQL Rocks`**.

c. In AlbumTitle, immettere **`Another Album Title`**.

d. Per Awards (Premi), inserisci **`8`**.

AWS CLI

L' AWS CLI esempio seguente crea diversi nuovi elementi nella `Music` tabella. È possibile farlo tramite l'API DynamoDB o PartiQL, un linguaggio di query compatibile con SQL per 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"}}'
```

Windows CMD

```
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'}"
```

Windows CMD

```
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'}"
```

Per ulteriori informazioni sulla scrittura di dati con PartiQL, consulta Istruzioni insert di PartiQL.

Per ulteriori informazioni sui tipi di dati supportati in DynamoDB, consulta Tipi di dati.

Per ulteriori informazioni su come rappresentare i tipi di dati DynamoDB in JSON, consulta Valori degli attributi.

AWS SDK

I seguenti esempi di codice mostrano come scrivere un elemento in una tabella DynamoDB utilizzando un SDK. AWS

.NET

SDK per .NET (v4)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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;
        }
    }
```

- Per i dettagli sull'API, PutItemconsulta AWS SDK per .NET API Reference.

Bash

AWS CLI con lo script Bash

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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

}
```

Le funzioni di utilità utilizzate in questo esempio.

```
###############################################################################
# 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
}
```

- Per i dettagli sull'API, consulta PutItem AWS CLICommand Reference.

C++

SDK per C++

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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);
}
```

Codice che attende che la tabella diventi attiva.

```
//! 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;
}
```

- Per i dettagli sull'API, PutItemconsulta AWS SDK per C++ API Reference.

CLI

AWS CLI

Esempio 1: per aggiungere un elemento a una tabella

L'`put-item`esempio seguente aggiunge un nuovo elemento alla MusicCollectiontabella.

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

Contenuto di `item.json`:

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

Output:

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

Per ulteriori informazioni, consulta Scrittura di un elemento nella Guida per gli sviluppatori di Amazon DynamoDB.

Esempio 2: per sovrascrivere in base a una condizione un elemento in una tabella

Nell'esempio seguente `put-item` sovrascrive un elemento esistente nella tabella `MusicCollection` solo se tale elemento include un attributo `AlbumTitle` con il valore `Greatest Hits`. Il comando restituisce il valore precedente dell'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
```

Contenuto di `item.json`:

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

Contenuto di `names.json`:

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

Contenuto di `values.json`:

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

Output:

```
{
    "Attributes": {
```

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

Se la chiave esiste già, verrà visualizzato l'output seguente:

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

Per ulteriori informazioni, consulta Scrittura di un elemento nella Guida per gli sviluppatori di Amazon DynamoDB.

- Per i dettagli sull'API, vedere PutItemin AWS CLI Command Reference.

Go

SDK per Go V2

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
}
```

Per definire una struttura di filmato utilizzata in questo esempio.

```
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"])
}
```

- Per i dettagli sull'API, PutItemconsulta AWS SDK per Go API Reference.

Java

SDK per Java 2.x

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Mette un elemento in una tabella utilizzando 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);
        }
    }
}
```

- Per i dettagli sull'API, vedi PutItemin AWS SDK for Java 2.x API Reference.

JavaScript

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Questo esempio utilizza il client di documenti per semplificare il lavoro con gli elementi in DynamoDB. Per i dettagli sull'API, consulta 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;
};
```

- Per i dettagli sulle API, consulta la PutItemsezione AWS SDK per JavaScript API Reference.

SDK per JavaScript (v2)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Inserisci un elemento in una tabella.

```
// 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);
  }
});
```

Inserisci un elemento in una tabella utilizzando il client documento 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);
  }
});
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK per JavaScript.
- Per i dettagli sull'API, PutItemconsulta AWS SDK per JavaScript API Reference.

Kotlin

SDK per Kotlin

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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.")
    }
}
```

- Per i dettagli sull'API, PutItemconsulta AWS SDK for Kotlin API reference.

PHP

SDK per PHP

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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);
    }
```

- Per i dettagli sull'API, PutItemconsulta AWS SDK per PHP API Reference.

PowerShell

Strumenti per PowerShell V4

Esempio 1: crea un nuovo elemento o sostituisce un elemento esistente con uno nuovo.

```
$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
```

- Per i dettagli sull'API, vedere PutItemin AWS Strumenti per PowerShell Cmdlet Reference (V4).

Strumenti per V5 PowerShell

Esempio 1: crea un nuovo elemento o sostituisce un elemento esistente con uno nuovo.

```
$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
```

- Per i dettagli sull'API, vedere PutItemin AWS Strumenti per PowerShell Cmdlet Reference (V5).

Python

SDK per Python (Boto3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
```

- Per i dettagli sull'API, consulta PutItem AWSSDK for Python (Boto3) API Reference.

Ruby

SDK per Ruby

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
```

- Per i dettagli sull'API, PutItemconsulta AWS SDK per Ruby API Reference.

Rust

SDK per Rust

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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,
    })
}
```

- Per i dettagli sulle API, consulta la PutItemguida di riferimento all'API AWS SDK for Rust.

SAP ABAP

SDK per SAP ABAP

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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.
```

- Per i dettagli sulle API, PutItemconsulta AWS SDK for SAP ABAP API reference.

Swift

SDK per Swift

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
    }
```

- Per i dettagli sull'API, consulta la PutItemguida di riferimento all'API AWS SDK for Swift.

Per ulteriori esempi di DynamoDB, consulta Esempi di codice per l'utilizzo di DynamoDB AWS SDKs.

Dopo aver inserito i dati nella tabella, proseguire con Fase 3: leggere i dati da una tabella DynamoDB.

Fase 3: leggere i dati da una tabella DynamoDB

In questa fase, leggerai uno degli elementi creati in Fase 2: scrivere dati in una tabella DynamoDB. È possibile utilizzare la console DynamoDB o AWS CLI leggere un elemento dalla tabella `Music` specificando e. `Artist` `SongTitle`

Per ulteriori informazioni sulle operazioni di lettura in DynamoDB, consulta Lettura di un elemento.

Console di gestione AWS

Seguire questi passaggi per leggere i dati nella tabella `Music` utilizzando la console DynamoDB.

1. Apri la console DynamoDB all'indirizzo. https://console.aws.amazon.com/dynamodb/
2. Nel riquadro di navigazione a sinistra, selezionare Tables (Tabelle).
3. Nella pagina Tabelle, seleziona la tabella Musica.
4. Scegli Explore table items (Esplora elementi della tabella).
5. Nella sezione Elementi restituiti, visualizza l'elenco di elementi archiviati nella tabella, ordinati per `Artist` e `SongTitle`. Il primo elemento dell'elenco è quello con l'artista di nome Acme Band e il SongTitlePartiQL Rocks.

AWS CLI

L' AWS CLI esempio seguente legge un elemento da. `Music` È possibile farlo tramite l'API DynamoDB o PartiQL, un linguaggio di query compatibile con SQL per DynamoDB.

DynamoDB API

> **Note**
>
> Il comportamento predefinito per DynamoDB è lettura a consistenza finale. Il parametro `consistent-read` viene utilizzato qui di seguito per dimostrare le letture consistenti.

Linux

```
aws dynamodb get-item --consistent-read \
```

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

Windows CMD

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

Utilizzando `get-item` viene restituito il seguente risultato di esempio.

```
{
    "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'"
```

Windows CMD

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

Utilizzando l'istruzione `Select` PartiQL viene restituito il seguente risultato di esempio.

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

Per ulteriori informazioni sulla lettura di dati con PartiQL, consulta Istruzioni select di PartiQL.

AWS SDK

I seguenti esempi di codice mostrano come leggere un elemento da una tabella DynamoDB utilizzando un SDK. AWS

.NET

SDK per .NET (v4)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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;
        }
    }
```

- Per i dettagli sull'API, GetItemconsulta AWS SDK per .NET API Reference.

Bash

AWS CLI con lo script Bash

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
}
```

Le funzioni di utilità utilizzate in questo esempio.

```
###############################################################################
# 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
}
```

- Per i dettagli sull'API, consulta GetItem AWS CLICommand Reference.

C++

SDK per C++

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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();
}
```

- Per i dettagli sull'API, GetItemconsulta AWS SDK per C++ API Reference.

CLI

AWS CLI

Esempio 1: per leggere un elemento in una tabella

Nell'esempio seguente `get-item` recupera un elemento dalla tabella `MusicCollection`. La tabella ha una chiave hash-and-range primaria (`ArtisteSongTitle`), quindi è necessario specificare entrambi questi attributi. Il comando richiede anche informazioni sulla capacità di scrittura utilizzata dall'operazione.

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

Contenuto di `key.json`:

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

Output:

```
{
    "Item": {
        "AlbumTitle": {
            "S": "Songs About Life"
        },
        "SongTitle": {
            "S": "Happy Day"
        },
        "Artist": {
            "S": "Acme Band"
        }
    },
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 0.5
    }
}
```

Per ulteriori informazioni, consulta Lettura di un elemento nella Guida per gli sviluppatori di Amazon DynamoDB.

Esempio 2: per leggere un elemento utilizzando una lettura consistente

L'esempio seguente recupera un elemento dalla tabella `MusicCollection` tramite operazioni a elevata consistenza di lettura.

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

Contenuto di `key.json`:

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

Output:

```
{
    "Item": {
        "AlbumTitle": {
            "S": "Songs About Life"
        },
        "SongTitle": {
            "S": "Happy Day"
        },
        "Artist": {
            "S": "Acme Band"
        }
    },
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 1.0
    }
}
```

Per ulteriori informazioni, consulta Lettura di un elemento nella Guida per gli sviluppatori di Amazon DynamoDB.

Esempio 3: per recuperare attributi specifici di un elemento

Nell'esempio seguente viene utilizzata un'espressione di proiezione per recuperare solo tre attributi dell'elemento desiderato.

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

Contenuto di `names.json`:

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

Output:

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

Per ulteriori informazioni, consulta Lettura di un elemento nella Guida per gli sviluppatori di Amazon DynamoDB.

- Per i dettagli sull'API, consulta GetItem AWS CLICommand Reference.

Go

SDK per Go V2

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
}
```

Per definire una struttura di filmato utilizzata in questo esempio.

```
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"])
}
```

- Per i dettagli sull'API, GetItemconsulta AWS SDK per Go API Reference.

Java

SDK per Java 2.x

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Ottiene un elemento da una tabella utilizzando 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);
        }
    }
}
```

- Per i dettagli sull'API, vedere GetItemin AWS SDK for Java 2.x API Reference.

JavaScript

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Questo esempio utilizza il client di documenti per semplificare il lavoro con gli elementi in DynamoDB. Per i dettagli sull'API, consulta 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;
};
```

- Per i dettagli sulle API, consulta la GetItemsezione AWS SDK per JavaScript API Reference.

SDK per JavaScript (v2)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Ottieni un elemento da una tabella.

```
// 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);
  }
});
```

Ottieni un elemento da una tabella utilizzando il client documento 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);
  }
});
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK per JavaScript.
- Per i dettagli sull'API, GetItemconsulta AWS SDK per JavaScript API Reference.

Kotlin

SDK per Kotlin

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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)
        }
    }
}
```

- Per i dettagli sull'API, GetItemconsulta AWS SDK for Kotlin API reference.

PHP

SDK per PHP

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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,
        ]);
    }
```

- Per i dettagli sull'API, GetItemconsulta AWS SDK per PHP API Reference.

PowerShell

Strumenti per PowerShell V4

Esempio 1: restituisce l'elemento DynamoDB con la chiave di partizione e la SongTitle chiave di ordinamento Artist.

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

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

Output:

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

- Per i dettagli sull'API, vedere GetItemin AWS Strumenti per PowerShell Cmdlet Reference (V4).

Strumenti per V5 PowerShell

Esempio 1: restituisce l'elemento DynamoDB con la chiave di partizione e la SongTitle chiave di ordinamento Artist.

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

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

Output:

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

- Per i dettagli sull'API, vedere GetItemin AWS Strumenti per PowerShell Cmdlet Reference (V5).

Python

SDK per Python (Boto3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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"]
```

- Per i dettagli sull'API, consulta GetItem AWSSDK for Python (Boto3) API Reference.

Ruby

SDK per Ruby

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
```

- Per i dettagli sull'API, GetItemconsulta AWS SDK per Ruby API Reference.

SAP ABAP

SDK per SAP ABAP

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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.
```

- Per i dettagli sulle API, GetItemconsulta AWS SDK for SAP ABAP API reference.

Swift

SDK per Swift

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
        }
    }
```

- Per i dettagli sull'API, consulta la GetItemguida di riferimento all'API AWS SDK for Swift.

Per ulteriori esempi di DynamoDB, consulta Esempi di codice per l'utilizzo di DynamoDB AWS SDKs.

Per aggiornare i dati nella tua tabella, prosegui con Fase 4: aggiornare i dati in una tabella.

Fase 4: aggiornare i dati in una tabella

In questa fase, aggiorna una voce che hai creato in Fase 2: scrivere dati in una tabella DynamoDB. È possibile utilizzare la console DynamoDB o AWS CLI aggiornare `AlbumTitle` un elemento nella tabella `Artist` specificando`SongTitle`, e `Music` l'aggiornamento. `AlbumTitle`

Per ulteriori informazioni sulle operazioni di scrittura, consulta Scrittura di un elemento.

Console di gestione AWS

Puoi utilizzare la console DynamoDB per aggiornare i dati nella tabella `Music`.

1. Apri la console DynamoDB all'indirizzo. https://console.aws.amazon.com/dynamodb/
2. Nel riquadro di navigazione a sinistra, selezionare Tables (Tabelle).
3. Nell'elenco delle tabelle, scegli la tabella Musica.
4. Scegli Explore table items (Esplora elementi della tabella).
5. In Oggetti restituiti, per la riga dell'elemento con Acme Band Artist e Happy Day SongTitle, procedi come segue:
    a. Posiziona il cursore sul AlbumTitletitolo Songs About Life.
    b. Seleziona l'icona Modifica.
    c. Nella finestra popup Modifica stringa, inserisci **`Songs of Twilight`**.

d. Scegli Save (Salva).

> **Tip**
>
> In alternativa, per aggiornare un elemento, procedi come segue nella sezione Elementi restituiti:
>
> 1. Scegli la riga dell'elemento con l'artista di nome Acme Band e il SongTitlenome Happy Day.
> 2. Dal menù a discesa Operazioni, seleziona Modifica elemento.
> 3. Per entrare AlbumTitle, inserisci**`Songs of Twilight`**.
> 4. Selezionare Save and close (Salva e chiudi).

AWS CLI

L' AWS CLI esempio seguente aggiorna un elemento nella `Music` tabella. È possibile farlo tramite l'API DynamoDB o PartiQL, un linguaggio di query compatibile con SQL per 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
```

Windows CMD

```
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
```

L'utilizzo di `update-item` restituisce il seguente risultato di esempio perché è stato specificato `return-values ALL_NEW`.

```
{
    "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 *"
```

Windows CMD

```
aws dynamodb execute-statement --statement "UPDATE Music SET AlbumTitle='Updated
 Album Title' WHERE Artist='Acme Band' AND SongTitle='Happy Day' RETURNING ALL NEW
 *"
```

L'utilizzo dell'istruzione `Update` restituisce il seguente risultato di esempio perché è stato specificato `RETURNING ALL NEW *`.

```
{
    "Items": [
        {
            "AlbumTitle": {
```

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

Per ulteriori informazioni sull'aggiornamento dei dati con PartiQL, consulta Istruzioni update di PartiQL.

AWS SDK

I seguenti esempi di codice mostrano come aggiornare un elemento in una tabella DynamoDB utilizzando un SDK. AWS

.NET

SDK per .NET (v4)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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;
        }
    }
```

- Per i dettagli sull'API, UpdateItemconsulta AWS SDK per .NET API Reference.

Bash

AWS CLI con lo script Bash

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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

}
```

Le funzioni di utilità utilizzate in questo esempio.

```
################################################################################
# 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
}
```

- Per i dettagli sull'API, consulta UpdateItem AWS CLICommand Reference.

C++

SDK per C++

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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);
}
```

Codice che attende che la tabella diventi attiva.

```
//! 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;
}
```

- Per i dettagli sull'API, UpdateItemconsulta AWS SDK per C++ API Reference.

CLI

AWS CLI

Esempio 1: per aggiornare un elemento in una tabella

L'esempio `update-item` seguente aggiorna un elemento nella tabella `MusicCollection`. Aggiunge un nuovo attributo (`Year`) e modifica l'attributo `AlbumTitle`. Tutti gli attributi dell'elemento come appaiono dopo l'aggiornamento vengono restituiti nella risposta.

```
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
```

Contenuto di `key.json`:

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

Contenuto di `expression-attribute-names.json`:

```
{
    "#Y":"Year", "#AT":"AlbumTitle"
}
```

Contenuto di `expression-attribute-values.json`:

```
{
    ":y":{"N": "2015"},
    ":t":{"S": "Louder Than Ever"}
}
```

Output:

```
{
    "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
        ]
    }
}
```

Per ulteriori informazioni, consulta Scrittura di un elemento nella Guida per gli sviluppatori di Amazon DynamoDB.

Esempio 2: per aggiornare un elemento in base a una condizione

Nell'esempio seguente viene aggiornato un elemento nella tabella `MusicCollection`, ma solo se l'elemento esistente non dispone già di un attributo `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)"
```

Contenuto di `key.json`:

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

Contenuto di `expression-attribute-names.json`:

```
{
    "#Y":"Year",
    "#AT":"AlbumTitle"
}
```

Contenuto di `expression-attribute-values.json`:

```
{
    ":y":{"N": "2015"},
    ":t":{"S": "Louder Than Ever"}
}
```

Se l'elemento include già un attributo `Year`, DynamoDB restituisce il seguente output.

```
An error occurred (ConditionalCheckFailedException) when calling the UpdateItem
 operation: The conditional request failed
```

Per ulteriori informazioni, consulta Scrittura di un elemento nella Guida per gli sviluppatori di Amazon DynamoDB.

- Per i dettagli sull'API, consulta UpdateItem AWS CLICommand Reference.

Go

SDK per Go V2

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
}
```

Per definire una struttura di filmato utilizzata in questo esempio.

```
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"])
}
```

- Per i dettagli sull'API, UpdateItemconsulta AWS SDK per Go API Reference.

Java

SDK per Java 2.x

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Aggiorna un elemento in una tabella utilizzando 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!");
    }
}
```

- Per i dettagli sull'API, consulta la UpdateItemsezione AWS SDK for Java 2.x API Reference.

JavaScript

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Questo esempio utilizza il client di documenti per semplificare il lavoro con gli elementi in DynamoDB. Per i dettagli sull'API, consulta 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;
};
```

- Per i dettagli sulle API, consulta la UpdateItemsezione AWS SDK per JavaScript API Reference.

Kotlin

SDK per Kotlin

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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")
    }
}
```

- Per i dettagli sull'API, UpdateItemconsulta AWS SDK for Kotlin API reference.

PHP

SDK per PHP

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
                ]
            ],
        ]);
    }
```

- Per i dettagli sull'API, UpdateItemconsulta AWS SDK per PHP API Reference.

PowerShell

Strumenti per PowerShell V4

Esempio 1: imposta l'attributo genre su 'Rap' sull'elemento DynamoDB con la chiave di partizione e la SongTitle chiave di ordinamento 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
```

Output:

```
Name                           Value
----                           -----
Genre                          Rap
```

- Per i dettagli sull'API, vedere UpdateItemin AWS Strumenti per PowerShell Cmdlet Reference (V4).

Strumenti per V5 PowerShell

Esempio 1: imposta l'attributo genre su 'Rap' sull'elemento DynamoDB con la chiave di partizione e la SongTitle chiave di ordinamento 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
```

Output:

```
Name                           Value
----                           -----
Genre                          Rap
```

- Per i dettagli sull'API, vedere UpdateItemin AWS Strumenti per PowerShell Cmdlet Reference (V5).

Python

SDK per Python (Boto3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Aggiorna un elemento utilizzando un'espressione di aggiornamento.

```
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"]
```

Aggiorna un elemento utilizzando un'espressione di aggiornamento che include un'operazione aritmetica.

```
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"]
```

Aggiorna un elemento solo quando soddisfa determinate condizioni.

```
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"]
```

- Per i dettagli sull'API, consulta UpdateItem AWSSDK for Python (Boto3) API Reference.

Ruby

SDK per Ruby

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
```

- Per i dettagli sull'API, UpdateItemconsulta AWS SDK per Ruby API Reference.

SAP ABAP

SDK per SAP ABAP

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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.
```

- Per i dettagli sulle API, UpdateItemconsulta AWS SDK for SAP ABAP API reference.

Swift

SDK per Swift

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
        }
    }
```

- Per i dettagli sull'API, consulta la UpdateItemguida di riferimento all'API AWS SDK for Swift.

Per ulteriori esempi di DynamoDB, consulta Esempi di codice per l'utilizzo di DynamoDB AWS SDKs.

Per eseguire una query dei dati nella tabella `Music`, continua con Fase 5: eseguire query sui dati in una tabella DynamoDB.

Fase 5: eseguire query sui dati in una tabella DynamoDB

In questa fase, esegui le query dei dati che hai scritto nella tabella `Music` in the section called “Fase 2: scrittura dei dati” specificando `Artist`. In questo modo verranno visualizzate tutte le canzoni associate alla chiave di partizione: `Artist`.

Per ulteriori informazioni sulle operazioni delle query, consulta Esecuzione di query in DynamoDB.

Console di gestione AWS

Segui questi passaggi per utilizzare la console DynamoDB per eseguire query dei dati nella tabella `Music`.

1. Apri la console DynamoDB all'indirizzo. https://console.aws.amazon.com/dynamodb/
2. Nel riquadro di navigazione a sinistra, selezionare Tables (Tabelle).
3. Nell'elenco delle tabelle, scegli la tabella Musica.
4. Scegli Explore table items (Esplora elementi della tabella).
5. In Scansiona o interroga gli elementi, assicurati che sia selezionata l’opzione Query.
6. In Partition key (Chiave di partizione), inserisci **`Acme Band`**, quindi scegli Run (Esegui).

AWS CLI

L' AWS CLI esempio seguente interroga un elemento nella tabella. `Music` È possibile farlo tramite l'API DynamoDB o PartiQL, un linguaggio di query compatibile con SQL per DynamoDB.

DynamoDB API

È possibile eseguire una query su un elemento tramite l'API DynamoDB utilizzando `query` e fornendo la chiave di partizione.

Linux

```
aws dynamodb query \
    --table-name Music \
    --key-condition-expression "Artist = :name" \
    --expression-attribute-values  '{":name":{"S":"Acme Band"}}'
```

Windows CMD

```
aws dynamodb query ^
    --table-name Music ^
    --key-condition-expression "Artist = :name" ^
    --expression-attribute-values  "{\":name\":{\"S\":\"Acme Band\"}}"
```

L'utilizzo di `query` restituisce tutte le canzoni associate a questo particolare `Artist`.

```
{
    "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

È possibile eseguire una query su PartiQL utilizzando l'istruzione `Select` e fornendo la chiave di partizione.

Linux

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

Windows CMD

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

L'utilizzo della dichiarazione `Select` in questo modo restituisce tutte le canzoni associate a questo particolare `Artist`.

```
{
    "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"
            }
        }
    ]
}
```

Per ulteriori informazioni sull'esecuzione di query sui dati con PartiQL, consulta Istruzioni select di PartiQL.

AWS SDK

I seguenti esempi di codice mostrano come eseguire una query su una tabella DynamoDB utilizzando un SDK AWS .

.NET

SDK per .NET (v4)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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;
        }
    }
```

- Per ulteriori informazioni sulle API, consulta Query nella Documentazione di riferimento delle API AWS SDK per .NET .

Bash

AWS CLI con lo script Bash

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
}
```

Le funzioni di utilità utilizzate in questo esempio.

```
###############################################################################
# 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
}
```

- Per informazioni dettagliate sulle API, consulta Query nella Documentazione di riferimento di AWS CLI .

C++

SDK per C++

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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;
}
```

- Per ulteriori informazioni sulle API, consulta Query nella Documentazione di riferimento delle API AWS SDK per C++ .

CLI

AWS CLI

Esempio 1: per eseguire una query su una tabella

Nell'esempio seguente `query` esegue query sulle voce della tabella `MusicCollection`. La tabella ha una chiave hash-and-range primaria (`ArtistandSongTitle`), ma questa query specifica solo il valore della chiave hash. Restituisce i titoli dei brani dell'artista "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
```

Contenuto di `expression-attributes.json`:

```
{
    ":v1": {"S": "No One You Know"}
}
```

Output:

```
{
    "Items": [
```

```
        {
            "SongTitle": {
                "S": "Call Me Today"
            },
            "SongTitle": {
                "S": "Scared of My Shadow"
            }
        }
    ],
    "Count": 2,
    "ScannedCount": 2,
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 0.5
    }
}
```

Per ulteriori informazioni, consulta Utilizzo delle query in DynamoDB nella Guida per gli sviluppatori di Amazon DynamoDB.

Esempio 2: per eseguire una query su una tabella utilizzando operazioni a elevata consistenza di lettura e attraversare l'indice in ordine decrescente

Nell'esempio seguente viene eseguita la stessa query del primo esempio, ma vengono restituiti i risultati in ordine inverso e vengono utilizzate operazioni a elevata consistenza di lettura.

```
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
```

Contenuto di `expression-attributes.json`:

```
{
    ":v1": {"S": "No One You Know"}
}
```

Output:

```
{
    "Items": [
        {
            "SongTitle": {
                "S": "Scared of My Shadow"
            }
        },
        {
            "SongTitle": {
                "S": "Call Me Today"
            }
        }
    ],
    "Count": 2,
    "ScannedCount": 2,
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 1.0
    }
}
```

Per ulteriori informazioni, consulta Utilizzo delle query in DynamoDB nella Guida per gli sviluppatori di Amazon DynamoDB.

Esempio 3: per filtrare risultati specifici

Nell'esempio seguente viene eseguita una query su `MusicCollection` ma vengono esclusi i risultati con valori specifici nell'attributo `AlbumTitle`. Si noti che ciò non influisce su `ScannedCount` o `ConsumedCapacity`, poiché il filtro viene applicato dopo la lettura degli elementi.

```
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
```

Contenuto di `values.json`:

```
{
    ":v1": {"S": "No One You Know"},
    ":v2": {"S": "Blue Sky Blues"},
    ":v3": {"S": "Greatest Hits"}
}
```

Contenuto di `names.json`:

```
{
    "#n1": "Artist",
    "#n2": "AlbumTitle"
}
```

Output:

```
{
    "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
    }
}
```

Per ulteriori informazioni, consulta Utilizzo delle query in DynamoDB nella Guida per gli sviluppatori di Amazon DynamoDB.

Esempio 4: per recuperare solo il conteggio degli elementi

Nell'esempio seguente viene recuperato il conteggio degli elementi corrispondenti alla query, ma non viene recuperato nessuno degli elementi stessi.

```
aws dynamodb query \
    --table-name MusicCollection \
    --select COUNT \
    --key-condition-expression "Artist = :v1" \
    --expression-attribute-values file://expression-attributes.json
```

Contenuto di `expression-attributes.json`:

```
{
    ":v1": {"S": "No One You Know"}
}
```

Output:

```
{
    "Count": 2,
    "ScannedCount": 2,
    "ConsumedCapacity": null
}
```

Per ulteriori informazioni, consulta Utilizzo delle query in DynamoDB nella Guida per gli sviluppatori di Amazon DynamoDB.

Esempio 5: per eseguire una query su un indice

Nell'esempio seguente viene eseguita una query su un indice locale secondario `AlbumTitleIndex`. La query restituisce tutti gli attributi della tabella di base proiettati nell'indice secondario locale. Si noti che durante l'esecuzione di query su un indice secondario locale o su un indice secondario globale è necessario fornire anche il nome della tabella di base mediante il parametro `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
```

Contenuto di `expression-attributes.json`:

```
{
    ":v1": {"S": "No One You Know"}
}
```

Output:

```
{
    "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
            }
        }
    }
}
```

Per ulteriori informazioni, consulta Utilizzo delle query in DynamoDB nella Guida per gli sviluppatori di Amazon DynamoDB.

- Per informazioni dettagliate sulle API, consulta Query nella Documentazione di riferimento di AWS CLI .

Go

SDK per Go V2

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
}
```

Per definire una struttura di filmato utilizzata in questo esempio.

```
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"])
}
```

- Per ulteriori informazioni sulle API, consulta Query nella Documentazione di riferimento delle API AWS SDK per Go .

Java

SDK per Java 2.x

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Interroga una tabella utilizzando 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;
    }
}
```

Esegue query su una tabella utilizzando `DynamoDbClient` e un indice secondario.

```
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);
        }
    }
}
```

- Per ulteriori informazioni sulle API, consulta Query nella Documentazione di riferimento delle API AWS SDK for Java 2.x .

JavaScript

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Questo esempio utilizza il client di documenti per semplificare il lavoro con gli elementi in DynamoDB. Per i dettagli sull'API, consulta 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;
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK per JavaScript.
- Per ulteriori informazioni sulle API, consulta Query nella Documentazione di riferimento delle API AWS SDK per JavaScript .

SDK per JavaScript (v2)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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);
  }
});
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK per JavaScript.
- Per ulteriori informazioni sulle API, consulta Query nella Documentazione di riferimento delle API AWS SDK per JavaScript .

Kotlin

SDK per Kotlin

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
    }
}
```

- Per informazioni dettagliate sulle API, consulta Query nella Documentazione di riferimento per le API di SDK AWS per Kotlin.

PHP

SDK per PHP

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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);
    }
```

- Per ulteriori informazioni sulle API, consulta Query nella Documentazione di riferimento delle API AWS SDK per PHP .

PowerShell

Strumenti per PowerShell V4

Esempio 1: richiama una query che restituisce elementi DynamoDB con l'elemento Artist specificato. SongTitle

```
$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
```

Output:

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

- Per informazioni dettagliate sull'API, consulta Query nella documentazione di riferimento dei cmdlet di AWS Strumenti per PowerShell (V4).

Strumenti per V5 PowerShell

Esempio 1: richiama una query che restituisce elementi DynamoDB con l'elemento Artist specificato. SongTitle

```
$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
```

Output:

```
Name                           Value
----                           -----
```

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

- Per informazioni dettagliate sull'API, consulta Query nella documentazione di riferimento dei cmdlet di AWS Strumenti per PowerShell (V5).

Python

SDK per Python (Boto3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Esegui la query degli elementi utilizzando un'espressione di condizione chiave.

```
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"]
```

Esegui la query degli elementi e proiettali per restituire un sottoinsieme di dati.

```
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"]
```

- Per informazioni dettagliate sulle API, consulta Query nella Documentazione di riferimento per l'API SDK for Python (Boto3)AWS .

Ruby

SDK per Ruby

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
```

- Per ulteriori informazioni sulle API, consulta Query nella Documentazione di riferimento delle API AWS SDK per Ruby .

Rust

SDK per Rust

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Trova i filmati realizzati nell'anno specificato.

```
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![])
    }
}
```

- Per ulteriori informazioni sulle API, consulta Query nella Documentazione di riferimento delle API AWS .

SAP ABAP

SDK per SAP ABAP

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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.
```

- Per informazioni dettagliate sull'API, consulta Query nella Documentazione di riferimento dell'API dell'AWS SDK per SAP ABAP.

Swift

SDK per Swift

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
        }
    }
```

- Per informazioni dettagliate sulle API, consulta Query nella Documentazione di riferimento per le API di SDK AWS per Swift.

Per ulteriori esempi di DynamoDB, consulta Esempi di codice per l'utilizzo di DynamoDB AWS SDKs.

Per creare un indice globale secondario per la tua tabella, procedi con Fase 6: (facoltativo) eliminare la tabella DynamoDB per eseguire la pulizia delle risorse.

Fase 6: (facoltativo) eliminare la tabella DynamoDB per eseguire la pulizia delle risorse

Se non hai più bisogno della tabella Amazon DynamoDB creata per il tutorial, puoi eliminarla. In questo modo hai la certezza che non ti vengano addebitati costi per risorse che non stai utilizzando. È possibile utilizzare la console DynamoDB o eliminare AWS CLI `Music` la tabella in cui è stata creata. Fase 1: creare una tabella DynamoDB

Per ulteriori informazioni sulle operazioni della tabella in DynamoDB, consulta Utilizzo di tabelle e dati in DynamoDB.

Console di gestione AWS

Per eliminare la tabella `Music` utilizzando la console:

1. Apri la console DynamoDB all'indirizzo. https://console.aws.amazon.com/dynamodb/
2. Nel riquadro di navigazione a sinistra, selezionare Tables (Tabelle).
3. Seleziona la casella di spunta accanto alla tabella Musica nell'elenco delle tabelle.
4. Scegli Elimina.

AWS CLI

L' AWS CLI esempio seguente elimina la `Music` tabella utilizzando. `delete-table`

```
aws dynamodb delete-table --table-name Music
```

AWS SDK

I seguenti esempi di codice mostrano come eliminare una tabella DynamoDB usando un SDK AWS .

.NET

SDK per .NET (v4)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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;
        }
    }
```

- Per i dettagli sull'API, DeleteTableconsulta AWS SDK per .NET API Reference.

Bash

AWS CLI con lo script Bash

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
}
```

Le funzioni di utilità utilizzate in questo esempio.

```
###############################################################################
# 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
}
```

- Per i dettagli sull'API, consulta DeleteTable AWS CLICommand Reference.

C++

SDK per C++

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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();
}
```

- Per i dettagli sull'API, DeleteTableconsulta AWS SDK per C++ API Reference.

CLI

AWS CLI

Come eliminare una tabella

Nell'esempio seguente `delete-table` elimina la tabella `MusicCollection`.

```
aws dynamodb delete-table \
    --table-name MusicCollection
```

Output:

```
{
    "TableDescription": {
        "TableStatus": "DELETING",
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableName": "MusicCollection",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "WriteCapacityUnits": 5,
            "ReadCapacityUnits": 5
        }
    }
}
```

Per ulteriori informazioni, consulta Eliminazione di una tabella nella Guida per gli sviluppatori di Amazon DynamoDB.

- Per i dettagli sull'API, consulta DeleteTable AWS CLICommand Reference.

Go

SDK per Go V2

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
}
```

- Per i dettagli sull'API, DeleteTableconsulta AWS SDK per Go API Reference.

Java

SDK per Java 2.x

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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!");
    }
}
```

- Per i dettagli sull'API, DeleteTableconsulta AWS SDK for Java 2.x API Reference.

JavaScript

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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;
};
```

- Per i dettagli sull'API, DeleteTableconsulta AWS SDK per JavaScript API Reference.

SDK per JavaScript (v2)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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);
  }
});
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK per JavaScript.
- Per i dettagli sull'API, DeleteTableconsulta AWS SDK per JavaScript API Reference.

Kotlin

SDK per Kotlin

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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")
    }
```

```
}
```

- Per i dettagli sull'API, DeleteTableconsulta AWS SDK for Kotlin API reference.

PHP

SDK per PHP

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
    public function deleteTable(string $TableName)
    {
        $this->customWaiter(function () use ($TableName) {
            return $this->dynamoDbClient->deleteTable([
                'TableName' => $TableName,
            ]);
        });
    }
```

- Per i dettagli sull'API, DeleteTableconsulta AWS SDK per PHP API Reference.

PowerShell

Strumenti per PowerShell V4

Esempio 1: elimina la tabella specificata. Prima di procedere con l'operazione, viene richiesta una conferma.

```
Remove-DDBTable -TableName "myTable"
```

Esempio 2: elimina la tabella specificata. Prima di procedere con l'operazione, non viene richiesta una conferma.

```
Remove-DDBTable -TableName "myTable" -Force
```

- Per i dettagli sull'API, vedere DeleteTablein AWS Strumenti per PowerShell Cmdlet Reference (V4).

Strumenti per V5 PowerShell

Esempio 1: elimina la tabella specificata. Prima di procedere con l'operazione, viene richiesta una conferma.

```
Remove-DDBTable -TableName "myTable"
```

Esempio 2: elimina la tabella specificata. Prima di procedere con l'operazione, non viene richiesta una conferma.

```
Remove-DDBTable -TableName "myTable" -Force
```

- Per i dettagli sull'API, vedere DeleteTablein AWS Strumenti per PowerShell Cmdlet Reference (V5).

Python

SDK per Python (Boto3)

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
```

- Per i dettagli sull'API, consulta DeleteTable AWSSDK for Python (Boto3) API Reference.

Ruby

SDK per Ruby

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
```

- Per i dettagli sull'API, DeleteTableconsulta AWS SDK per Ruby API Reference.

Rust

SDK per Rust

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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())),
    }
}
```

- Per i dettagli sulle API, consulta la DeleteTableguida di riferimento all'API AWS SDK for Rust.

SAP ABAP

SDK per SAP ABAP

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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.
```

- Per i dettagli sulle API, DeleteTableconsulta AWS SDK for SAP ABAP API reference.

Swift

SDK per Swift

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
```

```
        }
    }
```

- Per i dettagli sull'API, consulta la DeleteTableguida di riferimento all'API AWS SDK for Swift.

Per ulteriori esempi di DynamoDB, consulta Esempi di codice per l'utilizzo di DynamoDB AWS SDKs.

Conoscenze aggiuntive su DynamoDB

Per ulteriori informazioni sull'utilizzo di Amazon DynamoDB, consulta i seguenti argomenti:

- Utilizzo di tabelle e dati in DynamoDB
- Utilizzo di elementi e attributi in DynamoDB
- Esecuzione di query in DynamoDB
- Utilizzo degli indici secondari globali in DynamoDB
- Utilizzo delle transazioni
- Accelerazione in memoria con DynamoDB Accelerator (DAX)
- Programmazione con DynamoDB e SDK AWS

Genera codice di infrastruttura per Amazon DynamoDB utilizzando Console-to-Code

La Console-to-Code funzionalità di Amazon Q Developer semplifica la gestione dell'infrastruttura per Amazon DynamoDB trasformando le fasi di creazione manuale delle tabelle in codice di automazione riproducibile. Questa funzionalità aiuta gli sviluppatori a scalare in modo efficiente la configurazione delle risorse di database nei loro ambienti. Per ulteriori informazioni, consulta Automating Servizi AWS with Amazon Q Developer Console-to-Code.

Console-to-Code acquisisce configurazioni dettagliate delle tabelle DynamoDB, incluse chiavi di partizione, chiavi di ordinamento, impostazioni di throughput assegnate e indici secondari, e le converte in modelli precisi. infrastructure-as-code Utilizzando l'IA generativa, lo strumento assicura che il codice generato mantenga la compatibilità dei parametri stabilita durante il flusso di lavoro della console.

Gli sviluppatori possono generare il codice dell'infrastruttura DynamoDB in diversi formati, come:

- AWS Cloud Development Kit (AWS CDK) in TypeScript Python e Java
- CloudFormation in YAML o JSON

Questo approccio consente ai team di:

- Standardizzare la gestione delle risorse del database
- Implementare un'infrastruttura con controllo delle versioni
- Ridurre gli errori dovuti alla configurazione manuale

Console-to-Code for Amazon DynamoDB è disponibile in tutte le regioni AWS commerciali e offre una potente soluzione per trasformare le configurazioni manuali in codice di infrastruttura automatizzato e riproducibile.

Come funziona

Quando si utilizza Console-to-Code con DynamoDB, il processo prevede in genere:

1. Prototipazione nella console: utilizzo della console DynamoDB per creare e configurare risorse come le tabelle. Per ulteriori informazioni, consulta Connect to Amazon DynamoDB.
2. Registrazione delle azioni: Console-to-Code registra queste azioni mentre le esegui.
3. Generazione di codice: la funzionalità utilizza le funzionalità di IA generativa di Amazon Q Developer per trasformare le azioni della console in codice riutilizzabile nel formato preferito.
4. Personalizzazione del codice: è possibile quindi copiare o scaricare questo codice e personalizzarlo ulteriormente per i propri carichi di lavoro di produzione.

Vantaggi dell'utilizzo Console-to-Code con DynamoDB

Automazione semplificata

Conversione della creazione e della configurazione manuali di tabelle DynamoDB in codice riutilizzabile con un solo clic.

Best practice

Il codice generato segue le migliori pratiche AWS guidate per implementazioni affidabili.

Un ponte tra console e codice

Non è necessario scegliere tra l'utilizzo di Console di gestione AWS o Infrastructure-as-Code (IaC). È possibile utilizzare entrambi gli approcci insieme.

Sviluppo più veloce

Avvio rapido con un codice di automazione che può essere ulteriormente personalizzato per l'uso in produzione.

Casi d'uso di esempio

- Creazione di tabelle DynamoDB con attributi, chiavi e impostazioni di capacità specifici
- Configurazione di indici secondari globali e indici secondari locali
- Configurazione di policy di dimensionamento automatico per le tabelle DynamoDB
- Creazione di configurazioni di backup e ripristino
- Creazione e gestione di flussi DynamoDB

Nozioni di base

Per iniziare a utilizzare Console-to-Code con DynamoDB:

1. Accedi Console di gestione AWS e apri la console DynamoDB all'indirizzo. https://console.aws.amazon.com/DynamoDB/
2. Inizia a creare o modificare le risorse DynamoDB tramite l'interfaccia della console.
3. Usa la Console-to-Code funzione per generare codice per le tue azioni nel tuo formato preferito.
4. Copia o scarica il codice generato e personalizzalo in base alle esigenze specifiche.

Per ulteriori informazioni e istruzioni dettagliate sull'uso Console-to-Code, consulta Automating Servizi AWS with Amazon Q Developer Console-to-Code nella Amazon Q Developer User Guide.

Amazon DynamoDB: come funziona

Nelle seguenti sezioni viene riportata una panoramica dei componenti del servizio Amazon DynamoDB e di come interagiscono tra loro.

Argomenti

- Cheat sheet per DynamoDB
- Componenti principali di Amazon DynamoDB
- API DynamoDB
- Tipi di dati e regole di denominazione supportati in Amazon DynamoDB
- Classi di tabella DynamoDB
- Partizioni e distribuzione dei dati in DynamoDB
- Come passare da SQL a NoSQL
- Risorse e strumenti di apprendimento di Amazon DynamoDB

Cheat sheet per DynamoDB

Questo cheat sheet fornisce un riferimento rapido per lavorare con Amazon DynamoDB e le sue varie applicazioni. AWS SDKs

Configurazione iniziale

1. Iscriviti per. AWS
2. Ottenimento di una chiave di accesso AWS per accedere a DynamoDB in modo programmatico.
3. Configurazione delle credenziali DynamoDB.

Consulta anche:

- Configurazione di DynamoDB (servizio Web)
- Nozioni di base su DynamoDB
- Panoramica di base dei componenti principali

SDK o CLI

Scegli l'SDK preferito o configurare l'AWS CLI.

> **Note**
>
> Quando usi Windows, una barra rovesciata (\) che non è inclusa in un preventivo viene considerata una restituzione. AWS CLI Inoltre, è necessario evitare le virgolette e le parentesi tra virgolette all'interno di altre virgolette. Per un esempio, consulta la scheda Windows in "Creazione di una tabella" nella sezione seguente.

Consulta anche:

- AWS CLI con DynamoDB
- Nozioni di base su DynamoDB - fase 2

Operazioni di base

Questa sezione fornisce il codice per le attività di base di DynamoDB. Per ulteriori informazioni su queste attività, vedere Guida introduttiva a DynamoDB e. AWS SDKs

Creare una tabella

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
```

Scrittura di elementi su una tabella

```
aws dynamodb put-item \ --table-name Music \ --item file://item.json
```

Lettura di elementi da una tabella

```
aws dynamodb get-item \ --table-name Music \ --item file://item.json
```

Eliminazione di elementi da una tabella

```
aws dynamodb delete-item --table-name Music --key file://key.json
```

Esecuzione di una query su una tabella

```
aws dynamodb query --table-name Music
--key-condition-expression "ArtistName=:Artist and SongName=:Songtitle"
```

Eliminazione di una tabella

```
aws dynamodb delete-table --table-name Music
```

Elenco dei nomi delle tabelle

```
aws dynamodb list-tables
```

Regole di denominazione

- Tutti i nomi devono essere codificati in UTF-8 e fanno distinzione tra maiuscole e minuscole.

- I nomi delle tabelle e i nomi degli indici devono avere una lunghezza compresa tra 3 e 255 caratteri e possono contenere solo i seguenti caratteri:
  - `a-z`
  - `A-Z`
  - `0-9`
  - `_` (carattere di sottolineatura)
  - `-`(trattino)
  - `.` (punto)
- I nomi degli attributi devono contenere almeno un carattere e non devono superare i 64 KB.

Per ulteriori informazioni, consulta Regole di denominazione.

Nozioni di base sulle quote del servizio

Unità di lettura e scrittura

- Unità di capacità di lettura (RCU): una lettura ad elevata consistenza di lettura al secondo o due letture a consistenza finale al secondo, per elementi di dimensioni fino a 4 KB.
- Unità di capacità di scrittura (WCU): una scrittura al secondo per elementi di dimensioni fino a 1 KB.

Limiti della tabella

- Dimensioni della tabella: non vi è un limite pratico sulle dimensioni della tabella. Le tabelle non hanno restrizioni in termini di numero di item o di byte.
- Numero di tabelle: per ogni AWS account, esiste una quota iniziale di 2.500 tabelle per regione. AWS
- Limite di dimensione della pagina per query e scansione: esiste un limite di 1 MB per pagina, per query o scansione. Se i parametri della query o l'operazione di scansione su una tabella generano più di 1 MB di dati, DynamoDB restituisce gli elementi corrispondenti iniziali. Restituisce anche una proprietà `LastEvaluatedKey` che è possibile utilizzare in una nuova richiesta per leggere la pagina successiva.

Indici

- Indici secondari locali (LSIs): è possibile definire un massimo di cinque indici secondari locali. LSIs sono utili principalmente quando un indice deve avere una forte coerenza con la tabella di base.
- Indici secondari globali (GSIs): esiste una quota predefinita di 20 indici secondari globali per tabella.
- Attributi degli indici secondari globali proiettati per tabella: è possibile proiettare un massimo di 100 attributi in tutti gli indici secondari locali e globali di una tabella. Tale approccio si applica solo agli attributi proiettati specificati dall'utente.

Chiavi di partizione

- La lunghezza minima del valore di una chiave di partizione è di 1 byte. La lunghezza massima è 2048 byte.
- Non vi è un limite pratico relativo al numero dei valori di chiavi di partizione distinte, sia per le tabelle che per gli indici secondari.
- La lunghezza minima del valore di una chiave di ordinamento è di 1 byte. La lunghezza massima è 1024 byte.
- In generale, non vi è un limite pratico relativo al numero dei valori delle chiavi di ordinamento distinte per ogni valore della chiave di partizione. Fanno eccezione le tabelle con indici secondari.

Per ulteriori informazioni sugli indici secondari, sulla progettazione delle chiavi di partizione e delle chiavi di ordinamento, consulta Best practice.

Limiti per i tipi di dati di uso comune

- Stringa: la lunghezza di una stringa è vincolata dal limite massimo della dimensione dell'elemento, che è di 400 KB. Le stringhe sono Unicode con codifica binaria UTF-8.
- Numero: un numero può avere fino a 38 cifre di precisione e può essere positivo, negativo o zero.
- Binario: la lunghezza di un dato binario è vincolata dal limite massimo della dimensione dell'elemento, che è di 400 KB. Le applicazioni che funzionano con gli attributi di tipo binario devono codificare i dati nel formato Base64 prima di inviarli a DynamoDB.

Per un elenco completo dei tipi di dati supportati, consulta Tipi di dati. Per ulteriori informazioni, consulta anche Service Quotas.

Elementi, attributi e parametri di espressione

La dimensione massima di un elemento in DynamoDB è 400 KB, che include sia la lunghezza binaria del nome dell'attributo (lunghezza UTF-8) che le lunghezze binarie dei valori dell'attributo (lunghezza UTF-8). Il nome attributo viene conteggiato per il limite di dimensione.

Non vi è alcun limite al numero di valori in un attributo List, Map o Set purché l'elemento contenente i valori rientri nel limite di dimensione dell'elemento di 400 KB.

Per i parametri di espressione, la lunghezza massima di qualsiasi stringa di espressione è di 4 KB.

Per ulteriori informazioni sulle dimensioni degli elementi, sugli attributi e sui parametri di espressione, consulta Service Quotas.

Ulteriori informazioni

- Sicurezza
- Monitoraggio e registrazione
- Utilizzo di flussi
- Backup e ripristino Point-in-time
- Integrazione con altri servizi AWS
- Documentazione di riferimento dell'API
- Centro di architettura: best practice per i database
- Tutorial video
- Forum DynamoDB

Componenti principali di Amazon DynamoDB

In DynamoDB, le tabelle, le voci e gli attributi sono i componenti principali da utilizzare. Una tabella è una raccolta di elementi e ogni elemento è una raccolta di attributi. DynamoDB utilizza le chiavi primarie per identificare in modo univoco ciascun elemento in una tabella. Per catturare gli eventi di modifica dei dati nelle tabelle DynamoDB è possibile utilizzare DynamoDB Streams.

Esistono dei limiti in DynamoDB. Per ulteriori informazioni, consulta Quote in Amazon DynamoDB.

Il seguente video ti fornirà un'introduzione a tabelle, elementi ed attributi.

Tabelle, elementi e attributi

Tabelle, elementi e attributi

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"}

Di seguito sono elencati i componenti di base di DynamoDB:

- Tabelle: analogamente ad altri sistemi di database, DynamoDB memorizza i dati in tabelle. Una tabella è una raccolta di dati. Ad esempio, vedi la tabella di esempio chiamata People che puoi utilizzare per memorizzare le informazioni di contatto personali su amici, familiari o chiunque altro ti interessi. Potresti avere anche una tabella Cars per memorizzare le informazioni sui veicoli guidati dalle persone.
- Elementi: ogni tabella contiene zero o più elementi. Un item è un set di attributi identificabili in modo univoco tra tutti gli altri item. Nella tabella People, ogni item rappresenta una persona. Per la tabella Cars, ogni item rappresenta un veicolo. Gli elementi in DynamoDB sono simili per molti aspetti alle righe, ai record o alle tuple in altri sistemi di database. In DynamoDB, non c'è limite al numero di elementi che è possibile memorizzare in una tabella.
- Attributi: ogni elemento è composto da uno o più attributi. Un attributo è un elemento dati fondamentale che non ha bisogno di essere ulteriormente suddiviso. Ad esempio, un elemento in una tabella Persone contiene attributi denominati PersonID, LastNameFirstName, e così via. Per una tabella Department, un elemento potrebbe avere attributi come DepartmentID, Name, Manager e così via. Gli attributi in DynamoDB sono simili per molti aspetti ai campi o alle colonne presenti in altri sistemi di database.

Il diagramma seguente mostra una tabella denominata People con alcuni item e attributi di esempio.

```
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"
}
```

Tieni presente quanto segue sulla tabella People:

- Ogni item nella tabella ha un identificatore univoco, o chiave primaria, che distingue l'item da tutti gli altri nella tabella. Nella tabella People, la chiave primaria consiste in un attributo (PersonID).
- Oltre alla chiave primaria, la tabella People è schematica, il che significa che né gli attributi né i loro tipi di dati devono essere definiti in anticipo. Ogni item può avere i propri attributi distinti.

- La maggior parte degli attributi è scalare, il che significa che possono avere un solo valore. Le stringhe e i numeri sono esempi comuni di scalari.
- Alcuni elementi hanno un attributo nidificato (Indirizzo). DynamoDB supporta gli attributi nidificati fino a un massimo di 32 livelli di profondità.

Di seguito è riportata un'altra tabella di esempio denominata Music che puoi utilizzare per tenere traccia della tua raccolta musicale.

```
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"
}
```

Tieni presente quanto segue sulla tabella Music:

- La chiave primaria per Music è composta da due attributi (Artista e SongTitle). Ogni item nella tabella deve avere questi due attributi. La combinazione di Artista e SongTitledistingue ogni elemento della tabella da tutti gli altri.
- Oltre alla chiave primaria, la tabella Music è schematica, il che significa che né gli attributi né i loro tipi di dati devono essere definiti in anticipo. Ogni item può avere i propri attributi distinti.
- Uno degli elementi ha un attributo nidificato (PromotionInfo), che contiene altri attributi nidificati. DynamoDB supporta gli attributi nidificati fino a un massimo di 32 livelli di profondità.

Per ulteriori informazioni, consulta Utilizzo di tabelle e dati in DynamoDB.

Chiave primaria

Quando crei una tabella, oltre al nome della tabella, è necessario specificare la chiave primaria della tabella. La chiave primaria identifica in modo univoco ogni elemento della tabella, in modo che due elementi non possano avere la stessa chiave.

DynamoDB supporta due diversi tipi di chiavi primarie:

- Chiave di partizione: una chiave primaria semplice, composta da un attributo noto come chiave di partizione.

DynamoDB utilizza il valore della chiave di partizione come input per una funzione hash interna. L'output dalla funzione hash determina la partizione (spazio di archiviazione fisico interno a DynamoDB) in cui verrà memorizzato l'elemento.

In una tabella che ha solo una chiave di partizione, non è possibile che due item abbiano lo stesso valore di chiave di partizione.

La tabella People descritta in Tabelle, elementi e attributi è un esempio di una tabella con una chiave primaria semplice (PersonID). È possibile accedere direttamente a qualsiasi elemento nella tabella Persone fornendo il PersonIdvalore per tale elemento.

- Chiave di partizione e chiave di ordinamento: indicati come chiave primaria composita, questo tipo di chiave è costituito da due attributi. Il primo attributo è la chiave di partizione e il secondo attributo è la chiave di ordinamento.

  DynamoDB utilizza il valore della chiave di partizione come input per una funzione hash interna. L'output dalla funzione hash determina la partizione (spazio di archiviazione fisico interno a DynamoDB) in cui verrà memorizzato l'elemento. Tutti gli elementi con lo stesso valore della chiave di partizione vengono memorizzati insieme, in ordine per valore di chiave di ordinamento.

  In una tabella che ha una chiave di partizione e una chiave di ordinamento, è possibile che più elementi abbiano lo stesso valore della chiave di partizione. Tuttavia, questi elementi devono avere valori delle chiavi di ordinamento diversi.

  La tabella Music descritta in Tabelle, elementi e attributi è un esempio di tabella con una chiave primaria composita (Artist and SongTitle). È possibile accedere direttamente a qualsiasi elemento della tabella Musica, se si forniscono l'Artista e SongTitlei valori per tale elemento.

  Una chiave primaria composita offre maggiore flessibilità durante la query sui dati. Ad esempio, se si fornisci solo il valore per Artist, DynamoDB recupera tutti i brani di quell'artista. Per recuperare solo un sottoinsieme di brani di un particolare artista, puoi fornire un valore per Artista insieme a un intervallo di valori per. SongTitle

> **Note**
>
> La chiave di partizione di un item è anche nota come attributo hash. Il termine attributo hash deriva dall'uso di una funzione hash interna in DynamoDB che distribuisce uniformemente gli elementi di dati tra le partizioni, in base ai valori delle chiavi delle partizioni.

La chiave di ordinamento di un item è anche nota come attributo di intervallo. Il termine attributo di intervallo deriva dal modo in cui DynamoDB memorizza gli elementi con la stessa chiave di partizione fisicamente vicini, ordinati in base al valore della chiave di ordinamento.

Ogni attributo chiave primaria deve essere scalare (ovvero può contenere solo un singolo valore). Gli unici tipi di dati consentiti per gli attributi della chiave primaria sono stringa, numero o binario. Non ci sono restrizioni di questo tipo per altri attributi non di chiave.

Indici secondari

Puoi creare uno o più indici secondari su una tabella. Un indice secondario consente di eseguire query sui dati nella tabella utilizzando una chiave alternativa, oltre alle query sulla chiave primaria. DynamoDB non richiede l'utilizzo di indici, ma offre maggiore flessibilità alle applicazioni durante l'esecuzione di query sui dati. Dopo aver creato un indice secondario su una tabella, puoi leggere i dati dall'indice più o meno allo stesso modo della tabella.

DynamoDB supporta due diversi tipi di indici:

- Indice secondario globale: un indice con una chiave di partizione e una chiave di ordinamento che possono essere differenti da quelle presenti sulla tabella. I valori della chiave primaria negli indici secondari globali non devono necessariamente essere univoci.
- Indice secondario locale: un indice con la stessa chiave di partizione della tabella ma con una chiave di ordinamento diversa.

In DynamoDB, gli indici secondari globali GSIs () sono indici che coprono l'intera tabella e consentono di eseguire query su tutte le chiavi di partizione. Gli indici secondari locali (LSIs) sono indici che hanno la stessa chiave di partizione della tabella di base ma una chiave di ordinamento diversa.

Ogni tabella in DynamoDB ha una quota di 20 indici secondari globali (quota predefinita) e 5 indici secondari locali.

Nella tabella Music di esempio mostrata in precedenza, è possibile interrogare gli elementi di dati per Artista (chiave di partizione) o per Artista e SongTitle(chiave di partizione e chiave di ordinamento). E se volessi interrogare i dati anche per genere e? AlbumTitle Per fare ciò, potreste creare un indice su Genre e AlbumTitlequindi interrogare l'indice più o meno allo stesso modo in cui interroghereste la tabella Music.

Il diagramma seguente mostra la tabella Music di esempio, con un nuovo indice chiamato GenreAlbumTitle. Nell'indice, Genre è la chiave di partizione e AlbumTitlela chiave di ordinamento.

Tabella musicale	GenreAlbumTitle

{
"Artist": "No One You Know",
"SongTitle": "My Dog Spot",
"AlbumTitle": "Hey Now",
"Price": 1.98,
"Genre": "Country",
"CriticRating": 8.4
}
	{
	"Genre": "Country",
	"AlbumTitle": "Hey Now",
	"Artist": "No One You Know",
	"SongTitle": "My Dog Spot"
	}

{
"Artist": "No One You Know",
"SongTitle": "Somewhere Down The Road",
"AlbumTitle": "Somewhat Famous",
"Genre": "Country",
"CriticRating": 8.4,
"Year": 1984
}
	{
	"Genre": "Country",
	"AlbumTitle": "Somewhat Famous",
	"Artist": "No One You Know",
	"SongTitle": "Somewhere Down The Road"
	}

{
"Artist": "The Acme Band",
"SongTitle": "Still in Love",
"AlbumTitle": "The Buck Starts Here",
"Price": 2.47,
"Genre": "Rock",
"PromotionInfo": {
"RadioStationsPlaying": {
"KHCR",
"KQBX",
"WTNR",
	{
	"Genre": "Rock",
	"AlbumTitle": "The Buck Starts Here",
	"Artist": "The Acme Band",
	"SongTitle": "Still In Love"
	}

Tabella musicale	GenreAlbumTitle

<pre>"WJJH"
},
"TourDates": {
"Seattle": "20150622",
"Cleveland": "20150630"
},
"Rotation": "Heavy"
}
}</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>

Nota quanto segue sull'GenreAlbumTitleindice:

- Ogni indice appartiene a una tabella chiamata la tabella di base dell'indice. Nell'esempio precedente, Music è la tabella di base per l'GenreAlbumTitleindice.
- DynamoDB conserva gli indici automaticamente. Quando si aggiunge, aggiorna o elimina un elemento nella tabella di base, DynamoDB aggiunge, aggiorna o elimina l'elemento corrispondente in tutti gli indici che appartengono a quella tabella.
- Quando crei un indice, puoi specificare quali attributi verranno copiati o proiettati, dalla tabella di base all'indice. Come minimo, DynamoDB proietta gli attributi della chiave dalla tabella di base nell'indice. Questo è il caso di `GenreAlbumTitle`, in cui vengono proiettati solo gli attributi della chiave della tabella `Music` nell'indice.

È possibile eseguire una query nell'GenreAlbumTitleindice per trovare tutti gli album di un particolare genere (ad esempio, tutti gli album Rock). Puoi eseguire la query sull'indice per trovare tutti gli album di un determinato genere (ad esempio, tutti gli album Country con il titolo che inizia con la lettera H).

Per ulteriori informazioni, consulta Miglioramento dell'accesso ai dati con gli indici secondari in DynamoDB.

DynamoDB Streams

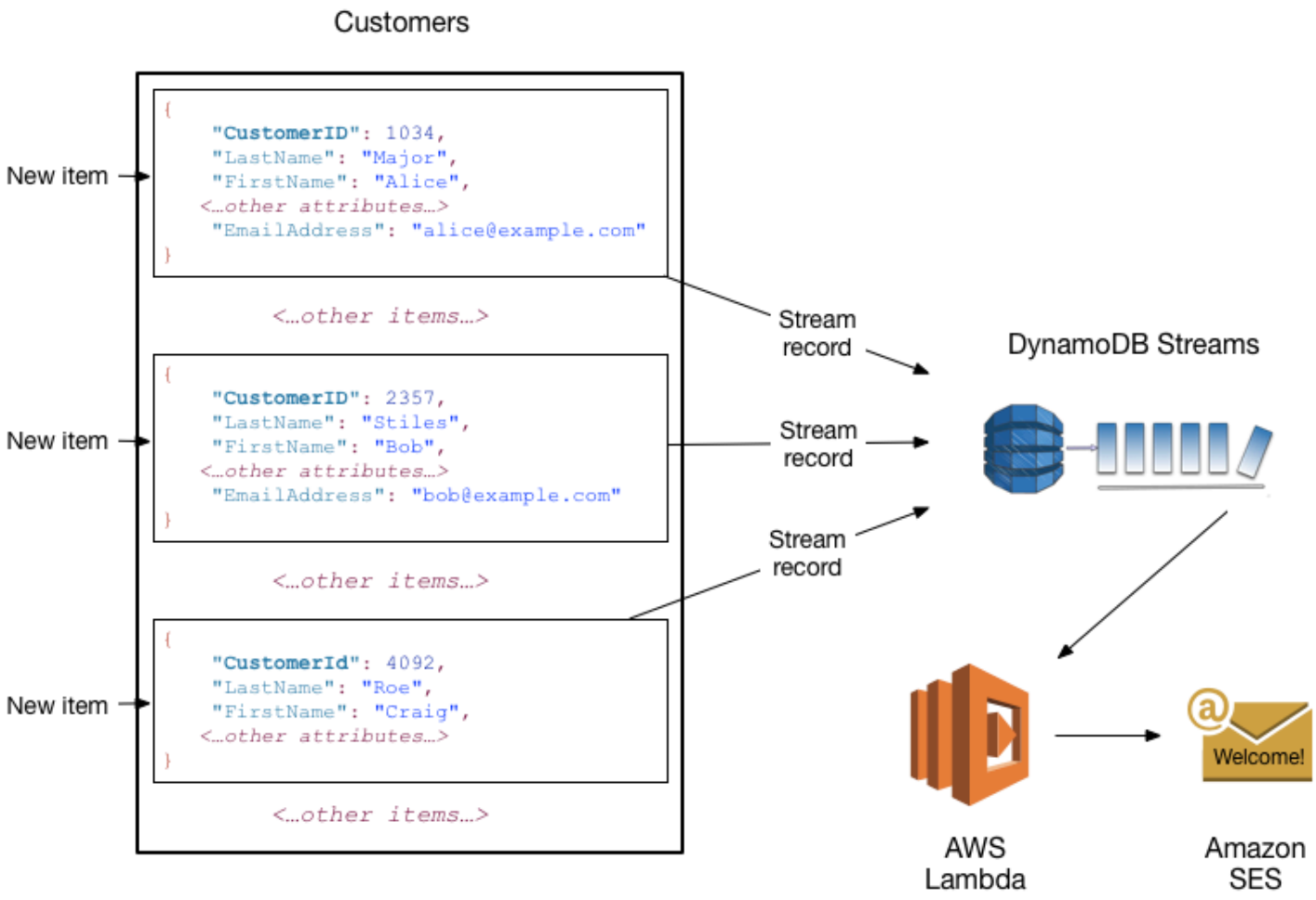
DynamoDB Streams è una funzionalità opzionale che cattura gli eventi di modifica dei dati nelle tabelle DynamoDB. I dati relativi a questi eventi vengono visualizzati nel flusso pressoché in tempo reale e nell'ordine in cui si sono verificati gli eventi.

Ogni evento è rappresentato da un record di flusso. Se su una tabella viene abilitato un flusso, DynamoDB Streams scrive un record di flusso ogni volta che si verifica uno dei seguenti eventi:

- Una nuova voce viene aggiunta alla tabella, il flusso acquisisce un'immagine dell'intera voce, inclusi tutti i suoi attributi.
- Una voce viene aggiornata, il flusso acquisisce l'immagine "prima" e "dopo" della modifica di qualsiasi attributo nella voce.
- Una voce viene eliminata dalla tabella, il flusso acquisisce un'immagine dell'intera voce prima che venga cancellata.

Ogni record di flusso contiene anche il nome della tabella, la data e l'ora dell'evento e altri metadati. I record di flusso hanno una durata di 24 ore e dopo vengono automaticamente rimossi dal flusso.

Puoi utilizzare DynamoDB Streams AWS Lambda insieme per creare un codice trigger che viene eseguito automaticamente ogni volta che un evento di interesse appare in uno stream. Ad esempio, considera la tabella Customers contenente le informazioni sui clienti di un'azienda. Supponiamo di voler inviare una e-mail di benvenuto a ogni nuovo cliente. È possibile abilitare un flusso su quella tabella e quindi associare il flusso a una funzione Lambda. La funzione Lambda viene eseguita ogni volta che viene visualizzato un nuovo record di flusso, ma vengono elaborati solo i nuovi elementi aggiunti alla tabella Customers. Per ogni elemento che ha un attributo `EmailAddress`, la funzione Lambda richiama Amazon Simple Email Service (Amazon SES) per inviare un messaggio e-mail a quell'indirizzo.



> **Note**
>
> In questo esempio, l'ultimo cliente, Craig Roe, non riceverà un messaggio e-mail perché non ha un `EmailAddress`.

Oltre ai trigger, DynamoDB Streams consente soluzioni potenti come la replica dei dati all'interno e AWS tra le regioni, le viste materializzate dei dati nelle tabelle DynamoDB, l'analisi dei dati utilizzando le viste materializzate Kinesis e molto altro.

Per ulteriori informazioni, consulta Acquisizione dei dati di modifica per DynamoDB Streams.

API DynamoDB

Per utilizzare Amazon DynamoDB, la propria applicazione deve usare alcune semplici operazioni API. Di seguito è riportato un riepilogo delle operazioni organizzate per categoria.

> **Note**
>
> Per l'elenco completo delle operazioni, consulta la Documentazione di riferimento dell'API Amazon DynamoDB.

Argomenti

- Piano di controllo
- Piano dati
- DynamoDB Streams
- Transazioni

Piano di controllo

Le operazioni del piano di controllo consentono di creare e gestire le tabelle DynamoDB. Ti permettono anche di utilizzare indici, flussi e altri oggetti che dipendono dalle tabelle.

- `CreateTable`: crea una nuova tabella. Se si desidera, è possibile creare uno o più indici secondari e abilitare DynamoDB Streams per la tabella.
- `DescribeTable`: restituisce informazioni su una tabella, come lo schema della chiave primaria, le impostazioni di velocità effettiva e le informazioni sugli indici.
- `ListTables`: restituisce i nomi di tutte le tabelle in un elenco.
- `UpdateTable`: modifica le impostazioni di una tabella o dei relativi indici, crea o rimuove nuovi indici su una tabella o modifica le impostazioni di DynamoDB Streams per una tabella.
- `DeleteTable`: rimuove una tabella e tutti i relativi oggetti dipendenti da DynamoDB.

Piano dati

Le operazioni piano dei dati consentono di eseguire operazioni di creazione, lettura, aggiornamento ed eliminazione (chiamate anche CRUD) sui dati in una tabella. Alcune operazioni del piano dati consentono inoltre di leggere i dati da un indice secondario.

È possibile utilizzarePartiQL: un linguaggio di query compatibile con SQL per Amazon DynamoDB, per eseguire queste operazioni CRUD oppure è possibile utilizzare il classico CRUD di DynamoDB che separa ogni operazione in una chiamata API distinta. APIs

PartiQL: un linguaggio di query compatibile con SQL

- `ExecuteStatement`: legge più elementi da una tabella. È inoltre possibile scrivere o aggiornare un singolo elemento da una tabella. Quando si scrive o si aggiorna un singolo elemento, è necessario specificare gli attributi della chiave primaria.
- `BatchExecuteStatement`: scrive, aggiorna o legge più elementi da una tabella. Questa operazione è più efficiente di `ExecuteStatement` perché l'applicazione ha bisogno solo di un singolo viaggio di andata e ritorno sulla rete per scrivere o leggere gli elementi.

Classico APIs

Creazione dei dati

- `PutItem`: scrive un singolo elemento in una tabella. È necessario specificare gli attributi della chiave primaria, ma non è necessario specificare altri attributi.
- `BatchWriteItem`: scrive fino a 25 elementi in una tabella. Questa operazione è più efficiente della chiamata ripetuta di `PutItem` perché la tua applicazione ha bisogno solo di un singolo round trip di rete per scrivere gli elementi.

Lettura dei dati

- `GetItem`: recupera un singolo elemento da una tabella. È necessario specificare la chiave primaria per l'item desiderato. Puoi recuperare l'intero item o solo un sottoinsieme dei suoi attributi.
- `BatchGetItem`: richiama fino a 100 elementi da una o più tabelle. Questa operazione è più efficiente della chiamata ripetuta di `GetItem` perché la tua applicazione ha bisogno solo di un singolo round trip di rete per leggere gli elementi.
- `Query`: recupera tutti gli elementi che hanno una chiave di partizione specifica. È necessario specificare il valore della chiave di partizione. Puoi recuperare gli interi item o solo un sottoinsieme dei loro attributi. Facoltativamente, puoi applicare una condizione ai valori delle chiavi di ordinamento, in modo da recuperare solo un sottoinsieme di dati con la stessa chiave di partizione. Puoi utilizzare questa operazione su una tabella, a condizione che la tabella abbia sia una chiave di partizione che una chiave di ordinamento. Puoi anche utilizzare questa operazione su un indice, a condizione che l'indice abbia sia una chiave di partizione che una chiave di ordinamento.
- `Scan`: recupera tutti gli elementi nella tabella o nell'indice specificati. Puoi recuperare gli interi item o solo un sottoinsieme dei loro attributi. Facoltativamente, puoi applicare una condizione di filtro per restituire solo i valori che ti interessano e scartare il resto.

Aggiornamento dei dati

- `UpdateItem`: modifica uno o più attributi in un elemento. Dovrai specificare la chiave primaria per l'item che vuoi modificare. Puoi aggiungere nuovi attributi e modificare o rimuovere gli attributi esistenti. Puoi inoltre eseguire aggiornamenti condizionali, in modo che l'aggiornamento abbia esito positivo solo quando viene soddisfatta una condizione definita dall'utente. Facoltativamente, puoi implementare un contatore atomico che incrementa o decrementa un attributo numerico senza interferire con altre richieste di scrittura.

Eliminazione di dati

- `DeleteItem`: elimina un singolo elemento da una tabella. Dovrai specificare la chiave primaria per l'item che vuoi eliminare.
- `BatchWriteItem`: elimina fino a 25 elementi da una o più tabelle. Questa operazione è più efficiente della chiamata ripetuta di `DeleteItem` perché la tua applicazione ha bisogno solo di un singolo round trip di rete per eliminare gli elementi.

> **Note**
>
> È possibile utilizzare `BatchWriteItem` sia per creare che per eliminare i dati.

DynamoDB Streams

Le operazioni di DynamoDB Streams consentono di abilitare o disabilitare un flusso su una tabella e consentire l'accesso ai record di modifica dei dati contenuti in un flusso.

- `ListStreams`: restituisce un elenco di tutti i flussi o solo il flusso per una tabella specifica.
- `DescribeStream`: restituisce le informazioni su un flusso, come ad esempio il suo Amazon Resource Name (ARN) e dove l'applicazione può iniziare a leggere i primi record di flusso.
- `GetShardIterator`: restituisce un iteratore di partizioni che è una struttura dati che l'applicazione utilizza per recuperare i record dal flusso.
- `GetRecords`: recupera uno o più record di flusso, utilizzando un determinato iteratore di partizioni.

Transazioni

La transazioni forniscono atomicità, coerenza, isolamento e durabilità (ACID), consentendoti di mantenere più facilmente dati corretti nelle applicazioni.

È possibile utilizzarePartiQL: un linguaggio di query compatibile con SQL per Amazon DynamoDB, per eseguire operazioni transazionali oppure è possibile utilizzare il classico CRUD di DynamoDB APIs che separa ogni operazione in una chiamata API distinta.

PartiQL: un linguaggio di query compatibile con SQL

- `ExecuteTransaction`— Un'operazione in batch che consente le operazioni CRUD su più elementi all'interno e tra tabelle con un risultato garantito. all-or-nothing

Classico APIs

- `TransactWriteItems`— Un'operazione in batch che consente di `Put` `Delete` eseguire operazioni su più elementi sia all'interno che tra tabelle diverse con un all-or-nothing risultato garantito. `Update`
- `TransactGetItems`: un'operazione in batch che consente alle operazioni `Get` di recuperare più elementi da una o più tabelle.

Tipi di dati e regole di denominazione supportati in Amazon DynamoDB

In questa sezione vengono descritte le regole di denominazione in Amazon DynamoDB e i vari tipi di dati supportati da DynamoDB. Sono presenti limiti che si applicano ai tipi di dati. Per ulteriori informazioni, consulta Tipi di dati.

Argomenti

- Regole di denominazione
- Tipi di dati
- Descrittori del tipo di dati

Regole di denominazione

Le tabelle, gli attributi e altri oggetti in DynamoDB devono avere un nome. I nomi devono essere significativi e concisi, ad esempio nomi come Prodotti, Libri e Autori sono auto-esplicativi.

Di seguito sono illustrate le regole di denominazione per DynamoDB:

- Tutti i nomi devono essere codificati in UTF-8 e rispettano la differenza tra maiuscole e minuscole.
- I nomi delle tabelle e i nomi degli indici devono avere una lunghezza compresa tra 3 e 255 caratteri e possono contenere solo i seguenti caratteri:
  - `a-z`
  - `A-Z`
  - `0-9`
  - `_` (carattere di sottolineatura)
  - `-` (trattino)
  - `.` (punto)
- I nomi degli attributi devono contenere almeno un carattere e non devono superare i 64 KB. Una best practice consiste nel mantenere i nomi degli attributi i più brevi possibile. Questo aiuta a ridurre le unità di richiesta di lettura utilizzate, poiché i nomi degli attributi sono inclusi nella misurazione dell'utilizzo dell'archiviazione e della velocità di trasmissione effettiva.

  Di seguito sono elencate le eccezioni. Questi nomi di attributo non devono superare i 255 caratteri:
  - Nomi delle chiavi di partizione degli indici secondari
  - Nomi delle chiavi di ordinamento degli indici secondari
  - I nomi degli attributi proiettati specificati dall'utente (applicabile solo agli indici secondari locali).

Parole riservate e caratteri speciali

DynamoDB ha una lista di parole riservate e caratteri speciali. Per un elenco completo, consulta Parole riservate in DynamoDB. Inoltre, i seguenti caratteri hanno un significato speciale in DynamoDB: # (cancelletto) e : (due punti).

Sebbene DynamoDB consenta di utilizzare le parole riservate e i caratteri speciali per i nomi, consigliamo di evitare di farlo perché comporta la definizione di variabili segnaposto ogni volta che questi nomi vengono utilizzati in un'espressione. Per ulteriori informazioni, consulta Nomi di attributi di espressione (alias) in DynamoDB.

Tipi di dati

DynamoDB supporta molti tipi di dati diversi per gli attributi all'interno di una tabella. Possono essere classificati come segue:

- Tipi scalari: un tipo scalare può rappresentare esattamente un valore. I tipi scalari sono numero, stringa, binario, booleano e null.
- Tipi di documento: un tipo di documento può rappresentare una struttura complessa con attributi nidificati, ad esempio in un documento JSON. I tipi di documento sono elenco e mappa.
- Tipi di set: un tipo di set può rappresentare più valori scalari. I tipi di set sono set di stringhe, set di numeri e set di binari.

Quando si crea una tabella o un indice secondario, è necessario specificare i nomi e i tipi di dati di ciascun attributo di chiave primaria (chiave di partizione e chiave di ordinamento). Inoltre, ogni attributo chiave primaria deve essere definito come tipo stringa, numero o binario.

DynamoDB è un database NoSQL senza schema. Ciò significa che, oltre agli attributi della chiave primaria, non è necessario definire attributi o tipi di dati quando si creano tabelle. In confronto, i database relazionali richiedono di definire i nomi e i tipi di dati di ogni colonna quando crei una tabella.

Le seguenti sono descrizioni di ciascun tipo di dati, insieme a esempi in formato JSON.

Tipi scalari

I tipi scalari sono numero, stringa, binario, booleano e null.

Numero

I numeri possono essere positivi, negativi o zero. I numeri possono avere una precisione fino a 38 cifre. Il superamento di questo limite comporta un'eccezione. Se è necessaria una maggiore precisione di quella a 38 cifre, è possibile utilizzare le stringhe.

- Intervallo positivo: da 1E-130 a 9.99999999999999999999999999999999999999E+125
- Intervallo negativo: da -9.99999999999999999999999999999999999999E+125 a -1E-130

In DynamoDB, i numeri sono rappresentati come lunghezza variabile. Gli zero iniziali e finali vengono tagliati.

Tutti i numeri vengono inviati a DynamoDB attraverso la rete come stringhe in modo da aumentare al massimo la compatibilità tra linguaggi e librerie. Tuttavia, DynamoDB li tratta come attributi di tipo number per operazioni matematiche.

Puoi utilizzare il tipo di dati numero per rappresentare una data o un timestamp. Un modo per eseguire questa operazione consiste nell'utilizzare il tempo epoch, ovvero il numero di secondi trascorsi dalle 00:00:00 UTC del 1° gennaio 1970. Ad esempio, il tempo epoca (Unix epoch) `1437136300` rappresenta le 12:31:40 UTC del 17 luglio 2015.

Per ulteriori informazioni, vedere http://en.wikipedia. org/wiki/Unix_ora.

Stringa

Le stringhe sono Unicode con codifica binaria UTF-8. La lunghezza minima di una stringa può essere pari a zero se l'attributo non viene utilizzato come chiave per un indice o una tabella ed è vincolato dal limite di dimensione massima dell'elemento DynamoDB pari a 400 KB.

Le seguenti limitazioni addizionali si applicano agli attributi della chiave primaria che sono definiti come tipo string:

- Per una chiave primaria semplice, la lunghezza massima del valore del primo attributo (la chiave di partizione) è 2.048 bytes.
- Per una chiave primaria composita, la lunghezza massima del valore del secondo attributo (la chiave di ordinamento) è 1.024 byte.

DynamoDB raccoglie e confronta le stringhe utilizzando i byte della codifica di stringa UTF-8 sottostante. Ad esempio, "a" (0x61) è maggiore di "A" (0x41) e "¿" (0xC2BF) è maggiore di "z" (0x7A).

Puoi utilizzare il tipo di dati stringa per rappresentare una data o un timestamp. Un modo per farlo è utilizzando le stringhe ISO 8601, come mostrato in questi esempi:

- `2016-02-15`
- `2015-12-21T17:42:34Z`
- `20150311T122706Z`

Per ulteriori informazioni, consulta http://en.wikipedia. org/wiki/ISO_8601.

> **Note**
>
> A differenza dei database relazionali tradizionali, DynamoDB non supporta a livello nativo dati di tipo data/ora. Può invece risultare utile per archiviare i dati di data e ora come tipo di dati numerici, utilizzando il formato Unix Epoch.

Binario

Gli attributi di tipo binario possono memorizzare qualsiasi tipo di dati binari, ad esempio testi compressi, dati crittografati o immagini. Quando DynamoDB confronta i valori binari tratta ciascun byte dei dati binari come non firmato.

La lunghezza di un attributo binario può essere zero, se l'attributo non viene utilizzato come chiave per un indice o una tabella ed è vincolato dal limite di dimensione massima dell'elemento DynamoDB pari a 400 KB.

Se definisci un attributo di chiave primaria come attributo di tipo binario, si applicano i seguenti vincoli aggiuntivi:

- Per una chiave primaria semplice, la lunghezza massima del valore del primo attributo (la chiave di partizione) è 2.048 bytes.
- Per una chiave primaria composita, la lunghezza massima del valore del secondo attributo (la chiave di ordinamento) è 1.024 byte.

Le applicazioni devono codificare i valori binari in formato codificato Base64 prima di inviarli a DynamoDB. Quando vengono ricevuti questi valori, DynamoDB decodifica i dati in una matrice di byte non firmata e li usa come lunghezza dell'attributo binario.

L'esempio seguente è un attributo binario che utilizza il testo con codifica Base64:

```
dGhpcyB0ZXh0IGlzIGJhc2U2NC1lbmNvZGVk
```

Booleano

Un attributo di tipo booleano può memorizzare `true` o `false`.

Null

Null rappresenta un attributo con uno stato sconosciuto o non definito.

Tipi di documento

I tipi di documento sono elenco e mappa. Questi tipi di dati possono essere annidati l'uno nell'altro per rappresentare strutture dati complesse fino a una profondità di 32 livelli.

Non esiste alcun limite al numero di valori in un elenco o in una mappa, purché l'elemento contenente i valori rientri nel limite di dimensione dell'elemento di DynamoDB (400 KB).

Un valore di attributo può essere una stringa vuota o un valore binario se l'attributo non viene utilizzato per una tabella o una chiave dell'indice. Il valore attributo non può essere un set vuoto (String Set, Number Set o Binary Set), tuttavia sono consentite liste e mappe vuote. Valori String e Binary vuoti sono consentiti all'interno di elenchi e mappe. Per ulteriori informazioni, consulta Attributes.

List

Un attributo di tipo elenco può memorizzare una raccolta di valori ordinata. Gli elenchi sono racchiusi tra parentesi quadre: `[ ... ]`

Un elenco è simile a una matrice JSON. Non ci sono restrizioni sui tipi di dati che possono essere memorizzati in un elemento di elenco e gli elementi di un elemento di elenco non devono essere dello stesso tipo.

L'esempio seguente mostra un elenco che contiene due stringhe e un numero:

```
FavoriteThings: ["Cookies", "Coffee", 3.14159]
```

> **Note**
>
> DynamoDB consente di utilizzare i singoli elementi all'interno di elenchi, anche se tali elementi sono profondamente annidati. Per ulteriori informazioni, consulta Utilizzo di espressioni in DynamoDB.

Eseguire la mappatura

Un attributo di tipo mappa può memorizzare una raccolta di coppie nome-valore non ordinata. Le mappe sono racchiuse tra parentesi graffe: `{ ... }`

Una mappa è simile a un oggetto JSON. Non ci sono restrizioni sui tipi di dati che possono essere memorizzati in un elemento di mappa e gli elementi in una mappa non devono essere dello stesso tipo.

Le mappe sono ideali per la memorizzazione di documenti JSON in DynamoDB. L'esempio seguente mostra una mappa che contiene una stringa, un numero e un elenco annidato che contiene un'altra mappa.

```
{
    Day: "Monday",
    UnreadEmails: 42,
    ItemsOnMyDesk: [
        "Coffee Cup",
        "Telephone",
        {
            Pens: { Quantity : 3},
            Pencils: { Quantity : 2},
            Erasers: { Quantity : 1}
        }
    ]
}
```

> **Note**
>
> DynamoDB consente di utilizzare i singoli elementi all'interno di mappe, anche se tali elementi sono profondamente annidati. Per ulteriori informazioni, consulta Utilizzo di espressioni in DynamoDB.

Set

DynamoDB supporta tipi che rappresentano set di valori number, string o binary. Tutti gli elementi di un set devono essere dello stesso tipo. Ad esempio, un attributo di tipo Number Set può contenere solo numeri, String Set può contenere solo stringhe e così via.

Non esiste alcun limite al numero di valori in un set, purché l'elemento contenente i valori rientri nel limite di dimensione dell'elemento di DynamoDB (400 KB).

Ogni valore all'interno di un set deve essere univoco. L'ordine dei valori all'interno di un set non viene mantenuto. Pertanto, le tue applicazioni non devono fare affidamento su nessun ordine di elementi

particolare all'interno del set. DynamoDB non supporta set vuoti, tuttavia, all'interno di un set sono consentiti stringhe vuote e valori binari.

L'esempio seguente mostra un set di stringhe, un set di numeri e un set binario:

```
["Black", "Green", "Red"]

[42.2, -19, 7.5, 3.14]

["U3Vubnk=", "UmFpbnk=", "U25vd3k="]
```

Descrittori del tipo di dati

Il protocollo API DynamoDB di basso livello utilizza descrittori del tipo di dati come token che indicano a DynamoDB come interpretare ogni attributo.

Di seguito è riportato un elenco completo dei descrittori del tipo di dati di DynamoDB:

- **`S`**: String
- **`N`**: Number
- **`B`**: binario
- **`BOOL`**: Boolean
- **`NULL`**: null
- **`M`**: Map
- **`L`**: list
- **`SS`**: set string
- **`NS`**: set numerico
- **`BS`**: set binario

Classi di tabella DynamoDB

DynamoDB offre due classi di tabelle progettate per aiutarti a ottimizzare i costi. La classe di tabella DynamoDB Standard è quella predefinita ed è consigliata per la maggior parte dei carichi di lavoro. La classe di tabella DynamoDB Standard-Infrequent Access (DynamoDB Standard (accesso infrequente)) è ottimizzata per le tabelle in cui l'archiviazione è il costo principale. Ad esempio, le

tabelle che archiviano dati a cui si accede raramente, come i registri delle applicazioni, i vecchi post sui social media, la cronologia degli ordini di e-commerce e i risultati di gioco precedenti, sono buoni candidati per la classe di tabella Standard (accesso infrequente). Per i dettagli sui prezzi, consulta Prezzi di Amazon DynamoDB.

Ogni tabella DynamoDB è associata a una classe di tabella (DynamoDB Standard, per impostazione predefinita). Tutti gli indici secondari associati alla tabella utilizzano la stessa classe di tabella. Ogni classe di tabella offre prezzi diversi per l'archiviazione dati e le richieste di lettura e scrittura. Puoi selezionare la classe di tabella più conveniente per la tua tabella in base ai relativi modelli di utilizzo di archiviazione e throughput.

La scelta di una classe di tabella non è permanente: puoi modificare questa impostazione utilizzando la Console di gestione AWS AWS CLI o l'SDK. AWS DynamoDB supporta anche la gestione della classe di tabelle AWS CloudFormation utilizzando tabelle a regione singola e tabelle globali. Per ulteriori informazioni sulla selezione della classe di tabella, consulta Considerazioni sulla scelta di una classe di tabella in DynamoDB.

Partizioni e distribuzione dei dati in DynamoDB

Amazon DynamoDB memorizza i dati nelle partizioni. Una partizione è un'allocazione di storage per una tabella, supportata da unità a stato solido (SSDs) e replicata automaticamente su più zone di disponibilità all'interno di una regione. AWS La gestione delle partizioni è gestita interamente da DynamoDB: non è mai necessario gestire le partizioni da soli.

Quando crei una tabella, lo stato iniziale della tabella è `CREATING`. Durante questa fase, DynamoDB assegna partizioni sufficienti alla tabella in modo che possa gestire i requisiti di velocità effettiva assegnata. Puoi iniziare a scrivere e leggere i dati della tabella dopo che lo stato della tabella diventa `ACTIVE`.

DynamoDB alloca partizioni aggiuntive a una tabella nelle seguenti situazioni:

- Se aumenti le impostazioni di throughput assegnato della tabella oltre quelle supportate dalle partizioni esistenti.
- Se una partizione esistente esaurisce la capacità ed è necessario più spazio di storage.

La gestione delle partizioni avviene automaticamente in background ed è trasparente per le tue applicazioni. La tua tabella rimane disponibile e supporta completamente i requisiti di throughput assegnati.

Per ulteriori dettagli, consultare Progettazione delle chiavi di partizione.

Anche gli indici secondari globali in DynamoDB sono costituiti da partizioni. I dati in un indice secondario globale sono memorizzati separatamente dai dati della tabella di base, ma le partizioni dell'indice si comportano in modo simile alle partizioni della tabella.

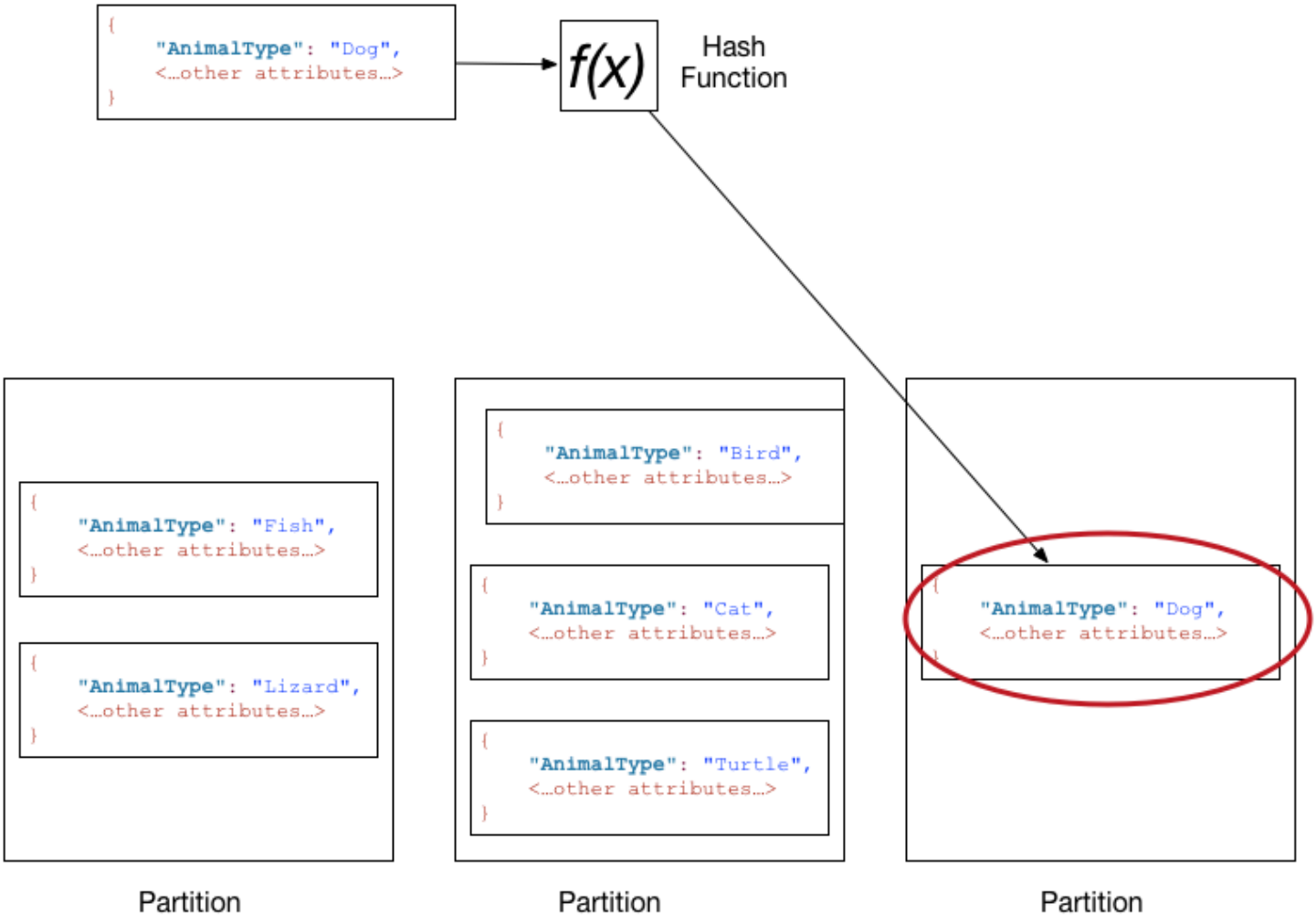
Distribuzione dei dati: chiave di partizione

Se la tabella ha una chiave primaria semplice (solo chiave di partizione), DynamoDB memorizza e recupera ciascun elemento in base al valore della chiave di partizione.

Per scrivere un elemento nella tabella, DynamoDB utilizza il valore della chiave di partizione come input per una funzione hash interna. Il valore di output dalla funzione hash determina la partizione in cui verrà memorizzato l'item.

Per leggere un elemento della tabella, è necessario specificare il valore della chiave di partizione per l'elemento. DynamoDB utilizza questo valore come input per la funzione hash, producendo così la partizione in cui è possibile trovare l'elemento.

Il diagramma seguente mostra una tabella denominata Pets con più partizioni. La chiave primaria della tabella è AnimalType(viene mostrato solo questo attributo chiave). DynamoDB utilizza la funzione hash per determinare dove memorizzare un nuovo elemento, in questo caso in base al valore hash della stringa Dog. Tieni presenti che gli elementi non sono memorizzati in ordine. La posizione di ciascun item è determinata dal valore hash della chiave di partizione.



> **Note**
>
> DynamoDB è ottimizzato per una distribuzione uniforme degli elementi sulle partizioni di una tabella, indipendentemente dal numero di partizioni presenti. Ti consigliamo di scegliere una chiave di partizione che possa avere un numero elevato di valori distinti rispetto al numero di item nella tabella.

Distribuzione dei dati: chiave di partizione e chiave di ordinamento

Se la tabella ha una chiave primaria composita (chiave di partizione e chiave di ordinamento), DynamoDB calcola il valore hash della chiave di partizione come descritto in Distribuzione dei dati: chiave di partizione. Tuttavia, tende a mantenere gli elementi che hanno lo stesso valore della chiave di partizione vicini e ordinati in base al valore dell'attributo della chiave di ordinamento. L'insieme di elementi che hanno lo stesso valore della chiave di partizione è denominato raccolta di elementi. Le

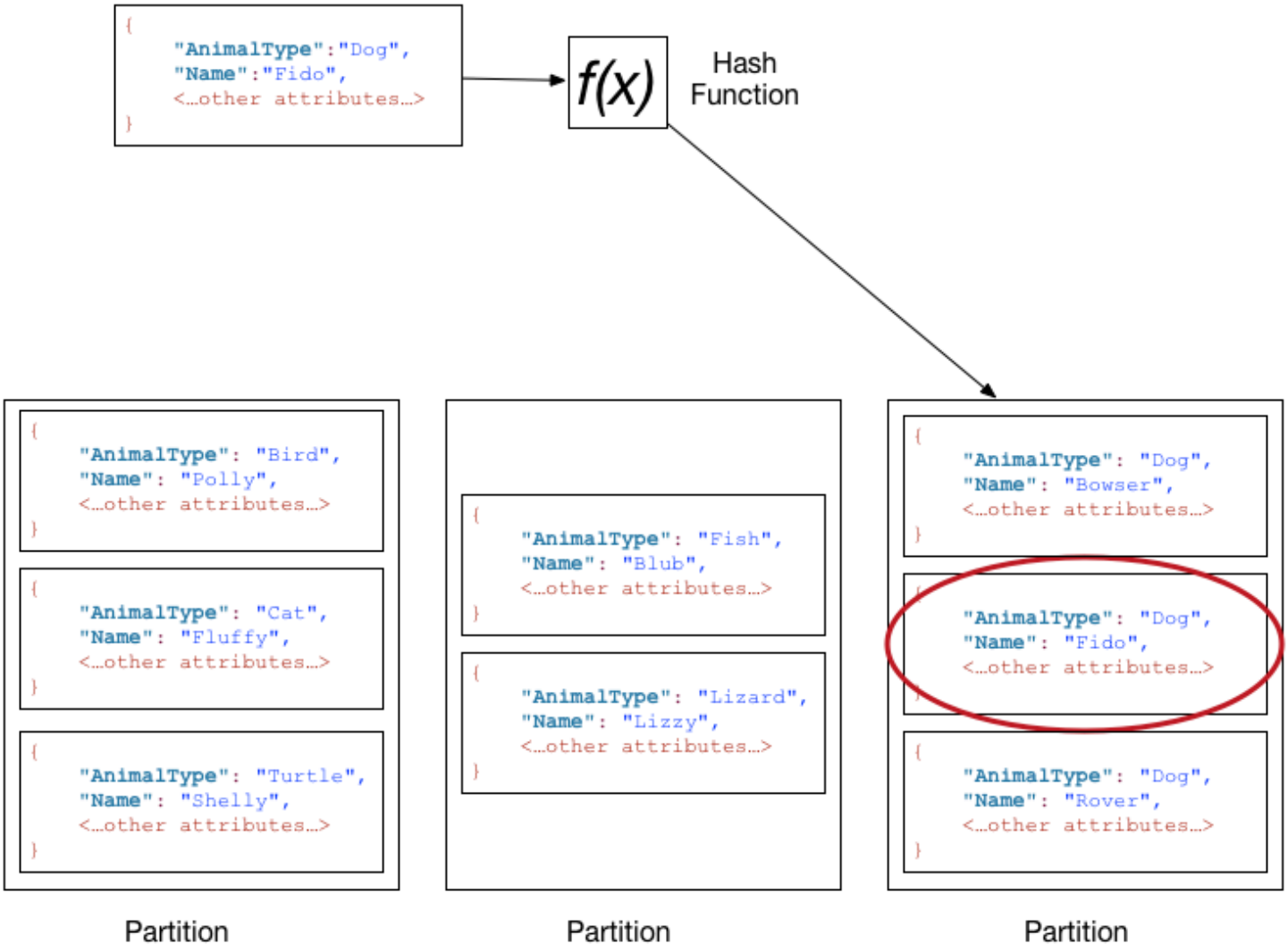
raccolte di elementi sono ottimizzate per il recupero efficiente degli intervalli di elementi all'interno della raccolta. Se la tabella non ha indici secondari locali, DynamoDB suddividerà automaticamente la raccolta di elementi su tutte le partizioni necessarie per archiviare i dati e garantire il throughput di lettura e scrittura.

Per scrivere un elemento nella tabella, DynamoDB calcola il valore hash della chiave di partizione per determinare quale partizione lo deve contenere. In quella partizione, diversi item potrebbero avere lo stesso valore della chiave di partizione. Pertanto, DynamoDB memorizza l'elemento tra gli altri con la stessa chiave di partizione, in ordine crescente e per chiave di ordinamento.

Per leggere un elemento della tabella, è necessario specificare il valore della chiave di partizione e della chiave di ordinamento. DynamoDB calcola il valore hash della chiave di partizione, producendo la partizione in cui è possibile trovare l'elemento.

Se gli elementi desiderati hanno lo stesso valore della chiave di partizione è possibile leggere più elementi dalla tabella in una singola operazione (`Query`). DynamoDB restituisce tutti gli elementi con tale valore della chiave di partizione. Facoltativamente, puoi applicare una condizione alla chiave di ordinamento in modo che restituisca solo gli elementi all'interno di un determinato intervallo di valori.

Supponiamo che la tabella Pets abbia una chiave primaria composita composta da AnimalType(chiave di partizione) e Name (chiave di ordinamento). Il seguente diagramma mostra DynamoDB che scrive un elemento con un valore della chiave di partizione uguale a Dog e un valore della chiave di ordinamento uguale a Fido.



Per leggere lo stesso elemento dalla tabella Pets, DynamoDB calcola il valore hash di Dog, producendo la partizione in cui sono memorizzati questi elementi. DynamoDB esegue quindi la scansione dei valori degli attributi della chiave di ordinamento finché non trova Fido.

Per leggere tutti gli elementi con un AnimalTypeof Dog, puoi eseguire un'`Query`operazione senza specificare una condizione di chiave di ordinamento. Come impostazione predefinita, le voci vengono restituite nell'ordine in cui sono memorizzate (ovvero in ordine crescente per chiave di ordinamento). Facoltativamente, puoi richiedere l'ordine decrescente.

Per eseguire la query solo di alcune voci Dog, puoi applicare una condizione alla chiave di ordinamento (ad esempio, solo le voci Dog dove Name inizia con una lettera compresa nell'intervallo da A a K).

> **Note**
>
> In una tabella DynamoDB, non esiste un limite superiore per il numero di valori delle chiavi di ordinamento distinti per valore di chiave di partizione. Se c'è bisogno di memorizzare miliardi di elementi Dog nella tabella Pets, DynamoDB assegna automaticamente l'archiviazione sufficiente per gestire questo requisito.

Come passare da SQL a NoSQL

Se sei uno sviluppatore di applicazioni, potresti avere una certa esperienza nell'uso dei sistemi di gestione di database relazionali (RDBMS) e Structured Query Language (SQL). Quando si inizia a lavorare con Amazon DynamoDB, è possibile che siano riscontrate molte somiglianze, ma anche molte cose diverse. NoSQL è un termine usato per descrivere i sistemi di database non relazionali altamente disponibili, scalabili e ottimizzati per prestazioni elevate. Anziché il modello relazionale, i database NoSQL (ad esempio DynamoDB) utilizzano modelli alternativi per la gestione dei dati, ad esempio, coppie chiave-valore o archiviazione di documenti. Per ulteriori informazioni, consulta Che cos'è NoSQL?.

Amazon DynamoDB supporta PartiQL, un linguaggio di query open source compatibile con SQL che semplifica la query dei dati in modo efficiente, indipendentemente da dove o in quale formato sono memorizzati. Con PartiQL, è possibile elaborare facilmente dati strutturati da database relazionali, dati semi-strutturati e nidificati in formati open data e persino dati senza schema in database NoSQL o documenti che consentono attributi diversi per righe diverse. Per ulteriori informazioni, consulta Linguaggio di query PartiQL.

Le seguenti sezioni descrivono le attività comuni dei database, confrontando le istruzioni SQL con le equivalenti operazioni DynamoDB.

> **Note**
>
> Gli esempi SQL in questa sezione sono compatibili con MySQL RDBMS.
> Gli esempi DynamoDB in questa sezione mostrano il nome dell'operazione DynamoDB insieme ai parametri per l'operazione in formato JSON.

Argomenti

- Scelta tra relazionale (SQL) e NoSQL
- Differenze nell'accesso a un database relazionale (SQL) e a DynamoDB
- Differenze tra un database relazionale (SQL) e DynamoDB durante la creazione di una tabella
- Differenze tra l'ottenimento di informazioni sulle tabelle da un database relazionale (SQL) e DynamoDB
- Differenze tra un database relazionale (SQL) e DynamoDB durante la scrittura di dati in una tabella
- Differenze tra un database relazionale (SQL) e DynamoDB durante la lettura di dati in una tabella
- Differenze tra un database relazionale (SQL) e DynamoDB nella gestione degli indici
- Differenze tra un database relazionale (SQL) e DynamoDB durante la modifica di dati in una tabella
- Differenze tra un database relazionale (SQL) e DynamoDB durante l'eliminazione di dati da una tabella
- Differenze tra un database relazionale (SQL) e DynamoDB durante la rimozione di una tabella

Scelta tra relazionale (SQL) e NoSQL

Le applicazioni di oggi hanno requisiti più impegnativi che mai. Ad esempio, un gioco online potrebbe iniziare con pochi utenti e una piccola quantità di dati. Tuttavia, se il gioco ha successo, può facilmente superare le risorse del sistema di gestione del database sottostante. Non è raro che le applicazioni Web abbiano centinaia, migliaia o milioni di utenti simultanei, con terabyte di nuovi dati generati al giorno. I database per tali applicazioni devono gestire decine (o centinaia) di migliaia di letture e scritture al secondo.

Amazon DynamoDB è adatto per questi tipi di carichi di lavoro. Gli sviluppatori possono iniziare con una piccola quantità e aumentare gradualmente l'utilizzo man mano che l'applicazione diventa più popolare. DynamoDB si adatta perfettamente per gestire quantità di dati molto grandi e un numero molto elevato di utenti.

Per ulteriori informazioni sulla modellazione di database relazionali tradizionali e su come adattarla per DynamoDB, consulta Best practice per la modellazione dei dati relazionali in DynamoDB.

La seguente tabella mostra alcune differenze dettagliate tra un sistema di gestione di database relazionali (RDBMS) e DynamoDB.

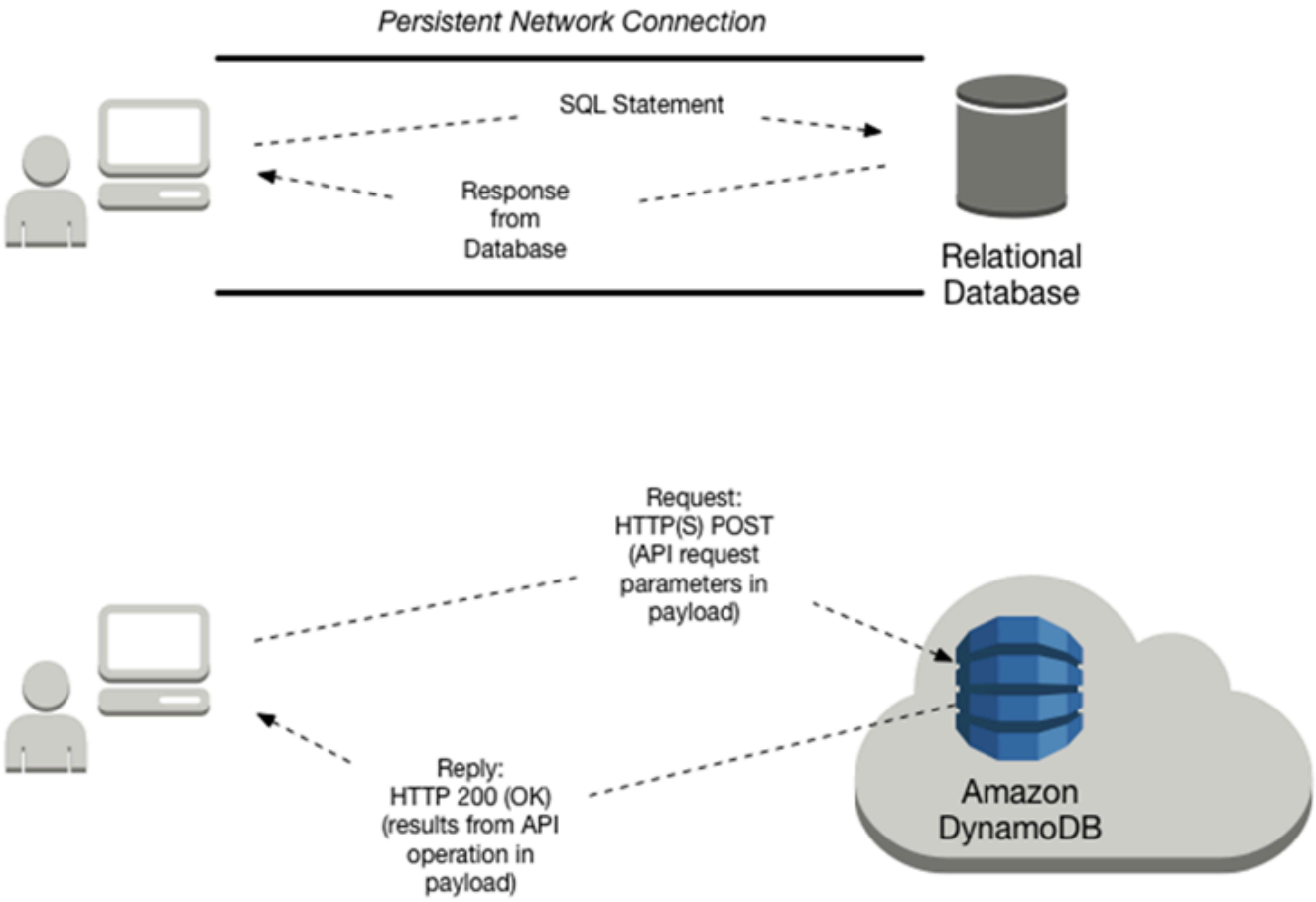
Caratteristica	Sistema di gestione di database relazionali (RDBMS)	Amazon DynamoDB
Carichi di lavoro ottimali	Query ad hoc, data warehousing, OLAP (Online Analytical Processing).	Applicazioni Web-scale, inclusi i social network, i giochi, la condivisione di file multimediali e IoT (Internet of Things).
Modello di dati	Il modello relazionale richiede uno schema ben definito, in cui i dati vengono normalizzati in tabelle, righe e colonne. Inoltre, tutte le relazioni sono definite tra tabelle, colonne, indici e altri elementi del database.	DynamoDB è privo di schema. Ogni tabella deve avere una chiave primaria per identificare in modo univoco ciascun item di dati, ma non esistono vincoli simili su altri attributi non di chiave. può gestire dati strutturati o semi strutturati, inclusi i documenti JSON. DynamoDB può gestire dati strutturati o semistrutturati, inclusi documenti JSON.
Accesso ai dati	SQL è lo standard per memorizzare e recuperare i dati. I database relazionali offrono un ampio set di strumenti per semplificare lo sviluppo di applicazioni basate su database, ma tutti questi strumenti utilizzano SQL.	È possibile utilizzare il Console di gestione AWS AWS CLI, the o WorkBench NoSQL per lavorare con DynamoDB ed eseguire attività ad hoc. PartiQL, un linguaggio di query compatibile con SQL, consente di selezionare, inserire, aggiornare ed eliminare i dati in DynamoDB. Le applicazioni possono utilizzare i kit di sviluppo AWS software (SDKs) per lavorare con DynamoDB utilizzando interfacce basate su oggetti,

Caratteristica	Sistema di gestione di database relazionali (RDBMS)	Amazon DynamoDB
		incentrate sui documenti o di basso livello.
Prestazioni	I database relazionali sono ottimizzati per lo storage, quindi le prestazioni generalmente dipendono dal sottosistema del disco. Gli sviluppatori e gli amministratori di database devono ottimizzare le query, gli indici e le strutture delle tabelle per ottenere prestazioni ottimali.	DynamoDB è ottimizzato per il calcolo, quindi le prestazioni sono principalmente una funzione dell'hardware sottostante e della latenza di rete. Come servizio gestito, DynamoDB isola l'utente e le sue applicazioni da questi dettagli di implementazione, in modo da potersi concentrare sulla progettazione e sulla realizzazione di applicazioni solide e con prestazioni elevate.
Dimensionamento	È più facile applicare il dimensionamento con hardware più veloce. È anche possibile che le tabelle del database si estendano su più host in un sistema distribuito, ma ciò richiede ulteriori investimenti. I database relazionali hanno dimensioni massime per il numero e la dimensione dei file che impongono limiti superiori alla scalabilità.	DynamoDB è progettato per aumentare orizzontalmente usando cluster distribuiti di hardware. Questo progetto consente un maggiore throughput senza una maggiore latenza. I clienti specificano i requisiti di velocità effettiva e DynamoDB assegna risorse sufficienti per soddisfare tali requisiti. Non ci sono limiti superiori per il numero di item per tabella, né per la dimensione totale della tabella.

Differenze nell’accesso a un database relazionale (SQL) e a DynamoDB

Per consentire all'applicazione di accedere a un database, deve essere autenticata per assicurarsi che possa utilizzare il database. Deve essere autorizzata in modo da poter eseguire solo le operazioni per le quali dispone di autorizzazioni.

Il diagramma seguente mostra l'interazione del client con un database relazionale e con Amazon DynamoDB.



La seguente tabella contiene ulteriori dettagli sulle attività di interazione del client:

Caratteristica	Sistema di gestione di database relazionali (RDBMS)	Amazon DynamoDB
Strumento per l'accesso al database	La maggior parte dei database relazionali fornisce un'interf accia a riga di comando (CLI)	Nella maggior parte dei casi, scrivi il codice dell'appl icazione. Puoi anche utilizzar

Caratteristica	Sistema di gestione di database relazionali (RDBMS)	Amazon DynamoDB
	per permetterti di immettere istruzioni SQL ad hoc e vedere immediatamente i risultati.	e Console di gestione AWS, the AWS Command Line Interface (AWS CLI) o NoSQL Workbench per inviare richieste ad hoc a DynamoDB e visualizzare i risultati. PartiQL, un linguaggi o di query compatibile con SQL, consente di seleziona re, inserire, aggiornare ed eliminare i dati in DynamoDB.
Connessione al database	Un programma applicativo stabilisce e mantiene una connessione di rete con il database. Al termine dell'appl icazione, termina anche la connessione.	DynamoDB è un servizio Web che offre interazioni stateless. Le applicazioni non hanno bisogno di mantenere connessioni di rete persistenti. L'interazione con DynamoDB invece si verifica utilizzando richieste e risposte HTTP(S).
Autenticazione	Un'applicazione non può connettersi al database finché non viene autentica ta. RDBMS può eseguire l'autenticazione autonomam ente oppure eseguire l'offload di questa attività sul sistema operativo host o su un servizio di directory.	Ogni richiesta a DynamoDB deve essere accompagnata da una firma crittografica che autentica la specifica richiesta . AWS SDKsForniscono tutta la logica necessaria per creare firme e richieste di firma. Per ulteriori informazioni, consulta Firmare le richieste AWS API in. Riferimenti generali di AWS

Caratteristica	Sistema di gestione di database relazionali (RDBMS)	Amazon DynamoDB
Autorizzazione	Le applicazioni possono eseguire solo operazioni per le quali sono state autorizzate. Gli amministratori di database o i proprietari di applicazioni possono utilizzare le istruzioni SQL `GRANT` e `REVOKE` per controllare l'accesso agli oggetti del database (come le tabelle), ai dati (come le righe all'interno di una tabella) o la possibilità di emettere alcune istruzioni SQL.	In DynamoDB, l'autorizzazione è gestita AWS Identity and Access Management da (IAM). È possibile scrivere una policy IAM per concedere le autorizzazioni su una risorsa DynamoDB (ad esempio una tabella), quindi consentire agli utenti e ai ruoli di utilizzare tale policy. IAM offre anche un controllo di accesso a granulometria fine per singoli elementi di dati nelle tabelle DynamoDB. Per ulteriori informazioni, consulta Identity and Access Management per Amazon DynamoDB.
Invio di una richiesta	L'applicazione emette un'istruzione SQL per ogni operazione di database che desidera eseguire. Alla ricezione dell'istruzione SQL, RDBMS verifica la sintassi, crea un piano per l'esecuzione dell'operazione ed esegue il piano.	L'applicazione invia le richieste HTTP(S) a DynamoDB. Le richieste contengono il nome dell'operazione da eseguire insieme ai parametri. DynamoDB esegue immediatamente la richiesta.
Ricezione di una risposta	RDBMS restituisce i risultati dell'istruzione SQL. Se si verifica un errore, RDBMS restituisce uno stato e un messaggio di errore.	DynamoDB restituisce una risposta HTTP(S) contenente i risultati dell'operazione. Se si verifica un errore, DynamoDB restituisce uno stato e messaggi di errore HTTP.

Differenze tra un database relazionale (SQL) e DynamoDB durante la creazione di una tabella

Le tabelle sono le strutture di dati fondamentali nei database relazionali e in Amazon DynamoDB. Un sistema di gestione di database relazionali (RDBMS) richiede la definizione dello schema della tabella al momento della sua creazione. Le tabelle DynamoDB invece sono prive di schema Oltre alla chiave primaria, non è necessario definire alcun attributo o tipo di dati extra al momento della creazione della tabella.

La sezione seguente confronta il modo in cui creeresti una tabella con SQL con il modo in cui la creeresti con DynamoDB.

Argomenti

- Creazione di una tabella con SQL
- Creazione di una tabella con DynamoDB

Creazione di una tabella con SQL

Con SQL useresti l'istruzione `CREATE TABLE` per creare una tabella, come illustrato nell'esempio seguente.

```
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 chiave principale di questa tabella è costituita da Artist e. SongTitle

Dovrai definire tutte le colonne e i tipi di dati della tabella e la chiave primaria della tabella. Puoi usare l'istruzione `ALTER TABLE` per modificare queste definizioni successivamente, se necessario.

Molte implementazioni SQL ti consentono di definire le specifiche di storage per la tua tabella, come parte dell'istruzione `CREATE TABLE`. Se non diversamente indicato, la tabella viene creata con le

impostazioni di storage predefinite. In un ambiente di produzione, un amministratore di database può aiutare a determinare i parametri di storage ottimali.

Creazione di una tabella con DynamoDB

Utilizza l'operazione `CreateTable` per creare una tabella con modalità assegnata, specificando i parametri come illustrato di seguito:

```
{
    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 chiave primaria per questa tabella è composta da Artist (chiave di partizione) e SongTitle(chiave di ordinamento).

Dovrai fornire i seguenti parametri a `CreateTable`:

- `TableName`: il nome della tabella.

- `KeySchema`— Attributi utilizzati per la chiave primaria. Per ulteriori informazioni, consultare Tabelle, elementi e attributi e Chiave primaria.
- `AttributeDefinitions`— Tipi di dati per gli attributi chiave dello schema.
- `ProvisionedThroughput (for provisioned tables)`— Numero di letture e scritture al secondo necessarie per questa tabella. DynamoDB riserva risorse di storage e di sistema sufficienti in modo che i requisiti di throughput siano sempre soddisfatti. Puoi usare l'operazione `UpdateTable` per modificare queste definizioni successivamente, se necessario. Non è necessario specificare i requisiti di archiviazione di una tabella perché l'allocazione dello spazio di archiviazione è gestita interamente da DynamoDB.

Differenze tra l'ottenimento di informazioni sulle tabelle da un database relazionale (SQL) e DynamoDB

Puoi verificare se una tabella è stata creata in base alle tue specifiche. In un database relazionale viene visualizzato tutto lo schema della tabella. Le tabelle Amazon DynamoDB sono prive di schema, perciò vengono mostrati solo gli attributi della chiave primaria.

Argomenti

- Ottenimento di informazioni su una tabella con SQL
- Ottenimento di informazioni su una tabella in DynamoDB

Ottenimento di informazioni su una tabella con SQL

La maggior parte dei sistemi di gestione dei database relazionali (RDBMS) consente di descrivere la struttura di una tabella, ovvero colonne, tipi di dati, definizione della chiave primaria e così via. Non esiste un metodo standard per fornire la descrizione in SQL. Tuttavia, molti sistemi di database forniscono un comando `DESCRIBE`. Di seguito è riportato un esempio da MySQL.

```
DESCRIBE Music;
```

Viene restituita la struttura della tabella, con tutti i nomi di colonna, i tipi di dati e le dimensioni.

```
+------------+-------------+------+-----+---------+-------+
| 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 chiave principale di questa tabella è costituita da Artist e. SongTitle

Ottenimento di informazioni su una tabella in DynamoDB

DynamoDB dispone di un'operazione `DescribeTable`, che è simile. L'unico parametro è il nome della tabella.

```
{
    TableName : "Music"
}
```

La risposta di `DescribeTable` sarà simile alla seguente.

```
{
  "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` restituisce anche informazioni sugli indici sulla tabella, le impostazioni del throughput assegnato, un conteggio approssimativo degli elementi e altri metadati.

Differenze tra un database relazionale (SQL) e DynamoDB durante la scrittura di dati in una tabella

Le tabelle dei database relazionali contengono righe di dati. Le righe sono costituiti dacolumns. Le tabelle Amazon DynamoDB contengonoELEMENTO. gli elementi sono costituiti da attributi.

In questa sezione viene descritto come scrivere una riga o un item in una tabella.

Argomenti

- Scrittura dei dati in una tabella con SQL
- Scrittura dei dati in una tabella in DynamoDB

Scrittura dei dati in una tabella con SQL

Una tabella in un database relazionale è una struttura di dati bidimensionale composta da righe e colonne. Alcuni sistemi di gestione di database forniscono anche il supporto per dati semi-strutturati, solitamente con tipi di dati JSON o XML nativi. Tuttavia, i dettagli di implementazione variano tra i fornitori.

In SQL, userai l'istruzione `INSERT` per aggiungere una riga a una tabella.

```
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 chiave principale di questa tabella è costituita da Artist e. SongTitle Dovrai specificare i valori per queste colonne.

> **Note**
>
> Questo esempio utilizza la colonna Tags per memorizzare dati semistrutturati sulle canzoni nella tabella Music. La colonna Tags è definita come di tipo TEXT e pertanto può memorizzare fino a 65.535 caratteri in MySQL.

Scrittura dei dati in una tabella in DynamoDB

In Amazon DynamoDB, puoi utilizzare l'API DynamoDB, o PartiQL, un linguaggio di query compatibile con SQL, per aggiungere un elemento a una tabella.

DynamoDB API

Con l'API DynamoDB, si utilizza l'operazione `PutItem` per aggiungere un elemento a una tabella.

```
{
    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 chiave principale di questa tabella è costituita da Artist e. SongTitle Dovrai specificare i valori per questi attributi.

Ecco alcune cose fondamentali da sapere su questo `PutItem` di esempio:

- DynamoDB fornisce il supporto nativo per i documenti utilizzando JSON. Questo rende DynamoDB ideale per l'archiviazione dei dati semistrutturati, ad esempio Tag. Puoi anche recuperare e modificare i dati dai documenti JSON.
- La tabella Music non ha attributi predefiniti, tranne la chiave primaria (Artista e SongTitle).
- La maggior parte dei database SQL è basata sulle transazioni. Quanto emetti un'istruzione `INSERT`, le modifiche dei dati non sono permanenti fino a quando non emetti un'istruzione `COMMIT`. Con Amazon DynamoDB, gli effetti di un'operazione `PutItem` diventano permanenti quando DynamoDB risponde con un codice di stato HTTP 200 (0K).

Di seguito vengono riportati altri esempi di `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**
>
> Oltre a `PutItem`, DynamoDB supporta un'operazione `BatchWriteItem` per la scrittura di più elementi contemporaneamente.

PartiQL for DynamoDB

Con PartiQL, utilizzi l'operazione `ExecuteStatement` per aggiungere un elemento a una tabella, utilizzando la dichiarazione `Insert` PartiQL.

```
INSERT into Music value {
    'Artist': 'No One You Know',
```

```
    'SongTitle': 'Call Me Today',
    'AlbumTitle': 'Somewhat Famous',
    'Year' : '2015',
    'Genre' : 'Acme'
}
```

La chiave primaria di questa tabella è composta da Artist e SongTitle. Dovrai specificare i valori per questi attributi.

> **Note**
>
> Per esempi di codice che utilizzano `Insert` e `ExecuteStatement`, consulta Istruzioni INSERT PartiQL per DynamoDB.

Differenze tra un database relazionale (SQL) e DynamoDB durante la lettura di dati in una tabella

Con SQL, userai l'istruzione `SELECT` per recuperare una o più righe da una tabella. Utilizza la clausola `WHERE` per determinare i dati che vengono restituiti.

Ciò è diverso dall'utilizzo di Amazon DynamoDB che fornisce le seguenti operazioni per la lettura dei dati:

- `ExecuteStatement` recupera uno o più elementi da una tabella. `BatchExecuteStatement` recupera più elementi da tabelle diverse in un'unica operazione. Entrambe queste operazioni utilizzano PartiQL, un linguaggio di query compatibile con SQL.
- `GetItem`: recupera un singolo elemento da una tabella. Questo è il modo più efficiente per leggere un singolo item, perché fornisce accesso diretto a una posizione fisica dell'item. (DynamoDB fornisce anche il`BatchGetItem`operazione, che consente di eseguire fino a 100`GetItem`In un'unica operazione.)
- `Query`: recupera tutti gli elementi che hanno una chiave di partizione specifica. In questi item è possibile applicare una condizione alla chiave di ordinamento e recuperare solo un sottoinsieme dei dati. `Query` offre accesso rapido ed efficace alle partizioni in cui i dati vengono memorizzati. Per ulteriori informazioni, consulta Partizioni e distribuzione dei dati in DynamoDB.
- `Scan`: recupera tutti gli elementi nella tabella o nell'indice specificati. Questa operazione non deve essere utilizzata con tabelle di grandi dimensioni in quanto può consumare notevoli quantità di risorse di sistema.

> **Note**
>
> Con un database relazionale puoi utilizzare l'istruzione `SELECT` per eseguire il join dei dati di più tabelle e restituire i risultati. I join sono fondamentali per il modello relazionale. Per garantire che i join vengano eseguiti in modo efficiente, il database e le applicazioni devono essere ottimizzati per le prestazioni su base continuativa. DynamoDB è un database NoSQL non relazionale che non supporta join di tabelle. Le applicazioni leggono i dati da una tabella alla volta.

Nelle sezioni seguenti vengono descritti diversi casi d'uso per la lettura dei dati e le operazioni da intraprendere per eseguire queste attività con un database relazionale e con DynamoDB.

Argomenti

- Differenze nella lettura di un elemento usando la chiave primaria
- Differenze nell'esecuzione di query su una tabella
- Differenze nella scansione di una tabella

Differenze nella lettura di un elemento usando la chiave primaria

Un modello di accesso comune per i database consiste nel leggere un singolo item da una tabella. Dovrai specificare la chiave primaria dell'item desiderato.

Argomenti

- Lettura di un elemento usando la chiave primaria con SQL
- Lettura di un elemento usando la chiave primaria in DynamoDB

Lettura di un elemento usando la chiave primaria con SQL

In SQL, userai l'istruzione `SELECT` per recuperare i dati da una tabella. Puoi richiedere una o più colonne nel risultato (o tutte se usi l'operatore `*`). La clausola `WHERE` determina le righe da restituire.

La seguente è un'istruzione `SELECT` per recuperare una singola riga dalla tabella Music. La clausola `WHERE` specifica i valori delle chiavi primarie.

```
SELECT *
FROM Music
```

```
WHERE Artist='No One You Know' AND SongTitle = 'Call Me Today'
```

Puoi modificare questa query per recuperare solo un sottoinsieme delle colonne:

```
SELECT AlbumTitle, Year, Price
FROM Music
WHERE Artist='No One You Know' AND SongTitle = 'Call Me Today'
```

Tieni presente che la chiave primaria per questa tabella è costituita da Artist e. SongTitle

Lettura di un elemento usando la chiave primaria in DynamoDB

In Amazon DynamoDB, puoi utilizzare l'API DynamoDB, o PartiQL, un linguaggio di query compatibile con SQL, per leggere un elemento da una tabella.

DynamoDB API

Con l'API DynamoDB, si utilizza l'operazione `PutItem` per aggiungere un elemento a una tabella.

DynamoDB offre l'operazione `GetItem` per il recupero di un elemento tramite la propria chiave primaria. `GetItem` è altamente efficiente perché fornisce accesso diretto a una posizione fisica dell'elemento. Per ulteriori informazioni, consulta Partizioni e distribuzione dei dati in DynamoDB.

Per impostazione predefinita, `GetItem` restituisce l'intero item con tutti gli attributi.

```
{
    TableName: "Music",
    Key: {
        "Artist": "No One You Know",
        "SongTitle": "Call Me Today"
    }
}
```

Puoi aggiungere un parametro `ProjectionExpression` per restituire solo alcuni degli attributi.

```
{
    TableName: "Music",
    Key: {
        "Artist": "No One You Know",
        "SongTitle": "Call Me Today"
    },
```

```
    "ProjectionExpression": "AlbumTitle, Year, Price"
}
```

Tieni presente che la chiave primaria di questa tabella è costituita da Artist e. SongTitle

L'operazione `GetItem` di DynamoDB è molto efficiente. Utilizza i valori delle chiavi primarie per determinare l'esatta posizione di archiviazione dell'elemento in questione e lo recupera direttamente da quella posizione. L'istruzione SQL `SELECT` è altrettanto efficiente per recuperare gli elementi tramite i valori delle chiavi primarie.

L'SQL`SELECT`supporta molti tipi di query e scansioni di tabelle. DynamoDB offre una funzionalità simile con le operazioni `Query` e`Scan`, che sono descritte in Differenze nell'esecuzione di query su una tabella e Differenze nella scansione di una tabella.

L'istruzione SQL `SELECT` può eseguire join di tabelle, consentendo di recuperare i dati da più tabelle contemporaneamente. I join sono più efficaci laddove le tabelle del database sono normalizzate e le relazioni tra le tabelle sono chiare. Tuttavia, se esegui il join di troppe tabelle in una sola istruzione `SELECT`, le prestazioni dell'applicazione possono risultare compromesse. Puoi risolvere questi problemi utilizzando la replica del database, le viste materializzate o le riscritture delle query.

DynamoDB è un database non relazionale e non supporta i join di tabella. Se si sta eseguendo la migrazione di un'applicazione esistente da un database relazionale a DynamoDB, è necessario denormalizzare il modello di dati per eliminare la necessità delle operazioni join.

PartiQL for DynamoDB

Con PartiQL, utilizzi l'operazione `ExecuteStatement` per leggere un elemento da una tabella, utilizzando la dichiarazione `Select` PartiQL.

```
SELECT AlbumTitle, Year, Price
FROM Music
WHERE Artist='No One You Know' AND SongTitle = 'Call Me Today'
```

Nota che la chiave primaria di questa tabella è composta da Artist e SongTitle.

> **Note**
>
> L'istruzione select PartiQL può essere utilizzata anche per eseguire query o scansione di una tabella DynamoDB

Per esempi di codice che utilizzano `Select` e `ExecuteStatement`, consulta Istruzioni SELECT PartiQL per DynamoDB.

Differenze nell'esecuzione di query su una tabella

Un altro modello di accesso comune è la lettura di più item da una tabella, in base ai criteri di query.

Argomenti

- Esecuzione di query su una tabella con SQL
- Esecuzione di query su una tabella in DynamoDB

Esecuzione di query su una tabella con SQL

Quando usi l'istruzione SQL `SELECT` puoi eseguire query su colonne chiave, colonne non chiave o una combinazione di esse. La clausola `WHERE` determina quali righe vengono restituite, come mostrato negli esempi seguenti.

```
/* 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;
```

Nota che la chiave primaria di questa tabella è composta da Artist e SongTitle.

Esecuzione di query su una tabella in DynamoDB

In Amazon DynamoDB, puoi utilizzare l'API DynamoDB, o PartiQL, un linguaggio di query compatibile con SQL, per eseguire una query di un elemento da una tabella.

DynamoDB API

Con Amazon DynamoDB puoi usare l'operazione `Query` per recuperare i dati in modo simile. L'operazione `Query` fornisce un accesso rapido ed efficiente alle posizioni fisiche in cui sono archiviati i dati. Per ulteriori informazioni, consulta Partizioni e distribuzione dei dati in DynamoDB.

È possibile utilizzare `Query` con qualsiasi tabella o indice secondario. Dovrai specificare una condizione di uguaglianza per il valore della chiave di partizione e facoltativamente fornire un'altra condizione per l'attributo della chiave di ordinamento, se definito.

Il parametro `KeyConditionExpression` specifica i valori delle chiavi su cui vuoi eseguire la query. Puoi utilizzare un `FilterExpression` facoltativo per rimuovere determinati item dai risultati prima che vengano restituiti.

In DynamoDB, è necessario utilizzare `ExpressionAttributeValues` come segnaposto nei parametri delle espressioni (come `KeyConditionExpression` e`FilterExpression`). Questo comportamento è analogo all'uso delle variabili di associazione nei database relazionali in cui sostituisci i valori effettivi nell'istruzione `SELECT` al runtime.

Nota che la chiave primaria di questa tabella è composta da Artist e SongTitle.

Di seguito vengono riportati altri esempi di `Query`.

```
// 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, puoi eseguire una query utilizzando l'operazione `ExecuteStatement` e l'istruzione `Select` sulla chiave di partizione.

```
SELECT AlbumTitle, Year, Price
FROM Music
WHERE Artist='No One You Know'
```

L'utilizzo della dichiarazione `SELECT` in questo modo restituisce tutte le canzoni associate a questo particolare `Artist`.

Per esempi di codice che utilizzano `Select` e `ExecuteStatement`, consulta Istruzioni SELECT PartiQL per DynamoDB.

Differenze nella scansione di una tabella

In SQL, un'istruzione `SELECT` senza una clausola `WHERE` restituirà ogni riga di una tabella. In Amazon DynamoDB, l'operazione `Scan` fa la stessa cosa. In entrambi i casi, puoi recuperare tutti gli elementi o solo alcuni di essi.

Se stai utilizzando un database SQL o NoSQL, le scansioni devono essere utilizzate con parsimonia perché possono consumare grandi quantità di risorse di sistema. A volte una scansione è appropriata (come la scansione di una piccola tabella) o inevitabile (come ad esempio un'esportazione di massa di dati). Tuttavia, come regola generale, devi progettare le applicazioni in modo da evitare di eseguire le scansioni. Per ulteriori informazioni, consulta Esecuzione di query in DynamoDB.

> **Note**
>
> L'esecuzione di un'esportazione in blocco crea anche almeno 1 file per partizione. Tutti gli elementi di ogni file provengono dal keyspace con hash di quella particolare partizione.

Argomenti

- Scansione di una tabella con SQL
- Scansione di una tabella in DynamoDB

Scansione di una tabella con SQL

Utilizzando SQL, puoi scansionare una tabella e recuperare tutti i dati usando un'istruzione `SELECT` senza specificare una clausola `WHERE`. Puoi richiedere una o più colonne nel risultato. In alternativa, puoi richiedere tutte le colonne se usi il carattere jolly (*).

Di seguito vengono illustrati alcuni esempi di utilizzo dell'istruzione `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;
```

Scansione di una tabella in DynamoDB

In Amazon DynamoDB, puoi utilizzare l'API DynamoDB, o PartiQL, un linguaggio di query compatibile con SQL, per eseguire una scansione su una tabella.

DynamoDB API

Con l'API di DynamoDB, puoi utilizzare l'operazione `Scan` per restituire una o più voci e i relativi attributi accedendo a ogni voce in una tabella o un indice secondario.

```
// Return all of the data in the table
{
    TableName:  "Music"
}
```

```
// Return all of the values for Artist and Title
{
    TableName:  "Music",
    ProjectionExpression: "Artist, Title"
}
```

L'operazione `Scan` fornisce anche il parametro `FilterExpression` che puoi utilizzare per scartare gli elementi che non vuoi visualizzare nei risultati. Un `FilterExpression` viene applicato dopo la scansione dell'intera tabella, ma prima che i risultati ti vengano restituiti. ti viene comunque addebitato l'intero `Scan`, anche se vengono restituiti solo pochi item corrispondenti.

PartiQL for DynamoDB

Con PartiQL, esegui una scansione utilizzando l'operazione `ExecuteStatement` per restituire tutti i contenuti di una tabella utilizzando la dichiarazione `Select`.

```
SELECT AlbumTitle, Year, Price
FROM Music
```

Si noti che questa istruzione restituirà tutti gli elementi per nella tabella Music.

Per esempi di codice che utilizzano `Select` e `ExecuteStatement`, consulta Istruzioni SELECT PartiQL per DynamoDB.

Differenze tra un database relazionale (SQL) e DynamoDB nella gestione degli indici

Gli indici consentono di accedere a modelli di query alternativi e possono velocizzare le query. In questa sezione viene eseguito il confronto delle operazioni di creazione e utilizzo dell'indice in SQL e in Amazon DynamoDB.

Che si stia utilizzando un database relazionale o DynamoDB, è necessario eseguire con attenzione la creazione dell'indice. Ogni volta che una scrittura si verifica su una tabella, tutti gli indici della

tabella devono essere aggiornati. In un ambiente di scrittura con tabelle di grandi dimensioni, questa operazione può consumare grandi quantità di risorse di sistema. In un ambiente di sola lettura o pressoché di sola lettura, questa operazione non costituisce un problema, ma devi assicurarti che gli indici vengano effettivamente utilizzati dall'applicazione e non occupino semplicemente spazio.

Argomenti

- Differenze tra un database relazionale (SQL) e DynamoDB durante la creazione di una tabella
- Differenze tra un database relazionale (SQL) e DynamoDB durante l'esecuzione di query e scansione di una tabella

Differenze tra un database relazionale (SQL) e DynamoDB durante la creazione di una tabella

Confrontare l'istruzione `CREATE INDEX` in SQL con l'operazione `UpdateTable` in Amazon DynamoDB.

Argomenti

- Creazione di un indice con SQL
- Creazione di un indice in DynamoDB

Creazione di un indice con SQL

In un database relazionale, un indice è una struttura di dati che ti consente di eseguire query veloci su colonne diverse in una tabella. Puoi utilizzare l'istruzione SQL `CREATE INDEX` per aggiungere un indice a una tabella esistente, specificando le colonne da indicizzare. Dopo aver creato l'indice, puoi eseguire una query sui dati della tabella come di consueto, ma ora il database può utilizzare l'indice per trovare rapidamente le righe specificate nella tabella anziché eseguire la scansione dell'intera tabella.

Dopo aver creato un indice, il database lo gestisce automaticamente. Ogni volta che modifichi i dati nella tabella, l'indice viene modificato automaticamente per riflettere le modifiche nella tabella.

In MySQL, puoi creare un indice simile al seguente.

```
CREATE INDEX GenreAndPriceIndex
ON Music (genre, price);
```

Creazione di un indice in DynamoDB

In , puoi creare e usare un per scopi simili.

Gli indici in DynamoDB sono diversi dalle rispettive controparti relazionali. Quando crei un indice secondario globale, specifica una chiave di partizione e opzionalmente una chiave di ordinamento. Dopo aver creato l'indice secondario, è possibile`Query`o`Scan`proprio come faresti con un tavolo. DynamoDB non dispone di un ottimizzatore di query, quindi un indice secondario viene utilizzato solo quando`Query`o`Scan`esso.

DynamoDB supporta due diversi tipi di indici:

- Indici secondari globali: la chiave primaria dell'indice può essere qualsiasi due attributi della relativa tabella.
- Indici secondari locali: la chiave di partizione dell'indice deve essere uguale a quella della chiave di partizione della tabella. Tuttavia, la chiave di ordinamento può essere un qualsiasi altro attributo.

assicura che i dati in un sono consistenti con la tabella. Puoi richiedere operazioni `Query` o `Scan` ad elevata consistenza su una tabella o un indice secondario locale. Tuttavia, gli indici secondari globali supportano solo la consistenza finale.

È possibile aggiungere un indice secondario globale a una tabella esistente utilizzando l'operazione `UpdateTable` e specificando `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
                }
            }
        }
    ]
}
```

Dovrai fornire i seguenti parametri a `UpdateTable`:

- `TableName`— La tabella a cui verrà associato l'indice.
- `AttributeDefinitions`— I tipi di dati per gli attributi dello schema chiave dell'indice.
- `GlobalSecondaryIndexUpdates`— Dettagli sull'indice che desideri creare:
  - `IndexName`: un nome per l'indice.
  - `KeySchema`— Gli attributi utilizzati per la chiave primaria dell'indice.
  - `Projection` - Specifica gli attributi che vengono copiati (proiettati) dalla tabella nell'indice. In questo caso, `ALL` significa che tutti gli attributi sono copiati.
  - `ProvisionedThroughput (for provisioned tables)`— Il numero di letture e scritture al secondo necessarie per questo indice. Questo valore è separato dalle impostazioni di throughput assegnato dalla tabella.

Parte di questa operazione comporta la compilazione dei dati dalla tabella nel nuovo indice. Durante la compilazione la tabella rimane disponibile. Tuttavia, l'indice non è pronto fino a quando l'attributo `Backfilling` cambia da true a false. Puoi utilizzare l'operazione `DescribeTable` per visualizzare questo attributo.

Differenze tra un database relazionale (SQL) e DynamoDB durante l’esecuzione di query e scansione di una tabella

Confrontare l'esecuzione di query e scansione di un indice usando l'istruzione SELECT (SELEZIONA) in SQL con le operazioni `Query` e `Scan` in Amazon DynamoDB.

Argomenti

- Esecuzione di query e scansione di un indice con SQL
- Esecuzione di query e scansione di un indice in DynamoDB

Esecuzione di query e scansione di un indice con SQL

In un database relazionale, non si lavora direttamente con gli indici. Si eseguono invece query sulle tabelle emettendo istruzioni `SELECT` in modo che l'ottimizzatore di query possa fare uso di qualsiasi indice.

Un ottimizzatore di query è un componente dei sistemi di gestione di database relazionali (RDBMS) che valuta gli indici disponibili e determina se possono essere utilizzati per accelerare una query. Se gli indici possono essere utilizzati per accelerare una query, RDBMS accede prima all'indice e quindi lo utilizza per individuare i dati nella tabella.

Ecco alcune istruzioni SQL che possono essere utilizzate GenreAndPriceIndexper migliorare le prestazioni. Supponiamo che la tabella Music contenga i dati sufficienti che l'ottimizzatore di query decida di usare questo indice, piuttosto che scansionare l'intera tabella.

```
/* 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;
```

Esecuzione di query e scansione di un indice in DynamoDB

In DynamoDB, le operazioni `Query` e `Scan` vengono eseguite direttamente sull'indice, nello stesso modo in cui vengono eseguite su una tabella. Puoi utilizzare l'API DynamoDB o PartiQL, un linguaggio di query compatibile con SQL, per eseguire query o effettuare la scansione dell'indice. Devi specificare `TableName` e `IndexName`.

Di seguito sono riportate alcune domande su GenreAndPriceIndexDynamoDB. Lo schema della chiave per l'indice è composto dagli attributi Genre e Price.

DynamoDB API

```
// All of the rock songs

{
```

```
    TableName: "Music",
    IndexName: "GenreAndPriceIndex",
    KeyConditionExpression: "Genre = :genre",
    ExpressionAttributeValues: {
        ":genre": "Rock"
    },
};
```

Questo esempio utilizza un `ProjectionExpression` per indicare che si vogliono solo alcuni degli attributi, piuttosto che tutti, nei risultati.

```
// 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"
};
```

Di seguito è riportata una scansione in corso. GenreAndPriceIndex

```
// Return all of the data in the index

{
    TableName:  "Music",
    IndexName: "GenreAndPriceIndex"
}
```

PartiQL for DynamoDB

Con PartiQL, puoi utilizzare l'istruzione `Select` PartiQL per eseguire query e scansioni sull'indice.

```
// 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
```

Quanto segue è una scansione in corso GenreAndPriceIndex.

```
// Return all of the data in the index

SELECT *
FROM Music.GenreAndPriceIndex
```

> **Note**
>
> Per esempi di codice che utilizzano `Select`, consulta Istruzioni SELECT PartiQL per DynamoDB.

Differenze tra un database relazionale (SQL) e DynamoDB durante la modifica di dati in una tabella

Il linguaggio SQL fornisce il`UPDATE`per la modifica dei dati. Amazon DynamoDB utilizza l'`UpdateItem`per eseguire attività simili.

Argomenti

- Modifica dei dati in una tabella con SQL
- Modifica dei dati in una tabella in DynamoDB

Modifica dei dati in una tabella con SQL

In SQL, usi l'istruzione `UPDATE` per modificare una o più righe. La clausola `SET` specifica i nuovi valori per una o più colonne e la clausola `WHERE` determina quali righe vengono modificate. Di seguito è riportato un esempio di :

```
UPDATE Music
SET RecordLabel = 'Global Records'
```

```
WHERE Artist = 'No One You Know' AND SongTitle = 'Call Me Today';
```

Se nessuna riga corrisponde alla clausola `WHERE`, l'istruzione `UPDATE` non ha alcun effetto.

Modifica dei dati in una tabella in DynamoDB

In DynamoDB, puoi utilizzare l'API DynamoDB o PartiQL, un linguaggio di query compatibile con SQL, per modificare un singolo elemento. Se vuoi modificare più elementi, devi utilizzare più operazioni.

DynamoDB API

Con l'API DynamoDB, l'operazione `UpdateItem` viene utilizzata per modificare un singolo elemento.

```
{
    TableName: "Music",
    Key: {
        "Artist":"No One You Know",
        "SongTitle":"Call Me Today"
    },
    UpdateExpression: "SET RecordLabel = :label",
    ExpressionAttributeValues: {
        ":label": "Global Records"
    }
}
```

È necessario specificare gli attributi `Key` dell'elemento da modificare e `UpdateExpression` per specificare i valori degli attributi. `UpdateItem` si comporta come un'operazione "upsert". L'elemento viene aggiornato se è presente nella tabella, ma in caso contrario viene aggiunto (inserito) un nuovo elemento.

`UpdateItem` supporta le scritture condizionali, in cui l'operazione ha esito positivo solo se una specifica `ConditionExpression` restituisce true. Ad esempio, la seguente operazione `UpdateItem` non esegue l'aggiornamento a meno che il prezzo della canzone non sia maggiore o uguale a 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` supporta anche i contatori atomici o gli attributi di tipo `Number` che possono essere incrementati o decrementati. I contatori atomici sono simili per molti versi ai generatori di sequenza, alle colonne di identità o ai campi a incremento automatico dei database SQL.

Di seguito è riportato un esempio di un'operazione `UpdateItem` per inizializzare un nuovo attributo (Plays) per tenere traccia del numero di volte in cui una canzone viene riprodotta.

```
{
    TableName: "Music",
    Key: {
        "Artist":"No One You Know",
        "SongTitle":"Call Me Today"
    },
    UpdateExpression: "SET Plays = :val",
    ExpressionAttributeValues: {
        ":val": 0
    },
    ReturnValues: "UPDATED_NEW"
}
```

Il parametro `ReturnValues` è impostato su `UPDATED_NEW` che restituisce i nuovi valori degli attributi aggiornati. In questo caso restituisce 0 (zero).

Ogni volta che qualcuno riproduce questa canzone, possiamo usare la seguente operazione `UpdateItem` per incrementare Plays di uno.

```
{
    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, utilizzi l'operazione `ExecuteStatement` per modificare un elemento in una tabella, utilizzando la dichiarazione `Update` PartiQL.

La chiave principale di questa tabella è costituita da Artist e. SongTitle Dovrai specificare i valori per questi attributi.

```
UPDATE Music
SET RecordLabel ='Global Records'
WHERE Artist='No One You Know' AND SongTitle='Call Me Today'
```

È inoltre possibile modificare più campi contemporaneamente, come nell'esempio seguente.

```
UPDATE Music
SET RecordLabel = 'Global Records'
SET AwardsWon = 10
WHERE Artist ='No One You Know' AND SongTitle='Call Me Today'
```

`Update` supporta anche i contatori atomici o gli attributi di tipo `Number` che possono essere incrementati o decrementati. I contatori atomici sono simili per molti versi ai generatori di sequenza, alle colonne di identità o ai campi a incremento automatico dei database SQL.

Di seguito è riportato un esempio di una dichiarazione `Update` per inizializzare un nuovo attributo (Plays) per tenere traccia del numero di volte in cui una canzone viene riprodotta.

```
UPDATE Music
SET Plays = 0
WHERE Artist='No One You Know' AND SongTitle='Call Me Today'
```

Ogni volta che qualcuno riproduce questa canzone, possiamo usare la seguente dichiarazione `Update` per incrementare Plays di uno.

```
UPDATE Music
```

```
SET Plays = Plays + 1
WHERE Artist='No One You Know' AND SongTitle='Call Me Today'
```

> **Note**
>
> Per esempi di codice che utilizzano `Update` e `ExecuteStatement`, consulta Istruzioni UPDATE PartiQL per DynamoDB.

Differenze tra un database relazionale (SQL) e DynamoDB durante l'eliminazione di dati da una tabella

Con SQL, userai l'istruzione `DELETE` per recuperare una o più righe da una tabella. Amazon DynamoDB utilizza l'operazione `DeleteItem` per eliminare un elemento alla volta.

Argomenti

- Eliminazione dei dati da una tabella con SQL
- Eliminazione dei dati da una tabella in DynamoDB

Eliminazione dei dati da una tabella con SQL

In SQL, usi l'istruzione `DELETE` per eliminare una o più righe. La clausola `WHERE` determina le righe che vuoi modificare. Di seguito è riportato un esempio di :

```
DELETE FROM Music
WHERE Artist = 'The Acme Band' AND SongTitle = 'Look Out, World';
```

Puoi modificare la clausola `WHERE` per eliminare più righe. Ad esempio, potresti eliminare tutti i brani di un artista particolare, come mostrato nell'esempio seguente.

```
DELETE FROM Music WHERE Artist = 'The Acme Band'
```

Eliminazione dei dati da una tabella in DynamoDB

In DynamoDB, puoi utilizzare l'API DynamoDB o PartiQL, un linguaggio di query compatibile con SQL, per eliminare un singolo elemento. Se vuoi modificare più elementi, devi utilizzare più operazioni.

DynamoDB API

Con l'API DynamoDB, utilizzi l'operazione `DeleteItem` per eliminare i dati da una tabella, un elemento alla volta. Devi specificare i valori delle chiavi primarie dell'item.

```
{
    TableName: "Music",
    Key: {
        Artist: "The Acme Band",
        SongTitle: "Look Out, World"
    }
}
```

> **Note**
>
> Oltre a `DeleteItem`, Amazon DynamoDB supporta un'operazione `BatchWriteItem` per l'eliminazione di più elementi contemporaneamente.

`DeleteItem` supporta le scritture condizionali, in cui l'operazione ha esito positivo solo se una specifica `ConditionExpression` restituisce true. Ad esempio, la seguente `DeleteItem` operazione elimina l'elemento solo se ha un attributo. RecordLabel

```
{
    TableName: "Music",
    Key: {
        Artist: "The Acme Band",
        SongTitle: "Look Out, World"
    },
   ConditionExpression: "attribute_exists(RecordLabel)"
}
```

PartiQL for DynamoDB

Con PartiQL, puoi utilizzare la dichiarazione `Delete` attraverso l'operazione `ExecuteStatement` per eliminare i dati da una tabella, un elemento alla volta. Devi specificare i valori delle chiavi primarie dell'item.

La chiave principale di questa tabella è costituita da Artist e SongTitle. Dovrai specificare i valori per questi attributi.

```
DELETE FROM Music
WHERE Artist = 'Acme Band' AND SongTitle = 'PartiQL Rocks'
```

Puoi anche specificare ulteriori condizioni per l'operazione. La seguente operazione DELETE elimina l'elemento solo se ha più di 11 Awards (Premi).

```
DELETE FROM Music
WHERE Artist = 'Acme Band' AND SongTitle = 'PartiQL Rocks' AND Awards > 11
```

> **Note**
>
> Per esempi di codice che utilizzano DELETE e ExecuteStatement, consulta Istruzioni DELETE PartiQL per DynamoDB.

Differenze tra un database relazionale (SQL) e DynamoDB durante la rimozione di una tabella

In SQL, userai l'istruzione DROP TABLE per rimuovere una tabella. In Amazon DynamoDB, si utilizza ilDeleteTableoperazione.

Argomenti

- Rimozione di una tabella con SQL
- Rimozione di una tabella in DynamoDB

Rimozione di una tabella con SQL

Quando non hai più bisogno di una tabella e vuoi eliminarla definitivamente, usa l'istruzione DROP TABLE in SQL.

```
DROP TABLE Music;
```

Dopo che una tabella è stata eliminata, non può più essere ripristinata. Alcuni database relazionali consentono di annullare un'operazione DROP TABLE, ma questa è una funzionalità specifica del fornitore e non è sempre implementata.

Rimozione di una tabella in DynamoDB

In DynamoDB, `DeleteTable` è un'operazione simile. Nell'esempio seguente, la tabella viene eliminata in modo permanente.

```
{
    TableName: "Music"
}
```

Risorse e strumenti di apprendimento di Amazon DynamoDB

Puoi utilizzare le seguenti risorse aggiuntive per comprendere e utilizzare DynamoDB.

Argomenti

- Strumenti per la codifica e la visualizzazione
- Articoli di linee guida prescrittive
- Articoli del Knowledge Center
- Post di blog, repository e guide
- Presentazioni di modellazione dei dati e di modelli di progettazione
- Corsi di formazione

Strumenti per la codifica e la visualizzazione

Puoi utilizzare i seguenti strumenti di codifica e visualizzazione per utilizzare DynamoDB:

- NoSQL Workbench per Amazon DynamoDB: uno strumento visivo unificato che consente di progettare, creare, eseguire query e gestire tabelle DynamoDB. Fornisce funzionalità di modellazione e visualizzazione dei dati e sviluppo di query.
- Dynobase: uno strumento desktop che facilita la visualizzazione e le operazioni con tabelle DynamoDB, la creazione di codice delle applicazioni e la modifica dei record con la convalida in tempo reale.
- DynamoDB Toolbox — Un progetto di Jeremy Daly che fornisce utili utilità per lavorare con la modellazione dei dati e Node.js. JavaScript
- DynamoDB Streams Processor: uno strumento semplice che facilita notevolmente le operazioni con i flussi DynamoDB.

Articoli di linee guida prescrittive

AWS Prescriptive Guidance fornisce strategie, guide e modelli collaudati per accelerare i tuoi progetti. Queste risorse sono state sviluppate da esperti di AWS tecnologia e dalla comunità globale di AWS partner, sulla base di anni di esperienza nell'aiutare i clienti a raggiungere i propri obiettivi aziendali.

Modellazione e migrazione dei dati

- Un modello di dati gerarchico in DynamoDB
- Modellazione dei dati con DynamoDB
- Migrare un database Oracle su DynamoDB utilizzando AWS DMS

Tabelle globali

- Utilizzo delle tabelle globali di Amazon DynamoDB

Serverless

- Implementa il modello di saga serverless con AWS Step Functions

Architettura SaaS

- Manage tenants across multiple SaaS products on a single control plane (Gestione dei tenant su più prodotti SaaS su un unico piano di controllo (control-plane))
- Tenant onboarding in SaaS architecture for the silo model using C# and AWS CDK (Integrazione dei tenant nell'architettura SaaS per il modello a silo con C# e AWS CDK)

Protezione e spostamento dei dati

- Configurazione dell'accesso multi-account in Amazon DynamoDB
- Opzioni di copia completa della tabella per DynamoDB
- Strategia di ripristino di emergenza per database su AWS

Miscellaneous (Varie)

- Aiuto per l'applicazione di tag in DynamoDB

Procedure guidate video delle linee guida prescrittive

- Utilizzo dell'architettura serverless per creare pipeline di dati
- Novartis - Buying Engine: AI-powered Procurement Portal
- Veritiv: abilita Insights per prevedere la domanda di vendita sui data lake AWS
- mimik: Utilizzo dell'Hybrid Edge Cloud per AWS supportare Edge Microservice Mesh
- Change Data Capture with Amazon DynamoDB

Per ulteriori articoli e video sulle linee guida prescrittive per DynamoDB, consulta le Linee guida prescrittive.

Articoli del Knowledge Center

Gli articoli e i video del AWS Knowledge Center riguardano le domande e le richieste più frequenti che riceviamo dai AWS clienti. Di seguito sono riportati alcuni articoli attuali del Knowledge Center su attività specifiche relative a DynamoDB:

Ottimizzazione dei costi

- Come posso ottimizzare i costi con Amazon DynamoDB?

Limitazione della larghezza di banda della rete e latenza

- Come faccio a risolvere i problemi di latenza elevata su una tabella Amazon DynamoDB?
- Perché la tabella DynamoDB viene limitata nella larghezza di banda della rete?
- Perché la tabella DynamoDB on demand viene limitata nella larghezza di banda della rete?

Paginazione

- Come posso implementare la paginazione in DynamoDB

Transazioni

- Perché la mia chiamata API `TransactWriteItems` non riesce in DynamoDB

Risoluzione dei problemi

- Come posso risolvere i problemi relativi al dimensionamento automatico di DynamoDB?
- Come posso risolvere gli errori HTTP 4XX in DynamoDB

Per ulteriori articoli e video su DynamoDB, consulta gli articoli del Knowledge Center.

Post di blog, repository e guide

Oltre alla Guida per gli sviluppatori di DynamoDB, sono disponibili molte risorse utili per utilizzare DynamoDB. Ecco alcuni post di blog, repository e guide selezionati per utilizzare DynamoDB:

- AWS archivio di esempi di codice DynamoDB in AWS vari linguaggi SDK: Node.js, Java, Python, .Net, Go e Rust.
- Il libro DynamoDB: una guida completa di DeBrie Alex che insegna un approccio strategico alla modellazione dei dati con DynamoDB.
- Guida DynamoDB: una guida aperta di DeBrie Alex che illustra i concetti di base e le funzionalità avanzate del database DynamoDB NoSQL.
- How to switch from RDBMS to DynamoDB in 20 easy steps (Come passare da RDBMS a DynamoDB in 20 semplici fasi): un elenco di fasi utili per l'apprendimento della modellazione dei dati di Jeremy Daly.
- Cheat sheet di JavaScript DocumentClient DynamoDB: un cheat sheet per aiutarti a iniziare a creare applicazioni con DynamoDB in un Node.js o in un ambiente. JavaScript
- DynamoDB Core Concept Videos (Video relativi ai concetti fondamentali di DynamoDB): questa playlist illustra molti dei concetti fondamentali di DynamoDB.

Presentazioni di modellazione dei dati e di modelli di progettazione

Puoi utilizzare le seguenti risorse sulla modellazione dei dati e sui modelli di progettazione per ottenere il massimo da DynamoDB:

- AWS re:Invent 2019: modellazione dei dati con DynamoDB
  - Una conferenza di Alex DeBrie che ti aiuta a iniziare con i principi della modellazione dei dati DynamoDB.
- AWS re:Invent 2020: Modellazione dei dati con DynamoDB — Parte 1
- AWS re:Invent 2020: Modellazione dei dati con DynamoDB — Parte 2
- AWS re:Invent 2017: modelli di progettazione avanzati

- AWS re:Invent 2018: modelli di progettazione avanzati
- AWS re:Invent 2019: modelli di progettazione avanzati
  - Jeremy Daly condivide i suoi 12 consigli chiave di questa sessione.
- AWS re:Invent 2020: DynamoDB advanced design patterns – Part 1 (Modelli di progettazione avanzata DynamoDB Parte 1)
- AWS re:Invent 2020: modelli di progettazione avanzati per DynamoDB — Parte 2
- Orari d'ufficio di DynamoDB su Twitch

> **Note**
>
> Ogni sessione copre diversi casi d'uso ed esempi.

Corsi di formazione

Esistono diversi corsi di formazione e opzioni didattiche per ulteriori informazioni su DynamoDB. Ecco alcuni esempi:

- Sviluppo con Amazon DynamoDB: progettato per portarti AWS dal principiante all'esperto nello sviluppo di applicazioni reali con modellazione dei dati per Amazon DynamoDB.
- Corso DynamoDB Deep-Dive: un corso di Pluralsight.
- Amazon DynamoDB: creazione di applicazioni basate su database NoSQL — Un corso del team di formazione e certificazione ospitato su edX. AWS

Letture e scritture di DynamoDB

Per letture e scritture di DynamoDB si intendono le operazioni che recuperano dati da una tabella (letture) e inseriscono, aggiornano o eliminano dati in una tabella (scritture). Per lavorare con DynamoDB è essenziale comprendere i concetti di letture e scritture, poiché influiscono direttamente sulle prestazioni e sui costi dell'applicazione.

Questo argomento illustra i dettagli sui diversi tipi di coerenza di lettura che si applicano a DynamoDB. Questo argomento descrive anche l'utilizzo di unità per le diverse operazioni di lettura e scrittura che è possibile eseguire.

Argomenti

- Coerenza di lettura di DynamoDB
- Operazioni di lettura e scrittura di DynamoDB

Coerenza di lettura di DynamoDB

Amazon DynamoDB legge dati da tabelle, indici secondari locali (LSI), indici secondari globali (GSI) e flussi. Per ulteriori informazioni, consulta Componenti principali di Amazon DynamoDB. Sia le tabelle che gli indici LSI forniscono due opzioni di coerenza di lettura: Alla fine coerente (impostazione predefinita) ed Elevata consistenza di lettura. Tutte le letture dagli indici GSI e dai flussi sono di tipo "Alla fine coerente".

Quando l'applicazione scrive i dati in una tabella DynamoDB e riceve una risposta HTTP 200 (OK), significa che la scrittura è stata completata correttamente ed è stata resa persistente. DynamoDB fornisce l'isolamento read-committed, che garantisce che le operazioni di lettura restituiscano sempre valori sottoposti a commit per un elemento. La lettura non offrirà mai una visione dell'elemento da una scrittura che alla fine non ha avuto successo. L'isolamento read-committed non impedisce le modifiche dell'item immediatamente dopo l'operazione di lettura.

Letture a coerenza finale

"Alla fine coerente" rappresenta il modello di coerenza di lettura predefinito per tutte le operazioni di lettura. Quando si eseguono letture a coerenza finale su una tabelle DynamoDB o un indice, le risposte potrebbero non riflettere i risultati di un'operazione di scrittura completata di recente. Se si ripete la richiesta di lettura dopo breve tempo, la risposta deve restituire alla fine i dati più recenti.

Le letture a coerenza finale sono supportate in tabelle, indici secondari locali e indici secondari globali. Si consideri inoltre che anche tutte le letture da un flusso DynamoDB alla fine sono di tipo Alla fine coerente.

Alla fine, le letture a coerenza finale costano la metà delle letture a elevata consistenza. Per ulteriori informazioni, consulta Prezzi di Amazon DynamoDB.

Operazioni a elevata consistenza di lettura

Le operazioni di lettura, come `GetItem`, `Query` e `Scan`, forniscono un parametro `ConsistentRead` facoltativo. Se si imposta `ConsistentRead` su true, DynamoDB restituisce una risposta con i dati più aggiornati che riflette gli aggiornamenti da tutte le operazioni di scrittura precedenti. Le letture a elevata consistenza sono supportate solo nelle tabelle e negli indici secondari locali. Non sono supportate letture a elevata consistenza da un indice secondario globale o da un flusso DynamoDB.

Consistenza di lettura per le tabelle globali

DynamoDB supporta anche tabelle globali per la replica multiattiva e multiregionale. Una tabella globale è composta da più tabelle di replica in diverse regioni AWS. Tutte le modifiche apportate a qualsiasi elemento in una tabella di replica vengono replicate in tutte le altre repliche all'interno della stessa tabella globale, in genere nel giro di un secondo. Tali modifiche sono di tipo "Alla fine coerente". Per ulteriori informazioni, consulta Modalità di coerenza.

Operazioni di lettura e scrittura di DynamoDB

Le operazioni di lettura di DynamoDB consentono di recuperare uno o più elementi da una tabella specificando il valore della chiave di partizione e, facoltativamente, il valore della chiave di ordinamento. Utilizzando le operazioni di scrittura di DynamoDB è possibile inserire, aggiornare o eliminare elementi in una tabella. Questo argomento illustra l'utilizzo di unità di capacità per queste due operazioni.

Argomenti

- Utilizzo di unità di capacità per le operazioni di lettura
- Utilizzo di unità di capacità per le operazioni di scrittura

Utilizzo di unità di capacità per le operazioni di lettura

Le richieste di lettura di DynamoDB possono essere fortemente coerenti, eventualmente coerenti o transazionali.

- Una richiesta di operazione a elevata consistenza di lettura di un elemento fino a 4 KB richiede un'unità di lettura.
- Una richiesta di lettura a coerenza finale di un elemento fino a 4 KB richiede mezza unità di lettura.
- Una richiesta di lettura transazionale di un elemento fino a 4 KB richiede due unità di lettura.

Per ulteriori informazioni sui modelli di consistenza di lettura di DynamoDB, consulta Coerenza di lettura di DynamoDB.

Le dimensioni degli elementi per le letture vengono arrotondate per eccesso al multiplo di 4 KB successivo. Ad esempio, la lettura di un elemento di 3.500 byte utilizza la stessa velocità effettiva della lettura di un elemento di 4 KB.

Se è necessario leggere un elemento che è più grande di 4 KB, DynamoDB necessita di unità di lettura aggiuntive. Il numero totale di unità di lettura richieste dipende dalla dimensione dell'elemento e dal fatto che si desideri o una lettura a coerenza finale o un'operazione a elevata consistenza di lettura. Ad esempio, se la dimensione dell'elemento è 8 KB, sono necessarie 2 unità di lettura per garantire un'operazione a elevata consistenza di lettura. Sarà necessaria 1 unità di lettura in caso di scelta di letture a coerenza finale o 4 unità di lettura per una richiesta di lettura transazionale.

Nell'elenco seguente viene descritto in che modo le operazioni di lettura di DynamoDB utilizzano le unità di lettura:

- GetItem: legge un singolo elemento da una tabella. Per determinare il numero di unità di lettura che saranno utilizzate da `GetItem`, arrotonda la dimensione dell'elemento per eccesso al successivo limite di 4 KB. Se è stata specificata un'operazione a elevata consistenza di lettura, questo è il numero di unità di lettura necessario. Per una lettura a coerenza finale, che è l'impostazione predefinita, dividi questo numero per due.

  Ad esempio, se si legge un elemento di 3,5 KB, DynamoDB ne arrotonda la dimensione a 4 KB. Se si legge un elemento di 10 KB, DynamoDB ne arrotonda la dimensione a 12 KB.
- BatchGetItem: legge fino a 100 elementi da una o più tabelle. DynamoDB elabora ogni elemento del batch come singola richiesta `GetItem`. Per prima cosa DynamoDB arrotonda per eccesso la dimensione di ciascun elemento al successivo limite di 4 KB, quindi calcola la dimensione totale. Il

risultato non coincide necessariamente con la dimensione totale di tutti gli elementi. Ad esempio, se `BatchGetItem` legge due elementi di 1,5 KB e 6,5 KB, DynamoDB calcola la dimensione come 12 KB (4 KB + 8 KB). DynamoDB non calcola la dimensione in 8 KB (1,5 KB + 6,5 KB).

- Query: legge più elementi che hanno lo stesso valore della chiave di partizione. Tutti gli elementi restituiti vengono considerati come un'unica operazione di lettura in cui DynamoDB calcola la dimensione totale di tutti gli elementi. DynamoDB quindi arrotonda la dimensione al limite di 4 KB successivo. Ad esempio, supponiamo che la query restituisca 10 elementi la cui dimensione combinata sia 40,8 KB. DynamoDB arrotonda le dimensioni dell'elemento per l'operazione a 44 KB. Se una query restituisce 1.500 elementi di 64 byte ciascuno, la dimensione cumulativa è 96 KB.
- Scan: legge tutti gli elementi in una tabella. DynamoDB considera la dimensione degli elementi che vengono valutati, non la dimensione degli elementi restituiti dalla scansione. Per ulteriori informazioni sulle operazioni Scan, consulta Scansione di tabelle in DynamoDB.

> **Important**
>
> Se si esegue un'operazione di lettura su un elemento che non esiste, DynamoDB continuerà a utilizzare il throughput effettivo come descritto sopra. Per le operazioni `Query`/`Scan`, verrà comunque addebitato un throughput di lettura aggiuntivo in base alla coerenza di lettura e al numero di partizioni ricercate per soddisfare la richiesta, anche se non esistono dati.

Per qualsiasi operazione che restituisce elementi puoi richiedere un sottoinsieme di attributi da recuperare. Questo però non influisce sui calcoli delle dimensioni degli elementi. Inoltre, `Query` e `Scan` possono restituire conteggi di elementi anziché valori di attributi. Il conteggio degli elementi richiede la stessa quantità di unità di lettura ed è soggetto agli stessi calcoli delle dimensioni degli elementi. Questo perché DynamoDB deve leggere ciascun elemento per incrementare il conteggio.

Utilizzo di unità di capacità per le operazioni di scrittura

Un'unità di scrittura rappresenta una scrittura per un elemento di dimensioni fino a 1 KB. Qualora sia necessario scrivere un elemento di dimensioni superiori a 1 KB, DynamoDB deve utilizzare unità di scrittura aggiuntive. Le richieste di scrittura transazionale richiedono 2 unità di scrittura per eseguire una scrittura per elementi fino a 1 KB. Il numero totale di unità di richiesta di scrittura necessarie dipendono dalla dimensione dell'item. Ad esempio, se la dimensione dell'elemento è pari a 2 KB, sono necessarie 2 unità di scrittura per sostenere una richiesta di scrittura o 4 unità di scrittura per una richiesta di scrittura transazionale.

Le dimensioni degli elementi per le scritture vengono arrotondate per eccesso al multiplo di 1 KB successivo. Ad esempio, la scrittura di un elemento di 500 byte utilizza la stessa velocità effettiva della scrittura di un elemento di 1 KB.

Nell'elenco seguente viene descritto in che modo le operazioni di scrittura di DynamoDB utilizzano le unità di scrittura:

- PutItem: scrive un singolo elemento in una tabella. Se nella tabella esiste già un elemento con la stessa chiave primaria, l'operazione sostituisce l'item. Per calcolare l'utilizzo del throughput assegnato, la dimensione presa in considerazione è quella dell'elemento più grande.
- UpdateItem: modifica un singolo elemento nella tabella. DynamoDB considera la dimensione dell'elemento come appare prima e dopo l'aggiornamento. Il throughput assegnato utilizzato riflette la dimensione maggiore tra quelle degli elementi. Anche se si aggiorna un sottoinsieme di attributi dell'elemento, `UpdateItem` consuma comunque la quantità completa del throughput allocato (la dimensione maggiore tra le dimensioni dell'elemento "prima" e "dopo").
- DeleteItem: rimuove un singolo elemento da una tabella. L'utilizzo del throughput assegnato si basa sulla dimensione dell'elemento eliminato.
- BatchWriteItem: scrive fino a 25 elementi in una o più tabelle. DynamoDB elabora ogni elemento del batch come singola richiesta `PutItem` o `DeleteItem` (gli aggiornamenti non sono supportati). Per prima cosa DynamoDB arrotonda per eccesso la dimensione di ciascun elemento al successivo limite di 1 KB, quindi calcola la dimensione totale. Il risultato non coincide necessariamente con la dimensione totale di tutti gli elementi. Ad esempio, se `BatchWriteItem` scrive due elementi di 500 byte e 3,5 KB, DynamoDB calcola la dimensione come 5 KB (1 KB + 4 KB). DynamoDB non calcola la dimensione come 4 KB (500 byte + 3,5 KB).

Per le operazioni `PutItem`, `UpdateItem` e `DeleteItem`, DynamoDB arrotonda la dimensione dell'elemento per eccesso al successivo KB. Ad esempio, se si inserisce o si elimina un elemento di 1,6 KB, DynamoDB ne arrotonda la dimensione a 2 KB.

Le operazioni `PutItem`, `UpdateItem` e `DeleteItem` consentono scritture condizionali, dove si specifica un'espressione che deve restituire true affinché l'operazione abbia esito positivo. Se l'espressione restituisce false, DynamoDB utilizza comunque le unità di capacità di scrittura della tabella. La quantità di unità di capacità di scrittura utilizzate dipende dalla dimensione dell'elemento. Questo elemento può essere un elemento esistente nella tabella o un elemento nuovo che si sta cercando di creare o aggiornare. Ad esempio, si supponga che un elemento esistente abbia una dimensione di 300 KB. Il nuovo elemento che si sta tentando di creare o aggiornare ha una

dimensione di 310 KB. Le unità di capacità di scrittura utilizzate saranno 310 KB per il nuovo elemento.

Capacità di throughput di DynamoDB

In questa sezione viene fornita una panoramica delle due modalità di throughput disponibili per le tabelle DynamoDB e alcune considerazioni sulla selezione della modalità di capacità appropriata per la propria applicazione. La modalità di throughput di una tabella determina come viene gestita la capacità della stessa. La modalità throughput determina anche il modo in cui ti vengono addebitate le operazioni di lettura e scrittura sulle tabelle. In Amazon DynamoDB, è possibile scegliere tra la modalità on demand e la modalità con provisioning per le proprie tabelle per soddisfare diversi requisiti di carico di lavoro.

Argomenti

- Modalità on demand
- Modalità provisioning
- Modalità di capacità on demand di DynamoDB
- Modalità con capacità allocata di DynamoDB
- Funzionamento del throughput di DynamoDB
- Capacità di espansione e capacità adattiva di DynamoDB
- Considerazioni sul passaggio tra modalità di capacità in DynamoDB

Modalità on demand

La modalità on demand di Amazon DynamoDB è un'opzione di throughput serverless che semplifica la gestione dei database e scala automaticamente per supportare le applicazioni con più requisiti dei clienti. DynamoDB on demand consente di creare una tabella senza preoccuparsi della pianificazione della capacità, del monitoraggio dell'utilizzo e della configurazione delle policy di dimensionamento. DynamoDB on-demand pay-per-request offre prezzi per le richieste di lettura e scrittura in modo da pagare solo per ciò che si utilizza. Per le tabelle in modalità on demand, non è necessario specificare la velocità effettiva di lettura e scrittura che si prevede l'applicazione esegua.

La modalità on demand è l'opzione di throughput predefinita e consigliata per la maggior parte dei carichi di lavoro DynamoDB. DynamoDB gestisce tutti gli aspetti della gestione e del dimensionamento del throughput per supportare carichi di lavoro che possono iniziare con dimensioni ridotte e scalare fino a milioni di richieste al secondo. È possibile leggere e scrivere sulle tabelle DynamoDB secondo necessità senza gestire la capacità di throughput sulla tabella. Per ulteriori informazioni, consulta Modalità di capacità on demand di DynamoDB.

Modalità provisioning

Nella modalità con provisioning è necessario specificare il numero di letture e scritture di dati al secondo richieste per l'applicazione. L'addebito avverrà in base alla capacità oraria di lettura e scrittura allocata, non alla quantità di quella capacità allocata effettivamente utilizzata. Ciò consente di regolare l'utilizzo di DynamoDB in modo tale da rimanere entro il tasso di richiesta definito, al fine di ottenere la prevedibilità dei costi.

È possibile scegliere di utilizzare la capacità allocata se si dispone di carichi di lavoro costanti con una crescita prevedibile e se è possibile prevedere in modo affidabile i requisiti di capacità per l'applicazione. Per ulteriori informazioni, consulta Modalità con capacità allocata di DynamoDB.

Modalità di capacità on demand di DynamoDB

Amazon DynamoDB on demand offre un'esperienza di database davvero serverless che scala automaticamente per soddisfare i carichi di lavoro più impegnativi senza dover pianificare la capacità. L'approccio on demand semplifica il processo di configurazione, elimina la gestione e il monitoraggio della capacità e offre un dimensionamento rapido e automatico. Per quanto riguarda pay-per-request i prezzi, non devi preoccuparti della capacità inattiva perché paghi solo per il throughput effettivamente utilizzato. La fatturazione viene effettuata in base alle richieste di lettura o scrittura, quindi i costi riflettono direttamente l'utilizzo effettivo.

Se si sceglie la modalità on demand, DynamoDB adatta istantaneamente i carichi di lavoro, siano essi aumentati o diminuiti, su qualsiasi livello di traffico raggiunto in precedenza. Se il livello di traffico di un carico di lavoro raggiunge un nuovo picco, DynamoDB scala automaticamente per soddisfare i requisiti di throughput aumentati. La modalità on demand è l'opzione di throughput predefinita e consigliata perché semplifica la creazione di applicazioni moderne serverless che possono iniziare in piccolo e scalare fino a milioni di richieste al secondo. Una volta aumentata orizzontalmente la tabella on demand, in futuro è possibile ripristinare istantaneamente lo stesso throughput senza limitazione (della larghezza di banda della rete). Se il traffico verso la tabella è pari a zero, con l'opzione on demand non viene addebitato alcun costo per il throughput. Per ulteriori informazioni sulle proprietà di dimensionamento della modalità on demand, consulta Throughput e proprietà di dimensionamento.

Le tabelle che utilizzano la modalità on demand offrono la stessa latenza di pochi millisecondi, lo stesso accordo sul livello di servizio (SLA) e la stessa sicurezza offerta da DynamoDB in modalità con provisioning.

> **Note**
>
> Per impostazione predefinita, DynamoDB protegge da un utilizzo indesiderato e incontrollato. Per scalare i limiti di throughput di lettura e scrittura a livello di tabella di 40.000 per tutte le tabelle dell'account, è possibile richiedere un aumento di questa quota. Le richieste di throughput che superano la quota di throughput predefinito della tabella sono sottoposte a limitazione (della larghezza di banda della rete). Per ulteriori informazioni, consulta Quote predefinite della velocità di trasmissione effettiva.

Facoltativamente, è possibile anche configurare il throughput massimo di lettura o scrittura (o entrambe) al secondo per singole tabelle on demand e singoli indici secondari globali. Configurando il throughput, è possibile mantenere limitati l'utilizzo e i costi a livello di tabella, proteggersi da un aumento involontario delle risorse utilizzate ed evitare un uso eccessivo per una gestione prevedibile dei costi. Le richieste di throughput che superano il throughput massimo della tabella sono sottoposte a limitazione (della larghezza di banda della rete). È possibile modificare il throughput massimo specifico della tabella in qualsiasi momento in base ai requisiti dell'applicazione. Per ulteriori informazioni, consulta Throughput massimo di DynamoDB per le tabelle on demand.

Per iniziare a utilizzare la modalità on demand, è possibile creare o aggiornare una tabella. Per ulteriori informazioni, consulta Operazioni di base sulle tabelle DynamoDB.

È possibile cambiare le tabelle dalla modalità con capacità allocata alla modalità on demand fino a quattro volte in una finestra variabile di 24 ore. È possibile passare le tabelle dalla modalità on demand alla modalità con capacità allocata in qualsiasi momento.

Per ulteriori informazioni sul passaggio dalla modalità di capacità di lettura a quella di scrittura, consulta Considerazioni sul passaggio tra modalità di capacità in DynamoDB. Per le quote delle tabelle on demand, consulta Throughput di lettura/scrittura.

Argomenti

- Unità di richiesta di lettura e unità di richiesta di scrittura
- Throughput e proprietà di dimensionamento
- Throughput massimo di DynamoDB per le tabelle on demand

Unità di richiesta di lettura e unità di richiesta di scrittura

DynamoDB addebita le letture e le scritture eseguite dall'applicazione sulle tabelle in termini di unità di richiesta di lettura e unità di richiesta di scrittura.

Un'unità di richiesta di lettura rappresenta un'operazione a elevata consistenza di lettura al secondo o due operazioni di lettura a coerenza finale al secondo per un elemento di dimensioni fino a 4 KB. Per ulteriori informazioni sui modelli di consistenza di lettura di DynamoDB, consulta Coerenza di lettura di DynamoDB.

Un'unità di richiesta di scrittura rappresenta un'operazione di scrittura per un elemento di dimensioni fino a 1 KB.

Per ulteriori informazioni su come vengono utilizzate le unità di lettura e scrittura, consulta Operazioni di lettura e scrittura di DynamoDB.

Throughput e proprietà di dimensionamento

Le tabelle DynamoDB che utilizzano la modalità di capacità on demand adattano automaticamente il volume di traffico dell'applicazione. Le nuove tabelle on demand saranno in grado di supportare fino a 4.000 scritture al secondo e 12.000 letture al secondo. La modalità di capacità on demand adatta automaticamente fino al doppio del precedente picco di traffico su una tabella. Ad esempio, si supponga che lo schema di traffico dell'applicazione vari tra 25.000 e 50.000 operazioni a elevata consistenza di lettura. Il picco di traffico precedente era di 50.000 letture al secondo. La modalità con capacità on demand supporta istantaneamente un traffico sostenuto fino a 100.000 letture al secondo. Se la propria applicazione registra un traffico di 100.000 letture al secondo, quel picco diventa il nuovo picco precedente. Questo picco precedente consente al traffico successivo di raggiungere fino a 200.000 letture al secondo.

Se il proprio carico di lavoro genera oltre il doppio del picco precedente sulla tabella, DynamoDB alloca automaticamente ulteriore capacità con l'aumentare del volume del traffico. Questa allocazione della capacità aiuta a garantire che il carico di lavoro non subisca una limitazione (della larghezza di banda della rete). Tuttavia, il throttling può verificarsi se si eccede del doppio il picco precedente entro 30 minuti. Ad esempio, si supponga che lo schema di traffico dell'applicazione vari tra 25.000 e 50.000 operazioni a elevata consistenza di lettura. Il picco di traffico precedentemente raggiunto era di 50.000 letture al secondo. Si raccomanda di pre-riscaldare la tabella oppure di scaglionare la crescita del traffico su un intervallo minimo di 30 minuti, prima di raggiungere un carico superiore a 100.000 letture al secondo. Per ulteriori informazioni sul pre-riscaldamento, consulta Funzionamento del throughput di DynamoDB.

DynamoDB non impone la restrizione di limitazione (della larghezza di banda della rete) di 30 minuti se il traffico di picco del carico di lavoro rimane entro il doppio del picco precedente. Se il traffico di picco supera il doppio del picco, assicurati che questa crescita si verifichi 30 minuti dopo l'ultima volta che è stato raggiunto il picco.

Throughput massimo di DynamoDB per le tabelle on demand

Per le tabelle su richiesta, puoi facoltativamente specificare la velocità massima di lettura o scrittura (o entrambe) al secondo su singole tabelle e sugli indici secondari globali associati (). GSIs La specificazione di un throughput massimo on demand aiuta a mantenere limitati l'utilizzo e i costi a livello di tabella. Per impostazione predefinita, le impostazioni di throughput massimo non vengono applicate e la velocità di throughput on demand è limitata da una quota di servizio AWS di throughput di lettura e scrittura a livello di tabella di 40.000 per tutte le tabelle dell'account. Se necessario, è possibile richiedere un aumento della quota di servizio.

Quando si configura il throughput massimo per una tabella on demand, le richieste di throughput che superano la quantità massima specificata verranno sottoposte a limitazione (della larghezza di banda della rete). È possibile modificare il throughput massimo a livello di tabella in qualsiasi momento in base ai requisiti dell'applicazione.

Di seguito sono riportati alcuni casi d'uso comuni in cui è possibile trarre vantaggio dall'utilizzo del throughput massimo per le tabelle on demand:

- Ottimizzazione dei costi di throughput: l'utilizzo del throughput massimo per le tabelle on demand offre un ulteriore livello di prevedibilità e gestibilità dei costi. Inoltre, offre una maggiore flessibilità nell'utilizzo della modalità on demand per supportare carichi di lavoro con modelli di traffico e budget diversi.
- Protezione dall'uso eccessivo: impostando il throughput massimo è possibile prevenire un aumento accidentale dell'utilizzo di lettura o scrittura, che potrebbe derivare da codice non ottimizzato o processi non autorizzati, rispetto a una tabella on demand. Questa impostazione a livello di tabella può proteggere le organizzazioni dall'utilizzo eccessivo di risorse entro un determinato periodo di tempo.
- Protezione dei servizi a valle: un'applicazione del cliente può includere tecnologie serverless e non serverless. L'architettura serverless può scalare rapidamente per soddisfare la domanda. Tuttavia, i componenti a valle con capacità fisse potrebbero risultare sovraccaricati. L'implementazione delle impostazioni di throughput massimo per le tabelle on demand può impedire la propagazione di grandi volumi di eventi a più componenti a valle con effetti collaterali imprevisti.

È possibile configurare la velocità effettiva massima per la modalità on demand per tabelle a regione singola e tabelle globali nuove ed esistenti e. GSIs È possibile anche configurare il throughput massimo durante il ripristino delle tabelle e l'importazione dei dati dai flussi di lavoro di Amazon S3.

È possibile specificare le impostazioni di throughput massimo per una tabella on demand utilizzando la console DynamoDB, la AWS CLI, AWS CloudFormation o l'API DynamoDB.

> **Note**
>
> Il throughput massimo per una tabella on demand viene applicato sulla base del principio del massimo impegno e dovrebbe essere considerato un obiettivo, anziché un massimale di richieste garantito. Il carico di lavoro potrebbe superare temporaneamente il throughput massimo specificato a causa della capacità di espansione. In alcuni casi, DynamoDB utilizza la capacità di espansione per accettare letture o scritture eccedenti rispetto alle impostazioni di throughput massimo della tabella. Con la capacità supplementare, le richieste di lettura o scrittura impreviste possono avere esito positivo anziché essere sottoposte a throttling.

Argomenti

- Considerazioni relative all'utilizzo del throughput massimo per la modalità on demand
- Richiedi limitazioni e metriche CloudWatch

Considerazioni relative all'utilizzo del throughput massimo per la modalità on demand

Quando si utilizza il throughput massimo per le tabelle in modalità on demand, si applicano le seguenti considerazioni:

- È possibile impostare in modo indipendente il throughput massimo per le letture e le scritture per qualsiasi tabella on demand o per un singolo indice secondario globale all'interno di tale tabella per ottimizzare l'approccio in base a requisiti specifici.
- Puoi utilizzare Amazon CloudWatch per monitorare e comprendere i parametri di utilizzo a livello di tabella di DynamoDB e per determinare le impostazioni di throughput massimo appropriate per la modalità on-demand. Per ulteriori informazioni, consulta Parametri e dimensioni di DynamoDB.
- Quando si specificano le impostazioni di throughput massimo di lettura o scrittura (o entrambe) su una replica di una tabella globale, le stesse impostazioni di throughput massimo vengono applicate automaticamente a tutte le tabelle di replica. Per garantire la corretta replica dei dati è necessario che le tabelle di replica e gli indici secondari in una tabella globale abbiano impostazioni

di throughput di scrittura identiche. Per ulteriori informazioni, consulta Best practice per le tabelle globali.

- Il throughput massimo di lettura o scrittura più piccolo che è possibile specificare è un'unità di richiesta al secondo.
- Il throughput massimo specificato deve essere inferiore alla quota di throughput predefinita disponibile per qualsiasi tabella on demand o singolo indice secondario globale all'interno di tale tabella.

Richiedi limitazioni e metriche CloudWatch

Se l'applicazione supera il throughput massimo di lettura o scrittura impostato nella tabella on demand, DynamoDB inizia a limitare la larghezza di banda della rete di tali richieste. Quando DynamoDB limita una lettura o una scrittura, restituisce una `ThrottlingException` al chiamante. È quindi possibile intraprendere le azioni appropriate, se necessario. Ad esempio, è possibile aumentare o disabilitare l'impostazione del throughput massimo della tabella o attendere un breve intervallo prima di ritentare la richiesta.

Per semplificare il monitoraggio, il throughput massimo configurato per una tabella o un indice secondario globale, CloudWatch fornisce le seguenti metriche: e. OnDemandMaxReadRequestUnits OnDemandMaxWriteRequestUnits

Modalità con capacità allocata di DynamoDB

Quando si crea una nuova tabella con provisioning in DynamoDB, è necessario specificare la capacità di throughput allocata, che è la quantità di throughput di lettura e scrittura che la tabella può supportare. L'addebito avverrà in base alla capacità oraria di lettura e scrittura allocata, non alla quantità di quella capacità allocata effettivamente utilizzata.

Con il cambiamento dei requisiti di accesso e dati dell'applicazione, può essere necessario modificare le impostazioni di throughput della tabella. È possibile utilizzare il dimensionamento automatico per regolare automaticamente la capacità assegnata della tabella in risposta alle modifiche del traffico. Il dimensionamento automatico di DynamoDB utilizza una policy di dimensionamento in Application Auto Scaling. Per configurare il dimensionamento automatico in DynamoDB, si impostano i livelli minimo e massimo di capacità di lettura e scrittura oltre alla percentuale di utilizzo di destinazione. Application Auto Scaling crea e gestisce gli CloudWatch allarmi che attivano gli eventi di scalabilità quando la metrica si discosta dall'obiettivo. Il dimensionamento automatico monitora l'attività della tabella e ne aumenta o diminuisce le impostazioni di capacità in base a soglie preconfigurate. La

scalabilità automatica si attiva quando la capacità consumata supera l'utilizzo previsto configurato per due minuti consecutivi. CloudWatch gli allarmi potrebbero avere un breve ritardo, fino a qualche minuto, prima di attivare il ridimensionamento automatico. Per ulteriori informazioni, consulta Gestione automatica della capacità effettiva di trasmissione con il dimensionamento automatico di DynamoDB.

Se si utilizza la scalabilità automatica di DynamoDB, le impostazioni di velocità effettiva vengono modificate automaticamente in base ai carichi di lavoro effettivi. Puoi anche utilizzare l'operazione UpdateTable per modificare la capacità di throughput della tabella manualmente. Ad esempio, questa potrebbe essere la scelta opportuna se si ha necessità di caricare in blocco i dati da un datastore esistente nella nuova tabella DynamoDB. Puoi creare la tabella con un'impostazione di throughput di scrittura elevata, per poi ridurla al termine del caricamento in blocco dei dati.

> **Note**
>
> Per impostazione predefinita, DynamoDB protegge da un utilizzo indesiderato e incontrollato. Per scalare i limiti di throughput di lettura e scrittura a livello di tabella di 40.000 per tutte le tabelle dell'account, è possibile richiedere un aumento di questa quota. Le richieste di throughput che superano la quota di throughput predefinito della tabella sono sottoposte a limitazione (della larghezza di banda della rete). Per ulteriori informazioni, consulta Quote predefinite della velocità di trasmissione effettiva.

È possibile cambiare le tabelle dalla modalità con capacità allocata alla modalità on demand fino a quattro volte in una finestra variabile di 24 ore. È possibile passare le tabelle dalla modalità on demand alla modalità con capacità allocata in qualsiasi momento.

Per ulteriori informazioni sul passaggio dalla modalità di capacità di lettura a quella di scrittura, consulta Considerazioni sul passaggio tra modalità di capacità in DynamoDB.

Argomenti

- Unità di capacità in lettura e unità di capacità in scrittura
- Scelta delle impostazioni di velocità di trasmissione effettiva iniziali
- Scalabilità automatica di DynamoDB
- Gestione automatica della capacità effettiva di trasmissione con il dimensionamento automatico di DynamoDB
- Capacità riservata di DynamoDB

Unità di capacità in lettura e unità di capacità in scrittura

Per le tabelle con modalità con provisioning, è possibile specificare i requisiti di throughput in termini di unità di capacità. Queste unità rappresentano la quantità di dati che un'applicazione deve leggere o scrivere al secondo. È possibile modificare queste impostazioni in un secondo momento, se necessario, o abilitare la scalabilità automatica di DynamoDB per la modifica automatica.

Per un elemento di dimensioni fino a 4 KB, un'unità di richiesta di lettura (RCU, Read Capacity Unit) rappresenta un'operazione a elevata consistenza di lettura al secondo o due operazioni di lettura a coerenza finale al secondo. Per ulteriori informazioni sui modelli di consistenza di lettura di DynamoDB, consulta Coerenza di lettura di DynamoDB.

Un'unità di capacità di scrittura (WCU, Write Capacity Units) rappresenta una scrittura al secondo per un elemento di dimensioni fino a 1 KB. Per ulteriori informazioni sulle differenti operazioni di lettura e scrittura, consulta Operazioni di lettura e scrittura di DynamoDB.

Scelta delle impostazioni di velocità di trasmissione effettiva iniziali

Ogni applicazione ha requisiti diversi di lettura e scrittura da e su un database. Nella determinazione delle impostazioni di throughput iniziali per una tabella DynamoDB, tieni presente quanto segue:

- Frequenze delle richieste di lettura e scrittura previste: è necessario stimare il numero di letture e scritture da eseguire al secondo.
- Dimensioni degli elementi: alcuni elementi sono abbastanza piccoli da poter essere letti o scritti utilizzando una sola unità di capacità. Elementi più grandi richiedono più unità di capacità. Stimando le dimensioni medie degli elementi nella propria tabella è possibile specificare impostazioni accurate per il throughput allocato.
- Requisiti di consistenza di lettura: le unità di capacità di lettura si basano su operazioni a elevata consistenza di lettura, che utilizzano il doppio delle risorse di database rispetto alle letture a coerenza finale. Devi determinare se l'applicazione richiede letture fortemente consistenti o se è possibile stabilire un requisito meno rigido ed eseguire invece letture consistenti finali. Le operazioni di lettura in DynamoDB sono a coerenza finale per impostazione predefinita. Se necessario, in questo caso è possibile richiedere operazioni a elevata consistenza di lettura.

Ad esempio, si supponga di voler leggere 80 elementi al secondo da una tabella. Gli elementi hanno una dimensione di 3 KB e si desiderano operazioni a elevata consistenza di lettura. In questo caso, ogni lettura richiede un'unità di capacità di lettura allocata. Per determinare questo numero, dividi la

dimensione dell'elemento dell'operazione per 4 KB. Quindi, arrotonda per eccesso al numero intero più vicino, come mostrato nell'esempio seguente:

- 3 KB / 4 KB = 0,75 o 1 unità di capacità di lettura

Pertanto, per leggere 80 elementi al secondo da una tabella, imposta il throughput di lettura allocato della tabella su 80 unità di capacità di lettura, come illustrato nell'esempio seguente:

- 1 unità di capacità di lettura per elemento × 80 letture al secondo = 80 unità di capacità di lettura

Si supponga ora di voler scrivere 100 elementi al secondo nella tabella e che ogni elemento abbia una dimensione di 512 byte. In questo caso, ogni lettura richiede un'unità di capacità di scrittura allocata. Per determinare questo numero, dividi la dimensione dell'elemento dell'operazione per 1 KB. Quindi, arrotonda per eccesso al numero intero più vicino, come mostrato nell'esempio seguente:

- 512 byte/1 KB = 0,5 o 1 unità di capacità di scrittura

Per scrivere 100 elementi al secondo nella tabella, imposta il throughput di scrittura allocato della tabella su 100 unità di capacità di scrittura:

- 1 unità di capacità di scrittura per elemento × 100 scritture al secondo = 100 unità di capacità di scrittura

Scalabilità automatica di DynamoDB

Il dimensionamento automatico di DynamoDB gestisce attivamente la capacità di throughput allocata per le tabelle e indici secondari globali. Con l'Auto Scaling definisci un intervallo (limiti superiore e inferiore) per le unità di capacità in lettura e scrittura. È inoltre possibile definire una percentuale di utilizzo di destinazione all'interno di tale intervallo. La scalabilità automatica di DynamoDB cerca di mantenere l'utilizzo di destinazione, anche se il carico di lavoro dell'applicazione aumenta o diminuisce.

Con la scalabilità automatica di DynamoDB, una tabella o un indice secondario globale può aumentare la capacità in lettura e scrittura assegnata per gestire improvvisi aumenti di traffico senza che si verifichi alcuna limitazione delle richieste. Quando il carico di lavoro diminuisce, la scalabilità automatica di DynamoDB può ridurre la velocità effettiva in modo da non pagare per la capacità assegnata inutilizzata.

> **Note**
>
> Se si utilizza il Console di gestione AWS per creare una tabella o un indice secondario globale, la scalabilità automatica di DynamoDB è abilitata per impostazione predefinita. È possibile gestire le impostazioni di ridimensionamento automatico in qualsiasi momento utilizzando la console AWS CLI, il o uno dei AWS SDKs. Per ulteriori informazioni, consulta Gestione automatica della capacità effettiva di trasmissione con il dimensionamento automatico di DynamoDB.

Tasso di utilizzo

Il tasso di utilizzo può aiutare a determinare se la capacità di provisioning è eccessiva, nel qual caso è opportuno ridurre la capacità della tabella per risparmiare sui costi. Al contrario, può anche aiutare a determinare se la capacità di provisioning è insufficiente. In questo caso, è consigliabile aumentare la capacità delle tabelle per evitare una potenziale limitazione (della larghezza di banda della rete) delle richieste durante istanze di traffico elevato impreviste. Per ulteriori informazioni, consulta Amazon DynamoDB auto scaling: Performance and cost optimization at any scale.

In caso di utilizzo del dimensionamento automatico di DynamoDB, sarà anche necessario impostare una percentuale di utilizzo di destinazione. Il dimensionamento automatico utilizzerà questa percentuale come destinazione per aumentare o diminuire la capacità. Si consiglia di impostare l'utilizzo di destinazione al 70%. Per ulteriori informazioni, consulta Gestione automatica della capacità effettiva di trasmissione con il dimensionamento automatico di DynamoDB.

Gestione automatica della capacità effettiva di trasmissione con il dimensionamento automatico di DynamoDB

Molti carichi di lavoro dei database sono per natura ciclici o sono difficili da prevedere in anticipo. Ad esempio, considera un'app di social network in cui la maggior parte degli utenti sono attivi durante le ore diurne. Il database deve essere in grado di gestire l'attività diurna e ciò non è necessario per gli stessi livelli di throughput durante la notte. Un altro esempio potrebbe essere una nuova app di gioco per dispositivi mobili che si sta improvvisamente diffondendo in modo rapido. Se il gioco si diffonde troppo, potrebbe superare le risorse di database disponibili, con un conseguente rallentamento delle prestazioni e diversi clienti insoddisfatti. Questi tipi di carichi di lavoro richiedono spesso l'intervento manuale per aumentare o diminuire le risorse di database in risposta alla variazione dei livelli di utilizzo.

La scalabilità automatica di Amazon DynamoDB utilizza il servizio Application AWS Auto Scaling per regolare dinamicamente la capacità di throughput assegnata per tuo conto, in risposta ai modelli di traffico effettivi. In tal modo una tabella o un indice secondario globale (GSI) può aumentare la capacità di lettura e scrittura allocata per gestire improvvisi aumenti di traffico senza limitazione (della larghezza di banda della rete). Quando il carico di lavoro diminuisce, Application Auto Scaling riduce la velocità effettiva in modo da non dover pagare per la capacità assegnata inutilizzata.

> **Note**
>
> Se si utilizza il Console di gestione AWS per creare una tabella o un indice secondario globale, la scalabilità automatica di DynamoDB è abilitata per impostazione predefinita. Puoi modificare le tue impostazioni di scalabilità automatica in qualsiasi momento. Per ulteriori informazioni, consulta Utilizzo della scalabilità Console di gestione AWS automatica con DynamoDB.
>
> Quando si elimina una tabella o una replica globale di una tabella, tutti gli obiettivi scalabili, le politiche di ridimensionamento o gli CloudWatch allarmi associati non vengono eliminati automaticamente con essa.

Con Application Auto Scaling, è possibile creare una policy di dimensionamento per una tabella o un indice secondario globale. La policy di dimensionamento specifica se desideri dimensionare la capacità di lettura o di scrittura (o e entrambe) e indica le impostazioni delle unità di capacità minime e massime assegnate per la tabella o l'indice.

La policy di dimensionamento contiene anche un utilizzo di destinazione: la percentuale di velocità effettiva assegnata consumata in un dato momento. Application Auto Scaling utilizza un monitoraggio degli obiettivi per regolare la velocità effettiva assegnata della tabella (o dell'indice) in alto o in basso, in risposta a carichi di lavoro effettivi in modo che l'utilizzo effettivo della capacità rimanga pari o vicino all'utilizzo di destinazione.

DynamoDB fornisce il throughput allocato utilizzato per periodi di un minuto. La scalabilità automatica si attiva quando la capacità consumata supera l'utilizzo previsto configurato per due minuti consecutivi. CloudWatch gli allarmi potrebbero avere un breve ritardo, fino a qualche minuto, prima di attivare il ridimensionamento automatico. Questo ritardo garantisce una valutazione metrica accurata CloudWatch . Se i picchi di throughput utilizzato distano più di un minuto, il dimensionamento automatico potrebbe non attivarsi. Analogamente, un evento di riduzione verticale può avvenire quando 15 punti dati consecutivi sono inferiori all'utilizzo di destinazione. In entrambi i casi, dopo i trigger di auto scaling, viene richiamata l'UpdateTableAPI. Sono quindi necessari alcuni minuti per

aggiornare la capacità allocata per la tabella o l'indice. Durante questo periodo, tutte le richieste che superano la capacità allocata precedente delle tabelle verranno sottoposte a limitazione (della larghezza di banda della rete).

> **Important**
>
> Non è possibile modificare il numero di punti dati da violare per attivare l'allarme sottostante (anche se il numero attuale potrebbe cambiare in futuro).

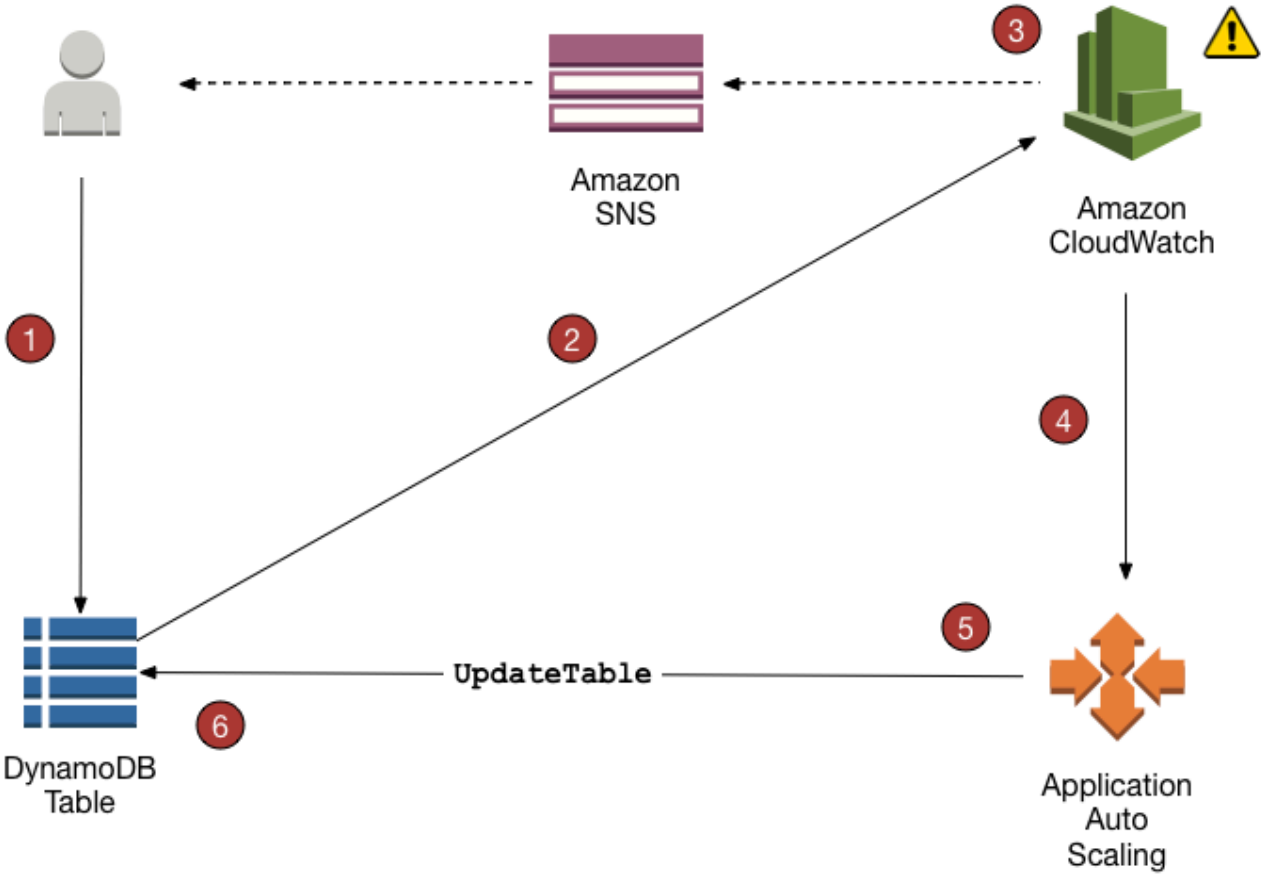
Puoi impostare i valori dell'utilizzo di destinazione del dimensionamento automatico tra il 20 e il 90 percento per la capacità in lettura e scrittura.

> **Note**
>
> Oltre alle tabelle, la scalabilità automatica di DynamoDB supporta anche gli indici secondari globali. Ogni indice secondario globale dispone della propria capacità di throughput assegnata, separata da quella della relativa tabella di base. Quando si crea una policy di dimensionamento per un indice secondario globale, Application Auto Scaling regola le impostazioni di velocità effettiva assegnata per l'indice per assicurare che il suo utilizzo effettivo rimanga uguale o prossimo alle proporzioni di utilizzo desiderate.

Funzionamento del dimensionamento automatico di DynamoDB

> **Note**
>
> Per iniziare a utilizzare rapidamente la scalabilità automatica di DynamoDB, consulta Utilizzo della scalabilità Console di gestione AWS automatica con DynamoDB.

Il seguente diagramma fornisce una panoramica dettagliata del modo in cui la scalabilità automatica di DynamoDB gestisce la capacità di throughput per una tabella:



Le fasi seguenti sintetizzano il processo di scalabilità automatica come illustrato nel diagramma precedente:

1. È possibile creare una policy di Application Auto Scaling per la tabella DynamoDB.
2. DynamoDB pubblica i parametri della capacità consumata su Amazon. CloudWatch
3. Se la capacità consumata della tabella supera l'utilizzo previsto (o scende al di sotto del target) per un periodo di tempo specifico, Amazon CloudWatch attiva un allarme. È possibile visualizzare l'allarme sulla console e ricevere notifiche tramite Amazon Simple Notification Service (Amazon SNS).
4. L' CloudWatch allarme richiama Application Auto Scaling per valutare la politica di scalabilità.
5. Application Auto Scaling invia una richiesta `UpdateTable` per regolare la velocità effettiva assegnata della tabella.
6. DynamoDB elabora la richiesta `UpdateTable`, aumentando (o diminuendo) in modo dinamico la capacità di throughput assegnata della tabella in modo che si avvicini all'utilizzo di destinazione.

Per comprendere come funziona la scalabilità automatica di DynamoDB, si supponga di avere una tabella denominata `ProductCatalog`. La tabella riceve carichi non frequenti di dati in blocco, perciò non esegue molta attività di scrittura. Tuttavia, ciò comporta un alto livello di attività di lettura che varia nel tempo. Monitorando i CloudWatch parametri di Amazon per`ProductCatalog`, stabilisci che la tabella richiede 1.200 unità di capacità di lettura (per evitare che DynamoDB limiti le richieste di lettura quando l'attività è al massimo). Inoltre, stabilisci che `ProductCatalog` richieda minimo 150 unità di capacità in lettura, quando il traffico di lettura è al livello minimo. Per ulteriori informazioni sulla limitazione (della larghezza di banda della rete), consulta Risoluzione dei problemi di limitazione (della larghezza di banda della rete) in Amazon DynamoDB.

All'interno dell'intervallo compreso tra 150 e 1.200 unità di capacità in lettura, decidi che una percentuale di utilizzo di destinazione del 70% sia adatta per la tabella `ProductCatalog`. L'utilizzo di destinazione viene espresso come proporzione tra le unità di capacità utilizzate e le unità di capacità assegnata, in percentuale. Application Auto Scaling utilizza il suo algoritmo di monitoraggio degli obiettivi per garantire che la capacità di lettura assegnata di `ProductCatalog` sia regolata come richiesto in modo che l'utilizzo rimanga più o meno al 70%.

> **Note**
>
> La scalabilità automatica di DynamoDB modifica le impostazioni della velocità di trasmissione effettiva assegnata solo quando il carico di lavoro effettivo rimane elevato (o basso) per un periodo continuativo di diversi minuti. L'algoritmo di monitoraggio degli obiettivi di Application Auto Scaling cerca di tenere l'utilizzo della destinazione pari o vicino al valore scelto a lungo termine.
>
> I picchi di attività improvvisi e di breve durata sono soddisfatti dalla capacità di ottimizzazione integrata della tabella. Per ulteriori informazioni, consulta Capacità di ottimizzazione.

Per abilitare la scalabilità automatica di DynamoDB per la tabella `ProductCatalog`, viene creata una policy di dimensionamento. La policy specifica quanto segue:

- La tabella o l'indice secondario globale che desideri gestire
- Il tipo di capacità da gestire (capacità di lettura o capacità di scrittura)
- I limiti superiore e inferiore per le impostazioni del throughput assegnato
- Utilizzo di destinazione

Quando crei una politica di scalabilità, Application Auto Scaling crea un paio di allarmi CloudWatch Amazon per tuo conto. Ogni coppia rappresenta i limiti superiore e inferiore per le impostazioni della velocità di trasmissione effettiva assegnata. Questi CloudWatch allarmi vengono attivati quando l'utilizzo effettivo della tabella si discosta dall'utilizzo previsto per un periodo di tempo prolungato.

Quando viene attivato uno degli CloudWatch allarmi, Amazon SNS ti invia una notifica (se l'hai abilitata). L' CloudWatch allarme richiama quindi Application Auto Scaling, che a sua volta notifica a DynamoDB di modificare la capacità fornita dalla tabella verso l'alto o verso il basso, a seconda `ProductCatalog` dei casi.

Durante un evento di scalabilità, viene addebitato per elemento di configurazione registrato. AWS Config Quando si verifica un evento di ridimensionamento, vengono creati quattro CloudWatch allarmi per ogni evento di auto-scaling di lettura e scrittura: alarms: e ProvisionedCapacity alarms:,. ProvisionedCapacityLow ProvisionedCapacityHigh ConsumedCapacity AlarmHigh AlarmLow Ciò dà luogo a un totale di otto allarmi. Pertanto, AWS Config registra otto elementi di configurazione per ogni evento di scalabilità.

> **Note**
>
> È possibile anche pianificare il dimensionamento di DynamoDB in modo che avvenga in determinati momenti. La procedura di base è disponibile qui.

Note per l'utilizzo

Prima di iniziare a utilizzare la scalabilità automatica di DynamoDB è necessario tenere presente quanto riportato di seguito:

- La scalabilità automatica di DynamoDB può aumentare la capacità di lettura o di scrittura in base alle necessità e secondo la policy di scalabilità automatica. Tutti le quote di DynamoDB restano in vigore, come descritto in Quote in Amazon DynamoDB.
- La scalabilità automatica di DynamoDB non impedisce di modificare manualmente le impostazioni di velocità effettiva assegnata. Queste regolazioni manuali non influiscono sugli CloudWatch allarmi esistenti correlati alla scalabilità automatica di DynamoDB.
- Se si abilita la scalabilità automatica di DynamoDB per una tabella che ha uno o più indici secondari globali, consigliamo di applicare la scalabilità automatica in maniera uniforme anche a quegli indici. Ciò contribuirà a garantire prestazioni migliori per la scrittura e la lettura delle tabelle e a evitare la limitazione della larghezza di banda della rete. È possibile abilitare la scalabilità

automatica selezionando Apply same settings to global secondary indexes (Applica le stesse impostazioni agli indici secondari globali) nella Console di gestione AWS. Per ulteriori informazioni, consulta Abilitazione del dimensionamento automatico di DynamoDB su tabelle esistenti.

- Quando si elimina una tabella o una replica globale di una tabella, tutti gli obiettivi scalabili, le policy di ridimensionamento o gli allarmi associati non vengono eliminati automaticamente con essa. CloudWatch
- Quando si crea un GSI per una tabella esistente, il dimensionamento automatico non è abilitato per il GSI. Dovrai gestire manualmente la capacità mentre il GSI è in fase di costruzione. Una volta completato il backfill sul GSI e raggiunto lo stato attivo, la scalabilità automatica funzionerà normalmente.

Utilizzo della scalabilità Console di gestione AWS automatica con DynamoDB

Quando usi il Console di gestione AWS per creare una nuova tabella, la scalabilità automatica di Amazon DynamoDB è abilitata per quella tabella per impostazione predefinita. È possibile utilizzare la console anche per abilitare la scalabilità automatica per le tabelle esistenti, modificare le impostazioni di scalabilità automatica o disabilitare la scalabilità automatica.

> **Note**
>
> Per funzionalità più avanzate come l'impostazione dei tempi di cooldown scale-in e scale-out, usa il AWS Command Line Interface () per AWS CLI gestire la scalabilità automatica di DynamoDB. Per ulteriori informazioni, consulta Utilizzo di AWS CLI per gestire la scalabilità automatica di DynamoDB.

Argomenti

- Prima di iniziare: concessione delle autorizzazioni utente per il dimensionamento automatico di DynamoDB
- Creazione di una nuova tabella con il dimensionamento automatico abilitato
- Abilitazione del dimensionamento automatico di DynamoDB su tabelle esistenti
- Visualizzazione delle attività di dimensionamento automatico sulla console
- Modifica o disabilitazione delle impostazioni di dimensionamento automatico di DynamoDB

Prima di iniziare: concessione delle autorizzazioni utente per il dimensionamento automatico di DynamoDB

In AWS Identity and Access Management (IAM), la policy AWS gestita `DynamoDBFullAccess` fornisce le autorizzazioni necessarie per l'utilizzo della console DynamoDB. Tuttavia, per dimensionamento automatico di DynamoDB, gli utenti IAM richiedono autorizzazioni aggiuntive.

> **Important**
>
> Le autorizzazioni sono necessarie per eliminare una tabella abilitata per il dimensionamento automatico. La policy AWS gestita `DynamoDBFullAccess` include tali autorizzazioni.

Per configurare un utente per l'accesso alla console DynamoDB e la scalabilità automatica di DynamoDB, crea un ruolo e aggiungi la politica di accesso a quel ruolo. AmazonDynamo DBFull Quindi assegna il ruolo a un utente.

Creazione di una nuova tabella con il dimensionamento automatico abilitato

> **Note**
>
> Il dimensionamento automatico di DynamoDB richiede la presenza di un ruolo collegato al servizio (`AWSServiceRoleForApplicationAutoScaling_DynamoDBTable`) che esegue operazioni di dimensionamento automatico per conto dell'utente. Questo ruolo viene creato automaticamente per te. Per ulteriori informazioni, consulta Ruoli collegati ai servizi per Application Auto Scaling nella Guida per l'utente di Application Auto Scaling.

Come creare una nuova tabella con la scalabilità automatica abilitata

1. Apri la console DynamoDB all'indirizzo. https://console.aws.amazon.com/dynamodb/
2. Scegliere Create table (Crea tabella).
3. Nella pagina Crea tabella, inserisci un Nome tabella e i dettagli della chiave primaria.
4. Selezionando Impostazioni predefinite, il dimensionamento automatico sarà abilitato nella nuova tabella.

   Altrimenti, seleziona Personalizza impostazioni ed effettua le seguenti operazioni per specificare le impostazioni personalizzate per la tabella:

a. Per Classe tabella, mantieni la selezione predefinita di DynamoDB Standard.

b. Per Impostazioni di capacità in scrittura/lettura, mantieni la selezione predefinita di Provisioning effettuato, quindi procedi come segue:

   i. Per la Capacità in lettura, assicurati che Auto Scaling sia impostato su Attivo.

   ii. Per la Capacità in scrittura, assicurati che Auto Scaling sia impostato su Attivo.

   iii. Per Capacità in lettura e Capacità in scrittura, imposta la policy di dimensionamento desiderata per la tabella e, facoltativamente, per tutti gli indici secondari globali della tabella.

   - Unità di capacità minima: inserisci il limite inferiore dell'intervallo di dimensionamento automatico.
   - Unità di capacità massima: inserisci il limite superiore dell'intervallo di dimensionamento automatico.
   - Utilizzo destinazione: inserisci la percentuale di utilizzo destinazione per la tabella.

> **Note**
>
> Se crei un indice secondario globale per la nuova tabella, la capacità dell'indice al momento della creazione sarà uguale alla capacità di base della tabella. Puoi modificare la capacità dell'indice nelle impostazioni della tabella dopo aver creato la tabella.

5. Scegliere Create table (Crea tabella). Questa azione crea la tabella con i parametri di dimensionamento automatico specificati.

Abilitazione del dimensionamento automatico di DynamoDB su tabelle esistenti

> **Note**
>
> Il dimensionamento automatico di DynamoDB richiede la presenza di un ruolo collegato al servizio (`AWSServiceRoleForApplicationAutoScaling_DynamoDBTable`) che esegue operazioni di dimensionamento automatico per conto dell'utente. Questo ruolo viene creato automaticamente per te. Per ulteriori informazioni, consulta Ruoli collegati ai servizi per Application Auto Scaling.

Come abilitare la scalabilità automatica DynamoDB per una tabella esistente

1. Apri la console DynamoDB all'indirizzo. https://console.aws.amazon.com/dynamodb/
2. Nel riquadro di navigazione sul lato sinistro della console scegli Tables (Tabelle).
3. Seleziona la tabella in cui desideri abilitare il dimensionamento automatico, quindi segui questa procedura:

   a. Vai alla scheda Impostazioni aggiuntive.

   b. Nella sezione Capacità di lettura/scrittura, scegli Modifica.

   c. Nella sezione Modalità di capacità, scegli Assegnata.

   d. Nella sezione Table capacity (Capacità tabella), imposta Auto Scaling (Scalabilità automatica) su On (Attiva) per Read capacity (Capacità di lettura), Write capacity (Capacità di scrittura) o entrambe. Per ognuna di queste, imposta la policy di dimensionamento desiderata per la tabella e, facoltativamente, tutti gli indici secondari globali della tabella.

      - Unità di capacità minima: inserisci il limite inferiore dell'intervallo di dimensionamento automatico.
      - Unità di capacità massima: inserisci il limite superiore dell'intervallo di dimensionamento automatico.
      - Utilizzo destinazione: inserisci la percentuale di utilizzo destinazione per la tabella.
      - Utilizza le stesse impostazioni di read/write capacità per tutti gli indici secondari globali: scegli se gli indici secondari globali devono utilizzare la stessa politica di scalabilità automatica della tabella di base.

      > **Note**
      >
      > Per prestazioni ottimali, si consiglia di abilitare l'opzione Usa le stesse impostazioni di read/write capacità per tutti gli indici secondari globali. Questa opzione consente alla scalabilità automatica di DynamoDB di dimensionare uniformemente tutti gli indici secondari globali nella tabella di base. Sono inclusi gli indici secondari globali esistenti e tutti gli altri che verranno creati per questa tabella in futuro.
      >
      > Se questa opzione è abilitata, non è possibile impostare una policy di dimensionamento su un singolo indice secondario globale.

4. Dopo aver selezionato le impostazioni desiderate, scegli Save (Salva).

Visualizzazione delle attività di dimensionamento automatico sulla console

Man mano che l'applicazione guida il traffico di lettura e scrittura nella tabella, la scalabilità automatica di DynamoDB modifica dinamicamente le impostazioni della velocità effettiva della tabella. Amazon CloudWatch tiene traccia della capacità fornita e consumata, degli eventi limitati, della latenza e di altri parametri per tutte le tabelle e gli indici secondari DynamoDB.

Per visualizzare queste metriche nella console DynamoDB, scegli la tabella che desideri utilizzare e seleziona la scheda Monitora. Per creare una visualizzazione personalizzabile delle metriche della tabella, seleziona Visualizza tutto in. CloudWatch

Modifica o disabilitazione delle impostazioni di dimensionamento automatico di DynamoDB

È possibile utilizzare il Console di gestione AWS per modificare le impostazioni di autoscaling di DynamoDB. Per eseguire questa operazione, vai alla scheda Impostazioni aggiuntive della tua tabella e seleziona Modifica nella sezione Capacità di lettura/scrittura. Per ulteriori informazioni su queste impostazioni, consultare Abilitazione del dimensionamento automatico di DynamoDB su tabelle esistenti.

Utilizzo di AWS CLI per gestire la scalabilità automatica di DynamoDB

Invece di usare il Console di gestione AWS, puoi usare il AWS Command Line Interface (AWS CLI) per gestire la scalabilità automatica di Amazon DynamoDB. Nel tutorial in questa sezione viene illustrato come installare e configurare la AWS CLI per la gestione della scalabilità automatica di DynamoDB. In questo tutorial, esegui quanto indicato di seguito:

- Crea una tabella DynamoDB denominata `TestTable`. Le impostazioni di velocità effettiva iniziali sono 5 unità di capacità di lettura e 5 unità di capacità di scrittura.
- Crea una policy di Application Auto Scaling per `TestTable`. La policy cerca di mantenere un rapporto obiettivo del 50% tra capacità di scrittura utilizzata e capacità di scrittura assegnata. L'intervallo per questo parametro è compreso tra 5 e 10 unità di capacità di scrittura. (Application Auto Scaling non può regolare la velocità effettiva oltre questo intervallo).
- Eseguire un programma Python per indirizzare il traffico di scrittura su `TestTable`. Quando il rapporto di destinazione supera il 50% per un periodo di tempo prolungato, Application Auto Scaling richiede a DynamoDB di regolare la velocità effettiva di `TestTable` verso l'alto in modo da mantenere l'utilizzo della destinazione al 50%.
- Verificare che DynamoDB abbia regolato correttamente la capacità di scrittura assegnata per `TestTable`.

> **Note**
>
> È possibile anche pianificare il dimensionamento di DynamoDB in modo che avvenga in determinati momenti. La procedura di base è disponibile qui.

Argomenti

- Prima di iniziare
- Fase 1: creazione di una tabella DynamoDB
- Fase 2: registrazione di una destinazione scalabile
- Fase 3: creazione di una policy di dimensionamento
- Passaggio 4: Indirizza il traffico di scrittura verso TestTable
- Fase 5: visualizzazione delle operazioni di Application Auto Scaling
- (Opzionale) Fase 6: pulizia

Prima di iniziare

Prima di iniziare il tutorial, completare le attività seguenti:

Installazione di AWS CLI

Se non l'hai ancora fatto, installa e configura AWS CLI. A questo scopo, seguire le istruzioni fornite nella Guida per l'utente di AWS Command Line Interface :

- Installazione dell' AWS CLI
- Configurazione della AWS CLI

Installazione di Python

Parte di questo tutorial richiede l'esecuzione di un programma Python (vedere Passaggio 4: Indirizza il traffico di scrittura verso TestTable). Se non è già installato, scaricare Python.

Fase 1: creazione di una tabella DynamoDB

In questo passaggio, usi il AWS CLI per creare. `TestTable` La chiave primaria è costituita da `pk` (chiave di partizione) e da `sk` (chiave di ordinamento). Entrambi questi attributi sono di tipo `Number`.

Le impostazioni di velocità effettiva iniziali sono 5 unità di capacità di lettura e 5 unità di capacità di scrittura.

1. Utilizzate il seguente AWS CLI comando per creare la tabella.

```
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. Per verificare lo stato della tabella, utilizza il comando seguente.

```
aws dynamodb describe-table \
    --table-name TestTable \
    --query "Table.[TableName,TableStatus,ProvisionedThroughput]"
```

   La tabella è pronta per l'uso quando lo stato diventa `ACTIVE`.

Fase 2: registrazione di una destinazione scalabile

Successivamente si registra la capacità di scrittura della tabella come destinazione scalabile con Application Auto Scaling. Ciò consente ad Application Auto Scaling di regolare la capacità di scrittura assegnata per TestTable, ma solo entro un intervallo di 5-10 unità di capacità.

> **Note**
>
> La scalabilità automatica di DynamoDB richiede la presenza di un ruolo collegato al servizio (`AWSServiceRoleForApplicationAutoScaling_DynamoDBTable`) che esegue operazioni di scalabilità automatica per conto tuo. Questo ruolo viene creato automaticamente per te. Per ulteriori informazioni, consulta Ruoli collegati ai servizi per Application Auto Scaling nella Guida per l'utente di Application Auto Scaling.

1. Utilizza il comando seguente per registrare la destinazione scalabile.

```
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. Utilizza il comando seguente per verificare la registrazione.

```
aws application-autoscaling describe-scalable-targets \
    --service-namespace dynamodb \
    --resource-id "table/TestTable"
```

> **Note**
>
> È inoltre possibile registrare una destinazione scalabile rispetto a un indice secondario globale. Ad esempio, per un indice secondario globale ("test-index"), l'ID risorsa e gli argomenti della dimensione scalabile vengono aggiornati di conseguenza.
>
> ```
> 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
> ```

Fase 3: creazione di una policy di dimensionamento

In questa fase, viene creata una policy di dimensionamento per `TestTable`. La policy definisce i dettagli in base ai quali Application Auto Scaling può regolare la velocità effettiva assegnata della tabella e le operazioni da intraprendere quando lo fa. Questa policy viene associata alla destinazione scalabile definita nel passaggio precedente (unità di capacità di scrittura per la tabella `TestTable`).

La policy contiene i seguenti elementi:

- `PredefinedMetricSpecification`: il parametro che può essere regolato da Application Auto Scaling. Per DynamoDB, i seguenti valori sono valori validi per`PredefinedMetricType`:
  - `DynamoDBReadCapacityUtilization`

  - `DynamoDBWriteCapacityUtilization`
- `ScaleOutCooldown`: la quantità minima di tempo (espressa in secondi) tra ogni evento Application Auto Scaling che aumenta la velocità effettiva assegnata. Questo parametro consente ad Application Auto Scaling di aumentare continuamente, ma non in modo aggressivo, la velocità effettiva in risposta ai carichi di lavoro reali. L'impostazione predefinita per `ScaleOutCooldown` è 0.
- `ScaleInCooldown`: la quantità minima di tempo (espressa in secondi) tra ogni evento Application Auto Scaling che diminuisce la velocità effettiva assegnata. Questo parametro consente ad Application Auto Scaling di ridurre la velocità effettiva in maniera graduale e prevedibile. L'impostazione predefinita per `ScaleInCooldown` è 0.
- `TargetValue`: Application Auto Scaling assicura che il rapporto tra la capacità utilizzata e la capacità assegnata rimanga pari o vicino a questo valore. `TargetValue` viene definito in percentuale.

> **Note**
>
> Per capire meglio come funziona `TargetValue`, supponiamo di avere una tabella con un'impostazione di velocità effettiva assegnata pari a 200 unità di capacità in scrittura. Si decide di creare una policy di dimensionamento per questa tabella, con un `TargetValue` del 70%.
>
> Si supponga ora di iniziare a guidare il traffico di scrittura verso la tabella in modo che la velocità effettiva di scrittura sia di 150 unità di capacità. Il consumed-to-provisioned rapporto è ora (150/ 200), ossia il 75 percento. Questo rapporto supera l'obiettivo, pertanto Application Auto Scaling aumenta la capacità di scrittura assegnata a 215 in modo che il rapporto sia (150/215) o 69,77%, il più vicino possibile a `TargetValue` ma senza superarlo.

Per `TestTable`, `TargetValue` si imposta al 50%. Application Auto Scaling regola il throughput assegnato dalla tabella entro un intervallo di 5-10 unità di capacità (vediFase 2: registrazione di una destinazione scalabile) in modo che il consumed-to-provisioned rapporto rimanga pari o vicino al 50 percento. I valori di `ScaleOutCooldown` e `ScaleInCooldown` vengono impostati su 60 secondi.

1. Crea un file denominato `scaling-policy.json` con i seguenti contenuti.

```
{
    "PredefinedMetricSpecification": {
```

```
        "PredefinedMetricType": "DynamoDBWriteCapacityUtilization"
    },
    "ScaleOutCooldown": 60,
    "ScaleInCooldown": 60,
    "TargetValue": 50.0
}
```

2. Utilizzare il comando seguente per creare la politica AWS CLI .

```
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. Nell'output, tieni presente che Application Auto Scaling ha creato due CloudWatch allarmi Amazon, uno per il limite superiore e inferiore dell'intervallo target di scalabilità.

4. Usa il seguente AWS CLI comando per visualizzare maggiori dettagli sulla politica di scalabilità.

```
aws application-autoscaling describe-scaling-policies \
    --service-namespace dynamodb \
    --resource-id "table/TestTable" \
    --policy-name "MyScalingPolicy"
```

5. Nell'output, verificare che le impostazioni delle policy corrispondano alle specifiche da Fase 2: registrazione di una destinazione scalabile e Fase 3: creazione di una policy di dimensionamento.

Passaggio 4: Indirizza il traffico di scrittura verso TestTable

Ora è possibile testare la policy di dimensionamento scrivendo i dati in `TestTable`. Per fare ciò, viene eseguito un programma Python.

1. Crea un file denominato `bulk-load-test-table.py` con i seguenti contenuti.

```
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. Per eseguire il programma, immettere il comando seguente.

   ```
   python bulk-load-test-table.py
   ```

   La capacità di scrittura assegnata per `TestTable` è molto bassa (5 unità di capacità di scrittura), quindi il programma si blocca occasionalmente a causa della limitazione della scrittura. Questo è il comportamento previsto.

   Lasciare che il programma continui a funzionare mentre si passa alla fase successiva.

Fase 5: visualizzazione delle operazioni di Application Auto Scaling

In questa fase vengono visualizzate le operazioni di Application Auto Scaling che vengono avviate per conto tuo. Viene inoltre verificato che Application Auto Scaling abbia aggiornato la capacità di scrittura assegnata per `TestTable`.

1. Immettere il comando seguente per visualizzare le operazioni di Application Auto Scaling.

   ```
   aws application-autoscaling describe-scaling-activities \
       --service-namespace dynamodb
   ```

Eseguire di nuovo questo comando occasionalmente, mentre il programma Python è in esecuzione. Occorrono diversi minuti prima che la policy di dimensionamento venga richiamata. Dovrebbe essere visualizzato l'output riportato di seguito.

```
...
{
    "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"
},
...
```

Ciò indica che Application Auto Scaling ha emesso una richiesta `UpdateTable` a DynamoDB.

2. Immettere il comando seguente per verificare che DynamoDB ha aumentato la capacità di scrittura della tabella.

```
aws dynamodb describe-table \
    --table-name TestTable \
    --query "Table.[TableName,TableStatus,ProvisionedThroughput]"
```

`WriteCapacityUnits` dovrebbe essersi dimensionato da `5` a `10`.

(Opzionale) Fase 6: pulizia

In questo tutorial sono state create diverse risorse. È possibile eliminare queste risorse se non sono più necessarie.

1. Eliminare la policy di dimensionamento per `TestTable`.

```
aws application-autoscaling delete-scaling-policy \
    --service-namespace dynamodb \
    --resource-id "table/TestTable" \
    --scalable-dimension "dynamodb:table:WriteCapacityUnits" \
    --policy-name "MyScalingPolicy"
```

2. Annullare la registrazione di una destinazione scalabile.

```
aws application-autoscaling deregister-scalable-target \
    --service-namespace dynamodb \
    --resource-id "table/TestTable" \
    --scalable-dimension "dynamodb:table:WriteCapacityUnits"
```

3. Eliminare la tabella `TestTable`.

```
aws dynamodb delete-table --table-name TestTable
```

Utilizzo dell' AWS SDK per configurare la scalabilità automatica sulle tabelle Amazon DynamoDB

Oltre a utilizzare Console di gestione AWS and the AWS Command Line Interface (AWS CLI), puoi scrivere applicazioni che interagiscono con la scalabilità automatica di Amazon DynamoDB. Questa sezione contiene due programmi Java che puoi utilizzare per verificare questa funzionalità:

- `EnableDynamoDBAutoscaling.java`
- `DisableDynamoDBAutoscaling.java`

Abilitazione di Application Auto Scaling per una tabella

Nel programma seguente viene riportato un esempio di configurazione di una policy di scalabilità automatica per una tabella DynamoDB (`TestTable`). Si procede nel seguente modo.

- Il programma registra unità di capacità in scrittura come un obiettivo scalabile per `TestTable`. L'intervallo per questo parametro è compreso tra 5 e 10 unità di capacità di scrittura.
- Dopo aver creato l'obiettivo scalabile, il programma sviluppa configurazioni di monitoraggio obiettivi. La policy cerca di mantenere un rapporto obiettivo del 50% tra capacità di scrittura utilizzata e capacità di scrittura assegnata.

- In seguito, il programma crea la policy di dimensionamento in base alla configurazione di monitoraggio obiettivi.

> **Note**
>
> Quando rimuovi manualmente una tabella o una replica globale di una tabella, non rimuovi automaticamente gli obiettivi scalabili, le politiche di scalabilità o gli allarmi associati. 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

Il programma richiede che venga specificato un Amazon Resource Name (ARN) per un ruolo collegato ai servizi di Application Auto Scaling valido. (Ad esempio: `arn:aws:iam::122517410325:role/AWSServiceRoleForApplicationAutoScaling_DynamoDBTable`.) Nel programma seguente, sostituisci `SERVICE_ROLE_ARN_GOES_HERE` con l'ARN reale.

```
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());
   }

  }

}
```

Disabilitazione di Application Auto Scaling per una tabella

Nel programma seguente, il processo precedente viene invertito. Esso rimuove la policy di scalabilità automatica e annulla la registrazione dell'obiettivo scalabile.

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());
 }

}
```

```
}
```

Capacità riservata di DynamoDB

Per le tabelle con capacità allocata che utilizzano la Classe tabella Standard, DynamoDB offre la possibilità di acquistare capacità riservata per lettura e scrittura. L'acquisto di capacità riservata è un accordo che prevede il pagamento di un importo minimo di capacità di throughput allocata per tutta la durata del contratto a fronte di prezzi scontati.

> **Note**
>
> Non è possibile acquistare capacità riservata per unità di capacità di scrittura replicate (r). WCUs La capacità riservata viene applicata solo alla Regione in cui è stata acquistata. La capacità riservata non è disponibile nemmeno per le tabelle che utilizzano la classe di tabella DynamoDB Standard (accesso infrequente) o la modalità di capacità on demand.

La capacità riservata viene acquistata in allocazioni di 100 WCUs o 100. RCUs L'offerta di capacità riservata più piccola è di 100 unità di capacità (lettura o scrittura). La capacità riservata di DynamoDB viene offerta con un impegno annuale o, in alcune Regioni, con un impegno triennale. È possibile risparmiare fino al 54% sulle tariffe standard per un periodo di un anno e fino al 77% sulle tariffe standard per un periodo di tre anni. Per ulteriori informazioni sulle tempistiche e le modalità di acquisto, consulta Amazon DynamoDB Reserved Capacity.

Durante l'acquisto di capacità riservata di DynamoDB viene effettuato un pagamento anticipato parziale una tantum a fronte di una tariffa oraria scontata per l'utilizzo allocato impegnato. Il pagamento è per l'intero utilizzo allocato impegnato, indipendentemente dall'utilizzo effettivo, pertanto i risparmi sui costi sono strettamente legati all'utilizzo. Qualsiasi capacità allocata in eccesso rispetto alla capacità riservata acquistata viene fatturata alle tariffe standard della capacità allocata. Prenotando in anticipo le unità di capacità di lettura e scrittura, hai un risparmio significativo sui costi della capacità assegnata.

Non è possibile vendere, annullare o trasferire la capacità riservata a un'altra Regione o account.

Funzionamento del throughput di DynamoDB

Il throughput a caldo si riferisce al numero di operazioni di lettura e scrittura che la tabella DynamoDB è in grado di supportare in modo istantaneo. Questi valori sono disponibili per impostazione predefinita per tutte le tabelle e gli indici secondari globali (GSI) e rappresentano il dimensionamento attuato in base all'utilizzo storico. Se si utilizza la modalità on demand o se si aggiorna il throughput allocato a questi valori, l'applicazione sarà in grado di emettere richieste fino a tali valori in modo istantaneo.

DynamoDB regolerà automaticamente i valori di throughput a caldo man mano che l'utilizzo aumenta. Tuttavia, è possibile anche aumentare questi valori in modo proattivo quando necessario, il che è particolarmente utile per i prossimi eventi di picco come il lancio o le vendite di prodotti. Per gli eventi di picco pianificati, in cui i tassi di richiesta alla tabella DynamoDB potrebbero aumentare di 10, 100 volte o più, ora è possibile valutare se l'attuale throughput a caldo è sufficiente per gestire il traffico previsto. In caso negativo, è possibile aumentare il valore di throughput a caldo senza modificare le impostazioni di throughput o la modalità di fatturazione. Questo processo, denominato pre-riscaldamento di una tabella, consente di impostare una baseline che le tabelle possano supportare in modo istantaneo. Ciò garantisce che le applicazioni siano in grado di gestire tassi di richieste più elevati dal momento in cui si verificano.

È possibile aumentare il valore di throughput a caldo per le operazioni di lettura, le operazioni di scrittura o entrambe. È possibile aumentare questo valore per tabelle a regione singola, tabelle globali e nuove ed esistenti. GSIs Per le tabelle globali, questa funzionalità è disponibile per la versione 2019.11.21 (Corrente) e le impostazioni di throughput a caldo configurate verranno applicate automaticamente a tutte le tabelle di replica nella tabella globale. Non è previsto alcun limite per il numero di tabelle DynamoDB che è possibile pre-riscaldare in qualsiasi momento. Il tempo necessario per completare il pre-riscaldamento dipende dai valori impostati e dalla dimensione della tabella o dell'indice. È possibile inviare richieste di pre-riscaldamento simultanee e tali richieste non interferiranno con le operazioni della tabella. È possibile preriscaldare il tavolo fino al limite di quota della tabella o dell'indice previsto per l'account in quella Regione. Utilizza la console Service Quotas per controllare i limiti attuali e aumentarli se necessario.

I valori di throughput a caldo sono disponibili per impostazione predefinita per tutte le tabelle e gli indici secondari senza alcun costo. Tuttavia, se si aumentano in modo proattivo questi valori di throughput caldo predefiniti per pre-riscaldare le tabelle, verranno addebitati i costi per tali richieste. Per ulteriori informazioni, consulta Prezzi di Amazon DynamoDB.

Per ulteriori informazioni sul throughput a caldo, consulta gli argomenti riportati di seguito:

Argomenti

- Check your DynamoDB table's current warm throughput
- Aumento del throughput a caldo di una tabella DynamoDB esistente
- Creazione di una nuova tabella DynamoDB con throughput caldo più elevato
- Funzionamento del throughput a caldo di DynamoDB in diversi scenari

Check your DynamoDB table's current warm throughput

Utilizza le seguenti istruzioni AWS CLI e quelle AWS della Console per controllare il valore attuale della velocità effettiva a caldo della tabella o dell'indice.

Console di gestione AWS

Per controllare il throughput a caldo di una tabella DynamoDB utilizzando la console DynamoDB:

1. Accedi Console di gestione AWS e apri la console DynamoDB all'indirizzo. https://console.aws.amazon.com/dynamodb/
2. Nel riquadro di navigazione a sinistra, selezionare Tables (Tabelle).
3. Nella pagina Tabelle, seleziona la tabella desiderata.
4. Seleziona Impostazioni aggiuntive per visualizzare il valore attuale del throughput a caldo. Questo valore viene visualizzato come unità di lettura al secondo e unità di scrittura al secondo.

AWS CLI

L' AWS CLI esempio seguente mostra come controllare il throughput a caldo della tabella DynamoDB.

1. Esegui l'operazione `describe-table` sulla tabella DynamoDB.

```
aws dynamodb describe-table --table-name GameScores
```

2. Riceverai una risposta simile a quella seguente. Le impostazioni sul `WarmThroughput` verranno visualizzate come `ReadUnitsPerSecond` e`WriteUnitsPerSecond`. `Status` risulterà `UPDATING` quando verrà aggiornato il valore di throughput a caldo e `ACTIVE` quando sarà stato impostato il nuovo valore di throughput a caldo.

```
{
```

```
    "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 throughput a caldo di una tabella DynamoDB esistente

Dopo aver verificato l'attuale valore di throughput a caldo della tabella DynamoDB, è possibile aggiornarla attraverso la seguente procedura:

Console di gestione AWS

Come controllare il valore di throughput a caldo della tabella DynamoDB utilizzando la console DynamoDB:

1. Accedi Console di gestione AWS e apri la console DynamoDB all'indirizzo. https://console.aws.amazon.com/dynamodb/
2. Nel riquadro di navigazione a sinistra, selezionare Tables (Tabelle).
3. Nella pagina Tabelle, seleziona la tabella desiderata.
4. Nel campo Throughput a caldo, seleziona Modifica.
5. Nella pagina Modifica il throughput effettivo, seleziona Aumentare il throughput a caldo.
6. Regola le Unità di lettura al secondo e le Unità di scrittura al secondo. Queste due impostazioni definiscono il throughput che la tabella può gestire in modo istantaneo.
7. Seleziona Salva.
8. Le unità di lettura al secondo e le unità di scrittura al secondo verranno aggiornate nel campo Throughput a caldo al termine dell'elaborazione della richiesta.

> **Note**
>
> L'aggiornamento del valore di throughput caldo è un'operazione asincrona. `Status` cambierà da `UPDATING` a `ACTIVE` quando l'aggiornamento sarà completo.

AWS CLI

L' AWS CLI esempio seguente mostra come aggiornare il valore di throughput caldo della tabella DynamoDB.

1. Esegui l'operazione `update-table` sulla tabella 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. Riceverai una risposta simile a quella seguente. Le impostazioni sul `WarmThroughput` verranno visualizzate come `ReadUnitsPerSecond` e`WriteUnitsPerSecond`. `Status` risulterà `UPDATING` quando verrà aggiornato il valore di throughput a caldo e `ACTIVE` quando sarà stato impostato il nuovo valore di throughput a caldo.

```
{
    "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

Gli esempi dell'SDK seguenti mostrano come aggiornare il valore di throughput a caldo di una tabella 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;
  }
}
```

Creazione di una nuova tabella DynamoDB con throughput caldo più elevato

È possibile regolare i valori di throughput a caldo quando si crea una tabella DynamoDB attraverso la procedura seguente. Queste fasi si applicano anche alla creazione di una tabella globale o di un indice secondario.

Console di gestione AWS

Per creare una tabella DynamoDB e regolare i valori di throughput a caldo tramite la console:

1. Accedi Console di gestione AWS e apri la console DynamoDB all'indirizzo. https://console.aws.amazon.com/dynamodb/
2. Seleziona Crea tabella.
3. Seleziona un Nome tabella, una Chiave di partizione e una Chiave di ordinamento (opzionale).
4. Per Impostazioni tabella, seleziona Personalizza impostazioni.
5. Nel campo Throughput a caldo, seleziona Aumentare il throughput a caldo.
6. Regola le Unità di lettura al secondo e le Unità di scrittura al secondo. Queste due impostazioni definiscono il throughput massimo che la tabella può gestire in modo istantaneo.
7. Continua ad aggiungere i dettagli rimanenti della tabella, quindi seleziona Crea tabella.

AWS CLI

L' AWS CLI esempio seguente mostra come creare una tabella DynamoDB con valori di throughput caldi personalizzati.

1. Esegui l'operazione `create-table` per creare la seguente tabella 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. Riceverai una risposta simile a quella seguente. Le impostazioni sul `WarmThroughput` verranno visualizzate come `ReadUnitsPerSecond` e`WriteUnitsPerSecond`. `Status` risulterà `UPDATING` quando verrà aggiornato il valore di throughput a caldo e `ACTIVE` quando sarà stato impostato il nuovo valore di throughput a caldo.

```
{
    "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

Gli esempi dell'SDK seguenti mostrano come creare una tabella DynamoDB con valori di throughput a caldo personalizzati.

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;
  }
}
```

Funzionamento del throughput a caldo di DynamoDB in diversi scenari

Di seguito sono riportati alcuni scenari diversi che possono verificarsi durante l'utilizzo del throughput a caldo di DynamoDB.

Argomenti

- Throughput a caldo e modelli di accesso irregolari
- Throughput a caldo per una tabella con provisioning
- Throughput a caldo per una tabella on demand
- Throughput a caldo per una tabella on demand con un throughput massimo configurato

Throughput a caldo e modelli di accesso irregolari

Una tabella può avere un throughput di 30.000 unità di lettura al secondo e 10.000 unità di scrittura al secondo, ma è comunque possibile che si verifichi una limitazione (della larghezza di banda della rete) delle operazioni di lettura o scrittura prima di raggiungere tali valori. Ciò è probabilmente dovuto a una partizione calda. Sebbene DynamoDB possa continuare a scalare per supportare un throughput praticamente illimitato, ogni singola partizione è limitata a 1.000 unità di scrittura al secondo e 3.000 unità di lettura al secondo. Se l'applicazione indirizza troppo traffico verso una piccola parte delle partizioni della tabella, la limitazione (della larghezza di banda della rete) può verificarsi anche prima di raggiungere i valori di throughput a caldo della tabella. Si consiglia di seguire le best practice di DynamoDB per garantire una scalabilità perfetta ed evitare partizioni calde.

Throughput a caldo per una tabella con provisioning

Si prenda in considerazione una tabella con provisioning con un throughput a caldo di 30.000 unità di lettura al secondo e 10.000 unità di scrittura al secondo, ma che attualmente ha un throughput allocato di 4.000 RCU e 8.000 WCU. È possibile scalare istantaneamente il throughput allocato alla tabella fino a 30.000 RCU o 10.000 WCU aggiornando le impostazioni di throughput allocato.

Quando si aumenta il throughput allocato oltre questi valori, il throughput a caldo si adeguerà automaticamente ai nuovi valori più alti, poiché è stato impostato un nuovo throughput di picco. Ad esempio, se si imposta il throughput allocato su 50.000 RCU, il throughput a caldo aumenterà a 50.000 unità di lettura al secondo.

```
"ProvisionedThroughput":
    {
        "ReadCapacityUnits": 4000,
        "WriteCapacityUnits": 8000
    }
"WarmThroughput":
    {
        "ReadUnitsPerSecond": 30000,
        "WriteUnitsPerSecond": 10000
    }
```

Throughput a caldo per una tabella on demand

Una nuova tabella on demand inizia con un throughput a caldo di 12.000 unità di lettura al secondo e 4.000 unità di scrittura al secondo. La tabella può supportare in modo istantaneo un traffico sostenuto fino a questi livelli. Quando le richieste superano le 12.000 unità di lettura al secondo o le 4.000 unità di scrittura al secondo, il throughput a caldo si adeguerà automaticamente a valori più alti.

```
"WarmThroughput":
    {
        "ReadUnitsPerSecond": 12000,
        "WriteUnitsPerSecond": 4000
    }
```

Throughput a caldo per una tabella on demand con un throughput massimo configurato

Si prenda in considerazione una tabella on demand con una throughput a caldo di 30.000 unità di lettura al secondo, ma con un throughput massimo configurato a 5.000 unità di richiesta di lettura (RRU). In questo scenario, il throughput della tabella sarà limitato al massimo di 5.000 RRU impostate. Tutte le richieste di throughput che superano questo limite subiranno una limitazione (della larghezza di banda della rete). Tuttavia, è possibile modificare il throughput massimo specifico della tabella in qualsiasi momento in base alle esigenze dell'applicazione.

```
"OnDemandThroughput":
```

```
    {
        "MaxReadRequestUnits": 5000,
        "MaxWriteRequestUnits": 4000
    }
"WarmThroughput":
    {
        "ReadUnitsPerSecond": 30000,
        "WriteUnitsPerSecond": 10000
    }
```

Capacità di espansione e capacità adattiva di DynamoDB

Per ridurre al minimo la limitazione (della larghezza di banda della rete) dovuta alle eccezioni di throughput, DynamoDB utilizza la capacità di espansione per gestire i picchi di utilizzo. DynamoDB utilizza la capacità adattiva per aiutare a gestire modelli di accesso irregolari.

Capacità di ottimizzazione

DynamoDB garantisce una certa flessibilità nell'assegnazione della velocità di trasmissione effettiva grazie alla capacità di espansione. Quando non si utilizza completamente il throughput di una partizione, DynamoDB riserva una parte della capacità inutilizzata per successive espansioni del throughput per la gestione dei picchi di traffico. Con la capacità supplementare, le richieste di lettura o scrittura impreviste possono avere esito positivo anziché essere sottoposte a throttling.

DynamoDB attualmente mantiene fino a cinque minuti (300 secondi) di capacità di lettura e scrittura non utilizzata. Durante un'espansione occasionale di attività di lettura o scrittura, queste unità di capacità extra possono essere utilizzate rapidamente, anche più velocemente della capacità di throughput allocata al secondo definita per la tabella.

DynamoDB può inoltre utilizzare la capacità di ottimizzazione per la manutenzione in background e altre attività senza preavviso.

Tieni presente che tali dettagli della capacità di ottimizzazione in futuro potrebbero cambiare.

Capacità adattiva

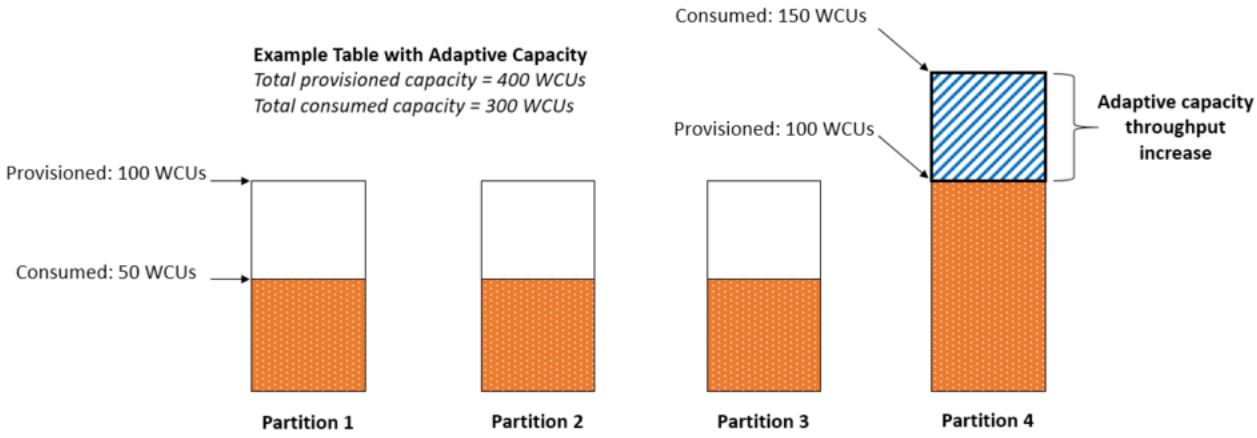
DynamoDB distribuisce automaticamente i dati tra le partizioni, che sono archiviate su più server nel Cloud AWS. Non sempre è possibile distribuire l'attività di lettura e scrittura in modo uniforme. Se l'accesso ai dati non è equilibrato, una partizione "hot" può ricevere un volume più elevato di traffico di lettura e scrittura rispetto ad altre partizioni. Poiché le operazioni di lettura e scrittura su

una partizione vengono gestite in modo indipendente, si verificherà la limitazione (della larghezza di banda della rete) se una singola partizione riceve più di 3000 operazioni di lettura o più di 1000 operazioni di scrittura. La capacità adattiva funziona aumentando automaticamente la capacità di throughput per le partizioni che ricevono più traffico.

Per gestire meglio i modelli di accesso non uniformi, la capacità adattiva di DynamoDB permette all'applicazione di continuare con le attività di lettura e scrittura sulle partizioni hot senza che si verifichi una limitazione, a condizione che tale traffico non superi la capacità totale assegnata alla tabella o la capacità massima della partizione. La capacità adattiva funziona aumentando automaticamente la capacità di throughput per le partizioni che ricevono più traffico.

Nel diagramma seguente viene illustrato il funzionamento della capacità adattiva. Nella tabella di esempio sono disponibili 400 partizioni distribuite in WCUs modo uniforme su quattro partizioni, che consentono a ciascuna partizione di supportare fino a 100 al secondo. WCUs Le partizioni 1, 2 e 3 ricevono ciascuna un traffico di scrittura pari a 50. WCU/sec. Partition 4 receives 150 WCU/sec. This hot partition can accept write traffic while it still has unused burst capacity, but eventually it throttles traffic that exceeds 100 WCU/sec

La capacità adattiva di DynamoDB risponde aumentando la capacità della partizione 4 in modo che possa sostenere il carico di lavoro più elevato di 150 senza subire limitazioni. WCU/sec



La capacità adattiva è abilitata automaticamente per tutte le tabelle DynamoDB, senza costi aggiuntivi. Non è necessario abilitarla o disabilitarla esplicitamente.

Isolare gli elementi con accesso frequente

Se l'applicazione instrada un traffico elevato in modo sproporzionato verso uno o più elementi, la capacità adattiva eseguirà un bilanciamento delle partizioni, affinché gli elementi ad accesso

frequente non siano ubicati all'interno della stessa partizione. Questo isolamento degli elementi ad accesso frequente riduce la probabilità di throttling delle richieste a causa del superamento della quota del throughput del carico di lavoro su una singola partizione. Puoi anche suddividere una raccolta di elementi in segmenti in base alla chiave di ordinamento, purché tale raccolta non generi traffico monitorato da un aumento o una diminuzione monotoni della chiave di ordinamento.

Se l'applicazione invia in modo coerente traffico elevato a un singolo elemento, la capacità adattiva può riequilibrare i dati in modo tale che una partizione contenga solo quel singolo elemento con accesso frequente. In questo caso, DynamoDB può fornire un throughput fino alla partizione massima di RCUs 3.000 e WCUs 1.000 alla chiave primaria di quel singolo elemento. La capacità adattiva non suddivide le raccolte di elementi su più partizioni della tabella quando è presente un indice secondario locale.

Considerazioni sul passaggio tra modalità di capacità in DynamoDB

Quando crei una tabella DynamoDB, devi selezionare la modalità di capacità on demand o la modalità di capacità assegnata.

È possibile cambiare le tabelle dalla modalità con capacità allocata alla modalità on demand fino a quattro volte in una finestra variabile di 24 ore. È possibile passare le tabelle dalla modalità on demand alla modalità con capacità allocata in qualsiasi momento.

Argomenti

- Passaggio dalla modalità con capacità allocata alla modalità con capacità on demand
- Passaggio dalla modalità con capacità on demand alla modalità con capacità allocata

Passaggio dalla modalità con capacità allocata alla modalità con capacità on demand

In modalità con provisioning è possibile impostare la capacità di lettura e scrittura in base alle esigenze previste dell'applicazione. Quando si aggiorna una tabella dalla modalità assegnata a quella on demand, non è necessario specificare quanto throughput di lettura e scrittura si prevede che l'applicazione esegua. DynamoDB on-demand offre prezzi pay-per-request semplici per le richieste di lettura e scrittura in modo da pagare solo per ciò che si utilizza, facilitando il bilanciamento di costi e prestazioni. Facoltativamente, è possibile configurare il throughput massimo di lettura o di scrittura (o entrambi) per le singole tabelle on demand e gli indici secondari globali associati per mantenere costi

e utilizzo limitati. Per ulteriori informazioni sull'impostazione del throughput massimo per una tabella o un indice specifico, consulta Throughput massimo di DynamoDB per le tabelle on demand.

Quando si passa dalla modalità con capacità allocata alla modalità con capacità on demand, DynamoDB apporta numerosi cambiamenti alla struttura della tabella e delle partizioni. Questo processo può richiedere alcuni minuti. Durante la durata del passaggio, la tabella assicura un throughput consistente con l'unità di capacità in scrittura precedentemente assegnata e le quantità di unità di capacità.

Velocità di trasmissione effettiva iniziale per la modalità di capacità on demand

Se una tabella esistente è recentemente passata alla modalità con capacità on demand per la prima volta, disporrà delle seguenti impostazioni del picco precedente, nonostante in precedenza la tabella non abbia gestito il traffico tramite la modalità con capacità on demand.

Di seguito sono riportati alcuni esempi di possibili scenari:

- Qualsiasi tabella con provisioning configurato al di sotto di 4000 WCU e 12.000 RCU, per cui non sia mai stato precedentemente allocato un numero superiore. Quando passi questa tabella a on-demand per la prima volta, DynamoDB si assicurerà che sia scalabile per supportare istantaneamente almeno 4.000 unità di scrittura e 12.000 unità di lettura/sec. units/sec
- Una tabella con provisioning configurato come 8.000 WCU e 24.000 RCU. Quando passi a questa tabella su richiesta, continuerà a essere in grado di supportare almeno 8.000 operazioni di scrittura e 24.000 letture in qualsiasi momento. units/sec units/sec
- Una tabella predisposta configurata con 8.000 WCU e 24.000 RCU, che ha consumato 6.000 operazioni di scrittura e 18.000 letture per un periodo prolungato. units/sec units/sec Quando passi a questa tabella su richiesta, continuerà a essere in grado di supportare almeno 8.000 unità di scrittura e 24.000 unità di lettura/sec. units/sec Il traffico precedente può inoltre consentire alla tabella di sostenere livelli di traffico molto più elevati senza limitazione della larghezza di banda della rete.
- Una tabella precedentemente assegnata con 10.000 WCU e 10.000 RCU, ma attualmente assegnata con 10 RCU e 10 WCU. Passando a questa tabella su richiesta, sarà in grado di supportare almeno 10.000 unità di scrittura e 10.000 unità di lettura/sec. units/sec

Impostazioni di dimensionamento automatico

Quando aggiorni una tabella dalla modalità assegnata a quella on demand:

- Se utilizzi la console, tutte le (eventuali) impostazioni di scalabilità automatica verranno eliminate.
- Se utilizzi l' AWS SDK AWS CLI o, tutte le impostazioni di ridimensionamento automatico verranno mantenute. Queste impostazioni possono essere applicate quando aggiorni nuovamente la tabella alla modalità di fatturazione assegnata.

Modifica della modalità di capacità in blocco nella console DynamoDB

È possibile modificare in blocco più tabelle perché passino dalla modalità con capacità allocata alla modalità con capacità on demand utilizzando la console DynamoDB. Come modificare in blocco la modalità di capacità:

1. Nella console DynamoDB apri la pagina Tabelle.
2. Seleziona le caselle di spunta per le tabelle che desideri aggiornare alla modalità con capacità on demand.
3. Seleziona Operazioni, quindi seleziona Aggiorna la modalità di capacità su richiesta.

Questa operazione in blocco consente di effettuare in modo efficiente il passaggio per più tabelle alla modalità con capacità on demand senza dover aggiornare ogni tabella singolarmente.

Passaggio dalla modalità con capacità on demand alla modalità con capacità allocata

Durante il ritorno alla modalità di capacità assegnata, a partire dalla modalità di capacità on demand, la tabella assicura un throughput consistente con il picco precedente raggiunto quando la tabella era impostata sulla modalità di capacità on demand.

Gestione della capacità

Quando aggiorni una tabella dalla modalità on demand a quella assegnata, considera quanto segue:

- Se utilizzi l' AWS SDK AWS CLI o, scegli le impostazioni di capacità assegnate corrette della tabella e degli indici secondari globali utilizzando Amazon CloudWatch per esaminare il consumo storico (`ConsumedWriteCapacityUnitse` le `ConsumedReadCapacityUnits` metriche) per determinare le nuove impostazioni di throughput.

> **Note**
>
> Se sposti una tabella globale alla modalità assegnata, osserva il consumo massimo tra tutte le repliche regionali per le tabelle di base e gli indici secondari globali quando stabilisci le nuove impostazioni di throughput.

- In caso di ritorno dalla modalità on demand alla modalità con capacità allocata, è necessario impostare un numero sufficiente di unità allocate iniziali in modo da gestire la capacità della tabella o dell'indice durante la transizione.

Gestione del dimensionamento automatico

Quando aggiorni una tabella dalla modalità on demand a quella assegnata:

- In caso di utilizzo della console, si consiglia di abilitare il dimensionamento automatico con le impostazioni predefinite seguenti:
  - Utilizzo di destinazione: 70%
  - Capacità minima assegnata: 5 unità
  - Capacità massima assegnata: il massimo delle regioni
- Se utilizzi l'SDK AWS CLI o, le impostazioni di ridimensionamento automatico precedenti (se presenti) vengono mantenute.

Programmazione con DynamoDB e SDK AWS

In questa sezione sono riportati argomenti per gli sviluppatori. Se invece desideri eseguire esempi di codice, consulta Esecuzione degli esempi di codice in questa guida per gli sviluppatori.

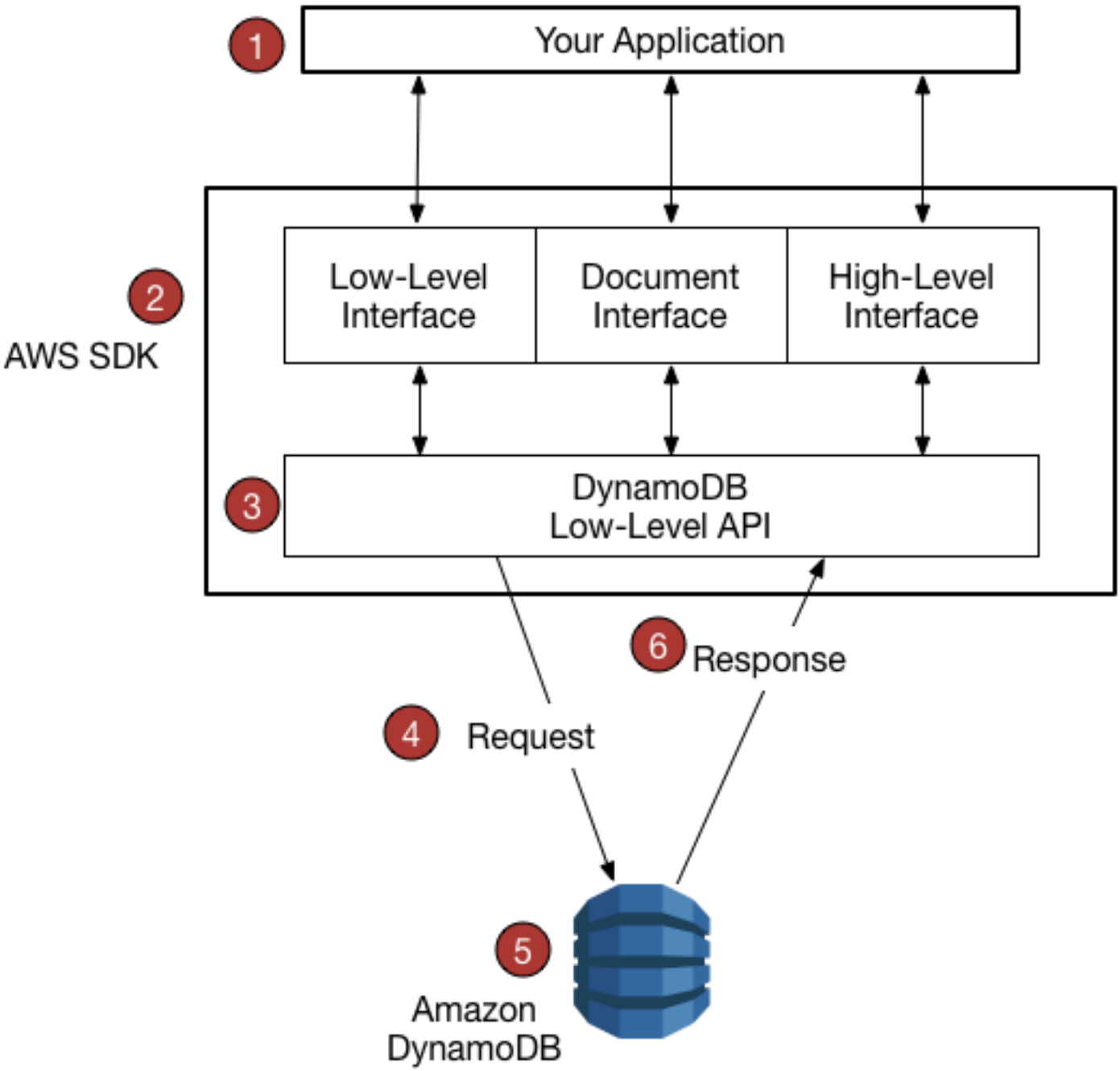
> **Note**
>
> Nel mese di dicembre 2017, AWS ha avviato il processo di migrazione di tutti gli endpoint Amazon DynamoDB in modo da utilizzare certificati sicuri emessi da Amazon Trust Services (ATS). Per ulteriori informazioni, consulta Risoluzione dei problemi di creazione della SSL/TLS connessione con DynamoDB.

Argomenti

- Panoramica del supporto degli SDK AWS per DynamoDB
- Programmazione di Amazon DynamoDB con Python e Boto3
- Programmazione di Amazon DynamoDB con JavaScript
- Programmazione di DynamoDB con AWS SDK for Java 2.x
- Gestione degli errori con DynamoDB
- Utilizzo di DynamoDB con un SDK AWS

Panoramica del supporto degli SDK AWS per DynamoDB

Il seguente diagramma fornisce una panoramica dettagliata della programmazione dell'applicazione Amazon DynamoDB con gli SDK AWS.



1. Si scrive un'applicazione utilizzando un SDK AWS per il linguaggio di programmazione in uso.
2. Ciascun SDK AWS fornisce una o più interfacce programmatiche per lavorare con DynamoDB. Le interfacce specifiche disponibili dipendono dal linguaggio di programmazione e dall'SDK AWS in uso. Le opzioni includono:
   - Interfacce di basso livello che funzionano con DynamoDB
   - Interfacce documentali che funzionano con DynamoDB
   - Interfacce di persistenza degli oggetti che funzionano con DynamoDB

- Interfacce di alto livello

3. L'SDK AWS costruisce richieste HTTP(S) per l'utilizzo con l'API DynamoDB di basso livello
4. L'SDK AWS invia la richiesta all'endpoint DynamoDB.
5. DynamoDB esegue la richiesta. Se la richiesta ha esito positivo, DynamoDB restituisce un codice di risposta HTTP 200 (OK). Se la richiesta ha esito negativo, DynamoDB restituisce un codice di errore HTTP e un messaggio di errore.
6. L'SDK AWS elabora la risposta e la propaga nuovamente all'applicazione.

Ciascuno degli SDK AWS fornisce servizi importanti per l'applicazione, incluso:

- Formattazione delle richieste HTTP(S) e serializzazione dei parametri di richiesta.
- Generazione di una firma di crittografia per ogni richiesta.
- Inoltro delle richieste a un endpoint DynamoDB e ricezione delle risposte da DynamoDB.
- Estrazione dei risultati da quelle risposte.
- Implementazione della logica dei nuovi tentativi di base in caso di errori.

Per nessuna di queste attività non occorre scrivere del codice.

> **Note**
>
> Per ulteriori informazioni sugli SDK AWS, incluse le istruzioni di installazione e la documentazione, consulta Strumenti per Amazon Web Services.

Supporto SDK per endpoint basati su account AWS

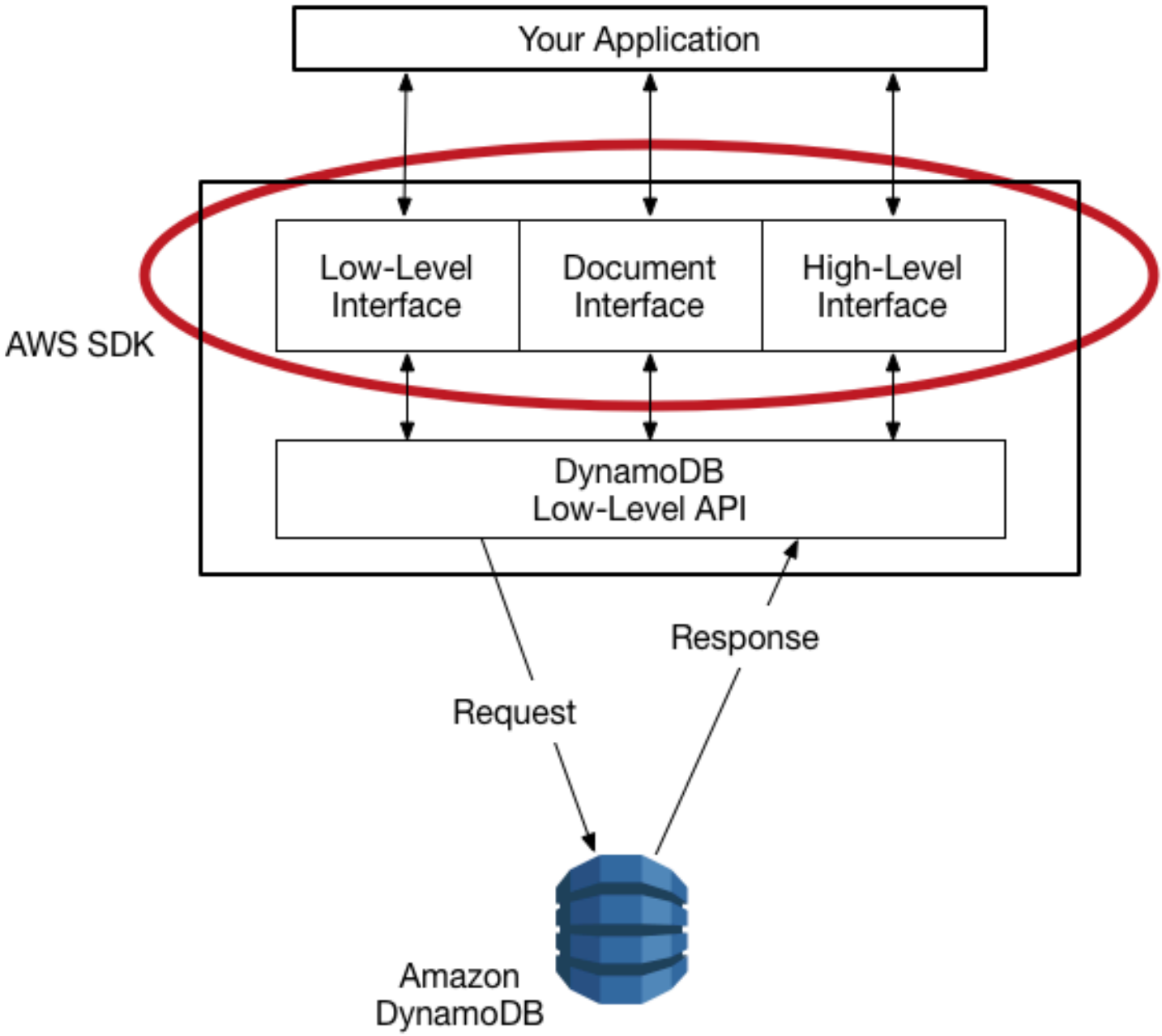
AWS sta implementando il supporto SDK per gli endpoint basati su account AWS per DynamoDB, a partire dall'SDK AWS per Java V1 il 4 settembre 2024. Questi nuovi endpoint aiutano AWS a garantire prestazioni e scalabilità elevate. Gli SDK aggiornati utilizzeranno automaticamente i nuovi endpoint, che hanno il formato `https://(account-id).ddb.(region).amazonaws.com`.

Se si utilizza una singola istanza di un client SDK per effettuare richieste a più account, l'applicazione avrà meno opportunità di riutilizzare le connessioni. AWS consiglia di modificare le applicazioni per connettersi a un minor numero di account per istanza del client SDK. Un'alternativa consiste

nell'impostare il client SDK in modo che continui a utilizzare gli endpoint Regionali utilizzando l'impostazione `ACCOUNT_ID_ENDPOINT_MODE`, come documentato nella Guida di riferimento per strumenti e SDK AWS.

Interfacce programmatiche che funzionano con DynamoDB

Ciascun SDK AWS fornisce una o più interfacce programmatiche per lavorare con Amazon DynamoDB. Queste interfacce spaziano da semplici wrapper DynamoDB di basso livello a livelli di persistenza basati sugli oggetti. Le interfacce specifiche disponibili dipendono dal linguaggio di programmazione e dall'SDK AWS in uso.



Nella sezione seguente sono descritte alcune delle interfacce disponibili con AWS SDK per Java come esempio Non tutte le interfacce sono disponibili in tutti gli SDK AWS.

Argomenti

- Interfacce di basso livello che funzionano con DynamoDB
- Interfacce documentali che funzionano con DynamoDB
- Interfacce di persistenza degli oggetti che funzionano con DynamoDB

Interfacce di basso livello che funzionano con DynamoDB

Ogni SDK AWS specifico del linguaggio fornisce un'interfaccia di basso livello per Amazon DynamoDB, con metodi che assomigliano molto alle richieste API DynamoDB di basso livello.

In alcuni casi, sarà necessario identificare i tipi di dati degli attributi utilizzando Descrittori del tipo di dati, ad esempio `S` per stringa o `N` per numero

> **Note**
>
> Un'interfaccia di basso livello è disponibile in ogni SDK AWS specifico per il linguaggio.

Il seguente programma Java utilizza l'interfaccia di basso livello di AWS SDK per Java.

Esempio di interfaccia di basso livello

```
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);
        }
    }
}
```

Interfacce documentali che funzionano con DynamoDB

Molti SDK AWS forniscono un'interfaccia documentale, che consente di eseguire operazioni del piano dati (creazione, lettura, aggiornamento, eliminazione) su tabelle e indici. Con un'interfaccia documentale, non è necessario specificare Descrittori del tipo di dati. I tipi di dati sono impliciti nella semantica dei dati stessi. Questi SDK AWS forniscono anche metodi per convertire facilmente documenti JSON da e verso tipi di dati nativi di Amazon DynamoDB.

> **Note**
>
> Le interfacce documentali sono disponibili negli SDK AWS per Java, .NET, Node.js e negli SDK per JavaScript.

Il seguente programma Java utilizza l'interfaccia documentale di AWS SDK per Java. Il programma crea un oggetto `Table` che rappresenta la tabella `Music` e quindi chiede all'oggetto di utilizzare `GetItem` per recuperare una canzone. Il programma restituisce quindi l'anno in cui è uscita la canzone.

La classe `software.amazon.dynamodb.document.DynamoDB` implementa l'interfaccia di documento di DynamoDB. Notare come `DynamoDB` funge da wrapper per il client di basso livello (`AmazonDynamoDB`).

Esempio di interfaccia documentale

```
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);

    }
}
```

Interfacce di persistenza degli oggetti che funzionano con DynamoDB

Alcuni SDK AWS forniscono un'interfaccia di persistenza degli oggetti in cui non si eseguono direttamente operazioni del piano dati. Invece, è possibile creare oggetti che rappresentano elementi

nelle tabelle e negli indici di Amazon DynamoDB e interagire solo con quegli oggetti. Ciò consente di scrivere codice incentrato sugli oggetti piuttosto che codice incentrato sul database.

> **Note**
>
> Le interfacce di persistenza degli oggetti sono disponibili negli SDK per Java e .NET di AWS. Per ulteriori informazioni, consulta Interfacce di programmazione di livello superiore per DynamoDB per Dynamo DB.

Esempio di interfaccia di persistenza degli oggetti

```
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();
    }
}
```

Interfacce di programmazione di livello superiore per DynamoDB

Gli SDK AWS forniscono applicazioni con interfacce di basso livello per lavorare con Amazon DynamoDB. Queste classi e metodi lato client corrispondono direttamente all'API DynamoDB di basso livello. Tuttavia, molti sviluppatori sperimentano un senso di disconnessione, o mancata corrispondenza dell'impedenza, quando è necessario mappare tipi di dati complessi agli elementi di una tabella di database. Con un'interfaccia di database di basso livello, gli sviluppatori devono scrivere metodi per leggere o scrivere i dati oggetto nelle tabelle di database e viceversa. La quantità di codice aggiuntivo richiesto per ogni combinazione di tipo di oggetto e tabella di database può sembrare travolgente.

Per semplificare lo sviluppo, gli SDK AWS per Java e .NET forniscono interfacce aggiuntive con livelli più elevati di astrazione. Le interfacce di livello superiore per DynamoDB consentono di definire le relazioni tra gli oggetti del programma e le tabelle di database che memorizzano i dati di tali oggetti. Dopo aver definito questa mappatura, vengono richiamati metodi oggetto semplici come `save`, `load` oppure`delete` e le operazioni di DynamoDB di basso livello sottostanti vengono richiamate automaticamente per conto dell'utente. Ciò consente di scrivere codice incentrato sugli oggetti piuttosto che codice incentrato sul database.

Le interfacce di programmazione di livello superiore per DynamoDB sono disponibili negli SDK per Java e .NET di AWS.

Java

- Java 1.x: DynamoDBMapper
- Java 2.x: Client avanzato DynamoDB

.NET

- Utilizzo del modello documentale .NET in DynamoDB
- Operazioni con il modello di persistenza degli oggetti di .NET e DynamoDB

Java 1.x: DynamoDBMapper

> **Note**
>
> L'SDK per Java è disponibile in due versioni: 1.x e 2.x. La fine del supporto per 1.x è stata annunciata il 12 gennaio 2024. La fine del supporto è prevista per il 31 dicembre 2025. Per sviluppare ex novo, si consiglia vivamente di utilizzare 2.x.

AWS SDK per Java fornisce una classe `DynamoDBMapper` che consente di mappare le classi lato client alle tabelle Amazon DynamoDB. Per utilizzare `DynamoDBMapper`, definire la relazione tra gli elementi di una tabella DynamoDB e le istanze di oggetto corrispondenti nel codice. La classe `DynamoDBMapper` consente di eseguire diverse operazioni di creazione, lettura, aggiornamento ed eliminazione (CRUD) sugli elementi e di eseguire query e scansioni sulle tabelle.

Argomenti

- Classe DynamoDBMapper
- Tipi di dati supportati per DynamoDBMapper per Java
- Annotazioni Java per DynamoDB
- Impostazioni di configurazione opzionali per DynamoDBMapper
- DynamoDB e blocco ottimistico con il numero di versione
- Mappatura dei dati arbitrari in DynamoDB
- Esempi della classe DynamoDBMapper

> **Note**
>
> La classe `DynamoDBMapper` non ti consente di creare, aggiornare o eliminare tabelle. Per eseguire queste attività, utilizza invece l'interfaccia SDK per Java di basso livello.

SDK per Java fornisce una serie di tipi di annotazione in modo che sia possibile mappare le classi alle tabelle. Ad esempio, considera una tabella `ProductCatalog` la cui chiave di partizione sia l'`Id`.

```
ProductCatalog(Id, ...)
```

Puoi mappare una classe nell'applicazione client alla tabella `ProductCatalog`, come mostrato nel seguente codice Java. Questo codice definisce un Plain Old Java Object (POJO), denominato `CatalogItem`, il quale utilizza annotazioni per mappare campi di oggetto ai nomi di attributo di 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; }
}
```

Nel codice precedente, l'annotazione `@DynamoDBTable` mappa la classe `CatalogItem` alla tabella `ProductCatalog`. Puoi archiviare istanze di classi individuali come item della tabella. Nella definizione della classe, l'annotazione `@DynamoDBHashKey` mappa la proprietà `Id` alla chiave primaria.

Per impostazione predefinita, le proprietà di classe si mappano agli stessi attributi dei nomi della tabella. Le proprietà `Title` e `ISBN` si mappano agli stessi attributi dei nomi della tabella.

L'annotazione `@DynamoDBAttribute` è facoltativa quando il nome dell'attributo DynamoDB corrisponde al nome della proprietà dichiarata nella classe. Quando questi sono differenti, utilizza questa annotazione con il parametro `attributeName` in modo da specificare a quale attributo DynamoDB corrisponde questa proprietà.

Nell'esempio precedente, l'annotazione `@DynamoDBAttribute` viene aggiunta a ogni proprietà per garantire che i nomi della proprietà corrispondano esattamente alle tabelle create in una fase precedente e che sia coerente con i nomi di attributo utilizzati in altri esempi di codice in questa guida.

La definizione della classe può avere proprietà che non si mappano ad alcun attributo della tabella. Identifica queste proprietà aggiungendo l'annotazione `@DynamoDBIgnore`. Nell'esempio precedente, la proprietà `SomeProp` viene contrassegnata con l'annotazione `@DynamoDBIgnore`. Quando carichi nella tabella l'istanza `CatalogItem`, l'istanza `DynamoDBMapper` non include la proprietà `SomeProp`. Inoltre, il mappatore non restituisce questo attributo quando recuperi un item dalla tabella.

Dopo aver definito la classe di mappatura, puoi utilizzare i metodi `DynamoDBMapper` per scrivere un'istanza di tale classe in un item corrispondente nella tabella `Catalog`. Il seguente esempio di codice dimostra questa tecnica.

```
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);
```

Il seguente esempio di codice mostra come recuperare l'item e accedere ad alcuni dei suoi attributi:

```
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` offre un modo intuitivo e naturale di lavorare con i dati di DynamoDB all'interno di Java. Fornisce inoltre diverse caratteristiche integrate, come il blocco ottimistico, transazioni ACID, valori di chiavi di partizione e di ordinamento generati automaticamente e la funzione Versioni multiple degli oggetti.

Classe DynamoDBMapper

La classe `DynamoDBMapper` è il punto di ingresso al database Amazon DynamoDB. Fornisce l'accesso a un endpoint DynamoDB e consente di accedere ai dati in diverse tabelle. Consente anche di eseguire diverse operazioni di creazione, lettura, aggiornamento ed eliminazione (CRUD) sugli elementi e di eseguire query e scansioni sulle tabelle. Questa classe fornisce i metodi seguenti per lavorare con DynamoDB.

Per la documentazione Javadoc corrispondente, consulta DynamoDBMapper nella Documentazione di riferimento delle API di AWS SDK per Java.

Argomenti

- save

- caricare
- Elimina
- query
- queryPage
- scan
- scanPage
- parallelScan
- batchSave
- batchLoad
- batchDelete
- BatchWrite
- transactionWrite
- transactionLoad
- count
- generateCreateTableRequest
- createS3Link
- getS3ClientCache

save

Salva l'oggetto specificato nella tabella. L'oggetto che intendi salvare è l'unico parametro obbligatorio di questo metodo. Puoi fornire parametri di configurazione opzionali utilizzando l'oggetto `DynamoDBMapperConfig`.

Se non esiste un item avente la stessa chiave primaria, questo metodo crea un nuovo item nella tabella. Se esiste un item avente la stessa chiave primaria, esso aggiorna l'item esistente. Se la chiave di partizione e la chiave di ordinamento sono del tipo String e sono annotate con `@DynamoDBAutoGeneratedKey`, viene loro assegnato un identificatore unico universale casuale (UUID) in caso di mancata inizializzazione. I campi di versione annotati con `@DynamoDBVersionAttribute` vengono incrementati di una unità. Inoltre, se viene aggiornato un campo di versione o viene generata una chiave, l'oggetto passato si aggiornerà a seguito dell'operazione.

Per impostazione predefinita, solo gli attributi corrispondenti alle proprietà della classe mappate vengono aggiornati. Glie eventuali attributi esistenti aggiuntivi su un item non sono interessati. Tuttavia, se specifichi `SaveBehavior.CLOBBER`, puoi forzare la sovrascrittura completa dell'item.

```
DynamoDBMapperConfig config = DynamoDBMapperConfig.builder()
    .withSaveBehavior(DynamoDBMapperConfig.SaveBehavior.CLOBBER).build();

mapper.save(item, config);
```

Se la funzione Versioni multiple è abilitata, le versioni dell'item lato client e lato server devono corrispondere. Tuttavia, non è necessario che la versione corrisponda se viene utilizzata l'opzione `SaveBehavior.CLOBBER`. Per ulteriori informazioni sulla funzione Controllo delle versioni, consulta DynamoDB e blocco ottimistico con il numero di versione.

caricare

Recupera un item da una tabella. È necessario fornire la chiave primaria dell'item che intendi recuperare. Puoi fornire parametri di configurazione opzionali utilizzando l'oggetto `DynamoDBMapperConfig`. Ad esempio, puoi richiedere facoltativamente letture fortemente consistenti per garantire che questo metodo recuperi solo i valori di item più recenti, come riportato nella seguente dichiarazione Java.

```
DynamoDBMapperConfig config = DynamoDBMapperConfig.builder()
    .withConsistentReads(DynamoDBMapperConfig.ConsistentReads.CONSISTENT).build();

CatalogItem item = mapper.load(CatalogItem.class, item.getId(), config);
```

Per impostazione predefinita, DynamoDB restituisce l'elemento che dispone di valori con consistenza finale. Per informazioni sul modello di consistenza finale di DynamoDB, consulta Coerenza di lettura di DynamoDB.

Elimina

Elimina un item dalla tabella. È necessario passare un'istanza di oggetto della classe mappata.

Se la funzione Versioni multiple è abilitata, le versioni dell'item lato client e lato server devono corrispondere. Tuttavia, non è necessario che la versione corrisponda se viene utilizzata l'opzione `SaveBehavior.CLOBBER`. Per ulteriori informazioni sulla funzione Controllo delle versioni, consulta DynamoDB e blocco ottimistico con il numero di versione.

query

Esegue una query su una tabella o un indice secondario.

Supponiamo che tu abbia una tabella, `Reply`, in cui vengono memorizzate le risposte dei thread di un forum. Ogni tema del thread può contenere 0 o più risposte. La chiave primaria della tabella `Reply` è costituita dai campi `Id` e `ReplyDateTime`, in cui l'`Id` è la chiave di partizione, mentre `ReplyDateTime` è la chiave di ordinamento della chiave primaria.

```
Reply ( Id, ReplyDateTime, ... )
```

Si supponga ora di aver creato una mappatura tra la classe `Reply` e la tabella `Reply` corrispondente in DynamoDB. Il seguente codice Java utilizza `DynamoDBMapper` per trovare tutte le risposte delle due settimane precedenti di uno specifico tema di un thread.

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 query restituisce una raccolta di oggetti `Reply`.

Per impostazione predefinita, il metodo `query` restituisce una raccolta a "caricamento differito". Inizialmente restituisce solo una pagina di risultati e in seguito effettua una chiamata di assistenza per

la pagina successiva, se necessario. Per ottenere tutti gli elementi corrispondenti, eseguire iterazioni sulla raccolta `latestReplies`.

Tieni presente che la chiamata del metodo `size()` sulla raccolta carica tutti i risultati al fine di fornire un conteggio accurato. Di conseguenza si potrebbe verificare un consumo eccessivo di throughput assegnato e nel caso di una tabella molto grande potrebbe persino esaurirsi tutta la memoria nella JVM.

Per eseguire query a un indice, è necessario in primo luogo modellarlo come classe di mappatore. Si supponga che la tabella `Reply` disponga di un indice secondario globale denominato PostedBy-Message-Index. La chiave di partizione di quest'indice è `PostedBy`, mentre la chiave di ordinamento è `Message`. La definizione di classe di un item dell'indice dovrebbe avere un aspetto simile al seguente:

```
@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.
}
```

L'annotazione `@DynamoDBTable` indica che questo indice è associato alla tabella `Reply`. L'annotazione `@DynamoDBIndexHashKey` denota la chiave di partizione (PostedBy) dell'indice, mentre `@DynamoDBIndexRangeKey` denota la chiave di ordinamento (Message) dell'indice.

Puoi ora utilizzare `DynamoDBMapper` per eseguire query all'indice, recuperando un sottoinsieme di messaggi pubblicati da un determinato utente. Non è necessario specificare il nome dell'indice se non sono presenti mappature in conflitto tra tabelle e indici e le mappature sono già state eseguite nel mappatore. Il mappatore deciderà in base alla chiave primaria e alla chiave di ordinamento.
Il codice seguente esegue una query su un indice secondario globale. Poiché gli indici secondari

globali supportano letture consistenti finali, ma non letture consistenti, è necessario specificare `withConsistentRead(false)`.

```
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 query restituisce una raccolta di oggetti `PostedByMessage`.

queryPage

Esegue query su una tabella o un indice secondario e restituisce una singola pagina di risultati corrispondenti. Come per il metodo `query`, è necessario specificare il valore di una chiave di partizione e un filtro query applicato all'attributo della chiave di ordinamento. Tuttavia, `queryPage` restituisce solo la prima "pagina" di dati, ovvero la quantità di dati che rientra in 1 MB.

scan

Esegue la scansione di un'intera tabella o di un indice secondario. Puoi specificare facoltativamente `FilterExpression` per filtrare il set di risultati.

Supponiamo che tu abbia una tabella, `Reply`, in cui vengono memorizzate le risposte dei thread di un forum. Ogni tema del thread può contenere 0 o più risposte. La chiave primaria della tabella `Reply` è costituita dai campi `Id` e `ReplyDateTime`, in cui l'`Id` è la chiave di partizione, mentre `ReplyDateTime` è la chiave di ordinamento della chiave primaria.

```
Reply ( Id, ReplyDateTime, ... )
```

Se hai mappato una classe Java alla tabella `Reply`, puoi utilizzare la `DynamoDBMapper` per eseguire la scansione della tabella. Ad esempio, il seguente codice Java esegue la scansione dell'intera tabella `Reply`, restituendo solo le risposte di un determinato anno.

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);
```

Per impostazione predefinita, il metodo `scan` restituisce una raccolta a "caricamento differito". Inizialmente restituisce solo una pagina di risultati e in seguito effettua una chiamata di assistenza per la pagina successiva, se necessario. Per ottenere tutti gli elementi corrispondenti, eseguire iterazioni sulla raccolta `replies`.

Tieni presente che la chiamata del metodo `size()` sulla raccolta carica tutti i risultati al fine di fornire un conteggio accurato. Di conseguenza si potrebbe verificare un consumo eccessivo di throughput assegnato e nel caso di una tabella molto grande potrebbe persino esaurirsi tutta la memoria nella JVM.

Per eseguire la scansione di un indice, è necessario in primo luogo modellarlo come classe di mappatore. Supponiamo che la tabella `Reply` abbia un indice secondario globale denominato `PostedBy-Message-Index`. La chiave di partizione di quest'indice è `PostedBy`, mentre la chiave di ordinamento è `Message`. Una classe di mappatore per questo indice viene mostrata nella sezione query. Utilizza le annotazioni `@DynamoDBIndexHashKey` e `@DynamoDBIndexRangeKey` per specificare la chiave di partizione e la chiave di ordinamento dell'indice.

L'esempio di codice seguente esegue la scansione di `PostedBy-Message-Index`. Esso non utilizza un filtro di scansione, così che ti vengano restituiti tutti gli elementi dell'indice.

```
DynamoDBScanExpression scanExpression = new DynamoDBScanExpression()
    .withIndexName("PostedBy-Message-Index")
    .withConsistentRead(false);

    List<PostedByMessage> iList =  mapper.scan(PostedByMessage.class, scanExpression);
    Iterator<PostedByMessage> indexItems = iList.iterator();
```

scanPage

Esegue una scansione su una tabella o un indice secondario e restituisce una singola pagina di risultati corrispondenti. Come per il metodo `scan`, puoi specificare facoltativamente `FilterExpression` per filtrare il set di risultati. Tuttavia, `scanPage` restituisce solo la prima "pagina" di dati, ovvero la quantità di dati che rientra in 1 MB.

parallelScan

Esegue una scansione parallela di un'intera tabella o di un indice secondario. Puoi specificare una serie di segmenti logici della tabella e un'espressione scan per filtrare i risultati. `parallelScan` divide l'attività di scansione tra più dipendenti, uno per ogni segmento logico; i dipendenti elaborano i dati in parallelo e restituiscono i risultati.

L'esempio di codice Java seguente esegue una scansione parallela sulla tabella `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

Salva gli oggetti in una o più tabelle tramite una o più chiamate al metodo `AmazonDynamoDB.batchWriteItem`. Questo metodo non fornisce garanzie sulle transazioni.

Il codice Java seguente salva due item (libri) nella tabella `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 più item da una o più tabelle utilizzando la loro chiave primaria.

Il seguente codice Java recupera due item da due tabelle differenti.

```
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 gli oggetti da una o più tabelle tramite una o più chiamate al metodo `AmazonDynamoDB.batchWriteItem`. Questo metodo non fornisce garanzie sulle transazioni.

Il seguente codice Java elimina due item (libri) dalla tabella `ProductCatalog`.

```
Book book1 = mapper.load(Book.class, 901);
Book book2 = mapper.load(Book.class, 902);
mapper.batchDelete(Arrays.asList(book1, book2));
```

BatchWrite

Salva gli oggetti o li elimina da una o più tabelle tramite una o più chiamate al metodo `AmazonDynamoDB.batchWriteItem`. Questo metodo non fornisce garanzie sulle transazioni né supporta la funzione Versioni multiple (inserimenti o eliminazioni condizionali).

Il seguente frammento di codice Java scrive un nuovo item nella tabella `Forum`, scrive un nuovo item nella tabella `Thread` ed elimina un item dalla tabella `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

Salva gli oggetti o li elimina da una o più tabelle tramite una o più chiamate al metodo `AmazonDynamoDB.transactWriteItems`.

Per un elenco di eccezioni specifiche delle transazioni, consulta la sezione relativa agli errori di TransactWriteItems.

Per ulteriori informazioni sulle transazioni DynamoDB e le garanzie ACID (Atomicity, Consistency, Isolation and Durability) fornite, consulta Amazon DynamoDB Transactions.

> **Note**
>
> Questo metodo non supporta i seguenti metodi:
>
> - DynamoDBMapperConfig.SaveBehavior.

Il seguente codice Java scrive un nuovo item in ogni tabella `Forum` e `Thread`, a livello di transazione.

```
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

Carica gli oggetti da una o più tabelle tramite una chiamata al metodo `AmazonDynamoDB.transactGetItems`.

Per un elenco di eccezioni specifiche delle transazioni, consulta la sezione relativa agli errori di TransactGetItems.

Per ulteriori informazioni sulle transazioni DynamoDB e le garanzie ACID (Atomicity, Consistency, Isolation and Durability) fornite, consulta Amazon DynamoDB Transactions.

Il seguente codice Java carica un elemento da ciascuna delle tabelle `Forum` e `Thread`, a livello di transazione.

```
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

Valuta l'espressione scan specificata e restituisce il numero di item corrispondenti. Non viene restituito alcun dato di item.

generateCreateTableRequest

Analizza una classe POJO che rappresenta una tabella DynamoDB e restituisce un `CreateTableRequest` per tale tabella.

createS3Link

Crea un collegamento a un oggetto in Amazon S3. È necessario specificare un nome del bucket e un nome della chiave, il quale identifica l'oggetto nel bucket in modo univoco.

Per utilizzare `createS3Link`, la classe di mappatore deve definire metodi getter e setter. Ciò viene illustrato nel seguente codice, nel quale si aggiunge alla classe `CatalogItem` un nuovo attributo e metodi getter/setter:

```
@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;
    }

...
}
```

Il seguente frammento di codice Java definisce un nuovo item da scrivere nella tabella `Product`. Questo elemento include un collegamento all'immagine del prodotto; i dati relativi all'immagine vengono caricati in 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 classe `S3Link` fornisce molti altri metodi per manipolare gli oggetti in Amazon S3. Per ulteriori informazioni, vedi la sezione relativa a Javadocs per `S3Link`.

getS3ClientCache

Restituisce `S3ClientCache` sottostante per accedere ad Amazon S3. Una `S3ClientCache` è una mappa intelligente per gli oggetti `AmazonS3Client`. Se si dispone di più client, un `S3ClientCache` può aiutare a tenere i client organizzati per regione AWS e creare nuovi client Amazon S3 on demand.

Tipi di dati supportati per DynamoDBMapper per Java

In questa sezione vengono descritti i tipi di dati Java primitivi, le raccolte e i tipi di dati arbitrari supportati in Amazon DynamoDB.

Amazon DynamoDB supporta i seguenti tipi di dati Java primitivi e classi wrapper primitive.

- `String`
- `Boolean`, `boolean`
- `Byte`, `byte`
- `Date` (come stringa con precisione pari al millisecondo ISO_8601, convertita in UTC)
- `Calendar` (come stringa con precisione pari al millisecondo ISO_8601, convertita in UTC)
- `Long`, `long`
- `Integer`, `int`
- `Double`, `double`
- `Float`, `float`
- `BigDecimal`

- `BigInteger`

> **Note**
>
> - Per ulteriori informazioni sulle regole di denominazione di DynamoDB e i vari tipi di dati supportati, consulta Tipi di dati e regole di denominazione supportati in Amazon DynamoDB.
> - I valori Binary vuoti sono supportati da DynamoDBMapper.
> - I valori String vuoti sono supportati da AWS SDK for Java 2.x.
>
> In AWS SDK per Java 1.x, DynamoDBMapper supporta la lettura dei valori degli attributi String vuoti, tuttavia, non scriverà valori degli attributi String vuoti perché questi attributi vengono eliminati dalla richiesta.

DynamoDB supporta i tipi di raccolta Java Set, List e Map. La tabella seguente riassume il modo in cui questi tipi Java vengono mappati ai tipi DynamoDB.

Tipo di Java	Tipo DynamoDB
Tutti i tipi di numeri	`N` (tipo numero)
Stringhe	`S` (tipo stringa)
Booleano	`BOOL` (Tipo booleano), 0 o 1.
ByteBuffer	`B` (tipo binario)
Data	`S` (tipo stringa). I valori Date vengono archiviati come stringhe in formato ISO-8601.
Tipi di raccolta Set	`SS` tipo (set di stringhe), `NS` tipo (set di numeri) o `BS` tipo (set binario).

L'interfaccia `DynamoDBTypeConverter` consente di mappare i propri tipi di dati arbitrari a un tipo di dati supportato in nativo da DynamoDB. Per ulteriori informazioni, consulta Mappatura dei dati arbitrari in DynamoDB.

Annotazioni Java per DynamoDB

In questa sezione vengono descritte le annotazioni disponibili per la mappatura delle classi e delle proprietà alle tabelle e agli attributi in Amazon DynamoDB.

Per la documentazione Javadoc corrispondente, consulta Riepilogo dei tipi di annotazioni nella Documentazione di riferimento delle API di AWS SDK per Java.

> **Note**
>
> Nelle seguenti annotazioni, sono obbligatori solo `DynamoDBTable` e `DynamoDBHashKey`.

Argomenti

- DynamoDBAttribute
- DynamoDBAutoGeneratedKey
- DynamoDBAutoGeneratedTimestamp
- DynamoDBDocument
- DynamoDBHashKey
- DynamoDBIgnore
- DynamoDBIndexHashKey
- DynamoDBIndexRangeKey
- DynamoDBRangeKey
- DynamoDBTable
- DynamoDBTypeConverted
- DynamoDBTyped
- DynamoDBVersionAttribute

DynamoDBAttribute

Mappa una proprietà all'attributo della tabella. Per impostazione predefinita, ogni proprietà di classe si mappa a un attributo item con lo stesso nome. Tuttavia, se i nomi non sono gli stessi, puoi utilizzare questa annotazione per mappare una proprietà all'attributo. Nel frammento Java seguente, `DynamoDBAttribute` mappa la proprietà `BookAuthors` al nome di attributo `Authors` della tabella.

```
@DynamoDBAttribute(attributeName = "Authors")
public List<String> getBookAuthors() { return BookAuthors; }
public void setBookAuthors(List<String> BookAuthors) { this.BookAuthors =
 BookAuthors; }
```

`DynamoDBMapper` utilizza `Authors` come nome di attributo quando salva l'oggetto nella tabella.

DynamoDBAutoGeneratedKey

Contrassegna una proprietà di chiave di partizione o di ordinamento come generata automaticamente. `DynamoDBMapper` genera un UUID casuale durante il salvataggio di questi attributi. Solo le proprietà String possono essere contrassegnate come chiavi generate automaticamente.

L'esempio seguente illustra l'utilizzo delle chiavi generate automaticamente.

```
@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 automaticamente un timestamp.

```
@DynamoDBAutoGeneratedTimestamp(strategy=DynamoDBAutoGenerateStrategy.ALWAYS)
public Date getLastUpdatedDate() { return lastUpdatedDate; }
public void setLastUpdatedDate(Date lastUpdatedDate) { this.lastUpdatedDate =
 lastUpdatedDate; }
```

Facoltativamente, la strategia di generazione automatica può essere definita fornendo un attributo di strategia. Il valore predefinito è `ALWAYS`.

DynamoDBDocument

Indica che una classe può essere serializzata come un documento di Amazon DynamoDB.

Ad esempio, si supponga di voler mappare un documento JSON a un attributo DynamoDB di tipo Map (`M`). Il seguente esempio di codice definisce un item contenente un attributo nidificato (Pictures) di 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; }

    }
}
```

Puoi quindi salvare un nuovo item `ProductCatalog`, con `Pictures`, come mostrato nell'esempio seguente.

```
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);
```

L'aspetto dell'item `ProductCatalog` risultante dovrebbe essere simile al seguente (in 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

Mappa una proprietà di classe alla chiave di partizione della tabella. La proprietà deve essere uno dei tipi di stringa, numero o binario scalari. La proprietà non può essere un tipo di raccolta.

Supponi di avere una tabella, `ProductCatalog`, che abbia come chiave primaria l'`Id`. Il seguente codice Java definisce una classe `CatalogItem` e mappa la sua proprietà `Id` alla chiave primaria della tabella `ProductCatalog` utilizzando il tag `@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 all'istanza `DynamoDBMapper` che la proprietà associata deve essere ignorata. Quando salvi i dati nella tabella, `DynamoDBMapper` non salva questa proprietà nella tabella.

Applicato al metodo getter o al campo della classe per una proprietà non modellata. Se l'annotazione viene applicata direttamente al campo della classe, il getter e il setter corrispondenti devono essere dichiarati nella stessa classe.

DynamoDBIndexHashKey

Mappa una proprietà di classe alla chiave di partizione di un indice secondario globale. La proprietà deve essere uno dei tipi di stringa, numero o binario scalari. La proprietà non può essere un tipo di raccolta.

Utilizza questa annotazione se è necessario eseguire una `Query` su un indice secondario globale. È necessario specificare il nome dell'indice (`globalSecondaryIndexName`). Se il nome della proprietà di classe è diverso dalla chiave di partizione dell'indice, è necessario inoltre specificare il nome di tale attributo dell'indice (`attributeName`).

DynamoDBIndexRangeKey

Associa una proprietà di classe alla chiave di ordinamento di un indice secondario globale o di un indice secondario locale. La proprietà deve essere uno dei tipi di stringa, numero o binario scalari. La proprietà non può essere un tipo di raccolta.

Utilizza questa annotazione se è necessario eseguire una `Query` su un indice secondario locale o globale e si desidera rifinire i risultati utilizzando la chiave di ordinamento dell'indice. È necessario specificare il nome dell'indice (`globalSecondaryIndexName`, oppure `localSecondaryIndexName`). Se il nome della proprietà di classe è diverso dalla chiave di ordinamento dell'indice, è necessario inoltre specificare il nome di tale attributo dell'indice (`attributeName`).

DynamoDBRangeKey

Mappa una proprietà di classe alla chiave di ordinamento della tabella. La proprietà deve essere uno dei tipi di stringa, numero o binario scalari. Non può essere un tipo di raccolta.

Se la chiave primaria è composita (chiave di partizione e chiave di ordinamento), puoi utilizzare questo tag per mappare il campo della classe alla chiave di ordinamento. Ad esempio, supponiamo che tu abbia una tabella `Reply` in cui vengono memorizzate le risposte dei thread di un forum. Ogni thread può contenere molte risposte. Pertanto, la chiave primaria di questa tabella è sia `ThreadId` sia `ReplyDateTime`. La chiave di partizione è `ThreadId` e la chiave di ordinamento è `ReplyDateTime`.

Il seguente codice Java definisce una classe `Reply` e la mappa alla tabella `Reply`. Esso utilizza sia il tag `@DynamoDBHashKey` che `@DynamoDBRangeKey` per identificare le proprietà di classe che si mappano alla chiave primaria.

```
@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 tabella di destinazione di DynamoDB. Ad esempio, il seguente codice Java definisce una classe `Developer` e la mappa alla tabella `People` in DynamoDB.

```
@DynamoDBTable(tableName="People")
public class Developer { ...}
```

L'annotazione `@DynamoDBTable` può essere ereditata. Tutte le nuove classi che ereditano dalla classe `Developer` vengono pure mappate alla tabella `People`. Ad esempio, supponiamo che tu abbia creato una classe `Lead` che erediti dalla classe `Developer`. Poiché hai mappato la classe `Developer` alla tabella `People`, anche gli oggetti della classe `Lead` vengono memorizzati nella stessa tabella.

`@DynamoDBTable` può anche essere sovrascritto. Tutte le nuove classi che ereditano dalla classe `Developer` vengono mappate alla stessa tabella `People` per impostazione predefinita. Tuttavia, puoi sovrascrivere questa mappatura predefinita. Ad esempio, se crei una classe che eredita dalla classe `Developer`, puoi mapparla esplicitamente a un'altra tabella aggiungendo l'annotazione `@DynamoDBTable`, come riportato nel seguente esempio di codice Java.

```
@DynamoDBTable(tableName="Managers")
public class Manager extends Developer { ...}
```

DynamoDBTypeConverted

Annotazione per contrassegnare che una proprietà utilizza un convertitore di tipi personalizzato. Può essere annotata su un'annotazione definita dall'utente per passare proprietà aggiuntive a `DynamoDBTypeConverter`.

L'interfaccia `DynamoDBTypeConverter` consente di mappare i propri tipi di dati arbitrari a un tipo di dati supportato in nativo da DynamoDB. Per ulteriori informazioni, consulta Mappatura dei dati arbitrari in DynamoDB.

DynamoDBTyped

Annotazione per sovrascrivere il vincolo del tipo di attributo standard. I tipi standard non richiedono l'annotazione se viene loro applicato il vincolo di attributo predefinito per quel tipo.

DynamoDBVersionAttribute

Identifica una proprietà di classe per archiviare un numero di versione di blocco ottimistico. `DynamoDBMapper` assegna un numero di versione a detta proprietà quando salva un nuovo item e lo incrementa ogni volta che lo aggiorni. Sono supportati soltanto i tipi scalari di numero. Per ulteriori informazioni sui tipi di dati, consulta Tipi di dati. Per ulteriori informazioni sulla funzione Controllo delle versioni, consulta DynamoDB e blocco ottimistico con il numero di versione.

Impostazioni di configurazione opzionali per DynamoDBMapper

Quando crei un'istanza di `DynamoDBMapper`, esso contiene determinati comportamenti predefiniti; puoi sovrascrivere tali impostazioni predefinite tramite la classe `DynamoDBMapperConfig`.

Nel seguente frammento di codice viene creato un `DynamoDBMapper` con impostazioni personalizzate:

```
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);
```

Per ulteriori informazioni, consulta DynamoDBMapperConfig nella Documentazione di riferimento delle API di AWS SDK per Java.

Puoi utilizzare i seguenti argomenti per un'istanza di `DynamoDBMapperConfig`:

- Un valore di enumerazione `DynamoDBMapperConfig.ConsistentReads`:
  - `EVENTUAL`: l'istanza del mappatore utilizza una richiesta di lettura a consistenza finale.
  - `CONSISTENT`: l'istanza del mappatore utilizza una richiesta di lettura fortemente consistente. Puoi utilizzare questa impostazione opzionale con le operazioni `load`, `query` o `scan`. Le letture

fortemente consistenti hanno implicazioni per le prestazioni e la fatturazione; consulta la pagina dei dettagli del prodotto di DynamoDB per ulteriori informazioni.

Se non specifichi un'impostazione di consistenza di lettura per l'istanza di mappatore, il valore predefinito sarà `EVENTUAL`.

> **Note**
>
> Questo valore viene applicato solo nelle operazioni `query`, `querypage`, `load` e `batch load` di DynamoDBMapper.

- Un valore di enumerazione `DynamoDBMapperConfig.PaginationLoadingStrategy`: controlla il modo in cui l'istanza del mappatore elabora un elenco impaginato di dati, ad esempio i risultati di una `query` o una `scan`:
  - `LAZY_LOADING`: l'istanza del mappatore carica i dati quando possibile e mantiene tutti i risultati caricati in memoria.
  - `EAGER_LOADING`: l'istanza del mappatore carica i dati non appena l'elenco viene inizializzato.
  - `ITERATION_ONLY`: è possibile utilizzare soltantoo un Iterator per leggere dall'elenco. Durante l'iterazione, l'elenco cancella tutti i risultati precedenti prima di caricare la pagina successiva, in modo che memorizzi al massimo una pagina di risultati caricati. Ciò significa, inoltre, che l'elenco può essere iterato una sola volta. Si consiglia di utilizzare questa strategia quando si gestiscono item di grandi dimensioni, al fine di ridurre il sovraccarico della memoria.

  Se non specifichi una strategia di caricamento della paginazione per l'istanza di mappatore, il valore predefinito sarà `LAZY_LOADING`.
- Un valore di enumerazione `DynamoDBMapperConfig.SaveBehavior`. Specifica il modo in cui l'istanza di mappatore deve gestire gli attributi durante le operazioni di salvataggio.
  - `UPDATE`: durante un'operazione di salvataggio, tutti gli attributi modellati vengono aggiornati e gli attributi non modellati non vengono influenzati. Tipi di numeri primitivi (byte, int, long) sono impostati su 0. I tipi di oggetto sono impostati su null.
  - `CLOBBER`: cancella e sostituisce tutti gli attributi, inclusi quelli non modellati, durante un'operazione di salvataggio. Questa operazione viene eseguita eliminando la voce e creandola nuovamente. Vengono ignorate anche le limitazioni dei campi con versione.

  Se non specifichi il comportamento di salvataggio per l'istanza di mappatore, il valore predefinito sarà `UPDATE`.

> **Note**
>
> Le operazioni transazionali DynamoDBMapper non supportano l'enumerazione `DynamoDBMapperConfig.SaveBehavior`.

- Un oggetto `DynamoDBMapperConfig.TableNameOverride`: indica all'istanza del mappatore di ignorare il nome della tabella specificato dall'annotazione `DynamoDBTable` della classe e utilizza invece un nome di tabella diverso fornito dall'utente. Ciò può essere utile quando esegui il partizionamento dei dati in più tabelle in fase di runtime.

Puoi sovrascrivere l'oggetto della configurazione predefinita per `DynamoDBMapper` in ogni operazione, a seconda delle necessità.

DynamoDB e blocco ottimistico con il numero di versione

Il blocco ottimistico è una strategia che assicura che l'elemento lato client che si sta aggiornando (o eliminando) sia lo stesso elemento in Amazon DynamoDB. Se utilizzi questa strategia le scritture del tuo database sono protette da eventuali sovrascritture da parte di altre scritture di terzi e viceversa.

Con il blocco ottimistico, ogni voce dispone di un attributo che funge da numero di versione. Se recuperi un item da una tabella, l'applicazione registra il numero di versione di tale item. Puoi aggiornare l'item, ma solo se il numero di versione lato server non è cambiato. In caso di mancata corrispondenza delle versioni, qualcun altro ha modificato l'item prima dell'utente. Il tentativo di aggiornamento non riesce, perché si dispone di una versione obsoleta dell'item. Se ciò accade, semplicemente riprova a recuperare l'elemento e tenta di aggiornarlo. Il blocco ottimistico evita di sovrascrivere accidentalmente le modifiche che sono state apportate da altri. Impedisce anche ad altri di sovrascrivere accidentalmente le modifiche.

Sebbene sia possibile implementare la propria strategia di blocco ottimistico, AWS SDK per Java fornisce l'annotazione `@DynamoDBVersionAttribute`. Nella classe di mappatura della tabella, puoi indicare una proprietà nella quale memorizzare il numero di versione e contrassegnarlo utilizzando questa annotazione. Quando si salva un oggetto, l'elemento corrispondente nella tabella DynamoDB conterrà un attributo che memorizza il numero di versione. `DynamoDBMapper` assegna un numero di versione quando salvi l'oggetto per la prima volta e incrementa automaticamente il numero di versione ogni volta che aggiorni l'item. Le richieste di aggiornamento o eliminazione avranno esito positivo solo se la versione dell'oggetto lato client corrisponde al numero di versione dell'elemento corrispondente nella tabella DynamoDB.

`ConditionalCheckFailedException` viene generato se:

- utilizzi il blocco ottimistico con `@DynamoDBVersionAttribute` e il valore della versione del server è diverso dal valore sul lato client;
- specifichi le tue limitazioni condizionali quando salvi i dati utilizzando `DynamoDBMapper` con `DynamoDBSaveExpression` e tali limitazioni non sono riuscite.

> **Note**
>
> - Le tabelle globali DynamoDB utilizzano una riconciliazione di tipo "last writer wins" per aggiornamenti contestuali. Se utilizzi le tabelle globali, vince la policy "last writer". Pertanto in questo caso la strategia di blocco non funziona come previsto.
> - Le operazioni di scrittura transazionali `DynamoDBMapper` non supportano l'annotazione e le espressioni di condizione `@DynamoDBVersionAttribute` all'interno dello stesso oggetto. Se un oggetto all'interno di una scrittura transazionale viene annotato con `@DynamoDBVersionAttribute` e ha anche un'espressione di condizione, viene generata un'eccezione SdkClientException.

Ad esempio, nel seguente codice Java si definisce una classe `CatalogItem` che possiede diverse proprietà. La proprietà `Version` viene etichettata con l'annotazione `@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;}
}
```

Puoi applicare l'annotazione `@DynamoDBVersionAttribute` ai tipi nullable disponibili nelle classi wrapper primitive che forniscono un tipo nullable, come `Long` e `Integer`.

Il blocco ottimistico ha il seguente impatto su questi metodi `DynamoDBMapper`:

- `save`: per un nuovo elemento, il `DynamoDBMapper` assegna il numero di versione iniziale uguale a 1. Se recuperi un item, aggiorni una o più delle sue proprietà e tenti di salvare le modifiche, l'operazione di salvataggio ha esito positivo solo se il numero di versione sul lato client e sul lato server corrispondono. `DynamoDBMapper` incrementa il numero di versione in modo automatico.
- `delete`: il metodo `delete` utilizza un oggetto come parametro e `DynamoDBMapper` esegue un controllo della versione prima di eliminare l'elemento. La verifica della versione può essere disabilitata se nella richiesta viene specificato `DynamoDBMapperConfig.SaveBehavior.CLOBBER`.

  L'implementazione interna del blocco ottimistico in `DynamoDBMapper` utilizza il supporto degli aggiornamenti e delle eliminazioni condizionali fornite da DynamoDB.
- `transactionWrite` —

- `Put`: per un nuovo elemento, il `DynamoDBMapper` assegna il numero di versione iniziale uguale a 1. Se recuperi un item, aggiorni una o più delle sue proprietà e tenti di salvare le modifiche, l'operazione put ha esito positivo solo se il numero di versione sul lato client e sul lato server corrispondono. `DynamoDBMapper` incrementa il numero di versione in modo automatico.
- `Update`: per un nuovo elemento, il `DynamoDBMapper` assegna il numero di versione iniziale uguale a 1. Se recuperi un item, aggiorni una o più delle sue proprietà e tenti di salvare le modifiche, l'operazione update ha esito positivo solo se il numero di versione sul lato client e sul lato server corrispondono. `DynamoDBMapper` incrementa il numero di versione in modo automatico.
- `Delete`: `DynamoDBMapper` esegue un controllo della versione prima di eliminare l'elemento. L'operazione delete riesce solo se il numero di versione sul lato client e sul lato server corrispondono.
- `ConditionCheck`: l'annotazione `@DynamoDBVersionAttribute` non è supportata per le operazioni `ConditionCheck`. Quando un item `ConditionCheck` viene annotato con `@DynamoDBVersionAttribute` viene generata un'eccezione SdkClientException.

Disabilitazione del blocco ottimistico

Per disabilitare il blocco ottimistico, puoi modificare il valore di enumerazione `DynamoDBMapperConfig.SaveBehavior` da `UPDATE` in `CLOBBER`. Puoi farlo creando un'istanza `DynamoDBMapperConfig` che non esegua la verifica della versione e utilizzi tale istanza per tutte le richieste. Per informazioni sui parametri `DynamoDBMapperConfig.SaveBehavior` e altri parametri opzionali `DynamoDBMapper`, consulta Impostazioni di configurazione opzionali per DynamoDBMapper .

Inoltre puoi impostare il comportamento di blocco per una sola operazione specifica. Ad esempio, nel seguente frammento di codice Java si utilizza `DynamoDBMapper` per salvare un item del catalogo. Specifica `DynamoDBMapperConfig.SaveBehavior` aggiungendo il parametro opzionale `DynamoDBMapperConfig` al metodo `save`.

> **Note**
>
> Il metodo transactionWrite non supporta la configurazione DynamoDBMapperConfig.SaveBehavior. Disabilitazione del blocco ottimistico per transactionWrite non supportata.

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));
```

Mappatura dei dati arbitrari in DynamoDB

Oltre ai tipi Java supportati (vedere Tipi di dati supportati per DynamoDBMapper per Java), è possibile utilizzare i tipi dell'applicazione per i quali non vi è una mappatura diretta ai tipi di Amazon DynamoDB. Per mappare questi tipi, è necessario fornire un'implementazione che converta il tipo complesso in un tipo supportato DynamoDB e viceversa; è inoltre necessario annotare il metodo di accesso del tipo complesso utilizzando l'annotazione `@DynamoDBTypeConverted`. Il codice del convertitore trasforma i dati quando gli oggetti vengono salvati o caricati. Viene utilizzato anche per tutte le operazioni che utilizzano tipi complessi. Quando si confrontano i dati durante le operazioni di query e di scansione, i confronti vengono fatti sui dati archiviati in DynamoDB.

Ad esempio, considera la seguente classe `CatalogItem`, che definisce una proprietà, `Dimension`, che è `DimensionType`. Questa proprietà archivia le dimensioni dell'item, quali altezza, larghezza e spessore. Supponiamo che si decida di memorizzare queste dimensioni dell'elemento come una stringa (ad esempio 8,5x11x0,5) in DynamoDB. Nell'esempio seguente viene fornito il codice del convertitore, il quale converte l'oggetto `DimensionType` in una stringa e una stringa in `DimensionType`.

> **Note**
>
> In questo esempio di codice si presuppone che siano già stati caricati dati in DynamoDB per l'account seguendo le istruzioni riportate nella sezione Creazione di tabelle e caricamento di dati per esempi di codice in DynamoDB.
> Per le istruzioni dettagliate per eseguire il seguente esempio, consultare Esempi di codice 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;
        }
    }
```

```
}
```

Esempi della classe DynamoDBMapper

L'SDK AWS per Java fornisce una classe `DynamoDBMapper` che consente di mappare le classi lato client alle tabelle DynamoDB. Per utilizzare `DynamoDBMapper`, definire la relazione tra gli elementi di una tabella DynamoDB e le istanze di oggetto corrispondenti nel codice. La classe `DynamoDBMapper` consente di eseguire diverse operazioni di creazione, lettura, aggiornamento ed eliminazione (CRUD) sugli elementi e di eseguire query e scansioni sulle tabelle.

Per ulteriori informazioni sull'utilizzo di `DynamoDBMapper`, consulta DynamoDB Examples Using the AWS SDK for Java nella Guida per gli sviluppatori di SDK AWS per Java 1.x.

Java 2.x: Client avanzato DynamoDB

Il client avanzato DynamoDB è una libreria di alto livello che fa parte di AWS SDK per Java versione 2 (v2). Offre un modo semplice per mappare le classi lato client alle tabelle DynamoDB. È possibile definire le relazioni tra le tabelle e le relative classi di modello corrispondenti nel codice. Dopo aver definito queste relazioni, è possibile eseguire in modo intuitivo varie operazioni di creazione, lettura, aggiornamento o eliminazione (CRUD) su tabelle o elementi in DynamoDB.

Per ulteriori informazioni su come utilizzare il client avanzato con DynamoDB, consulta l'argomento relativo all'utilizzo del client avanzato DynamoDB in AWS SDK per Java 2.x .

Utilizzo del modello documentale .NET in DynamoDB

AWS SDK per .NET fornisce classi di modelli di documenti che includono alcune operazioni di Amazon DynamoDB di basso livello che semplificano ulteriormente la codifica. Nel modello di documento, le classi primarie sono `Table` e `Document`. La classe `Table` fornisce metodi di operazioni di dati come `PutItem`, `GetItem` e `DeleteItem`. Essa inoltre fornisce i metodi `Query` e `Scan`. La classe `Document` rappresenta un singolo item in una tabella.

Le classi precedenti di modello del documento sono disponibili nello spazio dei nomi `Amazon.DynamoDBv2.DocumentModel`.

> **Note**
>
> Non puoi utilizzare le classi del modello di documento per creare, aggiornare ed eliminare tabelle. Tuttavia, il modello di documento supporta la maggior parte delle operazioni sui dati comuni.

Argomenti

- Tipi di dati supportati

Tipi di dati supportati

Il modello di documento supporta un insieme di tipi di dati .NET primitivi e tipi di raccolte dati. Il modello attualmente supporta i seguenti tipi di dati primitivi:

- `bool`
- `byte`
- `char`
- `DateTime`
- `decimal`
- `double`
- `float`
- `Guid`
- `Int16`
- `Int32`
- `Int64`
- `SByte`
- `string`
- `UInt16`
- `UInt32`
- `UInt64`

Nella tabella seguente viene riepilogata la mappatura dei tipi .NET precedenti ai tipi DynamoDB.

Tipo primitivo .NET	Tipo DynamoDB
Tutti i tipi di numeri	`N` (tipo numero)
Tutti i tipi stringa	`S` (tipo stringa)

Tipo primitivo .NET	Tipo DynamoDB
MemoryStream, byte[]	`B` (tipo binario)
bool	`N` (tipo di numero). 0 rappresenta false e 1 rappresenta true.
DateTime	`S` (tipo stringa). I valori `DateTime` vengono archiviati come stringhe in formato ISO-8601.
Guid	`S` (tipo stringa).
Tipi di raccolta (List, HashSet e array)	Tipo `BS` (set binario), tipo `SS` (set stringa) o tipo `NS` (set numerico)

AWS SDK per .NET definisce i tipi per la mappatura dei tipi booleani, null, list e map di DynamoDB all'API del modello di documento di .NET:

- Utilizza `DynamoDBBool` per il tipo Boolean.
- Utilizza `DynamoDBNull` per il tipo null.
- Utilizza `DynamoDBList` per il tipo list.
- Utilizza `Document` per il tipo map.

> **Note**
>
> - Sono supportati valori Binary vuoti.
> - È supportata la lettura dei valori String vuoti. I valori degli attributi String vuoti sono supportati all'interno dei valori degli attributi di stringa del tipo Set durante la scrittura su DynamoDB. I valori degli attributi String vuoti del tipo String e i valori String vuoti contenuti nel tipo List o Map vengono eliminati dalle richieste di scrittura

Operazioni con il modello di persistenza degli oggetti di .NET e DynamoDB

AWS SDK per .NET Fornisce un modello di persistenza degli oggetti che consente di mappare le classi lato client alle tabelle Amazon DynamoDB. Ogni istanza dell'oggetto viene quindi mappata a

un elemento nelle tabelle corrispondenti. Per salvare gli oggetti lato client nelle tabelle, il modello di persistenza degli oggetti fornisce la classe `DynamoDBContext`, un punto di ingresso a DynamoDB. Questa classe fornisce una connessione a DynamoDB e consente di accedere alle tabelle, eseguire varie operazioni CRUD ed eseguire query.

Il modello di persistenza degli oggetti fornisce un set di attributi per mappare le classi lato client alle tabelle e agli attributi delle tabelle. properties/fields

> **Note**
>
> Il modello di persistenza degli oggetti non fornisce un'API per creare, aggiornare o eliminare tabelle. Esso fornisce solo operazioni di dati. È possibile utilizzare solo l'API di AWS SDK per .NET basso livello per creare, aggiornare ed eliminare tabelle.

L'esempio seguente mostra come funziona il modello di persistenza degli oggetti. Inizia con tabella `ProductCatalog`. La tabella dispone di `Id` come chiave primaria.

```
ProductCatalog(Id, ...)
```

Si supponga di avere una classe `Book` con le proprietà `Title`, `ISBN` e `Authors`. È possibile mappare la classe `Book` alla tabella `ProductCatalog`aggiungendo gli attributi definiti dal modello di persistenza degli oggetti, come mostrato nel seguente esempio di codice 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; }
```

```
}
```

Nell'esempio precedente, l'attributo `DynamoDBTable` associa la classe `Book` alla tabella `ProductCatalog`.

Il modello di persistenza degli oggetti supporta sia la mappatura esplicita che quella predefinita tra le proprietà della classe e gli attributi della tabella.

- Mappatura esplicita: per mappare una proprietà a una chiave primaria, è necessario utilizzare gli attributi del modello di persistenza dell'oggetto `DynamoDBHashKey` e `DynamoDBRangeKey`. Inoltre, per gli attributi della chiave non primaria, se il nome di una proprietà nella classe e l'attributo tabella corrispondente a cui si desidera mapparlo non sono gli stessi, è necessario definire la mappatura aggiungendo esplicitamente l'attributo `DynamoDBProperty`.

  Nell'esempio precedente, la proprietà `Id` viene mappata alla chiave primaria con lo stesso nome e la proprietà `BookAuthors` viene mappata all'attributo `Authors` nella tabella `ProductCatalog`.
- Mappatura predefinita: per impostazione predefinita, il modello di persistenza degli oggetti mappa le proprietà della classe agli attributi con lo stesso nome nella tabella.

  Nell'esempio precedente, le proprietà `Title` e `ISBN` vengono mappate agli attributi con lo stesso nome nella tabella `ProductCatalog`.

Non è necessario mappare ogni singola proprietà della classe. Queste proprietà vengono identificate aggiungendo l'attributo `DynamoDBIgnore`. Quando si salva un'istanza `Book` nella tabella, `DynamoDBContext` non include la proprietà `CoverPage`. Inoltre, non restituisce questa proprietà quando si recupera l'istanza del libro.

È possibile mappare le proprietà di tipi primitivi .NET come int e string. È inoltre possibile mappare qualsiasi tipo di dati arbitrari purché si fornisca un convertitore appropriato per mappare i dati arbitrari a uno dei tipi DynamoDB. Per informazioni sulla mappatura di tipi arbitrari, consulta Mappatura di dati arbitrari con DynamoDB utilizzando il modello di persistenza degli oggetti AWS SDK per .NET.

Il modello di persistenza degli oggetti supporta il blocco ottimistico. Durante un'operazione di aggiornamento, questo garantisce di disporre della copia più recente dell'elemento che si sta per aggiornare. Per ulteriori informazioni, consulta Blocco ottimistico che utilizza DynamoDB e il modello di persistenza degli oggetti di AWS SDK per .NET.

Per ulteriori informazioni, consulta gli argomenti riportati di seguito.

Argomenti

- Tipi di dati supportati
- Attributi DynamoDB del modello di persistenza degli oggetti di .NET
- DBContext Classe Dynamo dal modello di persistenza degli oggetti.NET
- Blocco ottimistico che utilizza DynamoDB e il modello di persistenza degli oggetti di AWS SDK per .NET
- Mappatura di dati arbitrari con DynamoDB utilizzando il modello di persistenza degli oggetti AWS SDK per .NET

Tipi di dati supportati

Il modello di persistenza degli oggetti supporta un insieme di tipi di dati .NET primitivi e tipi di dati arbitrari. Il modello supporta i seguenti tipi di dati primitivi:

- `bool`
- `byte`
- `char`
- `DateTime`
- `decimal`
- `double`
- `float`
- `Int16`
- `Int32`
- `Int64`
- `SByte`
- `string`
- `UInt16`
- `UInt32`
- `UInt64`

Il modello di persistenza degli oggetti supporta anche i tipi di raccolte.NET. `DynamoDBContextè` in grado di convertire tipi di raccolta concreti e semplici oggetti Plain Old CLR ()POCOs.

Nella tabella seguente viene riepilogata la mappatura dei tipi .NET precedenti ai tipi DynamoDB.

Tipo primitivo .NET	Tipo DynamoDB
Tutti i tipi di numeri	`N` (tipo numero)
Tutti i tipi stringa	`S` (tipo stringa)
MemoryStream, byte []	`B` (tipo binario)
bool	`N` (tipo di numero). 0 rappresenta false e 1 rappresenta true.
Tipi di raccolta	Tipo `BS` (set binario), tipo `SS` (set stringa) o tipo `NS` (set numerico)
DateTime	`S` (tipo stringa). I valori `DateTime` vengono archiviati come stringhe in formato ISO-8601.

Il modello di persistenza degli oggetti supporta anche tipi di dati arbitrari. Tuttavia, è necessario fornire il codice del convertitore per mappare i tipi complessi ai tipi DynamoDB.

> **Note**
>
> - Sono supportati valori Binary vuoti.
> - È supportata la lettura dei valori String vuoti. I valori degli attributi String vuoti sono supportati all'interno dei valori degli attributi di stringa del tipo Set durante la scrittura su DynamoDB. I valori degli attributi String vuoti del tipo String e i valori String vuoti contenuti nel tipo List o Map vengono eliminati dalle richieste di scrittura

Attributi DynamoDB del modello di persistenza degli oggetti di .NET

In questa sezione vengono descritti gli attributi offerti dal modello di persistenza degli oggetti in modo che sia possibile mappare le classi e le proprietà alle tabelle e agli attributi DynamoDB.

> **Note**
>
> Negli attributi seguenti, solo `DynamoDBTable` e `DynamoDBHashKey` sono obbligatori.

Dinamo DBGlobal SecondaryIndexHashKey

Mappa una proprietà di classe alla chiave di partizione di un indice secondario globale. Utilizza questo attributo se è necessario eseguire una `Query` su un indice secondario globale.

Dinamo DBGlobal SecondaryIndexRangeKey

Associa una proprietà di classe alla chiave di ordinamento di un indice secondario globale. Utilizza questo attributo se è necessario eseguire una `Query` su un indice secondario globale e si desidera rifinire i risultati utilizzando la chiave di ordinamento dell'indice.

Chiave Dynamo DBHash

Mappa una proprietà della classe alla chiave di partizione della chiave primaria della tabella. Gli attributi della chiave primaria non possono essere un tipo di raccolta.

L'esempio di codice C# seguente mappa la classe `Book`alla tabella `ProductCatalog` e la proprietà `Id` alla chiave di partizione della chiave primaria della tabella.

```
[DynamoDBTable("ProductCatalog")]
public class Book
{
    [DynamoDBHashKey]
    public int Id { get; set; }

    // Additional properties go here.
}
```

Dinamo DBIgnore

Indica che la proprietà associata deve essere ignorata. Se non si desidera salvare nessuna delle proprietà di classe, è possibile aggiungere questo attributo per indicare a `DynamoDBContext` di non includere questa proprietà quando si salvano oggetti nella tabella.

Dinamo DBLocal SecondaryIndexRangeKey

Mappa una proprietà di classe alla chiave di ordinamento di un indice secondario globale. Utilizza questo attributo se è necessario eseguire una `Query` su un indice secondario locale e si desidera rifinire i risultati utilizzando la chiave di ordinamento dell'indice.

Dinamo DBProperty

Mappa una proprietà della classe all'attributo di una tabella. Se la proprietà della classe viene mappata a un attributo di tabella con lo stesso nome, non è necessario specificare l'attributo. Tuttavia, se i nomi non sono gli stessi, è possibile utilizzare questo tag per fornire la mappatura. Nella seguente istruzione C#, `DynamoDBProperty` mappa la proprietà `BookAuthors` all'attributo `Authors` della tabella.

```
[DynamoDBProperty("Authors")]
public List<string> BookAuthors { get; set; }
```

`DynamoDBContext` utilizza queste informazioni di mappatura per creare l'attributo `Authors` quando si salvano i dati dell'oggetto nella tabella corrispondente.

Dinamo DBRenamable

Specifica un nome alternativo per una proprietà di classe. Ciò è utile se si sta scrivendo un convertitore personalizzato per mappare dati arbitrari a una tabella DynamoDB in cui il nome di una proprietà della classe è diverso da un attributo di tabella.

Chiave Dynamo DBRange

Mappa una proprietà di classe alla chiave di ordinamento della chiave primaria della tabella. Se la tabella dispone di una chiave primaria composita (chiave di partizione e chiave di ordinamento), nella mappatura della classe è necessario specificare entrambi gli attributi `DynamoDBHashKey` e `DynamoDBRangeKey`.

Ad esempio, la tabella di esempio `Reply` ha una chiave primaria composta dalla chiave di partizione `Id` e dalla chiave di ordinamento `Replenishment`. Nell'esempio di codice C# seguente la classe `Reply` viene mappata alla tabella `Reply`. La definizione di classe indica anche che due delle sue proprietà vengono mappate alla chiave primaria.

```
[DynamoDBTable("Reply")]
```

```
public class Reply
{
   [DynamoDBHashKey]
   public int ThreadId { get; set; }
   [DynamoDBRangeKey]
   public string Replenishment { get; set; }

   // Additional properties go here.
}
```

Dinamo DBTable

Identifica la tabella di destinazione di DynamoDB a cui viene mappata la classe. Nell'esempio di codice C# seguente la classe `Developer` viene mappata alla tabella `People` in DynamoDB.

```
[DynamoDBTable("People")]
public class Developer { ...}
```

Questo attributo può essere ereditato o sovrascritto.

- L'attributo `DynamoDBTable` può essere ereditato. Nell'esempio precedente, se si aggiunge una nuova classe ,`Lead`, che eredita dalla classe `Developer`, la classe viene mappata anche alla tabella `People`. Entrambi gli oggetti `Developer` e `Lead` vengono archiviati nella tabella `People`.
- L'attributo `DynamoDBTable` può anche essere sovrascritto. Nel seguente esempio di codice C#, la classe `Manager`eredita dalla classe `Developer`. Tuttavia, l'aggiunta esplicita dell'attributo `DynamoDBTable` mappa la classe a un'altra tabella (`Managers`).

  ```
  [DynamoDBTable("Managers")]
  public class Manager : Developer { ...}
  ```

È possibile aggiungere il parametro opzionale, `LowerCamelCaseProperties`, per richiedere a DynamoDB di rendere minuscola la prima lettera del nome della proprietà quando si memorizzano gli oggetti in una tabella, come illustrato nell'esempio C# riportato di seguito.

```
[DynamoDBTable("People", LowerCamelCaseProperties=true)]
public class Developer
{
    string DeveloperName;
    ...
```

```
}
```

Quando si salvano le istanze della classe `Developer`, `DynamoDBContext` salva la proprietà `DeveloperName` come `developerName`.

Dinamo DBVersion

Identifica una proprietà di classe per l'archiviazione del numero di versione dell'elemento. Per ulteriori informazioni sulla funzione Controllo delle versioni, consulta Blocco ottimistico che utilizza DynamoDB e il modello di persistenza degli oggetti di AWS SDK per .NET.

DBContext Classe Dynamo dal modello di persistenza degli oggetti.NET

La classe `DynamoDBContext` è il punto di ingresso al database Amazon DynamoDB. Fornisce una connessione a DynamoDB e permette di accedere ai dati in varie tabelle, eseguire diverse operazioni CRUD ed eseguire query. La classe `DynamoDBContext` fornisce i metodi seguenti:

Argomenti

- Crea MultiTable BatchGet
- Crea MultiTable BatchWrite
- CreateBatchGet
- CreateBatchWrite
- Elimina
- Elimina
- ExecuteBatchGet
- ExecuteBatchWrite
- FromDocument
- FromQuery
- FromScan
- GetTargetTable
- Carica
- Query
- Save (Salva)

- Scan
- ToDocument
- Specificazione dei parametri opzionali per Dynamo DBContext

Crea MultiTable BatchGet

Crea un oggetto `MultiTableBatchGet`, composto da più oggetti `BatchGet`. Ognuno di questi oggetti `BatchGet` può essere utilizzato per recuperare elementi da una singola tabella DynamoDB.

Per recuperare gli elementi dalle tabelle, utilizza il metodo `ExecuteBatchGet`, passando l'oggetto `MultiTableBatchGet` come parametro.

Crea MultiTable BatchWrite

Crea un oggetto `MultiTableBatchWrite`, composto da più oggetti `BatchWrite`. Ognuno di questi oggetti `BatchWrite` può essere utilizzato per scrivere o eliminare elementi in una singola tabella DynamoDB.

Per scrivere sulle tabelle, utilizza il metodo `ExecuteBatchWrite`, passando l'oggetto `MultiTableBatchWrite` come parametro.

CreateBatchGet

Crea un oggetto `BatchGet` da usare per recuperare più elementi da una tabella.

CreateBatchWrite

Crea un oggetto `BatchWrite` che è possibile utilizzare per inserire più elementi in una tabella o per eliminare più elementi da una tabella.

Elimina

Elimina un item dalla tabella. Il metodo richiede la chiave primaria dell'elemento che si desidera eliminare. È possibile fornire il valore della chiave primaria o un oggetto lato client contenente un valore di chiave primaria come parametro per questo metodo.

- Se si specifica un oggetto lato client come parametro ed è stato abilitato il blocco ottimistico, l'eliminazione avrà esito positivo solo se le versioni lato client e lato server dell'oggetto corrispondono.

- Se si specifica come parametro solo il valore della chiave primaria, l'eliminazione avrà esito positivo indipendentemente dal fatto che sia stato abilitato o meno il blocco ottimistico.

> **Note**
>
> Per eseguire questa operazione in background, utilizza invece il metodo `DeleteAsync`.

Elimina

Elimina tutte le risorse gestite e non gestite.

ExecuteBatchGet

Legge i dati da una o più tabelle, elaborando tutti gli oggetti `BatchGet` in un `MultiTableBatchGet`.

> **Note**
>
> Per eseguire questa operazione in background, utilizza invece il metodo `ExecuteBatchGetAsync`.

ExecuteBatchWrite

Scrive o elimina i dati in una o più tabelle, elaborando tutti gli oggetti `BatchWrite` in un `MultiTableBatchWrite`.

> **Note**
>
> Per eseguire questa operazione in background, utilizza invece il metodo `ExecuteBatchWriteAsync`.

FromDocument

Data un'istanza di un `Document`, il metodo `FromDocument` restituisce un'istanza di una classe lato client.

Ciò è utile se si desidera utilizzare le classi del modello di documento insieme al modello di persistenza degli oggetti per eseguire qualsiasi operazione sui dati. Per ulteriori informazioni sulle classi del modello di documento fornite da AWS SDK per .NET, vedereUtilizzo del modello documentale .NET in DynamoDB.

Si supponga di avere un oggetto `Document` denominato `doc`, che contiene una rappresentazione di un elemento `Forum`. Per vedere come costruire questo oggetto, consulta la descrizione del metodo `ToDocument` più avanti in questo argomento). È possibile utilizzare `FromDocument` per recuperare l'elemento `Forum`dal `Document`, come mostrato nell'esempio di codice C# seguente.

Example

```
forum101 = context.FromDocument<Forum>(101);
```

> **Note**
>
> Se l'oggetto `Document` implementa l'interfaccia `IEnumerable`, è possibile utilizzare invece il metodo `FromDocuments`. Ciò consente di eseguire iterazioni su tutte le istanze della classe nel `Document`.

FromQuery

Esegue un'operazione `Query`, con i parametri di query definiti in un oggetto `QueryOperationConfig`.

> **Note**
>
> Per eseguire questa operazione in background, utilizza invece il metodo `FromQueryAsync`.

FromScan

Esegue un'operazione `Scan`, con i parametri di scansione definiti in un oggetto `ScanOperationConfig`.

> **Note**
>
> Per eseguire questa operazione in background, utilizza invece il metodo `FromScanAsync`.

GetTargetTable

Recupera la tabella di destinazione per il tipo specificato. Ciò è utile se si sta scrivendo un convertitore personalizzato per mappare dati arbitrari a una tabella DynamoDB ed è necessario determinare quale tabella è associata a un tipo di dati personalizzato.

Carica

Recupera un item da una tabella. Il metodo richiede solo la chiave primaria dell'elemento che si desidera recuperare.

Per impostazione predefinita, DynamoDB restituisce l'elemento che dispone di valori che sono a consistenza finale. Per informazioni sul modello di consistenza finale, consulta Coerenza di lettura di DynamoDB.

Loador `LoadAsync` method chiama l'GetItemoperazione, che richiede di specificare la chiave primaria per la tabella. Poiché `GetItem` ignora il parametro `IndexName`, non è possibile caricare un elemento utilizzando la partizione o la chiave di ordinamento di un indice. Pertanto, è necessario utilizzare la chiave primaria della tabella per caricare un elemento.

> **Note**
>
> Per eseguire questa operazione in background, utilizza invece il metodo `LoadAsync`. Per visualizzare un esempio di utilizzo del metodo `LoadAsync` per eseguire operazioni CRUD di alto livello su una tabella DynamoDB, consulta l'esempio seguente.

```
/// <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");
```

```
            }
        }
    }
```

Query

Interroga una tabella in base ai parametri di query forniti.

È possibile eseguire una query su una tabella solo se la tabella dispone di una chiave primaria composita (una chiave di partizione e una chiave di ordinamento). Quando si esegue una query, è necessario specificare una chiave di partizione e una condizione che si applichi alla chiave di ordinamento.

Si supponga di disporre di una classe `Reply` lato client mappata alla tabella `Reply` in DynamoDB. L'esempio di codice C# seguente esegue una query sulla tabella`Reply` per ottenere le risposte a un thread del forum pubblicate negli ultimi 15 giorni. La tabella `Reply` ha una chiave primaria che ha la chiave di partizione `Id` e la chiave di ordinamento `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);
```

Restituisce una raccolta di oggetti `Reply`.

Il metodo `Query` restituisce una raccolta `IEnumerable` a "caricamento differito". Inizialmente restituisce solo una pagina di risultati e in seguito effettua una chiamata di assistenza per la pagina successiva, se necessario. Per ottenere tutti gli elementi corrispondenti, è necessario iterare solo su `IEnumerable`.

Se la tabella ha una chiave primaria semplice (chiave di partizione), non è possibile utilizzare il metodo `Query`. Invece, è possibile utilizzare il metodo `Load` e fornire la chiave di partizione per recuperare l'elemento.

> **Note**
>
> Per eseguire questa operazione in background, utilizza invece il metodo `QueryAsync`.

Save (Salva)

Salva l'oggetto specificato nella tabella. Se la chiave primaria specificata nell'oggetto di input non esiste nella tabella, il metodo aggiunge un nuovo elemento alla tabella. Se esiste la chiave primaria, il metodo aggiorna l'elemento esistente.

Se è stato configurato il blocco ottimistico, l'aggiornamento avrà esito positivo solo se le versioni lato client e lato server dell'elemento corrispondono. Per ulteriori informazioni, consulta Blocco ottimistico che utilizza DynamoDB e il modello di persistenza degli oggetti di AWS SDK per .NET.

> **Note**
>
> Per eseguire questa operazione in background, utilizza invece il metodo `SaveAsync`.

Scan

Esegue la scansione dell'intera tabella.

È possibile filtrare i risultati della scansione specificando una condizione di scansione. La condizione può essere valutata in base a qualsiasi attributo nella tabella. Si supponga di disporre di una classe `Book` lato client mappata alla tabella `ProductCatalog` in DynamoDB. Nell'esempio di C# seguente viene eseguita la scansione della tabella e vengono restituiti solo gli elementi del libro con un prezzo inferiore a 0.

Example

```
IEnumerable<Book> itemsWithWrongPrice = context.Scan<Book>(
                    new ScanCondition("Price", ScanOperator.LessThan, price),
                    new ScanCondition("ProductCategory", ScanOperator.Equal, "Book")
      );
```

Il metodo `Scan` restituisce una raccolta `IEnumerable` a "caricamento differito". Inizialmente restituisce solo una pagina di risultati e in seguito effettua una chiamata di assistenza per la pagina

successiva, se necessario. Per ottenere tutti gli elementi corrispondenti, è necessario iterare solo su `IEnumerable`.

Per motivi di prestazioni, è preferibile eseguire query sulle tabelle ed evitare le scansioni.

> **Note**
>
> Per eseguire questa operazione in background, utilizza invece il metodo `ScanAsync`.

ToDocument

Restituisce un'istanza della classe del modello di documento `Document` dall'istanza della classe.

Ciò è utile se si desidera utilizzare le classi del modello di documento insieme al modello di persistenza degli oggetti per eseguire qualsiasi operazione sui dati. Per ulteriori informazioni sulle classi del modello di documento fornite da AWS SDK per .NET, vedereUtilizzo del modello documentale .NET in DynamoDB.

Si supponga di disporre di una classe lato client mappata alla tabella `Forum` di esempio. È quindi possibile utilizzare un `DynamoDBContext` per ottenere un elemento come oggetto `Document` dalla tabella `Forum`, come mostrato nell'esempio di codice C# seguente.

Example

```
DynamoDBContext context = new DynamoDBContext(client);

Forum forum101 = context.Load<Forum>(101); // Retrieve a forum by primary key.
Document doc = context.ToDocument<Forum>(forum101);
```

Specificazione dei parametri opzionali per Dynamo DBContext

Quando si utilizza il modello di persistenza degli oggetti, è possibile specificare i parametri facoltativi che seguono per `DynamoDBContext`.

- **`ConsistentRead`**: quando si recuperano i dati utilizzando le operazioni `Load`, `Query` o `Scan`, è possibile aggiungere questo parametro facoltativo per richiedere i valori più recenti dei dati.
- **`IgnoreNullValues`**: questo parametro indica a `DynamoDBContext` di ignorare i valori nulli sugli attributi durante un'operazione `Save`. Se questo parametro è false (o se non è impostato), allora un valore null viene interpretato come una direttiva per eliminare l'attributo specifico.

- **`SkipVersionCheck`**: questo parametro indica a `DynamoDBContext` di non confrontare le versioni durante il salvataggio o l'eliminazione di un elemento. Per ulteriori informazioni sulla funzione Controllo delle versioni, consulta Blocco ottimistico che utilizza DynamoDB e il modello di persistenza degli oggetti di AWS SDK per .NET.
- **`TableNamePrefix`**: aggiunge un prefisso a tutti i nomi delle tabelle con una stringa specifica. Se questo parametro è null (o se non è impostato), allora non viene utilizzato alcun prefisso.
- `DynamoDBEntryConversion`: specifica lo schema di conversione utilizzato dal client. È possibile impostare questo parametro sulla versione V1 o V2. V1 è la versione predefinita.

  In base alla versione impostata, il comportamento di questo parametro cambia. Ad esempio:

  - In V1, il tipo di dati `bool` viene convertito nel tipo numerico `N`, dove 0 rappresenta false e 1 rappresenta true. In V2, `bool` viene convertito in `BOOL`.
  - Nella V2, gli elenchi e gli array non sono raggruppati con. HashSets Gli elenchi e gli array di valori numerici, tipi basati su stringhe e tipi a base binaria vengono convertiti nel tipo `L` (List, elenco), che può essere inviato vuoto per aggiornare un elenco. Il caso è diverso da V1, in cui un elenco vuoto non viene inviata in rete.

    Nella V1, i tipi di raccolta, come List HashSet, e gli array vengono trattati allo stesso modo. L'elenco e HashSet la matrice di numeri vengono convertiti nel tipo `NS` (set di numeri).

  L'esempio seguente imposta la versione dello schema di conversione su V2, che modifica il comportamento di conversione tra tipi .NET e tipi di dati DynamoDB.

```
var config = new DynamoDBContextConfig
{
    Conversion = DynamoDBEntryConversion.V2
};
var contextV2 = new DynamoDBContext(client, config);
```

L'esempio C# seguente crea un nuovo `DynamoDBContext` specificando due dei parametri facoltativi precedenti, `ConsistentRead` e `SkipVersionCheck`.

Example

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
...
DynamoDBContext context =
```

```
      new DynamoDBContext(client, new DynamoDBContextConfig { ConsistentRead = true,
 SkipVersionCheck = true});
```

`DynamoDBContext` include questi parametri facoltativi con ogni richiesta inviata utilizzando questo contesto.

Invece di impostare questi parametri al livello `DynamoDBContext`, è possibile specificarli per le singole operazioni eseguite utilizzando `DynamoDBContext`, come mostrato nell'esempio di codice C# seguente. L'esempio carica un elemento del libro specifico. Il metodo `Load` di `DynamoDBContext` specifica i parametri facoltativi `ConsistentRead` e `SkipVersionCheck`.

Example

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
...
DynamoDBContext context = new DynamoDBContext(client);
Book bookItem = context.Load<Book>(productId,new DynamoDBContextConfig{ ConsistentRead
 = true, SkipVersionCheck = true });
```

In questo caso, `DynamoDBContext` include questi parametri solo quando si invia la richiesta `Get`.

Blocco ottimistico che utilizza DynamoDB e il modello di persistenza degli oggetti di AWS SDK per .NET

Il supporto ottimistico del blocco nel modello di persistenza degli oggetti garantisce che la versione dell'elemento per l'applicazione corrisponda alla versione dell'elemento sul lato server prima di aggiornare o eliminare l'elemento. Si supponga di recuperare un elemento per l'aggiornamento. Tuttavia, prima di inviare nuovamente gli aggiornamenti, alcune altre applicazioni aggiornano lo stesso elemento. In questa situazione, la tua applicazione avrà una copia obsoleta dell'elemento. Senza il blocco ottimistico, qualsiasi aggiornamento eseguito sovrascriverà l'aggiornamento effettuato dall'altra applicazione.

La funzione di blocco ottimistica del modello di persistenza degli oggetti fornisce il tag `DynamoDBVersion` che è possibile utilizzare per abilitare il blocco ottimistico. Per utilizzare questa funzionalità, è necessario aggiungere una proprietà alla classe per memorizzare il numero di versione. Quindi aggiungere l'attributo `DynamoDBVersion` alla proprietà. Quando si salva l'oggetto per la prima volta, `DynamoDBContext` assegna un numero di versione che viene incrementato automaticamente ogni volta che si aggiorna l'elemento.

Le richieste di aggiornamento ed eliminazione hanno esito positivo solo se la versione dell'oggetto lato client corrisponde al numero di versione dell'elemento corrispondente del lato server. Se l'applicazione dispone di una copia non aggiornata, deve ottenere la versione più recente dal server prima di poter aggiornare o eliminare tale elemento.

L'esempio di codice C# seguente definisce una classe `Book` con attributi di persistenza dell'oggetto che la mappano alla tabella `ProductCatalog`. La proprietà `VersionNumber` nella classe lavorata con l'attributo `DynamoDBVersion` memorizza il valore del numero di versione.

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**
>
> Puoi applicare l'attributo `DynamoDBVersion` solo a un tipo primitivo numerico annullabile (come `int?`).

Il blocco ottimistico ha il seguente impatto sulle operazioni `DynamoDBContext`:

- Per un nuovo elemento, `DynamoDBContext` assegna il numero di versione iniziale uguale a 0. Se si recupera un elemento esistente, si aggiorna una o più delle sue proprietà e si prova a salvare le modifiche, l'operazione di salvataggio ha esito positivo solo se il numero di versione sul lato client e sul lato server corrispondono. `DynamoDBContext` incrementa quindi il numero di versione. Non è necessario impostare il numero di versione.

- Il metodo `Delete` fornisce overload che possono assumere il valore di una chiave primaria o un oggetto come parametro, come mostrato nel seguente esempio di codice 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);
  ```

  Se si specifica un oggetto come parametro, l'eliminazione ha esito positivo solo se la versione dell'oggetto corrisponde alla versione dell'elemento lato server corrispondente. Tuttavia, se si specifica un valore di chiave primaria come parametro, `DynamoDBContext` non è a conoscenza di alcun numero di versione ed elimina l'elemento senza effettuare il controllo della versione.

  Tenere presente che l'implementazione interna del blocco ottimistico nel codice del modello di persistenza degli oggetti utilizza l'aggiornamento condizionale e le operazioni API di eliminazione condizionale in DynamoDB.

Disabilitazione del blocco ottimistico

Per disabilitare il blocco ottimistico, utilizza la proprietà di configurazione `SkipVersionCheck`. È possibile impostare questa proprietà durante la creazione di `DynamoDBContext`. In questo caso, il blocco ottimistico è disabilitato per qualsiasi richiesta effettuata utilizzando il contesto. Per ulteriori informazioni, consulta Specificazione dei parametri opzionali per Dynamo DBContext .

Invece di impostare la proprietà a livello di contesto, è possibile disattivare il blocco ottimistico per un'operazione specifica, come mostrato nel seguente esempio di codice C#. Nell'esempio viene utilizzato il contesto per eliminare un elemento del libro. Il metodo `Delete` imposta la proprietà `SkipVersionCheck` facoltativa su true, disabilitando il controllo della versione.

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 });
```

Mappatura di dati arbitrari con DynamoDB utilizzando il modello di persistenza degli oggetti AWS SDK per .NET

Oltre ai tipi .NET supportati (vedere Tipi di dati supportati), è possibile utilizzare i tipi dell'applicazione per i quali non vi è una mappatura diretta ai tipi di Amazon DynamoDB. Il modello di persistenza degli oggetti supporta la memorizzazione di dati di tipi arbitrari purché si fornisca il convertitore per convertire i dati dal tipo arbitrario al tipo DynamoDB e viceversa. Il codice convertitore trasforma i dati sia durante il salvataggio che durante il caricamento degli oggetti.

È possibile creare qualsiasi tipo sul lato client. Tuttavia, i dati memorizzati nelle tabelle sono uno dei tipi DynamoDB e durante la query e la scansione, tutti i confronti di dati effettuati sono rispetto ai dati memorizzati in DynamoDB.

L'esempio di codice C# seguente definisce una classe `Book` con le proprietà `Id`, `Title`, `ISBN` e `Dimension`. La proprietà `Dimension` è di `DimensionType` che descrive le proprietà `Height`, `Width` e `Thickness`. Il codice di esempio fornisce i metodi del convertitore`ToEntry` e `FromEntry` per convertire i dati tra `DimensionType` e i tipi di stringa DynamoDB. Ad esempio, quando si salva un'istanza `Book`, il convertitore crea una stringa `Dimension` del libro come "8.5x11x.05". Quando recuperi un libro, converte la stringa in un'istanza `DimensionType`.

Nell'esempio viene mappato il tipo `Book` alla tabella `ProductCatalog`. Salva un'istanza `Book` di esempio, la recupera, aggiorna le sue dimensioni e salva di nuovo il `Book` aggiornato.

Per step-by-step istruzioni su come testare il seguente esempio, consulta. Esempi di codice .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;
        }
    }
}
```

Esecuzione degli esempi di codice in questa guida per gli sviluppatori

Gli SDK AWS forniscono ampio supporto per Amazon DynamoDB nei seguenti linguaggi:

- Java
- JavaScript nel browser
- .NET
- Node.js
- PHP
- Python
- Ruby
- C++
- Go
- Android
- iOS

L'esempio di codice in questa guida per gli sviluppatori fornisce una copertura più dettagliata delle operazioni di DynamoDB, usando i seguenti linguaggi di programmazione:

- Esempi di codice Java
- Esempi di codice .NET

Prima di iniziare con questo esercizio, sarà necessario creare un account AWS, ottenere la chiave di accesso e la chiave segreta e impostare AWS Command Line Interface (AWS CLI) sul computer. Per ulteriori informazioni, consulta Configurazione di DynamoDB (servizio Web) .

> **Note**
>
> Se si sta usando la versione scaricabile di DynamoDB, è necessario utilizzare la AWS CLI per creare le tabelle e i dati di esempio. Inoltre, necessiti di specificare il parametro `--endpoint-url` con ogni comando dell'AWS CLI. Per ulteriori informazioni, consulta Impostazione dell'endpoint locale .

Creazione di tabelle e caricamento di dati per esempi di codice in DynamoDB

Di seguito sono riportate le nozioni di base sulla creazione di tabelle in DynamoDB, sul caricamento di un set di dati di esempio, sulle query sui dati e sull'aggiornamento dei dati.

- Fase 1: creare una tabella DynamoDB
- Fase 2: scrivere dati in una tabella DynamoDB
- Fase 3: leggere i dati da una tabella DynamoDB
- Fase 4: aggiornare i dati in una tabella

Esempi di codice Java

Argomenti

- Java: Impostazioni delle credenziali AWS
- Java: Configurazione dell'endpoint e della regione AWS

Questa guida per gli sviluppatori contiene frammenti di codice Java e programmi pronti per l'esecuzione. Puoi trovare questi esempi di codice nelle seguenti sezioni:

- Utilizzo di elementi e attributi in DynamoDB
- Utilizzo di tabelle e dati in DynamoDB
- Esecuzione di query in DynamoDB
- Scansione di tabelle in DynamoDB
- Miglioramento dell’accesso ai dati con gli indici secondari in DynamoDB
- Java 1.x: DynamoDBMapper
- Acquisizione dei dati di modifica per DynamoDB Streams

Puoi iniziare velocemente usando Eclipse con AWS Toolkit for Eclipse. Oltre a un IDE completo, otterrai AWS SDK per Java con aggiornamenti automatici e modelli preconfigurati, per creare le applicazioni AWS.

Per eseguire esempi di codice Java (usando Eclipse)

1. Scaricare e installare l'IDE Eclipse;
2. Scarica e installa AWS Toolkit for Eclipse.
3. Avviare Eclipse e dal menu Eclipse scegliere File, New (Nuovo) e quindi Other (Altro).
4. In Seleziona una procedura guidata, scegli AWS, quindi Progetto Java per AWS, infine Successivo.
5. In Crea un Java AWS, effettua le seguenti operazioni:

   a. In Project name (Nome progetto) immettere un nome per il progetto.

   b. In Select Account (Seleziona un account) scegliere il profilo per le credenziali dall'elenco.

      Se è la prima volta che utilizzi AWS Toolkit for Eclipse, scegli Configura account AWS per configurare le credenziali AWS.
6. Scegli Fine per creare il progetto.
7. Dal menu Eclipse scegliere File, New (Nuovo) e infine Class (Classe).
8. In Java Class immettere un nome per la classe in Name (Nome) (usare lo stesso nome dell'esempio di codice che si desidera eseguire) e infine scegliere Finish (Fine) per creare la classe.
9. Copiare l'esempio di codice dalla pagina della documentazione nell'editor di Eclipse.
10. Per eseguire il codice, scegliere Run (Esegui) nel menu Eclipse.

L'SDK per Java fornisce client sicuri per lavorare con DynamoDB. Come best practice, le tue applicazioni dovrebbero creare un client e riutilizzarlo tra i thread.

Per ulteriori informazioni, consulta AWS SDK per Java.

> **Note**
>
> Gli esempi di codice in questa guida sono pensati per l'utilizzo con la versione più recente di AWS SDK per Java.

> Se si sta usando la AWS Toolkit for Eclipse, è possibile configurare gli aggiornamenti automatici per l'SDK per Java. Per far ciò in Eclipse, passa a Preferenze e scegli Kit di strumenti AWS, AWS SDK per Java, Scarica automaticamente nuovi SDK.

Java: Impostazioni delle credenziali AWS

SDK per Java richiede che siano fornite le credenziali AWS all'applicazione durante il runtime. Gli esempi di codice in questa guida presuppongono che stia usando un file di credenziali AWS, come descritto in Configurazione delle credenziali AWS nella Guida per gli sviluppatori di AWS SDK per Java.

Di seguito, è riportato un esempio di un file di credenziali AWS, denominato `~/.aws/credentials`, dove il segno della tilde (~) rappresenta la tua directory home.

```
[default]
aws_access_key_id = AWS access key ID goes here
aws_secret_access_key = Secret key goes here
```

Java: Configurazione dell'endpoint e della regione AWS

Per impostazione predefinita, gli esempi di codice accedono a DynamoDB nella regione Stati Uniti occidentali (Oregon). Puoi modificare la regione modificando le proprietà `AmazonDynamoDB`.

Il seguente esempio di codice crea un'istanza di un nuovo `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();
```

È possibile utilizzare il metodo `withRegion` per eseguire il codice in DynamoDB in ogni regione in cui sia disponibile. Per l'elenco completo, consulta Regioni ed endpoint AWS in Riferimenti generali di Amazon Web Services.

Se si desidera eseguire gli esempi di codice usando DynamoDB in locale sul computer, impostare l'endpoint come riportato di seguito:

AWS SDK V1

```
AmazonDynamoDB client =
 AmazonDynamoDBClientBuilder.standard().withEndpointConfiguration(
new AwsClientBuilder.EndpointConfiguration("http://localhost:8000", "us-west-2"))
.build();
```

AWS SDK 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();
```

Esempi di codice .NET

Argomenti

- .NET: Configurazione delle credenziali AWS
- .NET: Configurazione dell'endpoint e della regione AWS

Questa guida contiene frammenti di codice .NET e programmi pronti per l'esecuzione. Puoi trovare questi esempi di codice nelle seguenti sezioni:

- Utilizzo di elementi e attributi in DynamoDB
- Utilizzo di tabelle e dati in DynamoDB
- Esecuzione di query in DynamoDB
- Scansione di tabelle in DynamoDB
- Miglioramento dell’accesso ai dati con gli indici secondari in DynamoDB
- Utilizzo del modello documentale .NET in DynamoDB
- Operazioni con il modello di persistenza degli oggetti di .NET e DynamoDB
- Acquisizione dei dati di modifica per DynamoDB Streams

È possibile iniziare velocemente usando AWS SDK per .NET con Toolkit for Visual Studio.

Esecuzione degli esempi di codice .NET (usando Visual Studio)

1. Scaricare e installare Microsoft Visual Studio.
2. Scaricare e installare Toolkit for Visual Studio.
3. Avvia Visual Studio. Scegli File, Nuovo, Progetto.
4. In Nuovo progetto scegli Progetto vuoto AWS, quindi scegli OK.
5. In Credenziali di accesso AWS scegliere Usa profilo esistente, seleziona il profilo delle credenziali dall'elenco, infine scegli OK.

   Se è la prima volta che si utilizza Toolkit for Visual Studio, scegli Usa un nuovo profilo per configurare le credenziali AWS.
6. Nel tuo progetto di Visual Studio, scegli la tabella per il codice sorgente del tuo programma (`Program.cs`). Copia l'esempio di codice dalla pagina della documentazione nell'editor di Visual Studio, sostituendo qualsiasi altro codice che vedi nell'editor.
7. Se vengono visualizzati messaggi di errore del moduloImpossibile trovare il tipo o il nome dello spazio dei nomi..., è necessario installare l'assieme di SDK AWSper DynamoDB come segue:

   a. In Esplora soluzioni aprire il menu contestuale (pulsante destro del mouse) per il progetto e scegliere Manage NuGet Packages (Gestisci pacchetti NuGet).

   b. In Gestione pacchetti NuGet scegliere Browse (Sfoglia).

   c. Nella casella di ricerca, immettere **`AWSSDK.DynamoDBv2`** e attendere il completamento della ricerca.

   d. Scegli AWSSDK.DynamoDBv2, quindi Installa.

   e. Quando l'installazione è stata completata, scegliere la tabella Program.cs per tornare al programma.
8. Per eseguire il codice, scegliere Start nella barra degli strumenti di Visual Studio.

SDK per .NET fornisce client sicuri per lavorare con DynamoDB. Come best practice, le tue applicazioni dovrebbero creare un client e riutilizzarlo tra i thread.

Per ulteriori informazioni, consulta AWS SDK for .NET.

> **Note**
>
> Gli esempi di codice in questa guida sono pensati per l'utilizzo con la versione più recente di AWS SDK per .NET.

.NET: Configurazione delle credenziali AWS

La SDK per .NET richiede di fornire le credenziali AWS all'applicazione durante il runtime. Gli esempi di codice in questa guida presuppongono che stia usando SDK Store per gestire il tuo file delle credenziali AWS, come descritto in Utilizzo di SDK Store nella Guida per gli sviluppatori di AWS SDK per .NET.

Toolkit for Visual Studio supporta più set di credenziali da qualsiasi numero di account. Ogni set viene definito profilo. Visual Studio aggiunge voci al file `App.config` del progetto in modo che l'applicazione possa trovare le credenziali AWS al runtime.

L'esempio seguente mostra il file `App.config` predefinito che viene generato quando si crea un nuovo progetto usando 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>
```

In fase di runtime, il programma utilizza il set `default` delle credenziali AWS, come specificato dalla voce `AWSProfileName`. Le credenziali AWS sono conservate in SDK Store in forma crittografata. Toolkit for Visual Studio fornisce un'interfaccia grafica dell'utente per gestire le credenziali, tutto all'interno di Visual Studio. Per ulteriori informazioni, consulta Specifica delle credenziali nella Guida per l'utente di AWS Toolkit for Visual Studio.

> **Note**
>
> Per impostazione predefinita, gli esempi di codice accedono a DynamoDB nella regione Stati Uniti occidentali (Oregon). Puoi cambiare la regione modificando la voce `AWSRegion` nel file App.config. È possibile impostare `AWSRegion` su qualsiasi regione in cui è disponibile

DynamoDB. Per l'elenco completo, consulta Regioni ed endpoint AWS in Riferimenti generali di Amazon Web Services.

.NET: Configurazione dell'endpoint e della regione AWS

Per impostazione predefinita, gli esempi di codice accedono a DynamoDB nella regione Stati Uniti occidentali (Oregon). Puoi cambiare la regione modificando la voce `AWSRegion` nel file `App.config`. Oppure, puoi modificare la regione modificando le proprietà `AmazonDynamoDBClient`.

Il seguente esempio di codice crea un'istanza di un nuovo `AmazonDynamoDBClient`. Il client viene modificato in modo che il codice venga eseguito in DynamoDB in una regione differente.

```
AmazonDynamoDBConfig clientConfig = new AmazonDynamoDBConfig();
// This client will access the US East 1 region.
clientConfig.RegionEndpoint = RegionEndpoint.USEast1;
AmazonDynamoDBClient client = new AmazonDynamoDBClient(clientConfig);
```

Per l'elenco completo delle regioni, consulta Regioni ed endpoint AWS in Riferimenti generali di Amazon Web Services.

Se si desidera eseguire gli esempi di codice usando DynamoDB in locale sul computer, impostare l'endpoint come riportato di seguito:

```
AmazonDynamoDBConfig clientConfig = new AmazonDynamoDBConfig();
// Set the endpoint URL
clientConfig.ServiceURL = "http://localhost:8000";
AmazonDynamoDBClient client = new AmazonDynamoDBClient(clientConfig);
```

API DynamoDB di basso livello

L'API di basso livello di Amazon DynamoDB è l'interfaccia a livello di protocollo per DynamoDB. A questo livello, ogni richiesta HTTP(S) deve essere formattata correttamente e contenere una firma digitale valida.

Gli SDK AWS costruiscono le richieste API di basso livello di DynamoDB per conto tuo ed elaborano le risposte da DynamoDB. Ciò ti consente di concentrarti sulla logica dell'applicazione, anziché sui

dettagli di basso livello. Tuttavia, può essere comunque utile sapere in grandi linee come funziona l'API di livello basso di DynamoDB.

Per ulteriori informazioni sull'API DynamoDB di basso livello, consultaDocumentazione di riferimento dell'API Amazon DynamoDB.

> **Note**
>
> DynamoDB Streams ha una propria API di basso livello che è separata da quella di DynamoDB ed è completamente supportata dagli SDK AWS.
> Per ulteriori informazioni, consulta Acquisizione dei dati di modifica per DynamoDB Streams. Per informazioni sull'API DynamoDB Streams di basso livello, consulta la documentazioneRiferimento API Amazon DynamoDB Streams.

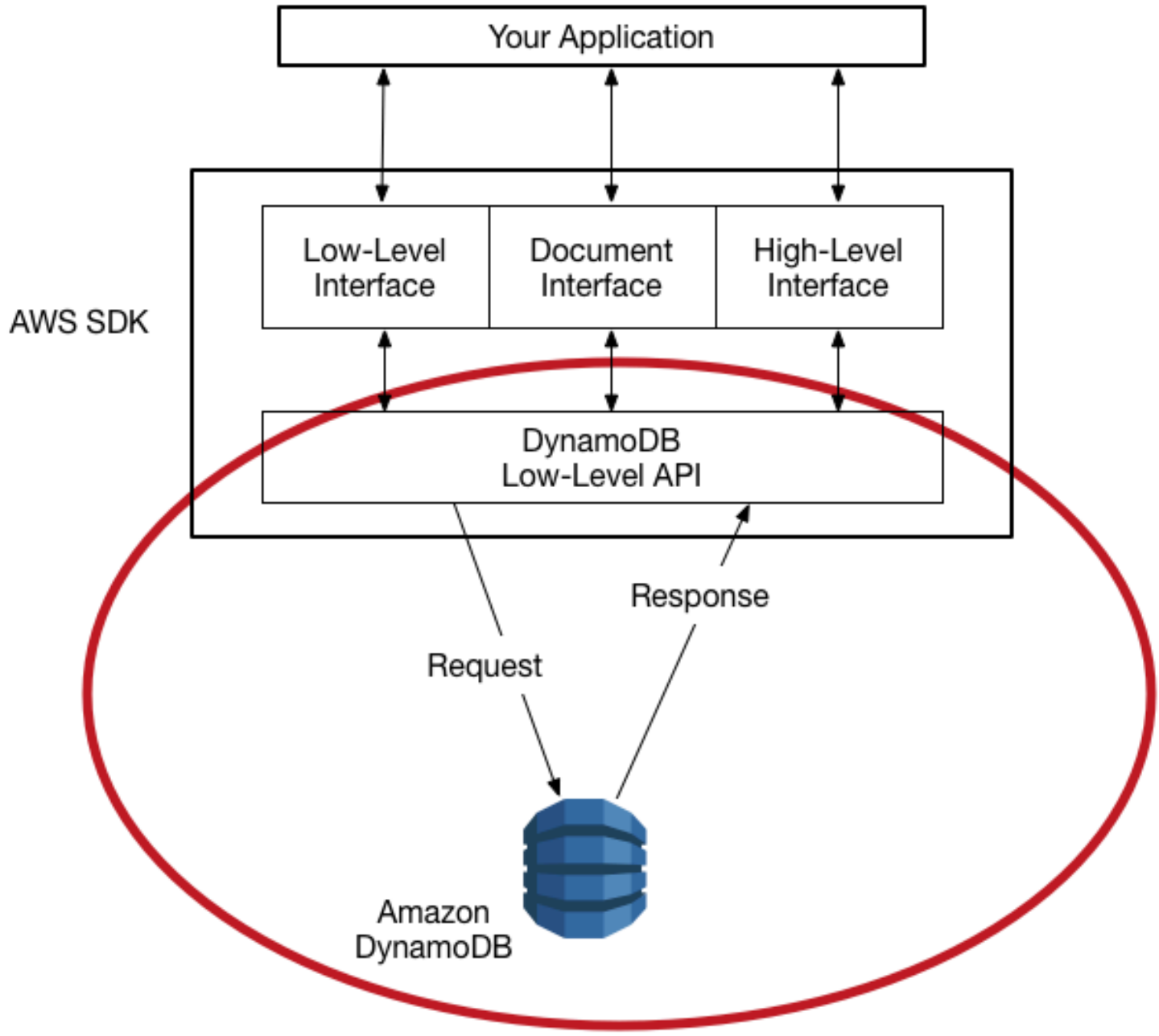
L'API di basso livello di DynamoDB utilizza JavaScript Object Notation (JSON) come formato di protocollo di collegamento. Con il formato JSON, i dati vengono presentati secondo una gerarchia che consente di trasmettere simultaneamente i valori dei dati e la struttura corrispondente. Le coppie nome-valore sono definite nel formato `name:value`. La gerarchia dei dati viene definita tramite parentesi annidate di coppie nome-valore.

DynamoDB usa JSON solo come protocollo di trasporto e non come formato di archiviazione. LaAWSSDK usano JSON per inviare i dati a DynamoDB e DynamoDB risponde con JSON. DynamoDB non memorizza i dati in modo persistente in formato JSON.

> **Note**
>
> Per ulteriori informazioni su JSON, consulta Presentazione di JSON nel sito Web `JSON.org`.

Argomenti

- Formato della richiesta
- Formato della risposta
- Descrittori del tipo di dati
- Dati numerici
- Dati binari



Formato della richiesta

L'API di basso livello di DynamoDB accetta le richieste HTTP(S) `POST` come input. Gli SDK AWS costruiscono queste richieste automaticamente.

Supponiamo di avere una tabella denominata `Pets` con uno schema di chiave costituito da `AnimalType` (chiave di partizione) e `Name` (chiave di ordinamento). Entrambi questi attributi sono di tipo `string`. Per recuperare un elemento da `Pets`, l'SDK AWS costruisce la richiesta seguente.

```
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"}
    }
}
```

Tieni presente le seguenti considerazioni sulla richiesta:

- L'intestazione `Authorization` contiene le informazioni necessarie per l'autenticazione della richiesta da parte di DynamoDB. Per ulteriori informazioni, consulta Firma delle richieste API AWS e Processo di firma Signature Version 4 in Riferimenti generali di Amazon Web Services.
- L'intestazione `X-Amz-Target` contiene il nome di un'operazione DynamoDB: `GetItem`. Il nome è accompagnato dalla versione API di basso livello, in questo caso `20120810`.
- Il payload (corpo) della richiesta contiene i parametri per l'operazione in formato JSON. Per l'operazione `GetItem`, i parametri sono `TableName` e `Key`.

Formato della risposta

Al ricevimento della richiesta, DynamoDB la elabora e restituisce una risposta. Per la richiesta mostrata in precedenza, il payload della risposta HTTP(S) contiene i risultati dell'operazione, come mostrato nell'esempio seguente:

```
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"}
    }
}
```

A questo punto, l'SDK AWS restituisce i dati della risposta all'applicazione per continuare l'elaborazione.

> **Note**
>
> Se l'esecuzione della richiesta non riesce, DynamoDB restituisce un codice di errore HTTP e un messaggio di errore. L'SDK AWS propaga la risposta all'applicazione sotto forma di eccezione. Per ulteriori informazioni, consulta Gestione degli errori con DynamoDB.

Descrittori del tipo di dati

Il protocollo API di basso livello di DynamoDB richiede che ogni attributo sia accompagnato da un descrittore del tipo di dati. I descrittori del tipo di dati sono token che indicano a DynamoDB come interpretare ogni attributo.

Gli esempi in Formato della richiesta e Formato della risposta mostrano come vengono usati i descrittori del tipo di dati. La richiesta `GetItem` specifica `S` per gli attributi dello schema della chiave `Pets` (`AnimalType` e `Name`) che sono di tipo `string`. La risposta `GetItem` contiene un item Pets con attributi di tipo `string` (`S`), `number` (`N`), `map` (`M`) e `list` (`L`).

Di seguito è riportato un elenco completo dei descrittori del tipo di dati di DynamoDB:

- **`S`**: String
- **`N`**: Number
- **`B`**: binario
- **`BOOL`**: Boolean
- **`NULL`**: null
- **`M`**: Map
- **`L`**: list
- **`SS`**: set string
- **`NS`**: set numerico
- **`BS`**: set binario

> **Note**
>
> Per una descrizione dettagliata dei tipi di dati di DynamoDB, consulta Tipi di dati.

Dati numerici

I vari linguaggi di programmazione offrono diversi livelli di supporto per JSON. In alcuni casi, potresti decidere di utilizzare una libreria di terze parti per convalidare e analizzare i documenti JSON.

Alcune librerie di terze parti si basano sul tipo di numero JSON, fornendo i propri tipi come `int`, `long` o `double`. Tuttavia, dal momento che il tipo di dati numerico nativo in DynamoDB non viene esattamente mappato a questi altri tipi di dati, pertanto queste distinzioni tra tipi possono causare

conflitti. Inoltre, molte librerie JSON non gestiscono valori numerici a precisione fissa e deducono automaticamente un doppio tipo di dati per le sequenze di cifre che contengono una virgola decimale.

Per risolvere questi problemi, DynamoDB fornisce un singolo tipo numerico senza perdita di dati. Per evitare conversioni implicite indesiderate a un valore double, DynamoDB usa le stringhe per il trasferimento dei dati di valori numerici. Questo approccio fornisce flessibilità per l'aggiornamento dei valori degli attributi pur mantenendo una corretta semantica di ordinamento, ad esempio mettendo i valori "01", "2" e "03" nella sequenza corretta.

Se la precisione del numero è importante per l'applicazione, sarà necessario convertire i valori numerici in stringhe prima di passarli a DynamoDB.

Dati binari

supporta gli attributi binari. Tuttavia, JSON non supporta in modo nativo la codifica dei dati binari. Per inviare dati binari in una richiesta, è necessario codificarli nel formato Base64. Dopo aver ricevuto la richiesta, DynamoDB decodifica i dati Base64 nuovamente in binario.

Lo schema di codifica base64 utilizzato da DynamoDB è descritto in RFC 4648 sul sito Web Internet Engineering Task Force (IETF).

Programmazione di Amazon DynamoDB con Python e Boto3

Questa guida fornisce un orientamento ai programmatori che desiderano utilizzare Amazon DynamoDB con Python. La guida copre i diversi livelli di astrazione, la gestione della configurazione, la gestione degli errori, il controllo delle policy di ripetizione dei tentativi, la gestione del keep-alive e altro ancora.

Argomenti

- Informazioni su Boto
- Utilizzo della documentazione Boto
- Informazioni sui livelli di astrazione del client e delle risorse
- Utilizzo della risorsa della tabella batch_writer
- Ulteriori esempi di codice che esplorano i livelli del client e delle risorse
- Informazioni sull’interazione degli oggetti Client e Resource con sessioni e thread
- Personalizzazione dell’oggetto Config
- Gestione degli errori

- Registrazione
- Hook degli eventi
- Impaginazione e impaginatore
- Waiter

Informazioni su Boto

È possibile accedere a DynamoDB da Python utilizzando l'SDK AWS ufficiale per Python, comunemente noto come Boto3. Il nome Boto deriva da un delfino d'acqua dolce originario del Rio delle Amazzoni. La libreria Boto3 è la terza versione principale della libreria, rilasciata per la prima volta nel 2015. La libreria Boto3 è piuttosto ampia, in quanto supporta tutti i servizi AWS, non solo DynamoDB. Questa guida si rivolge solo alle parti di Boto3 rilevanti per DynamoDB.

Boto è gestito e pubblicato da AWS come progetto open source ospitato su GitHub. È suddiviso in due pacchetti: Botocore e Boto3.

- Botocore fornisce funzionalità di basso livello. In Botocore si trovano il client, la sessione, le credenziali, la configurazione e le classi di eccezione.
- Boto3 si basa su Botocore. Offre un'interfaccia di livello superiore più in linea con Python. In particolare, presenta una tabella DynamoDB come risorsa e offre un'interfaccia più semplice ed elegante rispetto all'interfaccia client di livello inferiore e orientata ai servizi.

Poiché questi progetti sono ospitati su GitHub, è possibile visualizzare il codice sorgente, tenere traccia dei problemi aperti o inviare i propri problemi.

Utilizzo della documentazione Boto

Inizia a esplorare la documentazione Boto con le seguenti risorse:

- Inizia con la sezione Quickstart che costituisce un valido punto di partenza per l'installazione del pacchetto. Consulta questa pagina per istruzioni su come installare Boto3, se non è già installato (Boto3 è spesso disponibile automaticamente nei servizi AWS come AWS Lambda).
- Dopodiché, concentrati sulla guida a DynamoDB della documentazione. La guida mostra come eseguire le attività di base di DynamoDB: creare ed eliminare una tabella, manipolare elementi, eseguire operazioni batch, eseguire una query ed eseguire una scansione. Gli esempi utilizzano l'interfaccia delle risorse. `boto3.resource('dynamodb')` indica che si sta utilizzando l'interfaccia delle risorse di livello superiore.

- Dopo la guida, è possibile consultare la documentazione di riferimento di DynamoDB. Questa landing page fornisce un elenco esaustivo delle classi e dei metodi disponibili. In alto viene mostrata la classe `DynamoDB.Client`. Questa fornisce un accesso di basso livello a tutte le operazioni del piano di controllo (control-plane) e del piano dati. In basso si trova la classe `DynamoDB.ServiceResource`. Questa è l'interfaccia di livello superiore, in linea con Python. Con essa è possibile creare una tabella, eseguire operazioni batch tra tabelle o ottenere un'istanza `DynamoDB.ServiceResource.Table` per azioni specifiche della tabella.

Informazioni sui livelli di astrazione del client e delle risorse

Le due interfacce di lavoro sono l'interfaccia del client e l'interfaccia delle risorse.

- L'interfaccia del client di basso livello fornisce una mappatura uno a uno con l'API del servizio sottostante. Tutte le API offerte da DynamoDB sono disponibili tramite il client. Ciò significa che l'interfaccia del client può fornire funzionalità complete, ma è spesso più verbosa e complessa da usare.
- L'interfaccia delle risorse di livello superiore non fornisce una mappatura uno a uno dell'API del servizio sottostante. Tuttavia, fornisce metodi che semplificano l'accesso al servizio, ad esempio `batch_writer`.

Ecco un esempio di inserimento di un elemento utilizzando l'interfaccia del client. Tutti i valori vengono passati come mappa con la chiave che ne indica il tipo ("S" per stringa, "N" per numero) e il loro valore come stringa. Questo approccio è noto come formato JSON 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 ...
    }
)
```

Ecco la stessa operazione `PutItem` utilizzando l'interfaccia delle risorse. La tipizzazione dei dati è implicita:

```
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 ...
    }
)
```

Se necessario, è possibile eseguire la conversione tra JSON normale e JSON DynamoDB utilizzando le classi `TypeSerializer` e `TypeDeserializer` fornite 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()
    }
```

Ecco come eseguire una query utilizzando l'interfaccia del client. Esprime la query come un costrutto JSON. Utilizza una stringa `KeyConditionExpression` che richiede la sostituzione di variabili per gestire eventuali conflitti tra parole chiave:

```
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 stessa operazione di query utilizzando l'interfaccia delle risorse può essere abbreviata e semplificata:

```
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')
)
```

Come ultimo esempio, si immagini di voler ottenere la dimensione approssimativa di una tabella (ossia i metadati conservati nella tabella che viene aggiornata ogni 6 ore circa). Con l'interfaccia del client, si esegue un'operazione `describe_table()` e si estrae la risposta dalla struttura JSON restituita:

```
import boto3

dynamodb = boto3.client('dynamodb')

response = dynamodb.describe_table(TableName='YourTableName')
```

```
size = response['Table']['TableSizeBytes']
```

Con l'interfaccia delle risorse, la tabella esegue l'operazione di descrizione in modo implicito e presenta i dati direttamente come attributo:

```
import boto3

dynamodb = boto3.resource('dynamodb')

table = dynamodb.Table('YourTableName')
size = table.table_size_bytes
```

> **Note**
>
> Per valutare se sviluppare utilizzando l'interfaccia del client o quella delle risorse, si deve tenere presente che non verranno aggiunte nuove funzionalità all'interfaccia delle risorse in base alla documentazione sulle risorse: "Il team dell'SDK AWS per Python non intende aggiungere nuove funzionalità all'interfaccia delle risorse in boto3. Le interfacce esistenti continueranno a funzionare durante il ciclo di vita di boto3. I clienti possono accedere a nuove funzionalità di servizio tramite l'interfaccia del client".

Utilizzo della risorsa della tabella batch_writer

Una comodità disponibile solo con la risorsa della tabella di livello superiore è `batch_writer`. DynamoDB supporta operazioni di scrittura in batch che consentono fino a 25 operazioni di inserimento o eliminazione in una richiesta di rete. Questo tipo di operazioni in batch migliora l'efficienza riducendo al minimo i round trip di rete.

Con la libreria client di basso livello è possibile utilizzare l'operazione `client.batch_write_item()` per eseguire batch. È necessario suddividere manualmente il lavoro in batch da 25. Dopo ogni operazione, è necessario anche richiedere di ricevere un elenco di elementi non elaborati (alcune operazioni di scrittura potrebbero avere successo mentre altre potrebbero non andare a buon fine). È quindi necessario passare nuovamente gli elementi non elaborati a un'operazione `batch_write_item()` successivamente. È presente una quantità significativa di codice boilerplate.

Il metodo Table.batch_writer crea un gestore di contesto per scrivere oggetti in un batch. Questo presenta un'interfaccia in cui sembra che si stiano scrivendo gli elementi uno alla volta, ma internamente il gestore li memorizza nel buffer e li invia in batch. Gestisce inoltre implicitamente la ripetizione dei tentativi per gli elementi non elaborati.

```
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)
```

Ulteriori esempi di codice che esplorano i livelli del client e delle risorse

È possibile anche fare riferimento ai seguenti repository di esempi di codice che esplorano l'utilizzo delle varie funzioni, utilizzando sia client che risorse:

- Esempi di codice a singola azione AWS ufficiali.
- Esempi di codice orientato agli scenari AWS ufficiali.
- Esempi di codice a singola azione gestiti dalla community.

Informazioni sull'interazione degli oggetti Client e Resource con sessioni e thread

L'oggetto Resource non è thread-safe e non deve essere condiviso tra thread o processi. Consulta la guida su Resource per maggiori dettagli.

L'oggetto Client, al contrario, è generalmente thread-safe, a eccezione di specifiche funzionalità avanzate. Consulta la guida su Client per maggiori dettagli.

L'oggetto Session non è thread-safe. Pertanto, ogni volta che si creano Client o Resource in un ambiente multi-thread, prima sarebbe opportuno creare una nuova Session e quindi creare Client o Resource da Session. Consulta la guida su Session per maggiori dettagli.

Quando si chiama `boto3.resource()`, si sta implicitamente usando la Session predefinita. Ciò è utile per scrivere codice a thread singolo. In caso di scrittura di codice multi-thread, sarà necessario prima costruire una nuova Session per ogni thread e poi recuperare la risorsa da quella Session:

```
# Explicitly create a new Session for this thread
session = boto3.Session()
dynamodb = session.resource('dynamodb')
```

Personalizzazione dell'oggetto Config

Quando si costruisce un oggetto Client o Resource, è possibile passare parametri denominati opzionali per personalizzare il comportamento. Il parametro denominato `config` sblocca una serie di funzionalità. È un'istanza di `botocore.client.Config` e la documentazione di riferimento per Config mostra tutto ciò che espone affinché sia possibile controllarlo. La guida alla configurazione offre una buona panoramica.

> **Note**
>
> È possibile modificare molte di queste impostazioni comportamentali a livello di Session, all'interno del file di configurazione AWS o come variabili di ambiente.

Configurazione per i timeout

Un uso di una configurazione personalizzata è quello di regolare i comportamenti di rete:

- connect_timeout (float o int): il tempo in secondi prima che venga generata un'eccezione di timeout quando si tenta di stabilire una connessione. Il valore predefinito è 60 secondi.
- read_timeout (float o int): il tempo in secondi prima che venga generata un'eccezione di timeout quando si tenta di leggere da una connessione. Il valore predefinito è 60 secondi.

I timeout di 60 secondi sono eccessivi per DynamoDB. Ciò significa che un problema temporaneo della rete causerà un minuto di ritardo per il client prima di poter riprovare. Il codice seguente riduce i timeout a un secondo:

```
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)
```

Per un'utile discussione su questo argomento, consulta Tuning AWS Java SDK HTTP request settings for latency-aware DynamoDB applications. L'SDK per Java ha più configurazioni di timeout rispetto a Python.

Configurazione per keep-alive

Se si sta usando botocore 1.27.84 o successivo, è possibile anche controllare TCP Keep-Alive:

- tcp_keepalive (bool): abilita l'opzione del socket TCP Keep-Alive utilizzata durante la creazione di nuove connessioni se impostata su `True` (impostazione predefinita su `False`). Disponibile solo a partire da botocore 1.27.84.

L'impostazione di TCP Keep-Alive su `True` può ridurre le latenze medie. Ecco un esempio di codice che imposta in base a un condizione il Keep-Alive su true quando si dispone della versione botocore corretta:

```
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 è diverso da HTTP Keep-Alive. Con TCP Keep-Alive, il sistema operativo sottostante invia piccoli pacchetti tramite la connessione socket per mantenere attiva la connessione e rilevare immediatamente eventuali interruzioni. Con HTTP Keep-Alive, la

connessione web costruita sul socket sottostante viene riutilizzata. HTTP Keep-Alive è sempre abilitato con boto3.

Esiste un limite per la quantità di tempo in cui una connessione inattiva può essere mantenuta attiva. Prendi in considerazione l'invio di richieste periodiche, ad esempio ogni minuto, se hai una connessione inattiva ma desideri che la richiesta successiva utilizzi una connessione già stabilita.

Configurazione per le ripetizioni di tentativi

La configurazione accetta anche un dizionario denominato Nuovi tentativi in cui è possibile specificare il comportamento desiderato per la ripetizione dei tentativi. Le ripetizioni di tentativi avvengono all'interno dell'SDK quando l'SDK riceve un errore e l'errore è di tipo transitorio. Se dopo un errore viene effettuata una ripetizione di tentativo internamente, successivamente seguita da una risposta positiva), dal punto di vista del codice non c'è alcun errore, ma solo una latenza leggermente maggiore. Ecco i valori che è possibile specificare:

- max_attempts: un numero intero che rappresenta il numero massimo di ripetizioni dei tentativi che verranno effettuate su una singola richiesta. Ad esempio, impostando questo valore su 2, la richiesta verrà ritentata al massimo due volte dopo la richiesta iniziale. L'impostazione di questo valore su 0 non comporterà alcuna ripetizione dei tentativi dopo la richiesta iniziale.
- total_max_attempts: un numero intero che rappresenta il numero massimo di tentativi totali che verranno effettuati su una singola richiesta. Ciò include la richiesta iniziale, quindi il valore 1 indica che per nessuna richiesta verranno effettuate ripetizioni di tentativi. Se `total_max_attempts` e `max_attempts` sono entrambi forniti, `total_max_attempts` ha la precedenza. `total_max_attempts` è preferito rispetto a `max_attempts` perché è mappato sulla variabile di ambiente `AWS_MAX_ATTEMPTS` e sul valore del file di configurazione `max_attempts`.
- mode: una stringa che rappresenta il tipo di modalità di ripetizione dei tentativi che botocore dovrebbe usare. I valori validi sono:
  - legacy: la modalità predefinita. Attende 50 ms alla prima ripetizione di tentativo, quindi utilizza un backoff esponenziale con un fattore base pari a 2. Per DynamoDB, esegue fino a un massimo di 10 tentativi totali, a meno che non si esegua la sostituzione con quanto sopra).

    > **Note**
    >
    > Con il backoff esponenziale, l'ultimo tentativo attenderà quasi 13 secondi.

- standard: così denominata perché è più coerente con gli altri SDK AWS. Attende un periodo di tempo casuale compreso tra 0 ms e 1.000 ms per la prima ripetizione del tentativo. Se è necessaria una nuova ripetizione del tentativo, seleziona un altro periodo di tempo casuale tra 0 ms e 1.000 ms e lo moltiplica per due. Se è necessaria una nuova ripetizione del tentativo, esegue la stessa scelta casuale moltiplicata per quattro e così via. Ogni attesa è limitata a 20 secondi. Questa modalità eseguirà ripetizioni dei tentativi su un numero maggiore di condizioni di errore rilevate rispetto alla modalità `legacy`. Per DynamoDB, esegue fino a un massimo di tre tentativi totali, a meno che non si esegua la sostituzione con quanto sopra).
- adattiva: una modalità di ripetizione dei tentativi sperimentale che include tutte le funzionalità della modalità standard ma aggiunge la limitazione (della larghezza di banda della rete) automatica lato client. Grazie alla limitazione adattiva della frequenza, gli SDK possono rallentare la frequenza di invio delle richieste per soddisfare meglio la capacità dei servizi AWS. Si tratta di una modalità provvisoria il cui comportamento potrebbe cambiare.

Una definizione estesa di queste modalità di ripetizione dei tentativi è disponibile nella guida alla ripetizione dei tentativi e nell'argomento Retry behavior nella documentazione di riferimento dell'SDK.

Ecco un esempio che utilizza esplicitamente la policy di ripetizione dei tentativi `legacy` con un massimo di tre richieste totali (due ripetizioni di tentativi):

```
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)
```

Poiché DynamoDB è un sistema ad alta disponibilità e bassa latenza, potrebbe essere possibile volere maggiore aggressività con la velocità della ripetizione dei tentativi rispetto a quanto consentito dalle policy di ripetizione dei tentativi integrate. È possibile implementare la propria policy di ripetizione dei tentativi impostando il numero massimo di tentativi su 0, individuando personalmente

le eccezioni ed effettuando ripetizioni dei tentativi, se necessario, utilizzando il proprio codice invece di affidarsi a boto3 per eseguire ripetizioni di tentativi implicite.

Se si gestisce in autonomia la propria policy sulla ripetizione dei tentativi, si consiglia di distinguere tra limitazioni (della larghezza di banda della rete) ed errori:

- Una limitazione (della larghezza di banda della rete) (indicata da `ProvisionedThroughputExceededException` o`ThrottlingException`) indica un servizio funzionante che informa che è stata superata la capacità di lettura o scrittura su una tabella o partizione DynamoDB. Ogni millisecondo che passa viene resa disponibile altra capacità di lettura o scrittura, così che sia possibile effettuare ripetizioni di tentativi rapidamente, ad esempio ogni 50 ms, per tentare di accedere alla capacità appena rilasciata. Con le limitazioni (della larghezza di banda della rete) non è particolarmente necessario un backoff esponenziale, perché queste limitazioni sono leggere da restituire per DynamoDB e non comportano alcun costo per richiesta. Il backoff esponenziale assegna ritardi più lunghi ai thread client che hanno già atteso più a lungo, il che determina un aumento statistico di p50 e p99.
- Un errore (indicato da `InternalServerError` o `ServiceUnavailable`, tra gli altri) indica un problema temporaneo con il servizio. Questo può riguardare l'intera tabella o eventualmente solo la partizione da cui si sta leggendo o su cui si sta scrivendo. In caso di errori, è possibile sospendere più a lungo prima della ripetizione dei tentativi, ad esempio 250 ms o 500 ms, e utilizzare il jitter per scaglionare le ripetizioni di tentativi.

Configurazione per il numero massimo di connessioni nel pool

Infine, la configurazione consente di controllare la dimensione del pool di connessioni:

- max_pool_connections (int): il numero massimo di connessioni da mantenere in un pool di connessioni. Se non viene impostato alcun valore, viene utilizzato il valore predefinito 10.

Questa opzione controlla il numero massimo di connessioni HTTP da conservare in pool per il riutilizzo. Viene mantenuto un pool diverso per Session. Se si prevede che più di 10 thread utilizzino client o risorse creati sulla stessa Session, sarebbe opportuno prendere in considerazione la possibilità di aumentare questo valore, in modo che i thread non debbano attendere che altri thread utilizzino una connessione condivisa.

```
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
```

Gestione degli errori

Le eccezioni dei servizi AWS non sono tutte definite staticamente in Boto3. Ciò avviene perché gli errori e le eccezioni dei servizi AWS variano notevolmente e sono soggetti a modifiche. Boto3 racchiude tutte le eccezioni di servizio come `ClientError` ed espone i dettagli come JSON strutturato. Ad esempio, una risposta di errore potrebbe essere strutturata in questo modo:

```
{
    '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
    }
}
```

Il codice seguente rileva tutte eccezioni `ClientError` e analizza il valore della stringa di `Code` all'interno di `Error` per determinare l'azione da intraprendere:

```
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
```

Alcuni (ma non tutti) i codici di eccezione sono stati materializzati come classi di primo livello. È possibile scegliere di gestirli direttamente. Quando si utilizza l'interfaccia Client, queste eccezioni vengono popolate dinamicamente sul client ed è possibile intercettarle utilizzando l'istanza client in questo modo:

```
except ddb_client.exceptions.ProvisionedThroughputExceededException:
```

Quando si utilizza l'interfaccia Resource, è necessario utilizzare `.meta.client` per passare dalla risorsa al Client sottostante per accedere alle eccezioni, in questo modo:

```
except ddb_resource.meta.client.exceptions.ProvisionedThroughputExceededException:
```

Per esaminare l'elenco dei tipi di eccezioni materializzate, è possibile generarlo dinamicamente:

```
ddb = boto3.client("dynamodb")
print([e for e in dir(ddb.exceptions) if e.endswith('Exception') or
 e.endswith('Error')])
```

Quando si esegue un'operazione di scrittura con un'espressione condizionale, è possibile richiedere che, se l'espressione non va a buon fine, il valore dell'elemento venga restituito nella risposta di errore.

```
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'])
```

Per ulteriori informazioni sulla gestione degli errori e sulle eccezioni:

- La guida di boto3 sulla gestione degli errori contiene ulteriori informazioni sulle tecniche di gestione degli errori.
- La sezione della guida per sviluppatori di DynamoDB sugli errori di programmazione elenca gli errori che si possono riscontrare.
- La sezione sugli errori comuni nella documentazione di riferimento dell'API.
- La documentazione su ciascuna operazione API elenca gli errori che la chiamata potrebbe generare (ad esempio BatchWriteItem).

Registrazione

La libreria boto3 si integra con il modulo di registrazione di log integrato di Python per tracciare ciò che accade durante una sessione. Per controllare i livelli di registrazione di log, è possibile configurare il modulo relativo:

```
import logging

logging.basicConfig(level=logging.INFO)
```

Questo configura il logger root per registrare log di messaggi di livello `INFO` e superiore. I messaggi di registrazione di log che sono meno gravi di questo livello verranno ignorati. I livelli di logging includono `DEBUG`, `INFO`, `WARNING`, `ERROR` e `CRITICAL`. Il valore predefinito è `WARNING`.

I logger in boto3 sono gerarchici. La libreria utilizza alcuni logger diversi, ciascuno corrispondente a diverse parti della libreria. È possibile controllare separatamente il comportamento di ciascuno:

- boto3: il logger principale per il modulo boto3.
- botocore: il logger principale per il pacchetto botocore.
- botocore.auth: per registrare log della creazione di firme AWS per le richieste.

- botocore.credentials: per registrare log del processo di recupero e aggiornamento delle credenziali.
- botocore.endpoint: per registrare log della creazione di richieste prima che vengano inviate in rete.
- botocore.hooks: per registrare log degli eventi attivati nella libreria.
- botocore.loaders: per registrare log quando vengono caricate parti dei modelli di un servizio AWS.
- botocore.parsers: per registrare log delle risposte dei servizi AWS prima che vengano analizzate.
- botocore.retryhandler: per registrare log dell'elaborazione delle ripetizioni di tentativi delle richieste di servizi AWS (modalità legacy).
- botocore.retries.standard: per registrare log dell'elaborazione delle ripetizioni di tentativi delle richieste di servizi AWS (modalità standard o adattiva).
- botocore.utils: per registrare log di varie attività nella libreria.
- botocore.waiter: per registrare log delle funzionalità dei waiter, che effettuano il polling di un servizio AWS fino al raggiungimento di un determinato stato.

Anche altre librerie effettuano la registrazione di log. Internamente, boto3 utilizza l'urllib3 di terze parti per la gestione della connessione HTTP. Quando la latenza è importante, è possibile controllarne i log per assicurarsi che il pool venga utilizzato bene, vedendo quando urllib3 stabilisce una nuova connessione o ne chiude una inattiva.

- urllib3.connectionpool: per registrare log degli eventi di gestione del pool di connessioni.

Il seguente frammento di codice imposta la maggior parte della registrazione di log su `INFO` con la registrazione di log di livello `DEBUG` per l'attività degli endpoint e del pool di connessioni:

```
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)
```

Hook degli eventi

Botocore emette eventi durante varie fasi della sua esecuzione. È possibile registrare i gestori per questi eventi in modo che ogni volta che viene emesso un evento venga richiamato il gestore. Ciò consente di estendere il comportamento di botocore senza dover modificare i componenti interni.

Ad esempio, si supponga di voler tenere traccia di ogni chiamata di un'operazione `PutItem` su qualsiasi tabella DynamoDB dell'applicazione. È possibile registrarsi all'evento `'provide-client-params.dynamodb.PutItem'` per l'intercettazione e la registrazione di log ogni volta che viene invocata un'operazione `PutItem` sulla Session associata. Ecco un esempio:

```
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 ...
    }
)
```

All'interno del gestore, è possibile anche manipolare i parametri a livello di codice per modificare il comportamento:

```
params['TableName'] = "NewTableName"
```

Per ulteriori informazioni sugli eventi, consulta la documentazione botocore sugli eventi e la documentazione boto3 sugli eventi.

Impaginazione e impaginatore

Alcune richieste, come Query e Scan, limitano la dimensione dei dati restituiti in una singola richiesta e richiedono l'esecuzione di richieste ripetute per richiamare le pagine successive.

È possibile controllare il numero massimo di elementi da leggere per ogni pagina con il parametro `limit`. Ad esempio, se si desiderano gli ultimi 10 elementi, è possibile utilizzare `limit` per recuperare solo questi ultimi. Il limite è la quantità che deve essere letta dalla tabella prima di applicare qualsiasi filtro. Non c'è modo di specificare che si desiderano esattamente 10 elementi dopo il filtraggio; è possibile controllare il conteggio prefiltrato e controllare lato client solo quando sono stati effettivamente recuperati 10 elementi. Indipendentemente dal limite, ogni risposta ha sempre una dimensione massima di 1 MB.

Se la risposta include una `LastEvaluatedKey`, ciò indica che la risposta è terminata perché ha raggiunto un limite di conteggio o dimensione. La chiave è l'ultima chiave valutata per la risposta. È possibile recuperare questa `LastEvaluatedKey` e passarla a una chiamata successiva come `ExclusiveStartKey` in modo da leggere il blocco successivo da quel punto di partenza. Quando non viene restituita nessuna `LastEvaluatedKey`, significa che non ci sono più elementi corrispondenti a Query o Scan.

Ecco un semplice esempio (utilizzando l'interfaccia Resource, ma l'interfaccia Client ha lo stesso schema) che legge al massimo 100 elementi per pagina e si ripete fino a quando tutti gli elementi non sono stati letti.

```
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']
```

Per comodità, boto3 può occuparsi di questa operazione attraverso gli Impaginatori. Tuttavia, l'opzione funziona solo con l'interfaccia Client. Ecco il codice riscritto per usare gli Impaginatori:

```
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)
```

Per ulteriori informazioni, consulta la Guida sugli Impaginatori e la documentazione di riferimento dell'API per DynamoDB.Paginator.Query.

> **Note**
>
> Gli Impaginatori hanno anche le proprie impostazioni di configurazione denominate `MaxItems`, `StartingToken` e `PageSize`. Per l'impaginazione con DynamoDB, è necessario ignorare queste impostazioni.

Waiter

I waiter offrono la possibilità di attendere che un'operazione venga completata prima di procedere. Al momento, supportano solo l'attesa della creazione o dell'eliminazione di una tabella. In background, il waiter effettua un controllo ogni 20 secondi fino a 25 volte. È possibile fare questa operazione in autonomia, ma usare un waiter rende elegante il codice di automazione.

Questo codice mostra come attendere la creazione di una particolare tabella:

```
# 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']
```

Per ulteriori informazioni, consulta la Guida ai waiter e il Riferimento sui waiter.

Programmazione di Amazon DynamoDB con JavaScript

Questa guida fornisce un orientamento ai programmatori che desiderano utilizzare Amazon DynamoDB con JavaScript. Informazioni sull'AWS SDK per JavaScript, i livelli di astrazione disponibili, la configurazione delle connessioni, la gestione degli errori, la definizione delle policy di ripetizione dei tentativi, la gestione del keep-alive e altro ancora.

Argomenti

- Informazioni su AWS SDK per JavaScript
- Utilizzo di AWS SDK per JavaScript V3
- Accesso alla documentazione JavaScript
- Livelli di astrazione
- Utilizzo della funzione di utilità marshall
- Lettura degli elementi
- Scritture condizionali
- Impaginazione
- Specificare la configurazione
- Waiter

- Gestione degli errori
- Registrazione
- Considerazioni

Informazioni su AWS SDK per JavaScript

L'AWS SDK per JavaScript fornisce l'accesso ai Servizi AWS negli script del browser o su Node.js. Questa documentazione si concentra sull'ultima versione dell'SDK (V3). L'AWS SDK per JavaScript V3 è gestito da AWS come progetto open source ospitato su GitHub. I problemi e le richieste di funzionalità sono pubblici ed è possibile accedervi nella pagina dei problemi del repository GitHub.

JavaScript V2 è simile a V3, ma contiene differenze di sintassi. V3 è più modulare, facilita l'invio di dipendenze più piccole e dispone di un supporto TypeScript di primo livello. Si consiglia di utilizzare sempre la versione più recente dell'SDK.

Utilizzo di AWS SDK per JavaScript V3

È possibile aggiungere l'SDK all'applicazione Node.js tramite Node Package Manager. Gli esempi seguenti mostrano come aggiungere i pacchetti SDK più comuni per lavorare con DynamoDB.

- `npm install @aws-sdk/client-dynamodb`
- `npm install @aws-sdk/lib-dynamodb`
- `npm install @aws-sdk/util-dynamodb`

L'installazione dei pacchetti aggiunge riferimenti alla sezione relativa alle dipendenze del file di progetto package.json. È possibile utilizzare la sintassi del modulo ECMAScript più recente. Per ulteriori dettagli su questi due approcci, consulta la sezione Considerazioni.

Accesso alla documentazione JavaScript

Inizia a esplorare la documentazione JavaScript con le seguenti risorse:

- Accedi alla Developer guide per la documentazione di base su JavaScript. Le istruzioni di installazione si trovano nella sezione Setting up.
- Accedi alla documentazione di riferimento dell'API per esplorare tutte le classi e i metodi disponibili.
- L'SDK per JavaScript supporta molti Servizi AWS oltre a DynamoDB. Utilizza la seguente procedura per individuare una copertura dell'API specifica per DynamoDB:

1. Da Servizi, seleziona DynamoDB e librerie. Questo documenta il client di basso livello.
2. Seleziona lib-dynamodb. Questo documenta il client di basso livello. I due client rappresentano due diversi livelli di astrazione che è possibile scegliere di utilizzare. Consulta la sezione seguente per ulteriori informazioni sui livelli di astrazione.

Livelli di astrazione

L'SDK per JavaScript V3 ha un client di basso livello (`DynamoDBClient`) e un client di alto livello (`DynamoDBDocumentClient`).

Argomenti

- Client di basso livello (DynamoDBClient)
- Client di alto livello (DynamoDBDocumentClient)

Client di basso livello (**`DynamoDBClient`**)

Il client di basso livello non fornisce astrazioni aggiuntive rispetto al protocollo wire sottostante. Dà il pieno controllo su tutti gli aspetti della comunicazione, ma poiché non ci sono astrazioni, è necessario occuparsi manualmente di aspetti come la definizione degli elementi utilizzando il formato JSON di DynamoDB.

Come illustrato nell'esempio seguente, con questo formato i tipi di dati devono essere indicati in modo esplicito. Una S indica un valore di stringa e una N indica un valore numerico. I numeri in rete vengono sempre inviati come stringhe contrassegnate come tipi di numeri per evitare perdite di precisione. Le chiamate API di basso livello hanno uno schema di denominazione come `PutItemCommand` e `GetItemCommand`.

L'esempio seguente utilizza un client di basso livello con `Item` definito utilizzando DynamoDB JSON:

```
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();
```

Client di alto livello (**`DynamoDBDocumentClient`**)

Il client di documenti di alto livello DynamoDB offre funzionalità pratiche integrate, come l'eliminazione della necessità di eseguire manualmente il marshalling dei dati e la possibilità di letture e scritture dirette utilizzando oggetti JavaScript standard. La documentazione per `lib-dynamodb` illustra l'elenco dei vantaggi.

Per istanziare `DynamoDBDocumentClient`, costruisci un `DynamoDBClient` di basso livello e poi avvolgilo con un `DynamoDBDocumentClient`. La convenzione di denominazione delle funzioni differisce leggermente tra i due pacchetti. Ad esempio, il basso livello utilizza `PutItemCommand`, mentre l'alto livello utilizza `PutCommand`. I nomi distinti consentono a entrambi i set di funzioni di coesistere nello stesso contesto, permettendo di mescolarli entrambi nello stesso 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();
```

Il modello di utilizzo è coerente quando si leggono elementi utilizzando operazioni API come `GetItem`, `Query` o `Scan`.

Utilizzo della funzione di utilità marshall

È possibile utilizzare il client di basso livello ed eseguire manualmente le operazioni di marshall e unmarshall sui tipi di dati. Il pacchetto di utilità util-dynamodb ha una funzione di utilità `marshall()` che accetta JSON e produce DynamoDB JSON, oltre a una funzione `unmarshall()`, che fa l'opposto. L'esempio seguente utilizza il client di basso livello con il marshalling dei dati gestito dalla chiamata `marshall()`.

```
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();
```

Lettura degli elementi

Per leggere un singolo elemento da DynamoDB si utilizza l'operazione API `GetItem`. Analogamente al comando `PutItem`, è possibile scegliere di utilizzare il client di basso livello o il client di documenti di alto livello. L'esempio seguente dimostra l'utilizzo del client di documenti di alto livello per recuperare 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();
```

Utilizza l'operazione API `Query` per leggere più elementi. È possibile utilizzare il client di basso livello o il client di documenti. L'esempio seguente utilizza il client di documenti di alto livello.

```
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();
```

Scritture condizionali

Le operazioni di scrittura di DynamoDB possono specificare un'espressione di condizione logica che deve restituire true affinché la scrittura possa procedere. Se la condizione non restituisce true,

l'operazione di scrittura genera un'eccezione. L'espressione della condizione può verificare se l'elemento esiste già o se i suoi attributi soddisfano determinati vincoli.

```
ConditionExpression = "version = :ver AND size(VideoClip) < :maxsize"
```

Quando l'espressione condizionale non va a buon fine, è possibile utilizzare `ReturnValuesOnConditionCheckFailure` per richiedere che la risposta all'errore includa l'elemento che non soddisfa le condizioni per dedurre quale fosse il problema. Per ulteriori dettagli, consulta 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;
      }
  }
```

Esempi di codice aggiuntivi che mostrano altri aspetti dell'utilizzo dell'SDK per JavaScript V3 sono disponibili nella risorsa relativa alla documentazione dell'SDK per JavaScript V3 e nel repository GitHub DynamoDB-SDK-Examples.

Impaginazione

Argomenti

- Utilizzo del metodo di convenienza paginateScan

Richieste di lettura come `Scan` o `Query` probabilmente restituiranno più elementi in un set di dati. Se si esegue un'operazione `Scan` o `Query` con un parametro `Limit`, una volta che il sistema avrà letto quel numero di elementi, verrà inviata una risposta parziale e sarà necessario eseguire un'impaginazione per recuperare elementi aggiuntivi.

Il sistema leggerà solo un massimo di 1 megabyte di dati per richiesta. Se si include un'espressione `Filter`, il sistema continuerà a leggere al massimo un megabyte di dati dal disco, ma restituirà gli elementi di quel megabyte che corrispondono al filtro. L'operazione di filtro potrebbe restituire 0 elementi per una pagina, ma richiedere comunque un'ulteriore impaginazione prima che la ricerca sia esaurita.

È necessario cercare `LastEvaluatedKey` nella risposta e utilizzarlo come parametro `ExclusiveStartKey` in una richiesta successiva per continuare il recupero dei dati. Questa operazione funge da segnalibro, come indicato nell'esempio seguente.

> **Note**
>
> L'esempio restituisce un `lastEvaluatedKey` nullo come `ExclusiveStartKey` alla prima iterazione e questa operazione è consentita.

Esempio con `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);
```

```
});
```

Utilizzo del metodo di convenienza **`paginateScan`**

L'SDK fornisce metodi pratici denominati `paginateScan` e `paginateQuery` che funzionano automaticamente ed eseguono le richieste ripetute in background. Specifica il numero massimo di elementi da leggere per richiesta utilizzando il parametro standard `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**
>
> L'esecuzione regolare di scansioni complete della tabella non è un modello di accesso consigliato, a meno che la tabella non sia piccola.

Specificare la configurazione

Argomenti

- Configurazione per i timeout
- Configurazione per keep-alive
- Configurazione per le ripetizioni di tentativi

Per la configurazione di `DynamoDBClient`, è possibile specificare varie sostituzioni di configurazione passando un oggetto di configurazione al costruttore. Ad esempio, è possibile specificare la Regione a cui connettersi se non è già nota al contesto di chiamata o all'URL dell'endpoint da utilizzare. Ciò è utile se si desidera indirizzare un'istanza di DynamoDB locale per scopi di sviluppo.

```
const client = new DynamoDBClient({
  region: "eu-west-1",
  endpoint: "http://localhost:8000",
});
```

Configurazione per i timeout

DynamoDB utilizza HTTPS per la comunicazione client-server. È possibile controllare alcuni aspetti del livello HTTP fornendo un oggetto `NodeHttpHandler`. Ad esempio, è possibile regolare i valori di timeout chiave `connectionTimeout` e `requestTimeout`. `connectionTimeout` è la durata massima, in millisecondi, che il client aspetterà mentre tenta di stabilire una connessione prima di rinunciare.

`requestTimeout` definisce per quanto tempo il client aspetterà una risposta dopo l'invio di una richiesta, anch'esso in millisecondi. Le impostazioni predefinite per entrambi sono zero, il che significa che il timeout è disabilitato e non c'è limite al tempo di attesa del client se la risposta non arriva. È necessario impostare i timeout su un valore ragionevole in modo che, in caso di problemi di rete, la richiesta venga annullata e possa essere avviata una nuova richiesta. Per esempio:

```
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**
>
> L'esempio fornito utilizza l'importazione Smithy. Smithy è un linguaggio per definire servizi e SDK, open source e gestito da AWS.

Oltre a configurare i valori di timeout, è possibile impostare il numero massimo di socket, il che consente un numero maggiore di connessioni simultanee per origine. La guida per gli sviluppatori include dettagli sulla configurazione del parametro `maxSockets`.

Configurazione per keep-alive

Quando si utilizza HTTPS, la prima richiesta richiede sempre alcuni scambi preliminari per stabilire una connessione sicura. HTTP Keep-Alive consente alle richieste successive di riutilizzare la connessione già stabilita, rendendo le richieste più efficienti e riducendo la latenza. HTTP Keep-Alive è abilitato per impostazione predefinita con JavaScript V3.

Esiste un limite per la quantità di tempo in cui una connessione inattiva può essere mantenuta attiva. Prendi in considerazione l'invio di richieste periodiche, ad esempio ogni minuto, se hai una connessione inattiva ma desideri che la richiesta successiva utilizzi una connessione già stabilita.

> **Note**
>
> Nella V2, meno recente, dell'SDK, keep-alive era disattivato per impostazione predefinita, ossia ogni connessione veniva chiusa immediatamente dopo l'uso. Se si utilizza V2, è possibile ignorare questa impostazione.

Configurazione per le ripetizioni di tentativi

Quando l'SDK riceve una risposta di errore e l'errore è ripristinabile in base a quanto stabilito dall'SDK, ad esempio un'eccezione di limitazione (della larghezza di banda della rete) o un'eccezione

temporanea di servizio, ripeterà il tentativo. L'operazione avviene in modo invisibile al chiamante, sebbene sia possibile osservare un tempo maggiore per il completamento della richiesta.

L'SDK per JavaScript V3 effettuerà tre richieste totali, per impostazione predefinita, prima di rinunciare e restituire l'errore al contesto chiamante. È possibile modificare il numero e la frequenza di tali ripetizioni di tentativi.

Il costruttore `DynamoDBClient` accetta un'impostazione `maxAttempts` che limita il numero di tentativi da eseguire. L'esempio seguente aumenta il valore dal valore predefinito di 3 a un totale di 5. Se viene impostato su 0 o 1, significa l'assenza di intenzione di effettuare ripetizioni di tentativi automatiche e l'intenzione di gestire in autonomia eventuali errori ripristinabili all'interno del blocco catch.

```
const client = new DynamoDBClient({
  maxAttempts: 5,
});
```

È possibile anche controllare la tempistica delle ripetizioni di tentativi con una strategia relativa personalizzata. A tale scopo, importa il pacchetto di utilità `util-retry` e crea una funzione di backoff personalizzata che calcola il tempo di attesa tra le ripetizioni dei tentativi in base al numero di ripetizioni di tentativi corrente.

L'esempio seguente indica di effettuare un massimo di 5 tentativi con ritardi di 15, 30, 90 e 360 millisecondi se il primo tentativo fallisce. La funzione di backoff personalizzata, `calculateRetryBackoff`, calcola i ritardi accettando il numero di ripetizioni dei tentativi (inizia con 1 per il primo tentativo) e restituisce quanti millisecondi attendere per quella richiesta.

```
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.
  ),
});
```

Waiter

Il client DynamoDB include due utili funzioni waiter che possono essere utilizzate durante la creazione, la modifica o l'eliminazione di tabelle quando si desidera che il codice attenda di procedere fino al termine della modifica della tabella. Ad esempio, è possibile implementare una tabella, chiamare la funzione `waitUntilTableExists` e il codice si bloccherà finché la tabella non sarà resa ACTIVE. Il waiter interroga internamente il servizio DynamoDB con `describe-table` ogni 20 secondi.

```
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 funzionalità `waitUntilTableExists` restituisce il controllo solo quando può eseguire un comando `describe-table` che mostra lo stato della tabella ACTIVE. In questo modo è possibile utilizzare `waitUntilTableExists` per attendere il completamento della creazione, nonché apportare modifiche come l'aggiunta di un GSI, che potrebbe richiedere del tempo prima che la tabella ritorni allo stato ACTIVE.

Gestione degli errori

Nei primi esempi precedenti sono stati individuati tutti gli errori in generale. Tuttavia, nelle applicazioni pratiche, è importante distinguere tra vari tipi di errore e implementare una gestione degli errori più precisa.

Le risposte di errore di DynamoDB contengono metadati, che comprendono il nome dell'errore. È possibile intercettare gli errori e confrontarli con i possibili nomi delle condizioni di errore per stabilire come procedere. Per gli errori lato server, è possibile sfruttare l'operatore `instanceof` con i tipi di errore esportati dal pacchetto `@aws-sdk/client-dynamodb` per dedicarsi in modo efficiente alla gestione degli errori.

È importante notare che questi errori si manifestano solo dopo aver esaurito tutte le ripetizioni di tentativi. Se dopo un errore viene effettuata una ripetizione di tentativo, successivamente

seguita da una chiamata riuscita, dal punto di vista del codice non c'è alcun errore, ma solo una latenza leggermente maggiore. Le ripetizioni di tentativi verranno visualizzate nei grafici di Amazon CloudWatch come richieste non riuscite, ad esempio richieste di limitazione (della larghezza di banda della rete) o di errore. Se il client raggiunge il numero massimo di ripetizioni dei tentativi, rinuncerà e genererà un'eccezione. Con questa modalità il client comunica che non ripeterà i tentativi.

Di seguito è riportato uno snippet per intercettare l'errore e agire in base al tipo di errore restituito.

```
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.
      }
    }
}
```

Consulta the section called "Gestione degli errori" per le stringhe di errore più comuni nella Guida per gli sviluppatori di DynamoDB. Gli errori esatti possibili con una particolare chiamata API sono disponibili nella documentazione relativa a quella chiamata API, come Query API docs.

I metadati degli errori includono proprietà aggiuntive, a seconda dell'errore. Per un `TimeoutError`, i metadati includono il numero di tentativi effettuati e il `totalRetryDelay`, come illustrato di seguito.

```
{
  "name": "TimeoutError",
```

```
  "$metadata": {
    "attempts": 3,
    "totalRetryDelay": 199
  }
}
```

Se si gestisce in autonomia la propria policy sulla ripetizione dei tentativi, si consiglia di distinguere tra limitazioni (della larghezza di banda della rete) ed errori:

- Una limitazione (della larghezza di banda della rete) (indicata da `ProvisionedThroughputExceededException` o `ThrottlingException`) indica un servizio funzionante che informa che è stata superata la capacità di lettura o scrittura su una tabella o partizione DynamoDB. Ogni millisecondo che passa viene resa disponibile altra capacità di lettura o scrittura, così che sia possibile ripetere i tentativi rapidamente, ad esempio ogni 50 ms, per tentare di accedere alla capacità appena rilasciata.

  Con le limitazioni (della larghezza di banda della rete) non è particolarmente necessario un backoff esponenziale, perché queste limitazioni sono leggere da restituire per DynamoDB e non comportano alcun costo per richiesta. Il backoff esponenziale assegna ritardi più lunghi ai thread client che hanno già atteso più a lungo, il che determina un aumento statistico di p50 e p99.
- Un errore (indicato da `InternalServerError` o `ServiceUnavailable`, tra gli altri) indica un problema temporaneo con il servizio, probabilmente l'intera tabella o solo la partizione da cui si sta leggendo o scrivendo. In caso di errori, è possibile sospendere più a lungo prima della ripetizione dei tentativi, ad esempio 250 ms o 500 ms, e utilizzare il jitter per scaglionare le ripetizioni di tentativi.

Registrazione

Attiva la registrazione di log per ottenere maggiori dettagli sulle attività dell'SDK. È possibile impostare un parametro `DynamoDBClient` come mostrato nell'esempio seguente. Nella console verranno visualizzate ulteriori informazioni dei log che includono metadati come il codice dello stato e la capacità utilizzata. Eseguendo il codice in locale tramite una finestra del terminale, i log saranno mostrati direttamente in questo ambiente. Se si esegue il codice in AWS Lambda e i Amazon CloudWatch Logs è stato configurato, l'output della console verrà scritto lì.

```
const client = new DynamoDBClient({
  logger: console
});
```

È anche possibile collegarsi alle attività interne dell'SDK ed eseguire registrazioni di log personalizzate quando si verificano determinati eventi. L'esempio seguente utilizza i dati del client `middlewareStack` per intercettare ogni richiesta inviata dall'SDK e per eseguirne la registrazione di log man mano che avviene.

```
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",
  }
);
```

`MiddlewareStack` fornisce un potente strumento per osservare e controllare il comportamento dell'SDK. Per ulteriori informazioni, consulta il blog Introducing Middleware Stack in Modular AWS SDK per JavaScript.

Considerazioni

Per l'implementazione di AWS SDK per JavaScript in un progetto, ecco alcuni altri fattori da considerare.

Sistemi di moduli

L'SDK supporta due sistemi di moduli, CommonJS ed ES (ECMAScript). CommonJS utilizza la funzione `require`, mentre ES utilizza la parola chiave `import`.

1. Common JS – `const { DynamoDBClient, PutItemCommand } = require("@aws-sdk/client-dynamodb");`
2. ES (ECMAScript – `import { DynamoDBClient, PutItemCommand } from "@aws-sdk/client-dynamodb";`

Il tipo di progetto determina il sistema di moduli da utilizzare ed è specificato nella sezione relativa al tipo del file package.json. L'impostazione predefinita è CommonJS. Utilizza `"type": "module"` per indicare un progetto ES. In caso di un progetto Node.JS esistente che utilizza il formato di pacchetto CommonJS, è comunque possibile aggiungere funzioni con la più moderna

sintassi per l'importazione dell'SDK V3 denominando i file delle funzioni con l'estensione .mjs. Ciò consentirà al file di codice di essere trattato come ES (ECMAScript).

Operazioni asincrone

Molti esempi di codice utilizzano callback e promesse per gestire il risultato delle operazioni di DynamoDB. Con il moderno JavaScript questa complessità non è più necessaria e gli sviluppatori possono sfruttare la sintassi async/await più concisa e leggibile per le operazioni asincrone.

Runtime del browser web

Gli sviluppatori web e gli sviluppatori di applicazioni per dispositivi mobili che utilizzano React o React Native possono utilizzare l'SDK per JavaScript nei propri progetti. Con la versione precedente (V2) dell'SDK, gli sviluppatori web avrebbero dovuto caricare l'SDK completo nel browser, facendo riferimento a un'immagine SDK ospitata su https://sdk.amazonaws.com/js/

Con V3, è possibile raggruppare solo i moduli client V3 e tutte le funzioni JavaScript richiesti in un unico file JavaScript utilizzando webpack e aggiungerlo in un tag script <head> nelle pagine HTML, come spiegato nella sezione Getting started in a browser script della documentazione SDK.

Operazioni del piano dati DAX

Le operazioni del piano dati dell'acceleratore dei flussi Amazon DynamoDB (DAX, DynamoDB Accelerator) sono supportate dall'SDK per JavaScript V3.

Programmazione di DynamoDB con AWS SDK for Java 2.x

Questa guida alla programmazione fornisce un orientamento ai programmatori che desiderano utilizzare Amazon DynamoDB con Java. La guida tratta diversi concetti, tra cui i livelli di astrazione, la gestione della configurazione, la gestione degli errori, il controllo delle policy di ripetizione dei tentativi e la gestione del keep-alive.

Argomenti

- Informazioni su AWS SDK for Java 2.x
- Nozioni di base su AWS SDK for Java 2.x
- Analisi della documentazione di AWS SDK for Java 2.x
- Interfacce supportate
- Ulteriori esempi di codice

- Programmazione sincrona e asincrona
- Client HTTP
- Configurazione di un client HTTP
- Gestione degli errori
- AWS ID della richiesta
- Registrazione dei log
- Paginazione
- Annotazioni sulle classi di dati

Informazioni su AWS SDK for Java 2.x

È possibile accedere a DynamoDB da Java utilizzando il file ufficiale. AWS SDK per Java L'SDK per Java è disponibile in due versioni: 1.x e 2.x. La versione end-of-support 1.x è stata annunciata il 12 gennaio 2024. Entrerà in modalità manutenzione il 31 luglio 2024 e scadrà end-of-support il 31 dicembre 2025. Per le nuove implementazioni, si consiglia vivamente di utilizzare 2.x, che è stata rilasciata per la prima volta nel 2018. In questa guida viene trattata esclusivamente la versione 2.x e vengono illustrate solo le parti dell'SDK relative a DynamoDB.

Per informazioni sulla manutenzione e il supporto di AWS SDKs, consulta la politica di manutenzione di AWS SDK and Tools e la matrice di supporto della versione Tools nella AWS SDKs and Tools Reference AWS SDKs Guide.

AWS SDK for Java 2.x Si tratta di una riscrittura importante del codice base 1.x. L'SDK for Java 2.x supporta le moderne funzionalità Java, come la funzionalità I/O non bloccante introdotta in Java 8. L'SDK per Java 2.x aggiunge anche il supporto per implementazioni di client HTTP collegabili per fornire maggiore flessibilità di connessione di rete e più opzioni di configurazione.

Un cambiamento evidente tra l'SDK per Java 1.x e l'SDK per Java 2.x è l'uso di un nuovo nome pacchetto. L'SDK per Java 1.x utilizza il nome pacchetto `com.amazonaws`, mentre l'SDK per Java 2.x utilizza `software.amazon.awssdk`. Allo stesso modo, gli artefatti Maven per l'SDK per Java 1.x utilizzano il `groupId` `com.amazonaws`, mentre gli artefatti per l'SDK per Java 2.x utilizzano il `groupId` `software.amazon.awssdk`.

> **Important**
>
> La versione AWS SDK per Java 1.x ha un pacchetto DynamoDB denominato. `com.amazonaws.dynamodbv2` La dicitura "v2" nel nome pacchetto non indica che questo è

riservato a Java 2 (J2SE). Piuttosto, “v2” indica che il pacchetto supporta la seconda versione dell’API di basso livello di DynamoDB anziché la versione originale dell’API di basso livello.

Support per le versioni Java

AWS SDK for Java 2.x Fornisce il supporto completo per le versioni Java con supporto a lungo termine (LTS).

Nozioni di base su AWS SDK for Java 2.x

Il seguente tutorial mostra come utilizzare Apache Maven per definire le dipendenze per SDK per Java 2.x. Questo tutorial mostra anche come scrivere il codice che si collega a DynamoDB per elencare le tabelle DynamoDB disponibili. Il tutorial contenuto in questa guida si basa sul tutorial Get started with the AWS SDK for Java 2.x nella AWS SDK for Java 2.x Developer Guide. Questo tutorial è stato modificato per effettuare chiamate a DynamoDB anziché ad Amazon S3.

Per completare questo tutorial, segui questa procedura:

- Fase 1: prepararsi per il tutorial
- Fase 2: creare il progetto
- Fase 3: scrivere il codice
- Fase 4: creare ed eseguire l’applicazione

Fase 1: prepararsi per il tutorial

Prima di iniziare questo tutorial, servirà quanto segue:

- Autorizzazione ad accedere a DynamoDB.
- Un ambiente di sviluppo Java configurato con accesso Single Sign-On all'utilizzo di Servizi AWS . Portale di accesso AWS

Per prepararti per questo tutorial, segui le istruzioni riportate in Setup overview nella AWS SDK for Java 2.x Developer Guide. Dopo aver configurato l’ambiente di sviluppo con l’accesso Single Sign-On per l’SDK per Java e aver svolto una sessione attiva sul portale di accesso AWS, continua con la Fase 2 di questo tutorial.

Fase 2: creare il progetto

Per creare il progetto per questo tutorial, si esegue un comando Maven che richiede input su come configurare il progetto. Dopo aver inserito e confermato tutti gli input, Maven completa la creazione del progetto creando un file `pom.xml` e file Java stub.

1. Apri un terminale o una finestra del prompt dei comandi e accedi a una directory a tua scelta, ad esempio il `Desktop` o la cartella `Home`.
2. Immetti il comando seguente sul terminale, quindi premi Invio.

```
mvn archetype:generate \
   -DarchetypeGroupId=software.amazon.awssdk \
   -DarchetypeArtifactId=archetype-app-quickstart \
   -DarchetypeVersion=2.22.0
```

3. Per ogni prompt, inserisci il valore elencato nella seconda colonna.

Prompt	Valore da immettere
`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>*

Prompt	Valore da immettere
`Define value for property 'package' org.example:`	*<Enter>*

4. Dopo aver inserito l'ultimo valore, Maven elenca le scelte effettuate. Per confermare, inserisci Y (per Yes, Sì). In alternativa, inserisci N (per No), quindi inserisci nuovamente le tue scelte.

Maven crea una cartella di progetto denominata `getstarted` in base al valore `artifactId` inserito. All'interno della cartella `getstarted`, cerca un file con nome `README.md` che è possibile esaminare, un file `pom.xml` e una directory `src`.

Maven crea la seguente struttura di cartelle.

```
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
```

Di seguito viene mostrato il contenuto del file di progetto `pom.xml`.

`pom.xml`

La sezione `dependencyManagement` contiene una dipendenza all' AWS SDK for Java 2.x e la sezione `dependencies` ha una dipendenza per DynamoDB. La specificazione di queste dipendenze

impone a Maven di includere i file `.jar` pertinenti nel percorso della classe Java. Per impostazione predefinita, l' AWS SDK non include tutte le classi per tutti. Servizi AWS Per DynamoDB, se si utilizza l'interfaccia di basso livello, dovrebbe essere presente una dipendenza sull'artefatto `dynamodb`. Oppure, se si utilizza l'interfaccia di alto livello, dovrebbe essere presente sull'artefatto `dynamodb-enhanced`. Se non vengono incluse le dipendenze pertinenti, non è possibile compilare il codice.
Il progetto utilizza Java 1.8 a causa del valore `1.8` delle proprietà `maven.compiler.source` e `maven.compiler.target`.

```
<?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>
```

Fase 3: scrivere il codice

Il codice seguente mostra la classe App creata da Maven. Il metodo `main` è il punto di ingresso nell'applicazione, che crea un'istanza della classe `Handler` e quindi ne richiama il metodo `sendRequest`.

Classe **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 classe `DependencyFactory` creata da Maven contiene il metodo factory `dynamoDbClient` che crea e restituisce un'istanza `DynamoDbClient`. L'istanza `DynamoDbClient` utilizza un'istanza del client HTTP basato su Apache. Questo avviene perché si specifica `apache-client` quando Maven chiede quale client HTTP usare.

Il codice seguente mostra la classe `DependencyFactory`.

Classe 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 classe `Handler` contiene la logica principale del programma. Quando viene creata un'istanza di `Handler` nella classe App, `DependencyFactory` fornisce il servizio client `DynamoDbClient`. Il codice utilizza l'istanza `DynamoDbClient` per chiamare DynamoDB.

Maven genera la seguente classe `Handler` con un commento *`TODO`*. La fase successiva del tutorial sostituisce il commento *`TODO`* con il codice.

Classe **`Handler`**, generata da 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.
    }
}
```

Per compilare la logica, sostituisci l'intero contenuto della classe `Handler` con il codice seguente. Il metodo `sendRequest` viene compilato e vengono aggiunte le importazioni necessarie.

Classe **`Handler`**, implementata

Il codice seguente utilizza l'istanza `DynamoDbClient` per recuperare un elenco di tabelle esistenti. Se esistono tabelle per un determinato account e Regione AWS, il codice utilizza l'istanza `Logger` per registrare log dei nomi di queste tabelle.

```
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));
         }
     }
 }
```

Fase 4: creare ed eseguire l'applicazione

Dopo aver creato il progetto e nel momento in cui contiene la classe `Handler` completa, crea ed esegui l'applicazione.

1. Assicurati di avere una AWS IAM Identity Center sessione attiva. Per confermare, esegui il comando AWS Command Line Interface (AWS CLI) `aws sts get-caller-identity` e controlla la risposta. Se non è presente una sessione attiva, consulta Sign in using the AWS CLI per istruzioni.
2. Apri un terminale o una finestra del prompt dei comandi e vai alla directory del progetto `getstarted`.
3. Pre compilare il progetto esegui il comando seguente:

```
mvn clean package
```

4. Per eseguire l'applicazione, esegui il comando seguente:

```
mvn exec:java -Dexec.mainClass="org.example.App"
```

Dopo aver visualizzato il file, elimina l'oggetto e quindi elimina il bucket.

Riuscito

Se il progetto Maven è stato creato ed eseguito senza errori, congratulazioni! Hai creato con successo la tua prima applicazione Java utilizzando SDK per Java 2.x.

Pulizia

Per eliminare le risorse create durante questo tutorial, elimina la cartella del progetto `getstarted`.

Analisi della documentazione di AWS SDK for Java 2.x

La AWS SDK for Java 2.x Developer Guide copre tutti gli aspetti dell'SDK in tutti i Servizi AWS. Si consiglia di iniziare dagli argomenti seguenti:

- Migrate from version 1.x to 2.x: include una spiegazione dettagliata delle differenze tra 1.x e 2.x. Questo argomento contiene anche istruzioni su come utilizzare entrambe le versioni principali side-by-side.
- DynamoDB guide for Java 2.x SDK: mostra come eseguire le operazioni di base di DynamoDB: creazione di una tabella, manipolazione di elementi e recupero di elementi. Questi esempi utilizzano l'interfaccia di basso livello. Java ha diverse interfacce, come spiegato nella sezione seguente: Interfacce supportate.

> **Tip**
>
> Dopo aver esaminato questi argomenti, aggiungi la documentazione di riferimento dell'API AWS SDK for Java 2.x ai segnalibri. Copre tutto Servizi AWS e ti consigliamo di utilizzarlo come riferimento API principale.

Interfacce supportate

AWS SDK for Java 2.x Supporta le seguenti interfacce, a seconda del livello di astrazione desiderato.

Argomenti in questa sezione

- Interfaccia di basso livello
- Interfaccia di alto livello
- Interfaccia documentale
- Confronto delle interfacce con un esempio Query

Interfaccia di basso livello

L'interfaccia di basso livello fornisce una one-to-one mappatura all'API di servizio sottostante. Ogni API DynamoDB è disponibile tramite questa interfaccia. Ciò significa che l'interfaccia di basso livello può fornire funzionalità complete, ma è spesso più verbosa e complessa da usare. Ad esempio, è necessario utilizzare le funzioni `.s()` per contenere le stringhe e le funzioni `.n()` per contenere i numeri. Il seguente esempio di PutItem inserimento di un elemento utilizzando l'interfaccia di basso livello.

```
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)");
    }
}
```

Interfaccia di alto livello

L'interfaccia di alto livello di AWS SDK for Java 2.x si chiama DynamoDB Enhanced Client. Questa interfaccia offre un'esperienza di creazione del codice più idiomatica.

Il client avanzato offre una modalità di mappatura tra le classi di dati lato client e le tabelle DynamoDB progettate per archiviare tali dati. È possibile definire le relazioni tra le tabelle e le relative classi di modello corrispondenti nel codice. È quindi possibile fare affidamento sull'SDK per gestire la manipolazione dei tipi di dati. Per ulteriori informazioni sul client avanzato, consulta DynamoDB enhanced client API nella AWS SDK for Java 2.x Developer Guide.

L'esempio seguente PutItemutilizza l'interfaccia di alto livello. In questo esempio, `DynamoDbBean` con nome `YourItem` crea un file `TableSchema` che ne consente l'uso diretto come input per la chiamata `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;
        }
    }
}
```

L' AWS SDK per Java 1.x ha una propria interfaccia di alto livello, a cui spesso si fa riferimento con la sua classe principale. `DynamoDBMapper` AWS SDK for Java 2.x È pubblicato in un pacchetto separato (e artefatto Maven) denominato. `software.amazon.awssdk.enhanced.dynamodb` L'SDK per Java 2.x viene spesso identificato tramite la sua classe principale `DynamoDbEnhancedClient`.

Interfaccia di alto livello con classi di dati immutabili

La funzionalità di mappatura fornita dall'API avanzata del client DynamoDB funziona anche con classi di dati immutabili. Una classe immutabile ha solo getter e richiede una classe builder che l'SDK utilizza per creare istanze della classe. L'immutabilità in Java è uno stile privo di effetti collaterali comunemente usato che gli sviluppatori possono utilizzare per creare classi. Queste classi hanno un comportamento più prevedibile in applicazioni multi-thread complesse. Invece di utilizzare l'annotazione `@DynamoDbBean` come mostrato nell'High-level interface example, le classi immutabili utilizzano l'annotazione `@DynamoDbImmutable`, che accetta la classe builder come input.

L'esempio seguente utilizza la classe builder `DynamoDbEnhancedClientImmutablePutItem` come input per creare uno schema della tabella. L'esempio fornisce quindi lo schema come input per la chiamata 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)");
    }
}
```

L'esempio seguente mostra la classe di dati immutabili.

```
@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); }
    }
```

```
}
```

Interfaccia di alto livello che utilizza classi di dati immutabili e librerie di generazione di codice boilerplate di terze parti

Le classi di dati immutabili (mostrate nell'esempio precedente) richiedono codice boilerplate. Ad esempio, la logica di getter e setter sulle classi di dati, oltre alle classi `Builder`. Le librerie di terze parti, come Project Lombok, possono aiutare a generare quel tipo di codice boilerplate. La riduzione della maggior parte del codice standard consente di limitare la quantità di codice necessaria per lavorare con classi di dati immutabili e l'SDK. AWS Ciò determina inoltre una migliore produttività e leggibilità del codice. Per ulteriori informazioni, consulta Use third-party libraries, such as Lombok nella AWS SDK for Java 2.x Developer Guide.

L'esempio seguente mostra come Project Lombok semplifica il codice necessario per utilizzare l'API avanzata del client 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)");
    }
}
```

L'esempio seguente mostra l'oggetto di dati immutabili della classe di dati immutabili.

```
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 `YourImmutableLombokItem` classe utilizza le seguenti annotazioni fornite da Project Lombok e dall'SDK: AWS

- @Builder — Produce un generatore complesso APIs per le classi di dati fornito da Project Lombok.
- @ DynamoDbImmutable — Identifica la `DynamoDbImmutable` classe come annotazione di entità mappabile DynamoDB fornita dall'SDK. AWS
- @Value: la variante immutabile di `@Data`. Per impostazione predefinita, tutti i campi vengono resi privati e definitivi e i setter non vengono generati. Project Lombok fornisce questa annotazione.

Interfaccia documentale

L'interfaccia AWS SDK for Java 2.x Document evita la necessità di specificare i descrittori dei tipi di dati. I tipi di dati sono impliciti nella semantica dei dati stessi. Questa interfaccia Document è simile alla AWS SDK per Java 1.x, interfaccia Document, ma con un'interfaccia riprogettata.

Il seguente Document interface example illustra la chiamata `PutItem` espressa utilizzando l'interfaccia documentale. L'esempio utilizza anche. EnhancedDocument Per eseguire comandi su una tabella DynamoDB utilizzando l'API del documento avanzata, è necessario innanzitutto associare la tabella allo schema della tabella documentale per creare un oggetto risorsa `DynamoDBTable`. Il generatore di schemi della tabella documentale richiede la chiave dell'indice primario e i provider dei convertitori di attributi.

È possibile utilizzare `AttributeConverterProvider.defaultProvider()` per convertire gli attributi documentali di tipi predefiniti. È possibile modificare il comportamento predefinito generale con un'implementazione `AttributeConverterProvider` personalizzata. Inoltre, è possibile modificare il convertitore per un singolo attributo. La AWS SDKs and Tools Reference Guide fornisce maggiori dettagli ed esempi su come utilizzare i convertitori personalizzati. Il loro utilizzo principale è per gli attributi delle classi di dominio che non dispongono di un convertitore predefinito. Utilizzando un convertitore personalizzato, è possibile fornire all'SDK le informazioni necessarie per scrivere o leggere su 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)");
    }

}
```

Per convertire documenti JSON da e verso tipi di dati nativi di Amazon DynamoDB, è possibile utilizzare i seguenti metodi di utilizzo:

- `EnhancedDocument.fromJson(String json)`— Crea una nuova EnhancedDocument istanza da una stringa JSON.
- `EnhancedDocument.toJson()`: crea una rappresentazione in formato stringa JSON del documento che è possibile utilizzare nell'applicazione come qualsiasi altro oggetto JSON.

Confronto delle interfacce con un esempio **`Query`**

Questa sezione mostra la stessa chiamata `Query` espressa utilizzando le varie interfacce. Per ottimizzare i risultati di queste query, tieni presente quanto segue:

- DynamoDB ha come destinazione un valore della chiave di partizione specifico, quindi è necessario specificare la chiave di partizione in modo completo.
- Per fare in modo che la query abbia come destinazione solo gli articoli del carrello, la chiave di ordinamento contiene un'espressione di condizione della chiave che utilizza `begins_with`.
- Si utilizza `limit()` per limitare la richiesta a un massimo di 100 elementi restituiti.

- Si imposta il valore `scanIndexForward` su false. I risultati vengono restituiti in ordine di byte UTF-8, il che di solito significa che l'articolo nel carrello con il numero più basso viene restituito per primo. Impostando il valore `scanIndexForward` su false, si inverte l'ordine e l'articolo nel carrello con il numero più alto viene restituito per primo.
- Si applica un filtro per rimuovere qualsiasi risultato che non corrisponde ai criteri. I dati filtrati consumano la capacità di lettura indipendentemente dal fatto che l'elemento corrisponda al filtro.

Example **`Query`** con l'interfaccia di basso livello

L'esempio seguente esegue la query di una tabella denominata `YourTableName` utilizzando una `keyConditionExpression`. Ciò limita la query a un valore specifico della chiave di partizione e a un valore della chiave di ordinamento che iniziano con un valore di prefisso specifico. Queste condizioni della chiave limitano la quantità di dati letti da DynamoDB. Infine, la query applica un filtro sui dati recuperati da DynamoDB utilizzando 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`** con l'interfaccia documentale

L'esempio seguente esegue la query di una tabella denominata `YourTableName` utilizzando un'interfaccia documentale.

```
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`** con l'interfaccia di alto livello

L'esempio seguente esegue una query di una tabella denominata `YourTableName` utilizzando l'API avanzata del client 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;
        }
    }
}
```

Interfaccia di alto livello con classi di dati immutabili

Quando si esegue un'operazione `Query` con le classi di dati immutabili di alto livello, il codice è lo stesso dell'esempio di interfaccia di alto livello, a eccezione della costruzione della classe di entità `YourItem` o `YourImmutableItem`. Per ulteriori informazioni, consulta l'PutItemesempio.

Interfaccia di alto livello che utilizza classi di dati immutabili e librerie di generazione di codice boilerplate di terze parti

Quando si esegue un'operazione `Query` con le classi di dati immutabili di alto livello, il codice è lo stesso dell'esempio di interfaccia di alto livello, a eccezione della costruzione della classe di entità `YourItem` o `YourImmutableLombokItem`. Per ulteriori informazioni, consulta l'PutItemesempio.

Ulteriori esempi di codice

Per ulteriori esempi di utilizzo di DynamoDB con l'SDK per Java 2.x, fai riferimento ai seguenti repository di esempi di codice:

- Esempi di codice a singola azione AWS ufficiali

- Esempi di codice a singola azione gestiti dalla community
- Esempi di codice ufficiali AWS orientati allo scenario

Programmazione sincrona e asincrona

AWS SDK for Java 2.x Fornisce client sincroni e asincroni per, Servizi AWS come DynamoDB.

Le classi `DynamoDbClient` e `DynamoDbEnhancedClient` forniscono metodi sincroni che bloccano l'esecuzione del thread finché il client non riceve una risposta dal servizio. Questo client è il modo più semplice per interagire con DynamoDB se non sono necessarie operazioni asincrone.

Le classi `DynamoDbAsyncClient` e `DynamoDbEnhancedAsyncClient` forniscono metodi asincroni che terminano immediatamente, riassegnando il controllo al thread chiamante senza attendere una risposta. Il client senza blocchi ha un vantaggio che sfrutta per un'elevata simultaneità su pochi thread, il che consente una gestione efficiente delle richieste di I/O con risorse di calcolo minime. Ciò migliora il throughput e la reattività.

AWS SDK for Java 2.x Utilizza il supporto nativo per I/O non bloccanti. La versione 1.x doveva simulare I/O non bloccanti. AWS SDK per Java

I metodi asincroni terminano prima che sia disponibile una risposta, perciò occorre un modo per ottenere la risposta quando è pronta. I metodi asincroni in AWS SDK per Java restituiscono un `CompletableFuture`oggetto che contiene i risultati dell'operazione asincrona futura. Quando si chiamano query `get()` o `join()` su questi oggetti `CompletableFuture`, il codice si blocca finché il risultato non è disponibile. Se queste query vengono chiamate contemporaneamente alla richiesta, il comportamento è simile a una semplice chiamata sincrona.

Per ulteriori informazioni sulla programmazione asincrona, consulta Use asynchronous programming nella AWS SDK for Java 2.x Developer Guide.

Client HTTP

Per supportare ogni client, esiste un client HTTP che gestisce la comunicazione con i Servizi AWS. È possibile collegare client HTTP alternativi, scegliendone uno con le caratteristiche più adatte alla propria applicazione. Alcuni sono più leggeri, altri hanno più opzioni di configurazione.

Alcuni client HTTP supportano solo l'uso sincrono, mentre altri supportano solo l'uso asincrono. Per un diagramma di flusso che può aiutare a selezionare il client HTTP ottimale per uno specifico carico di lavoro, consulta HTTP client recommendations nella AWS SDK for Java 2.x Developer Guide.

L'elenco seguente presenta alcuni dei possibili client HTTP:

Argomenti

- Client HTTP basato su Apache
- Client HTTP basato su URLConnection
- Client HTTP basato su Netty
- AWS Client HTTP basato su CRT

Client HTTP basato su Apache

La classe `ApacheHttpClient` supporta client di servizio sincroni. È il client HTTP predefinito per l'uso sincrono. Per informazioni sulla configurazione della classe `ApacheHttpClient`, consulta Configure the Apache-based HTTP client nella AWS SDK for Java 2.x Developer Guide.

Client HTTP basato su `URLConnection`

La classe `UrlConnectionHttpClient` è un'altra opzione per i client sincroni. Si carica più velocemente rispetto al client HTTP basato su Apache, ma ha meno funzionalità. Per informazioni sulla configurazione della *`UrlConnectionHttpClient`* classe, consulta Configurare il client HTTP URLConnection basato nella Guida per gli sviluppatori.AWS SDK for Java 2.x

Client HTTP basato su Netty

La classe `NettyNioAsyncHttpClient` supporta client asincroni. È la scelta predefinita per l'uso asincrono. Per informazioni sulla classe `NettyNioAsyncHttpClient`, consulta Configure the Netty-based HTTP client nella AWS SDK for Java 2.x Developer Guide.

AWS Client HTTP basato su CRT

Le nuove `AwsCrtHttpClient` e `AwsCrtAsyncHttpClient` le classi delle librerie AWS Common Runtime (CRT) sono altre opzioni che supportano client sincroni e asincroni. Rispetto ad altri client HTTP, CRT offre: AWS

- Tempo di avvio del'SDK più rapido
- Ingombro di memoria minore
- Tempo di latenza ridotto
- Gestione dell'integrità delle connessioni
- Bilanciamento del carico DNS

Per informazioni sulla configurazione delle *`AwsCrtAsyncHttpClient`* classi *`AwsCrtHttpClient`* and, consultate Configurare i client HTTP AWS basati su CRT nella Guida per gli sviluppatori.AWS SDK for Java 2.x

Il client HTTP AWS basato su CRT non è l'impostazione predefinita perché ciò comprometterebbe la compatibilità con le versioni precedenti delle applicazioni esistenti. Tuttavia, per DynamoDB consigliamo di utilizzare il client HTTP basato su CRT sia per usi AWS sincronizzati che asincroni.

Per un'introduzione al client HTTP AWS basato su CRT, consulta Annuncio della disponibilità del client HTTP AWS CRT nel blog sugli strumenti per sviluppatori. AWS SDK for Java 2.xAWS

Configurazione di un client HTTP

Per la configurazione di un client, è possibile fornire diverse opzioni di configurazione, tra cui:

- Impostazione dei timeout per diversi aspetti delle chiamate API.
- Abilitazione di TCP Keep-Alive.
- Controllo della policy di ripetizione dei tentativi in caso di errori.
- Specifica degli attributi di esecuzione che le istanze degli strumenti di intercettazione dell'esecuzione possono modificare. Gli strumenti di intercettazione dell'esecuzione possono scrivere codice che intercetta l'esecuzione delle richieste e delle risposte dell'API. Ciò consente di eseguire attività come la pubblicazione di metriche e la modifica delle richieste in corso.
- Aggiunta o manipolazione delle intestazioni HTTP.
- Abilitazione del monitoraggio delle metriche delle prestazioni lato client. L'utilizzo di questa funzionalità ti aiuta a raccogliere metriche sui client di servizio nella tua applicazione e ad analizzare l'output in Amazon CloudWatch.
- Specificazione di un servizio di esecuzione alternativo da utilizzare per la pianificazione delle attività, ad esempio la ripetizione dei tentativi e attività di timeout asincroni.

È possibile controllare la configurazione fornendo un oggetto `ClientOverrideConfiguration` alla classe `Builder` del client di servizio. Questo verrà riportato in alcuni esempi di codice nelle sezioni seguenti.

`ClientOverrideConfiguration` fornisce scelte di configurazione standard. I diversi client HTTP collegabili hanno anche possibilità di configurazione specifiche per l'implementazione.

Argomenti in questa sezione

- Configurazione del timeout
- RetryMode
- DefaultsMode
- Configurazione Keep-Alive

Configurazione del timeout

È possibile regolare la configurazione del client per controllare vari timeout relativi alle chiamate del servizio. DynamoDB offre latenze inferiori rispetto ad altri Servizi AWS. Pertanto, si potrebbe voler modificare queste proprietà per ridurre i valori di timeout in modo da poter anticipare l'errore (fail fast) in caso di problemi di rete.

È possibile personalizzare il comportamento relativo alla latenza utilizzando `ClientOverrideConfiguration` sul client DynamoDB o modificando le opzioni di configurazione dettagliate sull'implementazione del client HTTP sottostante.

È possibile configurare le seguenti proprietà di impatto utilizzando `ClientOverrideConfiguration`:

- `apiCallAttemptTimeout`: il tempo di attesa per il completamento di un singolo tentativo di completare una richiesta HTTP prima della rinuncia e dell'interruzione.
- `apiCallTimeout`: il periodo di tempo a disposizione del client per eseguire completamente una chiamata API. Ciò include l'esecuzione del gestore delle richieste che consiste in tutte le richieste HTTP, incluse le ripetizioni di tentativi.

AWS SDK for Java 2.x Fornisce valori predefiniti per alcune opzioni di timeout, come il timeout della connessione e i timeout del socket. L'SDK non fornisce valori predefiniti per i timeout delle chiamate API o i timeout dei singoli tentativi di chiamata API. Se questi timeout non sono impostati in `ClientOverrideConfiguration`, l'SDK utilizza invece il valore di timeout del socket per il timeout complessivo delle chiamate API. La sessione ha un valore di timeout predefinito di 30 secondi.

RetryMode

Un'altra configurazione correlata alla configurazione del timeout da prendere in considerazione è l'oggetto di configurazione `RetryMode`. Questo oggetto di configurazione contiene una raccolta di comportamenti relativi alla ripetizione dei tentativi.

L'SDK per Java 2.x supporta le seguenti modalità di ripetizione dei tentativi:

- `legacy`: la modalità di ripetizione dei tentativi predefinita se non la si modifica esplicitamente. Questa modalità di ripetizione dei tentativi è specifica dell'SDK per Java. È caratterizzata da un massimo di tre ripetizioni di tentativi, o più per servizi come DynamoDB, che prevede fino a otto ripetizioni.
- `standard`— Chiamato «standard» perché è più coerente con gli altri. AWS SDKs Questa modalità attende un periodo di tempo casuale compreso tra 0 ms e 1.000 ms per la prima ripetizione di tentativo. Se è necessaria una nuova ripetizione di tentativo, questa modalità seleziona un altro periodo di tempo casuale da 0 ms a 1.000 ms e lo moltiplica per due. Se è necessaria una nuova ripetizione di tentativo, esegue la stessa scelta casuale moltiplicata per quattro e così via. Ogni attesa è limitata a 20 secondi. Questa modalità esegue ripetizioni di tentativi su un numero maggiore di condizioni di errore rilevate rispetto alla modalità `legacy`. Per DynamoDB, esegue fino a un massimo di tre tentativi, a meno che non si esegua la sostituzione con numRetries.
- `adaptive`: si basa sulla modalità `standard` e limita dinamicamente la frequenza delle richieste AWS per massimizzare la percentuale di successo. Ciò può avvenire a scapito della latenza delle richieste. Non si consiglia la modalità adattiva per la ripetizione dei tentativi quando la latenza prevedibile è importante.

È possibile trovare una definizione estesa di queste modalità di ripetizione nell'argomento sul comportamento Riprova nella Guida di riferimento agli strumenti AWS SDKs e agli strumenti.

Policy di ripetizione dei tentativi

Tutte le configurazioni `RetryMode` hanno una `RetryPolicy`, che è costruita sulla base di una o più configurazioni `RetryCondition`. `TokenBucketRetryCondition` è particolarmente importante per il comportamento di ripetizione dei tentativi dell'implementazione del client SDK di DynamoDB. Questa condizione limita il numero di ripetizioni di tentativi che l'SDK effettua utilizzando un algoritmo token bucket. A seconda della modalità di ripetizione dei tentativi selezionata, le eccezioni di limitazione (della larghezza di banda della rete) possono o meno sottrarre token dal `TokenBucket`.

Quando un client rileva un errore per cui è possibile ripetere il tentativo, ad esempio un'eccezione di limitazione (della larghezza di banda della rete) o un errore temporaneo del server, l'SDK ritenta automaticamente la richiesta. È possibile controllare quante volte e con che velocità avviene la ripetizione dei tentativi.

Quando si configura un client, è possibile fornire un client `RetryPolicy` che supporti i seguenti parametri:

- `numRetries`: il numero massimo di ripetizioni dei tentativi da applicare prima che una richiesta venga considerata non riuscita. Il valore predefinito è 8 indipendentemente dalla modalità di ripetizione dei tentativi utilizzata.

  > **Warning**
  >
  > Assicurati di modificare questo valore predefinito dopo debita considerazione.

- `backoffStrategy`: la `BackoffStrategy` da applicare alla ripetizione dei tentativi (`FullJitterBackoffStrategy` è la strategia predefinita). Questa strategia esegue un ritardo esponenziale tra ripetizioni di tentativi aggiuntive in base al numero di ripetizioni di tentativi correnti, un ritardo di base e un tempo massimo di backoff. Quindi aggiunge il jitter per avere della casualità. Il ritardo di base utilizzato nel ritardo esponenziale è di 25 ms indipendentemente dalla modalità di ripetizione dei tentativi.
- `retryCondition`: `RetryCondition` determina se ripetere i tentativi o meno su una richiesta. Per impostazione predefinita, ripete i tentativi con un set specifico di codici di stato HTTP ed eccezioni per cui ritiene che possano essere effettuate ripetizioni di tentativi. Per la maggior parte delle situazioni, la configurazione predefinita dovrebbe essere sufficiente.

Il codice seguente fornisce una policy di ripetizione dei tentativi alternativa. Specifica un totale di cinque ripetizioni di tentativo (sei richieste totali). La prima ripetizione di tentativo deve avvenire dopo un ritardo di circa 100 ms e ogni ripetizione di tentativo aggiuntiva raddoppia tale tempo in modo esponenziale, fino a un ritardo massimo di un secondo.

```
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

Le proprietà di timeout che `ClientOverrideConfiguration` e `RetryMode` non gestiscono sono in genere configurate implicitamente specificando una `DefaultsMode`.

La AWS SDK for Java 2.x (versione 2.17.102 o successiva) ha introdotto il supporto per. `DefaultsMode` Questa funzionalità fornisce un set di valori predefiniti per le impostazioni configurabili più comuni, come le impostazioni di comunicazione HTTP, il comportamento della ripetizione dei tentativi, le impostazioni Regionali degli endpoint del servizio e potenzialmente qualsiasi configurazione relativa all'SDK. Quando si utilizza questa funzionalità, è possibile ottenere nuove impostazioni di configurazione predefinite personalizzate per scenari di utilizzo comuni.

Le modalità predefinite sono standardizzate in tutti i. AWS SDKs L'SDK per Java 2.x supporta le seguenti modalità predefinite:

- `legacy`: fornisce impostazioni predefinite che variano in base all'SDK AWS e che esistevano prima della creazione di `DefaultsMode`.
- `standard`: fornisce impostazioni predefinite non ottimizzate per la maggior parte degli scenari.
- `in-region`— Si basa sulla modalità standard e include impostazioni personalizzate per le applicazioni che effettuano chiamate Servizi AWS dall'interno della stessa. Regione AWS
- `cross-region`: si basa sulla modalità standard e include impostazioni con timeout elevati per le applicazioni che chiamano Servizi AWS in una Regione diversa.
- `mobile`: si basa sulla modalità standard e include impostazioni con timeout elevati personalizzate per applicazioni mobili con latenze più elevate.
- `auto`: si basa sulla modalità standard e include funzionalità sperimentali. L'SDK tenta di individuare l'ambiente di runtime per determinare automaticamente le impostazioni appropriate. Il rilevamento automatico è basato sull'euristica e non garantisce una precisione del 100%. Se non è possibile determinare l'ambiente di runtime, viene utilizzata la modalità standard. Il rilevamento automatico potrebbe eseguire query sui metadati e i dati utente dell'istanza, il che potrebbe introdurre latenza. Se la latenza di avvio è fondamentale per l'applicazione, si consiglia invece di scegliere una `DefaultsMode` esplicita.

È possibile configurare la modalità predefinita nei seguenti modi:

- Direttamente su un client, tramite `AwsClientBuilder.Builder#defaultsMode(DefaultsMode)`.
- Su un profilo di configurazione, tramite la proprietà del file di profilo `defaults_mode`.

- A livello globale, tramite la proprietà di sistema `aws.defaultsMode`.
- A livello globale, tramite la variabile di ambiente `AWS_DEFAULTS_MODE`.

> **Note**
>
> Per qualsiasi modalità diversa da `legacy`, i valori predefiniti forniti potrebbero cambiare man mano che le best practice si evolvono. Pertanto, in caso di utilizzo di una modalità diversa da `legacy`, si consiglia di eseguire dei test all'aggiornamento dell'SDK.

Le impostazioni predefinite di Smart nella AWS SDKs and Tools Reference Guide forniscono un elenco delle proprietà di configurazione e dei relativi valori predefiniti nelle diverse modalità predefinite.

Scegliete il valore della modalità predefinita in base alle caratteristiche dell'applicazione e al tipo con Servizio AWS cui l'applicazione interagisce.

Questi valori sono configurati tenendo conto di un'ampia gamma di Servizi AWS opzioni. Per un'implementazione tipica di DynamoDB in cui sia le tabelle che l'applicazione DynamoDB sono implementate in un'unica Regione, la modalità predefinita `in-region` è la più rilevante tra le modalità predefinite `standard`.

Example Configurazione del client SDK di DynamoDB ottimizzata per chiamate a bassa latenza

L'esempio seguente regola i timeout su valori inferiori per una chiamata DynamoDB con bassa latenza 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();
```

L'implementazione individuale del client HTTP può fornire un controllo ancora più granulare sul timeout e sul comportamento di utilizzo della connessione. Ad esempio, per il client AWS basato su CRT, è possibile abilitare`ConnectionHealthConfiguration`, che consente al client di monitorare

attivamente lo stato delle connessioni utilizzate. Per ulteriori informazioni, consulta Configurazione avanzata dei client HTTP AWS basati su CRT nella Guida per gli sviluppatori.AWS SDK for Java 2.x

Configurazione Keep-Alive

L'abilitazione del keep-alive può ridurre le latenze riutilizzando le connessioni. Esistono due diversi tipi di keep-alive: HTTP Keep-Alive e TCP Keep-Alive.

- HTTP Keep-Alive tenta di mantenere la connessione HTTPS tra il client e il server in modo che le richieste successive possano riutilizzare tale connessione. In questo modo si evita la pesante autenticazione HTTPS nelle richieste successive. HTTP Keep-Alive è abilitato per impostazione predefinita su tutti i client.
- TCP Keep-Alive richiede che il sistema operativo sottostante invii piccoli pacchetti tramite la connessione del socket per fornire la garanzia aggiuntiva che il socket sia mantenuto attivo e per rilevare immediatamente eventuali interruzioni. Ciò garantisce che una richiesta successiva non perda tempo a cercare di utilizzare un socket interrotto. Per impostazione predefinita, TCP Keep-Alive è disabilitato su tutti i client. Gli esempi di codice seguenti mostrano come abilitarlo su ogni client HTTP. Se abilitato per tutti i client HTTP non basati su CRT, l'effettivo meccanismo di Keep-Alive dipende dal sistema operativo. Pertanto, è necessario configurare valori TCP Keep-Alive aggiuntivi, come il timeout e il numero di pacchetti, tramite il sistema operativo. È possibile farlo usando `sysctl` su Linux o macOS o i valori di registro su Windows.

Example per abilitare TCP Keep-Alive su un client HTTP basato su Apache

```
DynamoDbClient client = DynamoDbClient.builder()
    .httpClientBuilder(ApacheHttpClient.builder().tcpKeepAlive(true))
    .build();
```

Client HTTP basato su **`URLConnection`**

Qualsiasi client sincrono che utilizza il client HTTP basato su `URLConnection` `HttpURLConnection` non dispone di un meccanismo per abilitare il keep-alive.

Example per abilitare TCP Keep-Alive su un client HTTP basato su Netty

```
DynamoDbAsyncClient client = DynamoDbAsyncClient.builder()
    .httpClientBuilder(NettyNioAsyncHttpClient.builder().tcpKeepAlive(true))
    .build();
```

Example per abilitare TCP Keep-Alive su un client HTTP basato su CRT AWS

Con il client HTTP AWS basato su CRT, puoi abilitare TCP keep-alive e controllarne la durata.

```
DynamoDbClient client = DynamoDbClient.builder()
    .httpClientBuilder(AwsCrtHttpClient.builder()
    .tcpKeepAliveConfiguration(TcpKeepAliveConfiguration.builder()
        .keepAliveInterval(Duration.ofSeconds(50))
        .keepAliveTimeout(Duration.ofSeconds(5))
        .build()))
    .build();
```

Quando si utilizza il client DynamoDB asincrono, è possibile abilitare TCP Keep-Alive come illustrato nel codice seguente.

```
DynamoDbAsyncClient client = DynamoDbAsyncClient.builder()
    .httpClientBuilder(AwsCrtAsyncHttpClient.builder()
    .tcpKeepAliveConfiguration(TcpKeepAliveConfiguration.builder()
        .keepAliveInterval(Duration.ofSeconds(50))
        .keepAliveTimeout(Duration.ofSeconds(5))
        .build()))
    .build();
```

Gestione degli errori

Per quanto riguarda la gestione delle eccezioni, AWS SDK for Java 2.x utilizza eccezioni di runtime (non selezionate).

L'eccezione di base, che copre tutte le eccezioni dell'SDK, è `SdkServiceException`, che si estende da `RuntimeException` non controllata di Java. Se si intercetta questa eccezione, sarà possibile intercettare tutte le eccezioni generate dall'SDK.

`SdkServiceException` ha una sottoclasse denominata `AwsServiceException`. Questa sottoclasse indica qualsiasi problema di comunicazione con Servizio AWS. Ha una sottoclasse denominata `DynamoDbException`, che indica un problema nella comunicazione con DynamoDB. Se la si intercetta, sarà possibile intercettare tutte le eccezioni relative a DynamoDB, ma nessun'altra eccezione dell'SDK.

Sono disponibili tipi di eccezioni più specifici in `DynamoDbException`. Alcuni di questi tipi di eccezioni si applicano a operazioni sul piano di controllo (control-plane) come

`TableAlreadyExistsException`. Altri si applicano alle operazioni sul piano dati. Di seguito è riportato un esempio di eccezione comune relativa al piano dati:

- `ConditionalCheckFailedException`: si è specificata una condizione nella richiesta che ha restituito il valore false. Ad esempio, è possibile che si sia provato un aggiornamento condizionale su un elemento, ma il valore effettivo dell'attributo non corrispondesse al valore previsto nella condizione. Per una richiesta che non va a buon fine in questo modo non vengono effettuate ripetizioni di tentativi.

In altre situazioni non viene definita un'eccezione specifica. Ad esempio, quando le richieste vengono sottoposte a limitazione (della larghezza di banda della rete), potrebbe essere limitata la `ProvisionedThroughputExceededException` specifica, mentre in altri casi potrebbe essere limitata la `DynamoDbException` più generica. In entrambi i casi, è possibile determinare se la limitazione (della larghezza di banda della rete) ha causato l'eccezione controllando se `isThrottlingException()` restituisce `true`.

A seconda delle esigenze dell'applicazione, è possibile intercettare tutte le istanze `AwsServiceException` o `DynamoDbException`. Tuttavia, spesso è necessario un comportamento diverso in situazioni diverse. La logica utilizzata per gestire un errore nel controllo delle condizioni è diversa da quella per gestire la limitazione (della larghezza di banda della rete). Definisci quali percorsi eccezionali desideri gestire e assicurati di testare i percorsi alternativi. Questo aiuta ad assicurarsi di poter gestire tutti gli scenari pertinenti.

Per un elenco di errori comuni che si possono riscontrare, consulta Gestione degli errori con DynamoDB. Consulta anche Common Errors in Amazon DynamoDB API Reference. Il riferimento fornisce anche gli errori esatti possibili per ogni operazione API, ad esempio per l'operazione `Query`. Per informazioni sulla gestione delle eccezioni, consulta Exception handling for the AWS SDK for Java 2.x nella AWS SDK for Java 2.x Developer Guide.

AWS ID della richiesta

Ogni richiesta include un ID di richiesta: può essere utile recuperarlo se si sta lavorando con il Supporto AWS per diagnosticare un problema. Ogni eccezione derivata da `SdkServiceException` ha un metodo `requestId()` disponibile per recuperare l'ID della richiesta.

Registrazione dei log

L'utilizzo della registrazione di log fornita dall'SDK può essere utile sia per intercettare messaggi importanti dalle librerie client sia per scopi di debug più approfonditi. I logger sono gerarchici e l'SDK

utilizza `software.amazon.awssdk` come logger root. È possibile configurare il livello con uno dei seguenti: `TRACE`, `DEBUG`, `INFO`, `WARN`, `ERROR`, `ALL` o `OFF`. Il livello configurato si applica a quel logger e verso il basso nella gerarchia dei logger.

Per la sua registrazione, AWS SDK for Java 2.x utilizza Simple Logging Façade for Java (J). SLF4 Questa funge da livello di astrazione rispetto agli altri logger e può essere usata per collegare il logger preferito. Per istruzioni su come collegare i logger, consultate il manuale utente J. SLF4

Ogni logger ha un comportamento particolare. Per impostazione predefinita, il logger Log4j 2.x crea un `ConsoleAppender`, che aggiunge gli eventi del log a `System.out` e i valori predefiniti al livello del log `ERROR`.

Il SimpleLogger logger incluso in SLF4 J emette per impostazione predefinita su `System.err` e imposta di default il livello di log. `INFO`

Si consiglia di impostare il livello su `WARN` per `software.amazon.awssdk` affinché qualsiasi implementazione di produzione intercetti eventuali messaggi importanti dalle librerie client dell'SDK limitando al contempo la quantità di output.

Se SLF4 J non riesce a trovare un logger supportato nel percorso della classe (nessun binding SLF4 J), per impostazione predefinita è un'implementazione senza operazioni. Questa implementazione comporta la registrazione di messaggi che `System.err` spiegano che non sono riuscito a trovare un' SLF4implementazione del logger nel classpath. Per evitare questa situazione, è necessario aggiungere un'implementazione del logger. Per fare ciò, è possibile aggiungere una dipendenza in Apache Maven `pom.xml` verso gli artefatti, come `org.slf4j.slf4j-simple` o `org.apache.logging.log4j.log4j-slf4j2-imp`.

Per informazioni su come configurare la registrazione di log nell'SDK, compresa l'aggiunta di dipendenze di registrazione di log alla configurazione dell'applicazione, consulta Logging with the SDK for Java 2.x in the AWS SDK per Java Developer Guide.

La seguente configurazione nel file `Log4j2.xml` mostra come regolare il comportamento di registrazione di log se si utilizza il logger Apache Log4j 2. Questa configurazione imposta il livello del logger root su `WARN`. Tutti i logger della gerarchia ereditano questo livello di log, incluso il logger `software.amazon.awssdk`.

Per impostazione predefinita, l'output viene inviato a `System.out`. Nell'esempio seguente si sostituisce comunque l'appender di output predefinito Log4j per applicare un Log4j personalizzato `PatternLayout`.

Esempio di un file di configurazione **`Log4j2.xml`**

La seguente configurazione registra log di messaggi sulla console ai livelli `ERROR` e `WARN` per tutte le gerarchie di logger.

```
<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>
```

AWS richiedere la registrazione degli ID

Quando qualcosa va storto, puoi trovare la richiesta IDs tra le eccezioni. Tuttavia, se desideri la richiesta IDs per le richieste che non generano eccezioni, puoi utilizzare la registrazione.

Il `software.amazon.awssdk.request` logger emette la richiesta IDs a livello. `DEBUG` L'esempio seguente estende il precedente configuration example per mantenere il livello del logger root al livello `ERROR`, del `software.amazon.awssdk` al livello `WARN` e del `software.amazon.awssdk.request` al livello `DEBUG`. L'impostazione di questi livelli aiuta a catturare la richiesta IDs e altri dettagli relativi alla richiesta, come l'endpoint e il codice di stato.

```
<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>
```

Di seguito è riportato un esempio di output del log:

```
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
```

Paginazione

Alcune richieste, come `Query` e `Scan`, limitano la dimensione dei dati restituiti in una singola richiesta e richiedono l'esecuzione di richieste ripetute per richiamare le pagine successive.

È possibile controllare il numero massimo di elementi da leggere per ogni pagina con il parametro `Limit`. Ad esempio, è possibile utilizzare il parametro `Limit` per recuperare solo gli ultimi 10 elementi. Questo limite specifica il numero di elementi da leggere dalla tabella prima di applicare qualsiasi filtro. Se si desiderano esattamente 10 elementi dopo il filtraggio, non c'è modo di specificarlo. È possibile controllare solo il conteggio prefiltrato e controllare lato client quando sono stati effettivamente recuperati 10 elementi. Indipendentemente dal limite, le risposte hanno sempre una dimensione massima di 1 MB.

Nella risposta dell'API potrebbe essere inclusa una `LastEvaluatedKey`. Ciò indica che la risposta è terminata perché ha raggiunto un limite di conteggio o di dimensione. Questa chiave è l'ultima chiave valutata per quella risposta. Interagendo direttamente con l'API, è possibile recuperare questa `LastEvaluatedKey` e passarla a una chiamata successiva come `ExclusiveStartKey` in modo da leggere il blocco successivo da quel punto di partenza. Se non viene restituita alcuna `LastEvaluatedKey`, significa che non ci sono più elementi che corrispondono alla chiamata API `Query` o `Scan`.

L'esempio seguente utilizza l'interfaccia di basso livello per limitare gli elementi a 100 in base al parametro `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 Possono semplificare questa interazione con DynamoDB fornendo metodi di impaginazione automatica che effettuano più chiamate di servizio per ottenere automaticamente le pagine successive di risultati. Questo semplifica il codice, ma toglie un certo controllo sull'utilizzo delle risorse che si avrebbe con la lettura manuale delle pagine.

Utilizzando i metodi `Iterable` disponibili nel client DynamoDB, come `QueryPaginator` e `ScanPaginator`, l'SDK si occupa dell'impaginazione. Il tipo restituito da questi metodi è un iterabile personalizzato che è possibile utilizzare per iterare tutte le pagine. L'SDK si occupa di gestire internamente le chiamate del servizio. Utilizzando l'API per i flussi Java, è possibile gestire il risultato di `QueryPaginator` come mostrato nell'esempio seguente.

```
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();
```

Annotazioni sulle classi di dati

L'SDK per Java fornisce diverse annotazioni che è possibile inserire negli attributi della classe di dati. Queste annotazioni influenzano il modo in cui l'SDK interagisce con gli attributi. Aggiungendo un'annotazione, è possibile fare in modo che un attributo si comporti come un contatore atomico implicito, mantenga un valore di timestamp generato automaticamente o tenga traccia del numero di versione di un elemento. Per ulteriori informazioni, consulta Data class annotations.

Gestione degli errori con DynamoDB

Questa sezione descrive gli errori di runtime e come gestirli. Descrive inoltre i messaggi e codici di errore specifici di Amazon DynamoDB. Per un elenco degli errori comuni che si applicano a tutti i servizi AWS, consulta Gestione degli accessi

Argomenti

- Componenti degli errori
- Errori transazionali
- Messaggi e codici di errore
- Gestione degli errori nell'applicazione
- Ripetizione dei tentativi in caso di errore e backoff esponenziale
- Operazioni in batch e gestione degli errori

Componenti degli errori

Quando il tuo programma invia una richiesta, DynamoDB prova a elaborarla. Se la richiesta ha esito positivo, DynamoDB restituisce un codice di stato HTTP di operazione riuscita (`200 OK`), insieme ai risultati dell'operazione richiesta.

Se la richiesta ha esito negativo, DynamoDB restituisce un errore. Ogni errore comprende tre componenti:

- Un codice di stato HTTP (ad esempio, `400`).

- Un nome di eccezione (ad esempio, `ResourceNotFoundException`).
- Un messaggio di errore (ad esempio `Requested resource not found: Table: tablename not found`).

Gli SDK AWS si occupano della propagazione degli errori alla tua applicazione per consentirti di adottare le misure appropriate. Ad esempio, in un programma Java puoi scrivere logica `try-catch` per gestire un errore `ResourceNotFoundException`.

Se non stai utilizzando un SDK AWS, devi analizzare il contenuto della risposta di basso livello da DynamoDB. Di seguito è riportato un esempio di tale risposta:

```
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"}
```

Errori transazionali

Per informazioni sugli errori transazionali, consulta Gestione dei conflitti nelle transazioni in DynamoDB

Messaggi e codici di errore

Di seguito è riportato un elenco di eccezioni restituite da DynamoDB, raggruppate in base al codice di stato HTTP. Se OK to retry? (OK riprovare?) è Yes (Sì), puoi inviare nuovamente la stessa richiesta. Se OK to retry? (OK riprovare?) è No, è necessario risolvere il problema sul lato client prima di inviare una nuova richiesta.

Codice di stato HTTP 400

Un codice di stato HTTP `400` indica un problema della richiesta, ad esempio un errore di autenticazione, l'assenza di parametri obbligatori o il superamento del throughput assegnato di una tabella. È necessario risolvere il problema nella tua applicazione prima di inviare nuovamente la richiesta.

AccessDeniedException

Messaggio: Access denied.

Il client non ha firmato correttamente la richiesta. Se utilizzi un SDK AWS, le richieste vengono firmate automaticamente; in caso contrario, consulta Processo di firma in Signature version 4 in Riferimenti generali di AWS.

OK riprovare? No

ConditionalCheckFailedException

Messaggio: The conditional request failed.

Hai specificato una condizione che ha restituito il valore false. Ad esempio, è possibile che si sia provato un aggiornamento condizionale su un elemento, ma il valore effettivo dell'attributo non corrispondesse al valore previsto nella condizione.

OK riprovare? No

IncompleteSignatureException

Messaggio: The request signature does not conform to AWS standards.

La firma della richiesta non include tutti i componenti obbligatori. Se utilizzi un SDK AWS, le richieste vengono firmate automaticamente; in caso contrario, consulta Processo di firma in Signature Version 4 in Riferimenti generali di AWS.

OK riprovare? No

ItemCollectionSizeLimitExceededException

Messaggio: Collection size exceeded.

Per una tabella con un indice secondario locale, un gruppo di elementi con lo stesso valore di chiave di partizione ha superato il limite massimo di dimensione pari a 10 GB. Per ulteriori informazioni sulle raccolte di item, consulta Raccolte di elementi negli indici secondari locali.

OK riprovare? Sì

LimitExceededException

Messaggio: Too many operations for a given subscriber.

Ci sono troppe operazioni di piano di controllo simultanee. Il numero complessivo di tabelle e indici in stato `CREATING`, `DELETING` o `UPDATING` non può superare 500.

OK riprovare? Sì

MissingAuthenticationTokenException

Messaggio: Request must contain a valid (registered) AWS Access Key ID.

La richiesta non include l'intestazione di autorizzazione richiesta o il formato non è corretto. Consultare API DynamoDB di basso livello.

OK riprovare? No

ProvisionedThroughputExceededException

Messaggio: You exceeded your maximum allowed provisioned throughput for a table or for one or more global secondary indexes. To view performance metrics for provisioned throughput vs. consumed throughput, open the Amazon CloudWatch console.

L'errore include un elenco di campi `ThrottlingReason` che forniscono un contesto specifico sul motivo per cui si è verificata la limitazione (della larghezza di banda della rete), in base al formato `ResourceType+OperationType+LimitType` (ad esempio, `TableReadProvisionedThroughputExceeded`) e all'ARN della risorsa interessata. Ciò consente di identificare quale risorsa è sottoposta a limitazione (della larghezza di banda della rete) (tabella o indice), quale tipo di operazione ha attivato la limitazione (lettura o scrittura) e il limite specifico che è stato superato (in questo caso: capacità allocata). Per ulteriori informazioni sulla diagnosi e la risoluzione dei problemi relativi alla limitazione (della larghezza di banda della rete), consulta Diagnosi della limitazione (della larghezza di banda della rete).

Esempio: la frequenza di richieste è troppo elevata. Gli SDK AWS per DynamoDB riprovano automaticamente le richieste che ricevono questa eccezione. La richiesta ha infine esito positivo, a meno che la coda dei tentativi ripetuti sia troppo estesa per terminare. Riduci la frequenza delle richieste utilizzando Ripetizione dei tentativi in caso di errore e backoff esponenziale.

OK riprovare? Sì

ReplicatedWriteConflictException

Messaggio: One or more items in this request are being modified by a request in another Region.

Esempio: è stata richiesta un'operazione di scrittura per un elemento in una tabella globale a elevata consistenza multi-Regione (MRSC) che è attualmente in fase di modifica da una richiesta in un'altra Regione.

OK riprovare? Sì

RequestLimitExceeded

Messaggio: Throughput exceeds the current throughput limit for your account. To request a limit increase, contact AWS Support at https://aws.amazon.com/support (La velocità di trasmissione effettiva supera il limite corrente per l'account. Per richiedere un aumento del limite, contatta il supporto all'indirizzo https://aws.amazon.com/support.).

L'errore include un elenco di campi `ThrottlingReason` che forniscono un contesto specifico sul motivo per cui si è verificata la limitazione (della larghezza di banda della rete), in base al formato `ResourceType+OperationType+LimitType` (ad esempio, `TableWriteAccountLimitExceeded` o `IndexReadAccountLimitExceeded`) e all'ARN della risorsa interessata. Ciò consente di identificare quale risorsa viene sottoposta a limitazione (della larghezza di banda della rete) (tabella o indice), quale tipo di operazione ha attivato la limitazione (lettura o scrittura) e se sono state superate le quote di servizio a livello di account. Per ulteriori informazioni sulla diagnosi e la risoluzione dei problemi relativi alla limitazione (della larghezza di banda della rete), consulta Diagnosi della limitazione (della larghezza di banda della rete).

Esempio: la frequenza delle richieste on demand supera la velocità di trasmissione effettiva dell'account consentita e alla tabella non può essere applicato un ulteriore dimensionamento.

OK riprovare? Sì

ResourceInUseException

Messaggio: The resource which you are attempting to change is in use.

Esempio: hai provato a ricreare una tabella esistente o a eliminare una tabella attualmente in stato `CREATING`.

OK riprovare? No

ResourceNotFoundException

Messaggio: Requested resource not found.

Esempio: la tabella richiesta non esiste o si trova nella prima fase dello stato `CREATING`.

OK riprovare? No

ThrottlingException

Messaggio: Rate of requests exceeds the allowed throughput.

Questa eccezione viene restituita come risposta AmazonServiceException con un codice di stato THROTTLING_EXCEPTION. Questa eccezione potrebbe essere restituita se si eseguono operazioni API del piano di controllo troppo rapidamente.

Per le tabelle che utilizzano la modalità on demand, questa eccezione potrebbe essere restituita per qualsiasi operazione API del piano dati se la frequenza di richiesta è troppo alta. Per ulteriori informazioni sul dimensionamento on demand, consulta Throughput e proprietà di dimensionamento.

L'errore include un elenco di campi `ThrottlingReason` che forniscono un contesto specifico sul motivo per cui si è verificata la limitazione (della larghezza di banda della rete), in base al formato `ResourceType+OperationType +LimitType` (ad esempio, `TableReadKeyRangeThroughputExceeded` o `IndexWriteMaxOnDemandThroughputExceeded`) e all'ARN della risorsa interessata. Queste informazioni consentono di identificare quale risorsa viene sottoposta a limitazione (della larghezza di banda della rete) (tabella o indice), quale tipo di operazione ha attivato la limitazione (lettura o scrittura) e il limite specifico che è stato superato (limiti di partizione o throughput massimo on demand). Per ulteriori informazioni sulla diagnosi e la risoluzione dei problemi relativi alla limitazione (della larghezza di banda della rete), consulta Diagnosi della limitazione (della larghezza di banda della rete).

OK riprovare? Sì

UnrecognizedClientException

Messaggio: The Access Key ID or security token is invalid.

La firma della richiesta non è corretta. La causa più probabile è un ID chiave di accesso AWS o una chiave segreta non valida.

OK riprovare? Sì

ValidationException

Messaggio: variabile, in base all'errore o agli errori specifici rilevati

Questo errore può verificarsi per vari motivi, ad esempio per un parametro obbligatorio mancante, un valore non compreso nell'intervallo valido o tipi di dati non corrispondenti. Il messaggio di errore contiene dettagli sulla parte specifica della richiesta che ha causato l'errore.

OK riprovare? No

Codice di stato HTTP 5xx

Un codice di stato HTTP `5xx` indica un problema che deve essere risolto da AWS. Potrebbe trattarsi di un errore temporaneo, nel qual caso puoi riprovare la richiesta finché non riesce. In caso contrario, accedi al AWS Service Health Dashboard per verificare se ci sono problemi operativi relativi al servizio.

Per ulteriori informazioni, consulta Come risolvere gli errori HTTP 5xx in Amazon DynamoDB?

InternalServerError (HTTP 500)

DynamoDB non è riuscito a elaborare la richiesta.

OK riprovare? Sì

> **Note**
>
> Puoi rilevare errori interni del server durante la gestione degli item. Sono previsti nel corso della durata di una tabella. Le richieste non riuscite possono essere riprovate immediatamente.
> Quando ricevi un codice di stato 500 in un'operazione di scrittura, l'operazione potrebbe essere stata eseguita correttamente o non essere riuscita. Se l'operazione di scrittura è una richiesta `TransactWriteItem`, è possibile riprovare a eseguire l'operazione.
> Se l'operazione di scrittura è una richiesta di scrittura con un elemento singolo, ad

esempio `PutItem`, `UpdateItem` o `DeleteItem`, la tua applicazione dovrebbe leggere lo stato dell'elemento prima di riprovare l'operazione e/o utilizzare Esempio di CLI di espressione condizionale in DynamoDB per garantire che l'elemento rimanga in uno stato corretto dopo aver riprovato, indipendentemente dal fatto che l'operazione precedente abbia avuto esito positivo o non sia riuscita. Se l'idempotenza è un requisito per l'operazione di scrittura, utilizza `TransactWriteItem`, che supporta le richieste idempotenti specificando automaticamente un `ClientRequestToken` per disambiguare più tentativi di eseguire la stessa azione.

ServiceUnavailable (HTTP 503)

DynamoDB al momento non è disponibile. Dovrebbe trattarsi di uno stato temporaneo.

OK riprovare? Sì

Gestione degli errori nell'applicazione

Per il buon funzionamento dell'applicazione devi aggiungere logica che intercetti gli errori e risponda in modo adeguato. Gli approcci tipici includono l'utilizzo di blocchi `try-catch` o di istruzioni `if-then`.

Gli SDK AWS eseguono le proprie ripetizioni di tentativi e le proprie verifiche degli errori. Se rilevi un errore durante l'utilizzo di uno degli SDK AWS, il codice e la descrizione dell'errore possono aiutarti a risolverlo.

Dovresti anche vedere un `Request ID` nella risposta. L'`Request ID` può essere utile se devi collaborare con AWS Support per diagnosticare un problema.

Ripetizione dei tentativi in caso di errore e backoff esponenziale

Numerosi componenti di una rete, ad esempio server DNS, switch, sistemi di bilanciamento del carico e altri, possono generare errori in qualsiasi fase del ciclo di vita di una richiesta specifica. La tecnica che viene generalmente utilizzata per gestire queste risposte di errore in un ambiente di rete consiste nell'implementare nuovi tentativi nell'applicazione client. Questa tecnica aumenta l'affidabilità dell'applicazione.

Ogni SDK AWS implementa automaticamente la logica di ripetizione dei tentativi. Puoi modificare i parametri delle ripetizioni in base alle tue esigenze. Considera ad esempio un'applicazione Java

che richiede una strategia fail-fast, senza tentativi ripetuti in caso di errore. Con l'AWS SDK per Java potresti utilizzare la classe `ClientConfiguration` e specificare un valore `maxErrorRetry` uguale a `0` per disattivare i tentativi. Per ulteriori informazioni, consulta la documentazione sugli SDK AWS relativa al tuo linguaggio di programmazione.

Se non utilizzi un SDK AWS, dovresti riprovare le richieste originali che ricevono errori del server (5xx). Tuttavia, gli errori del client (4xx, tranne `ThrottlingException` o `ProvisionedThroughputExceededException`) indicano che devi modificare la richiesta per correggere il problema prima di riprovare. Per consigli dettagliati per affrontare scenari di limitazione (della larghezza di banda della rete) specifici, consulta la sezione Risoluzione dei problemi relativi alla limitazione (della larghezza di banda della rete) in DynamoDB.

Oltre a semplici tentativi, ogni SDK AWS implementa l'algoritmo di backoff esponenziale per migliorare il controllo del flusso. Il concetto che sottende al backoff esponenziale è di utilizzare attese progressivamente più lunghe tra i tentativi per le risposte di errore consecutive. Ad esempio, fino a 50 millisecondi prima del primo nuovo tentativo, fino a 100 millisecondi prima del secondo, fino a 200 millisecondi prima del terzo e così via. Tuttavia, dopo un minuto, se la richiesta non è riuscita, il problema potrebbe essere dovuto al fatto che la dimensione della richiesta ha superato il throughput assegnato e non essere causato dalla frequenza della richiesta. Imposta l'arresto del numero massimo di tentativi a circa un minuto. If the request is not successful, investigate your provisioned throughput options.

> **Note**
>
> Gli SDK AWS implementano automaticamente la logica di ripetizione dei tentativi e il backoff esponenziale.

La maggior parte degli algoritmi di backoff esponenziale utilizza il jitter (ritardo randomizzato) per evitare collisioni successive. Poiché in questi casi non stai provando a evitare tali collisioni, non è necessario utilizzare questo numero casuale. Tuttavia, se utilizzi client simultanei, il jitter può aiutare a completare più velocemente le richieste. Per ulteriori informazioni, consulta il post del blog relativo al jitter e al backoff esponenziale.

Operazioni in batch e gestione degli errori

L'API di basso livello di DynamoDB supporta operazioni in batch per le letture e le scritture. `BatchGetItem` legge elementi da una o più tabelle e `BatchWriteItem` inserisce o elimina gli

elementi in una o più tabelle. Queste operazioni in batch vengono implementate come wrapper di altre operazioni DynamoDB non batch. In altre parole, `BatchGetItem` richiama `GetItem` una volta per ogni item nel batch. Analogamente, `BatchWriteItem` richiama `DeleteItem` o `PutItem`, in base al caso, per ogni item nel batch.

Un'operazione batch può tollerare l'errore di singole richieste nel batch. Considera ad esempio una richiesta `BatchGetItem` per la lettura di cinque item. La mancata elaborazione di alcune richieste `GetItem` sottostanti non comporta l'errore dell'intera operazione `BatchGetItem`. Tuttavia, se tutte e cinque le operazioni non riescono, l'intero `BatchGetItem` non riesce.

Le operazioni batch restituiscono informazioni sulle singole richieste non riuscite, in modo che tu possa diagnosticare il problema e riprovare l'operazione. Per `BatchGetItem`, le tabelle e le chiavi primarie in questione vengono restituite nel valore `UnprocessedKeys` della risposta. Per `BatchWriteItem`, informazioni simili vengono restituite in `UnprocessedItems`.

La causa più probabile della mancata riuscita di una lettura o scrittura è il throttling. Per `BatchGetItem`, una o più tabelle nella richiesta batch non dispongono di capacità di lettura assegnata sufficiente a supportare l'operazione. Per `BatchWriteItem`, una o più tabelle nella richiesta batch non dispongono di capacità di scrittura assegnata sufficiente.

Se DynamoDB restituisce elementi non elaborati, dovresti riprovare l'operazione in batch su quegli elementi. Tuttavia, è fortemente consigliabile utilizzare un algoritmo di backoff esponenziale. Se riprovi l'operazione batch immediatamente, le richieste di lettura o scrittura sottostanti possono ancora non riuscire a causa del throttling sulle singole tabelle. Se ritardi le operazioni in batch utilizzando il backoff esponenziale, le singole richieste nel batch avranno una probabilità di riuscita molto maggiore.

Utilizzo di DynamoDB con un SDK AWS

I Software Development Kit (SDK) di AWS sono disponibili per molti dei linguaggi di programmazione più diffusi. Ogni SDK fornisce un'API, esempi di codice, e documentazione che facilitano agli sviluppatori la creazione di applicazioni nel loro linguaggio preferito.

Documentazione sugli SDK	Esempi di codice
AWS SDK per C++	Esempi di codice AWS SDK per C++
AWS CLI	Esempi di codice AWS CLI

Documentazione sugli SDK	Esempi di codice
AWS SDK per Go	Esempi di codice AWS SDK per Go
AWS SDK per Java	Esempi di codice AWS SDK per Java
AWS SDK per JavaScript	Esempi di codice AWS SDK per JavaScript
AWS SDK per Kotlin	Esempi di codice AWS SDK per Kotlin
AWS SDK per .NET	Esempi di codice AWS SDK per .NET
AWS SDK per PHP	Esempi di codice AWS SDK per PHP
AWS Strumenti per PowerShell	Esempi di codice AWS Strumenti per PowerShell
AWS SDK for Python (Boto3)	Esempi di codice AWS SDK for Python (Boto3)
AWS SDK per Ruby	Esempi di codice AWS SDK per Ruby
AWS SDK for Rust	Esempi di codice AWS SDK for Rust
AWS SDK for SAP ABAP	Esempi di codice AWS SDK for SAP ABAP
SDK AWS per Swift	Esempi di codice SDK AWS per Swift

Per esempi specifici relativi a DynamoDB, consulta Esempi di codice per l'utilizzo di DynamoDB AWS SDKs.

> **Esempio di disponibilità**
>
> Non riesci a trovare quello che ti serve? Richiedi un esempio di codice utilizzando il link Provide feedback (Fornisci un feedback) nella parte inferiore di questa pagina.

Utilizzo di tabelle, elementi, query, scansioni e indici

In questa sezione sono forniti i dettagli di utilizzo di tabelle, elementi, query e altri elementi in Amazon DynamoDB.

Argomenti

- Utilizzo di tabelle e dati in DynamoDB
- Tabelle globali: replica multi-attiva e multi-Regione
- Utilizzo di elementi e attributi in DynamoDB
- Miglioramento dell'accesso ai dati con gli indici secondari in DynamoDB
- Gestione di flussi di lavoro complessi con transazioni DynamoDB
- Change Data Capture con Amazon DynamoDB

Utilizzo di tabelle e dati in DynamoDB

In questa sezione viene descritto come utilizzare AWS Command Line Interface (AWS CLI) e gli SDK AWS per creare, aggiornare ed eliminare tabelle in Amazon DynamoDB.

> **Note**
>
> Puoi eseguire queste stesse attività utilizzando la Console di gestione AWS. Per ulteriori informazioni, consulta Utilizzo della console.

In questa sezione sono fornite inoltre ulteriori informazioni sulla capacità di throughput tramite l'uso della scalabilità automatica di DynamoDB o l'impostazione manuale della velocità effettiva assegnata.

Argomenti

- Operazioni di base sulle tabelle DynamoDB
- Considerazioni sulla scelta di una classe di tabella in DynamoDB
- Aggiunta di tag ed etichette alle risorse in DynamoDB

Operazioni di base sulle tabelle DynamoDB

Analogamente ad altri sistemi di database, Amazon DynamoDB archivia i dati in tabelle. Puoi gestire le tabelle utilizzando alcune operazioni di base.

Argomenti

- Creazione di una tabella
- Descrizione di una tabella
- Aggiornamento di una tabella
- Eliminazione di una tabella
- Uso della protezione da eliminazione
- Elenco dei nomi delle tabelle
- Descrizione delle quote di velocità di trasmissione effettiva assegnate

Creazione di una tabella

Utilizza l'operazione `CreateTable` per creare una tabella in Amazon DynamoDB. Per creare la tabella, è necessario fornire le informazioni riportate di seguito:

- Nome tabella. Il nome deve essere conforme alle regole di denominazione di DynamoDB e deve essere unico nell'account e nella regione AWS correnti. Ad esempio, è possibile creare una tabella `People` nella regione Stati Uniti orientali (Virginia settentrionale) e un'altra tabella `People` in Europa (Irlanda). Tuttavia, queste due tabelle sarebbero totalmente diverse l'una dall'altra. Per ulteriori informazioni, consulta Tipi di dati e regole di denominazione supportati in Amazon DynamoDB.
- Chiave primaria. La chiave primaria può consistere di un attributo (chiave di partizione) o due attributi (chiave di partizione e chiave di ordinamento). Devi fornire i nomi e i tipi di dati degli attributi e il ruolo di ciascun attributo: `HASH` (per una chiave di partizione) e `RANGE` (per una chiave di ordinamento). Per ulteriori informazioni, consulta Chiave primaria.
- Impostazioni di throughput (per tabelle assegnate). Se utilizzi la modalità assegnata, devi specificare le impostazioni iniziali di throughput di lettura e scrittura per la tabella. Queste impostazioni possono essere modificate successivamente oppure è possibile abilitare la scalabilità automatica di DynamoDB perché le impostazioni vengano gestite automaticamente. Per ulteriori informazioni, consultare Modalità con capacità allocata di DynamoDB e Gestione automatica della capacità effettiva di trasmissione con il dimensionamento automatico di DynamoDB.

Esempio 1: creazione di una tabella on demand

Per creare la stessa tabella `Music` utilizzando la modalità on demand:

```
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
```

L'operazione `CreateTable` restituisce i metadati per la tabella, come illustrato di seguito:

```
{
    "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**
>
> Quando si richiama `DescribeTable` su una tabella on demand, le unità di capacità in lettura e le unità di capacità in scrittura sono impostate su 0.

Esempio 2: creazione di una tabella assegnata

L'esempio AWS CLI seguente mostra come creare una tabella (`Music`). La chiave primaria è costituita da `Artist` (chiave di partizione) e `SongTitle` (chiave di ordinamento), entrambe con tipo di dati `String`. Il throughput massimo per questa tabella è 10 unità di capacità di lettura e 5 unità di capacità di scrittura.

```
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
```

L'operazione `CreateTable` restituisce i metadati per la tabella, come illustrato di seguito:

```
{
    "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
    }
}
```

L'elemento `TableStatus` indica lo stato corrente della tabella (`CREATING`). La creazione della tabella potrebbe richiedere del tempo, in base ai valori specificati per `ReadCapacityUnits` e `WriteCapacityUnits`. Valori maggiori richiedono l'allocazione di più risorse per la tabella da parte di DynamoDB.

Esempio 3: creazione di una tabella utilizzando la classe di tabella DynamoDB Standard (accesso infrequente)

Crea la stessa tabella `Music` utilizzando la classe di tabella DynamoDB Standard (accesso infrequente).

```
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
```

L'operazione `CreateTable` restituisce i metadati per la tabella, come illustrato di seguito:

```
{
    "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
    }
}
```

Descrizione di una tabella

Per visualizzare i dettagli su una tabella, utilizza l'operazione `DescribeTable`. Devi specificare il nome della tabella. Il formato dell'output da `DescribeTable` è identico a quello da `CreateTable`. Include il timestamp di creazione della tabella, lo schema delle chiavi, le impostazioni della velocità effettiva assegnata, le dimensioni stimate e gli eventuali indici secondari presenti.

> **Important**
>
> Quando si richiama `DescribeTable` su una tabella on demand, le unità di capacità in lettura e le unità di capacità in scrittura sono impostate su 0.

Example

```
aws dynamodb describe-table --table-name Music
```

La tabella è pronta per l'uso quando `TableStatus` cambia da `CREATING` ad `ACTIVE`.

> **Note**
>
> Se si emette una richiesta `DescribeTable` subito dopo una richiesta `CreateTable`, DynamoDB potrebbe restituire un errore (`ResourceNotFoundException`). Questo

accade perché `DescribeTable` utilizza una query consistente finale e i metadata della tabella potrebbero non essere disponibili in quel momento. Attendi qualche secondo e prova nuovamente la richiesta `DescribeTable`.
A scopo di fatturazione, i costi di archiviazione di DynamoDB includono un costo per elemento di 100 byte. Per ulteriori informazioni, consulta Prezzi di DynamoDB. Questi 100 byte extra per elemento non sono utilizzati nei calcoli delle unità di capacità o dall'operazione `DescribeTable`.

Aggiornamento di una tabella

L'operazione `UpdateTable` consente di effettuare una delle operazioni seguenti:

- Modificare le impostazioni di throughput assegnate di una tabella (per tabelle con modalità assegnata).
- Cambiare la modalità di capacità in lettura/scrittura della tabella.
- Utilizza gli indici secondari globali nella tabella (consulta Utilizzo degli indici secondari globali in DynamoDB).
- Abilitare o disabilitare DynamoDB Streams sulla tabella (vedere Acquisizione dei dati di modifica per DynamoDB Streams).

Example

Il seguente esempio di AWS CLI come modificare le impostazioni di throughput assegnato di una tabella.

```
aws dynamodb update-table --table-name Music \
    --provisioned-throughput ReadCapacityUnits=20,WriteCapacityUnits=10
```

> **Note**
>
> Quando emetti una richiesta `UpdateTable`, lo stato della tabella cambia da `AVAILABLE` a `UPDATING`. La tabella rimane completamente disponibile mentre lo stato è `UPDATING`. Al termine del processo, lo stato cambia da `UPDATING` ad `AVAILABLE`.

Example

Il seguente esempio di AWS CLI mostra come modificare la modalità di capacità in lettura/scrittura di una tabella in modalità on demand.

```
aws dynamodb update-table --table-name Music \
    --billing-mode PAY_PER_REQUEST
```

Eliminazione di una tabella

Puoi rimuovere una tabella inutilizzata con l'operazione `DeleteTable`. L'operazione di eliminazione di una tabella è irreversibile. Per eliminare una tabella utilizzando la Console di gestione AWS, consulta the section called “Passaggio 6: pulizia (facoltativo)”.

Example

L'esempio AWS CLI seguente mostra come eliminare una tabella.

```
aws dynamodb delete-table --table-name Music
```

Quando emetti una richiesta `DeleteTable`, lo stato della tabella cambia da `ACTIVE` a `DELETING`. L'eliminazione della tabella può richiedere del tempo, in base alle risorse che utilizza (come i dati archiviati e i flussi o gli indici presenti).

A conclusione dell'operazione `DeleteTable`, la tabella sarà più presente in DynamoDB.

Uso della protezione da eliminazione

È possibile proteggere una tabella dall'eliminazione accidentale con la proprietà di protezione da eliminazione. L'attivazione di questa proprietà per una tabella aiuta a garantire che non venga eliminata accidentalmente durante le normali operazioni di gestione delle tabelle degli amministratori. In tal modo si contribuisce anche a prevenire le interruzioni delle normali operazioni aziendali.

Il proprietario della tabella o un amministratore autorizzato controlla la proprietà di protezione da eliminazione per ogni tabella che per impostazione predefinita è disattivata per tutte le tabelle, incluse le repliche globali e le tabelle ripristinate dai backup. Quando la protezione da eliminazione è disabilitata, la tabella può essere eliminata da qualsiasi utente autorizzato da una policy Identity and Access Management (IAM). Quando la protezione da eliminazione è abilitata, nessuno può eliminare la tabella.

Per modificare questa impostazione, vai alle Impostazioni aggiuntive della tabella, apri il pannello Protezione da eliminazione e seleziona Abilita la protezione da eliminazione.

La proprietà di protezione da eliminazione è supportata dalla console DynamoDB, dall'API, dalla CLI, dall'SDK e da CloudFormation. L'API `CreateTable` supporta la proprietà di protezione da eliminazione al momento della creazione della tabella e l'API `UpdateTable` supporta la modifica della proprietà di protezione da eliminazione per le tabelle esistenti.

> **Note**
>
> - Se un account AWS viene eliminato, tutti i dati dell'account, comprese le tabelle, vengono eliminati entro 90 giorni.
> - Se DynamoDB perde l'accesso a una chiave gestita dal cliente utilizzata per crittografare una tabella, archivia comunque la tabella. L'archiviazione comporta l'esecuzione di un backup della tabella e l'eliminazione dell'originale.

Elenco dei nomi delle tabelle

L'operazione `ListTables` restituisce i nomi delle tabelle DynamoDB nell'account e nella regione AWS correnti.

Example

Il seguente esempio della AWS CLI illustra come elencare i nomi delle tabelle DynamoDB.

```
aws dynamodb list-tables
```

Descrizione delle quote di velocità di trasmissione effettiva assegnate

L'operazione `DescribeLimits` restituisce le quote correnti di capacità di lettura e scrittura nell'account e nella regione AWS correnti.

Example

Il seguente esempio di AWS CLI mostra come descrivere le quote correnti di throughput assegnato.

```
aws dynamodb describe-limits
```

L'output mostra le quote superiori delle unità di capacità di lettura e scrittura nell'account e nella regione AWS correnti.

Per ulteriori informazioni sulle quote e su come richiedere un aumento delle quote, consulta Quote predefinite della velocità di trasmissione effettiva.

Considerazioni sulla scelta di una classe di tabella in DynamoDB

DynamoDB offre due classi di tabelle progettate per aiutarti a ottimizzare i costi. La classe di tabella DynamoDB Standard è quella predefinita ed è consigliata per la maggior parte dei carichi di lavoro. La classe di tabella DynamoDB Standard-Infrequent Access (DynamoDB Standard (accesso infrequente)) è ottimizzata per le tabelle in cui l'archiviazione è il costo principale. Ad esempio, le tabelle che archiviano dati a cui si accede raramente, come i registri delle applicazioni, i vecchi post sui social media, la cronologia degli ordini di e-commerce e i risultati di gioco precedenti, sono buoni candidati per la classe di tabella Standard (accesso infrequente).

Ogni tabella DynamoDB è associata a una classe di tabella. Tutti gli indici secondari associati alla tabella utilizzano la stessa classe di tabella. Puoi impostare la classe di tabella durante la creazione della tabella (DynamoDB Standard, di default) e aggiornare la classe di una tabella esistente utilizzando la Console di gestione AWS, AWS CLI o l'SDK AWS. DynamoDB supporta anche la gestione della classe di tabella utilizzando AWS CloudFormation per le tabelle a regione singola (tabelle che non sono globali). Ogni classe di tabella offre prezzi diversi per l'archiviazione dati e le richieste di lettura e scrittura. Quando scegli una classe di tabella per la tua tabella, tieni presente quanto segue:

- La classe di tabella DynamoDB Standard offre costi di throughput inferiori rispetto a DynamoDB Standard (accesso infrequente) ed è l'opzione più conveniente per le tabelle in cui il throughput è il costo dominante.
- La classe di tabella DynamoDB Standard (accesso infrequente) offre costi di storage inferiori rispetto a DynamoDB Standard ed è l'opzione più conveniente per le tabelle in cui l'archiviazione è il costo dominante. Quando l'archiviazione supera il 50% del costo del throughput (letture e scritture) di una tabella utilizzando la classe di tabella DynamoDB Standard, la classe di tabella DynamoDB Standard (accesso infrequente) può aiutare a ridurre il costo totale della tabella.
- Le tabelle DynamoDB Standard (accesso infrequente) offrono le stesse prestazioni, durata e disponibilità delle tabelle DynamoDB Standard.
- Il passaggio tra le classi di tabella DynamoDB Standard e Standard (accesso infrequente) non richiede la modifica del codice dell'applicazione. Si utilizzano le stesse API e gli endpoint di servizio DynamoDB indipendentemente dalla classe di tabella utilizzata dalle tabelle.

- Le tabelle DynamoDB Standard (accesso infrequente) sono compatibili con tutte le funzionalità di DynamoDB esistenti come il ridimensionamento automatico, la modalità on-demand, il time-to-live (TTL), i backup on-demand, il ripristino point-in-time (PITR) e gli indici secondari globali.

La classe di tabella più conveniente per la tabella dipende dai modelli di archiviazione e utilizzo del throughput previsti dalla tabella. Puoi esaminare il costo e l'utilizzo storico dell'archiviazione e del throughput della tua tabella con AWS Cost and Usage Reports (Report costi e utilizzo) e AWS Cost Explorer. Utilizza questi dati storici per determinare la classe di tabella più conveniente per la tua tabella. Per ulteriori informazioni sull'utilizzo di AWS Cost and Usage Reports (Report costi e utilizzo) e di AWS Cost Explorer, consulta la AWS Billing and Cost Management Documentation. Per dettagli sui prezzi della classe di tabella, consulta Amazon DynamoDB Pricing.

> **Note**
>
> Un aggiornamento della classe di tabella è un processo in background. Puoi comunque accedere normalmente alla tabella durante un aggiornamento della classe di tabella. Il tempo necessario per aggiornare la classe di tabella dipende dal traffico della tabella, dalle dimensioni di archiviazione e da altre variabili correlate. Non sono consentiti più di due aggiornamenti della classe di tabella in un periodo finale di 30 giorni.

Aggiunta di tag ed etichette alle risorse in DynamoDB

È possibile etichettare le risorse Amazon DynamoDB utilizzando i tag. I tag consentono di categorizzare le risorse in diversi modi, ad esempio, per scopo, proprietario, ambiente o altri criteri. I tag consentono di eseguire le seguenti operazioni:

- identificare rapidamente una risorsa in base ai tag a questa assegnati.
- Visualizza le AWS fatture suddivise per tag.

> **Note**
>
> Tutti gli indici secondari locali (LSI) e gli indici secondari globali (GSI) correlati alle tabelle con le etichette, sono etichettati con gli stessi tag in modo automatico. Attualmente, l'utilizzo di DynamoDB Streams non può essere taggato.

Il tagging è supportato da AWS servizi come Amazon EC2, Amazon S3, DynamoDB e altri. Un tagging efficiente può fornire informazioni dettagliate sui costi abilitando la creazione di report su servizi che recano tag specifici.

Per iniziare a utilizzare il tagging, procedi come segue:

1. comprendere Limitazioni dell'assegnazione di tag in DynamoDB;
2. creare tag utilizzando Assegnazione di tag alle risorse in DynamoDB;
3. Utilizzalo Utilizzo dei tag DynamoDB per creare report di allocazione dei costi per tenere traccia dei AWS costi per tag attivo.

Infine, è buona norma seguire strategie di tagging ottimali. Per informazioni, consulta Strategie di assegnazione di tag di AWS.

Limitazioni dell'assegnazione di tag in DynamoDB

Ogni tag consiste di una chiave e di un valore, entrambi personalizzabili. Le restrizioni si applicano come segue:

- Ogni tabella DynamoDB può avere solo un tag con la stessa chiave. Se provi ad aggiungere un tag esistente (con la stessa chiave), il valore del tag esistente viene caricato al nuovo valore;
- I valori e le chiavi dei tag rispettano la distinzione tra maiuscole e minuscole.
- La lunghezza massima della chiave è di 128 caratteri Unicode.
- La lunghezza massima del valore è di 256 caratteri Unicode.
- i caratteri consentiti sono lettere, spazi vuoti, numeri e i seguenti caratteri speciali: `+ - = . _ : /`
- Il numero massimo di tag per risorsa è 50.
- La dimensione massima supportata per tutti i tag in una tabella è 10 KB.
- AWS-ai nomi e ai valori dei tag assegnati viene assegnato automaticamente il `aws:` prefisso, che non puoi assegnare. AWS-i nomi dei tag assegnati non vengono conteggiati ai fini del limite di tag di 50 o del limite di dimensione massima di 10 KB. I nomi dei tag assegnati dall'utente hanno il prefisso `user:` nel report di allocazione dei costi;
- Non puoi retrodatare l'applicazione di un tag.

Assegnazione di tag alle risorse in DynamoDB

Puoi utilizzare la console Amazon DynamoDB o AWS Command Line Interface il AWS CLI() per aggiungere, elencare, modificare o eliminare tag. Puoi quindi attivare questi tag definiti dall'utente in modo che vengano visualizzati nella console Gestione dei costi e fatturazione AWS per il tracciamento dell'allocazione dei costi. Per ulteriori informazioni, consulta Utilizzo dei tag DynamoDB per creare report di allocazione dei costi.

Per la modifica in blocco, puoi anche utilizzare l'editor di tag nella Console di gestione AWS. Per ulteriori informazioni, consulta Utilizzo dell'editor di tag.

Per utilizzare l'API DynamoDB, consulta le seguenti operazioni nella Documentazione di riferimento delle API di Amazon DynamoDB:

- TagResource
- UntagResource
- ListTagsOfResource

Argomenti

- Impostazione delle autorizzazioni per filtrare in base ai tag
- Aggiunta di tag a tabelle nuove o esistenti (Console di gestione AWS)
- Aggiunta di tag a tabelle nuove o esistenti (AWS CLI)

Impostazione delle autorizzazioni per filtrare in base ai tag

Per utilizzare i tag per filtrare l'elenco delle tabelle nella console DynamoDB, assicurati che le policy dell'utente includano l'accesso alle seguenti operazioni:

- `tag:GetTagKeys`
- `tag:GetTagValues`

È possibile accedere a queste operazioni collegando una nuova policy IAM all'utente attenendosi alla procedura riportata di seguito.

1. Accedi alla Console IAM come utente amministratore.
2. Nel menu di navigazione a sinistra, seleziona "Policy".

3. Seleziona "Crea policy".
4. Incollare la seguente policy nell'editor JSON.

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "tag:GetTagKeys",
                "tag:GetTagValues"
            ],
            "Resource": "*"
        }
    ]
}
```

5. Completare la procedura guidata e assegnare un nome alla policy, ad esempio `TagKeysAndValuesReadAccess`.
6. Dal menu di navigazione a sinistra, scegli "Utenti".
7. Dall'elenco, seleziona l'utente normalmente utilizzato per accedere alla console DynamoDB.
8. Seleziona "Aggiungi autorizzazioni".
9. Seleziona "Collega direttamente le policy esistenti".
10. Seleziona quindi la policy creata in precedenza.
11. Completa la procedura guidata.

Aggiunta di tag a tabelle nuove o esistenti (Console di gestione AWS)

È possibile utilizzare la console DynamoDB per aggiungere, tag a nuove tabelle durante la loro creazione oppure aggiungere, modificare o eliminare i tag di tabelle esistenti.

Per assegnare tag alle risorse al momento della creazione (console)

1. Accedi Console di gestione AWS e apri la console DynamoDB all'indirizzo. https://console.aws.amazon.com/dynamodb/
2. Nel pannello di navigazione, scegli Tabelle, quindi seleziona Crea tabella.

3. Nella pagina Create DynamoDB table (Crea tabella DynamoDB), fornire un nome e una chiave primaria. Nella sezione Tags (Tag), scegli Add new tag (Aggiungi nuovo tag) e inserisci i tag che vuoi utilizzare.

   Per informazioni sulla struttura dei tag, consulta Limitazioni dell'assegnazione di tag in DynamoDB.

   Per ulteriori informazioni sulla creazione delle tabelle, consulta Operazioni di base sulle tabelle DynamoDB.

Per assegnare tag alle risorse esistenti (console)

Apri la console DynamoDB all'indirizzo. https://console.aws.amazon.com/dynamodb/

1. Nel pannello di navigazione, seleziona Tabelle.
2. Scegli una tabella nell'elenco, quindi scegli la scheda Additional settings (Impostazioni aggiuntive). Puoi aggiungere, modificare o eliminare i tag nella sezione Tags (Tag) nella parte inferiore della pagina.

Aggiunta di tag a tabelle nuove o esistenti (AWS CLI)

Gli esempi seguenti mostrano come utilizzare i tag AWS CLI per specificare quando si creano tabelle e indici e per etichettare le risorse esistenti.

Per assegnare tag alle risorse al momento della creazione (AWS CLI)

- Il seguente esempio crea una nuova tabella `Movies` e aggiunge il tag `Owner` con un valore `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
```

Per assegnare tag alle risorse esistenti (AWS CLI)

- L'esempio seguente aggiunge il tag `Owner` con un valore `blueTeam` della tabella `Movies`:

```
aws dynamodb tag-resource \
    --resource-arn arn:aws:dynamodb:us-east-1:123456789012:table/Movies \
    --tags Key=Owner,Value=blueTeam
```

Per elencare tutti i tag di una tabella (AWS CLI)

- L'esempio seguente elenca tutti i tag associati alla tabella `Movies`:

```
aws dynamodb list-tags-of-resource \
    --resource-arn arn:aws:dynamodb:us-east-1:123456789012:table/Movies
```

Utilizzo dei tag DynamoDB per creare report di allocazione dei costi

AWS utilizza i tag per organizzare i costi delle risorse nel rapporto di allocazione dei costi. AWS fornisce due tipi di tag di allocazione dei costi:

- Un AWS tag generato. AWS definisce, crea e applica questo tag per te.
- Tag definiti dall'utente. L'utente definisce, crea e applica questi tag.

È necessario attivare entrambi i tipi di tag separatamente per poterli visualizzare in Cost Explorer o in un report di allocazione dei costi.

Per attivare i tag AWS generati:

1. Accedi Console di gestione AWS e apri la console Billing and Cost Management https://console.aws.amazon.com/billing/a casa#/.
2. Nel riquadro di navigazione scegli Cost Allocation Tags (Tag per l'allocazione dei costi).
3. In Tag per l'allocazione dei costi generati da AWS scegli Attiva.

Per attivare i tag definiti dall'utente:

1. Accedi Console di gestione AWS e apri la console Billing and Cost Management https://console.aws.amazon.com/billing/a casa#/.
2. Nel riquadro di navigazione scegli Cost Allocation Tags (Tag per l'allocazione dei costi).
3. In Tag per l'allocazione dei costi definiti dall'utente scegli Attiva.

Dopo aver creato e attivato i tag, AWS genera un rapporto di allocazione dei costi con l'utilizzo e i costi raggruppati in base ai tag attivi. Il rapporto sull'allocazione dei costi include tutti i AWS costi per ogni periodo di fatturazione. Il report include sia le risorse taggate, sia quelle non taggate, per permetterti di organizzare in modo chiaro le spese per le risorse.

> **Note**
>
> Attualmente, tutti i dati trasferiti da DynamoDB non saranno suddivisi per tag nei report di allocazione dei costi.

Per ulteriori informazioni, consulta Utilizzo dei tag per l'allocazione dei costi.

Tabelle globali: replica multi-attiva e multi-Regione

Le tabelle globali Amazon DynamoDB sono una funzionalità di database completamente gestito, multi-Regione e multi-attivo che offre replica dei dati e prestazioni di lettura e scrittura rapide e localizzate per applicazioni scalate a livello globale.

Le tabelle globali replicano automaticamente i dati Regioni AWS delle tabelle DynamoDB senza richiedere la creazione e la manutenzione della propria soluzione di replica. Quando si scelgono le Regioni AWS per le repliche delle tabelle, le tabelle globali gestiscono automaticamente tutte le repliche. Le tabelle globali sono disponibili in tutte le Regioni in cui DynamoDB è disponibile.

Le tabelle globali offrono i seguenti vantaggi:

- Replica automaticamente i dati delle tabelle DynamoDB nelle aree AWS di tua scelta per localizzare i dati più vicino agli utenti
- Maggiore disponibilità delle applicazioni durante l'isolamento o la riduzione delle prestazioni Regionale
- Eliminazione della risoluzione dei conflitti di aggiornamento in modo da potersi concentrare sulla logica di business dell'applicazione

Le tabelle globali sono ideali per le applicazioni che richiedono continuità aziendale e alta disponibilità attraverso l'implementazione multi-Regione. Qualsiasi replica di tabella globale può essere utilizzata per operazioni di lettura e scrittura. Le applicazioni possono raggiungere un'elevata resilienza con un Obiettivo del punto di ripristino (RPO) basso o nullo spostando il traffico verso un'altra Regione se l'elaborazione delle applicazioni viene interrotta in una Regione.

È possibile configurare una tabella globale utilizzando la Console di gestione. AWS Le tabelle globali utilizzano APIs DynamoDB esistente per leggere e scrivere dati nelle tabelle, quindi non sono necessarie modifiche all'applicazione. Il pagamento avviene solo in base alle risorse allocate o utilizzate, senza costi anticipati o impegni. Per iniziare a utilizzare le tabelle globali, consulta Tutorial: creazione di tabelle globali.

Argomenti

- Come funzionano le tabelle globali DynamoDB
- Tutorial: creazione di tabelle globali
- Sicurezza per tabelle globali DynamoDB
- Informazioni sulla fatturazione di Amazon DynamoDB per le tabelle globali
- Versioni delle tabelle globali DynamoDB
- Best practice per le tabelle globali

Come funzionano le tabelle globali DynamoDB

Nelle sezioni seguenti sono riportate ulteriori informazioni sui concetti e i comportamenti delle tabelle globali in Amazon DynamoDB.

Concetti

Le tabelle globali sono una funzionalità di DynamoDB che replica i dati delle tabelle tra le regioni. AWS

Una tabella di replica (o replica) è una singola tabella DynamoDB che funziona come parte di una tabella globale. Una tabella globale è composta da due o più tabelle di replica in diverse regioni. AWS Ciascuna tabella globale può avere una sola tabella di replica per Regione AWS . Tutte le repliche in una tabella globale condividono lo stesso nome di tabella, lo stesso schema di chiave primaria e gli stessi dati degli elementi.

Quando un'applicazione scrive dati in una replica in una Regione, DynamoDB replica automaticamente la scrittura in tutte le altre repliche nella tabella globale. Per ulteriori informazioni su come iniziare a utilizzare le tabelle globali, consulta Tutorial: creazione di tabelle globali.

Versioni

Sono disponibili due versioni delle tabelle globali DynamoDB: versione 2019.11.21 (Corrente) e versione 2017.11.29 (Legacy). Sarebbe opportuno utilizzare la versione 2019.11.21 (Corrente) ogni

volta che è possibile. Le informazioni contenute in questa sezione della documentazione si riferiscono alla versione 2019.11.21 (Corrente). Per ulteriori informazioni, consulta Determinare la versione di una tabella globale.

Disponibilità

Le tabelle globali aiutano a migliorare la continuità aziendale semplificando l'implementazione di un'architettura ad alta disponibilità multi-Regione. Se un carico di lavoro in una singola AWS regione viene compromesso, è possibile spostare il traffico delle applicazioni in una regione diversa ed eseguire letture e scritture su una tabella di replica diversa nella stessa tabella globale.

Ogni tabella di replica in una tabella globale offre la stessa durata e disponibilità di una tabella DynamoDB a Regione singola. Le tabelle globali offrono un Accordo sul livello di servizio (SLA) con disponibilità del 99,999%, rispetto al 99,99% delle tabelle a Regione singola.

Modalità di coerenza

Quando si crea una tabella globale, è possibile configurarne la modalità di coerenza. Le tabelle globali supportano due modalità di coerenza: coerenza finale multi-Regione (MREC, Multi-Region Eventual Consistency) e elevata consistenza multi-Regione (MRSC, Multi-Region Strong Consistency).

Se non si specifica una modalità di coerenza durante la creazione di una tabella globale, la tabella globale utilizza come impostazione predefinita la coerenza finale multi-Regione (MREC). Una tabella globale non può contenere repliche configurate con modalità di coerenza diverse. Non è possibile modificare la modalità di coerenza di una tabella globale dopo la creazione.

Coerenza finale multi-Regione (MREC)

La coerenza finale multi-Regione (MREC) è la modalità di coerenza predefinita per le tabelle globali. Le modifiche agli elementi in una replica di tabella globale MREC vengono replicate in modo asincrono su tutte le altre repliche, in genere entro un secondo o meno. Nell'improbabile eventualità che una replica in una tabella globale MREC venga isolata o danneggiata, tutti i dati non ancora replicati in altre Regioni verranno replicati quando la replica diventa integra.

Se lo stesso elemento viene modificato in più Regioni contemporaneamente, DynamoDB risolverà il conflitto utilizzando la modifica con il timestamp interno più recente per elemento, denominato metodo di risoluzione dei conflitti basato sulla "priorità dell'ultima istanza di scrittura". Un elemento alla fine convergerà in tutte le repliche alla versione creata dall'ultima scrittura.

Le operazioni a elevata consistenza di lettura restituiscono la versione più recente di un elemento se tale elemento è stato aggiornato l'ultima volta nella Regione in cui è avvenuta la lettura, ma possono restituire dati non aggiornati se l'elemento è stato aggiornato l'ultima volta in una Regione diversa. Le scritture condizionali valutano l'espressione della condizione rispetto alla versione dell'elemento nella Regione.

Si crea una tabella globale MREC aggiungendo una replica a una tabella DynamoDB esistente. L'aggiunta di una replica non ha alcun impatto sulle prestazioni sulle tabelle DynamoDB a Regione singola esistenti o sulle repliche di tabelle globali. È possibile aggiungere repliche a una tabella globale MREC per espandere il numero di Regioni in cui i dati vengono replicati o rimuovere le repliche da una tabella globale MREC se non sono più necessarie. Una tabella globale MREC può avere una replica in qualsiasi Regione in cui è disponibile DynamoDB e può avere tante repliche quante sono le Regioni nella partizione AWS .

Elevata consistenza multi-Regione (MRSC)

È possibile configurare la modalità MRSC (Multi-region Strong Consistency) quando si crea una tabella globale. Le modifiche agli elementi in una replica di tabella globale MRSC vengono replicate in modo sincrono in almeno un'altra Regione prima che l'operazione di scrittura restituisca una risposta con esito positivo. Quando si converte una tabella a Regione singola esistente in una tabella globale MRSC, è necessario assicurarsi che la tabella sia vuota fino al completamento della conversione per garantire una corretta impostazione di inizializzazione e replica.

Le operazioni a elevata consistenza di lettura su qualsiasi replica MRSC restituiscono sempre la versione più recente di un elemento. Le scritture condizionali valutano sempre l'espressione della condizione rispetto alla versione più recente di un elemento.

Un'operazione di scrittura ha esito negativo con una `ReplicatedWriteConflictException` quando tenta di modificare un elemento che è già stato modificato in un'altra Regione. Per le scritture con esito negativo con la `ReplicatedWriteConflictException` è possibile procedere alla ripetizione del tentativo: le richieste avranno esito positivo se l'elemento non viene più modificato in un'altra Regione.

È possibile configurare una tabella globale MRSC con tre repliche o con due repliche e un testimone. Un testimone è un componente di una tabella globale MRSC che contiene dati scritti su repliche di tabelle globali e fornisce un'alternativa opzionale a una replica completa, supportando al contempo l'architettura di disponibilità di MRSC. Non è possibile eseguire operazioni di lettura o scrittura su un testimone. Un testimone si trova in una Regione diversa rispetto alle due repliche.

Quando si crea una tabella globale MRSC, si scelgono le Regioni sia per l'implementazione di replica che di testimone al momento della creazione della tabella MRSC. È possibile determinare se e in quale Regione una tabella globale MRSC ha un testimone configurato dall'output dell'API `DescribeTable`. Il testimone è di proprietà e gestito da DynamoDB e non verrà visualizzato nel AWS tuo account nella regione in cui è configurato.

Una tabella globale MRSC deve essere implementata esattamente in tre Regioni. Si crea una tabella globale MRSC aggiungendo una replica e un testimone o due repliche a una tabella DynamoDB esistente che non contiene dati. Non è possibile aggiungere repliche aggiuntive a una tabella globale MRSC esistente. Non è possibile eliminare una singola replica o un testimone da una tabella globale MRSC. È possibile eliminare due repliche o eliminare una replica e un testimone da una tabella globale MRSC, convertendo la replica rimanente in una tabella DynamoDB a Regione singola.

Le seguenti considerazioni si applicano alle tabelle globali MRSC:

- Quando si converte una tabella a Regione singola in una tabella globale MRSC, è necessario assicurarsi che la tabella sia vuota. La conversione di una tabella a Regione singola in una tabella globale MRSC con elementi esistenti non è supportata. Assicurati che non venga scritto alcun dato nella tabella durante il processo di conversione.
- Le tabelle globali MRSC sono disponibili nei seguenti set di Regioni:
  - Set della Regione degli Stati Uniti: Stati Uniti orientali (Virginia settentrionale), Stati Uniti orientali (Ohio), Stati Uniti occidentali (Oregon)
  - Set della Regione Europa: Europa (Irlanda), Europa (Londra), Europa (Parigi), Europa (Francoforte)
  - Set della Regione Asia Pacifico: Asia Pacifico (Tokyo), Asia Pacifico (Seoul) e Asia Pacifico (Osaka).
- Le tabelle globali MRSC non possono estendersi a set di Regioni (ad esempio, una tabella globale MRSC non può contenere repliche di set di Regioni degli Stati Uniti e dell'Europa).
- Il valore Time to Live (TTL) non è supportato per le tabelle globali MRSC.
- Gli indici secondari locali (LSIs) non sono supportati per le tabelle globali MRSC.
- CloudWatch Le informazioni di Contributor Insights vengono riportate solo per la regione in cui si è verificata un'operazione.

Scelta di una modalità di coerenza

Il criterio chiave per la scelta di una modalità di coerenza multi-Regione è se l'applicazione dia priorità alle scritture a bassa latenza e alle operazioni a elevata consistenza di lettura o dia priorità alla elevata consistenza globale.

Le tabelle globali MREC avranno latenze per operazioni di scrittura e per operazioni a elevata consistenza di lettura inferiori rispetto alle tabelle globali MRSC. Le tabelle globali MREC possono supportare un RPO pari al ritardo di replica tra le repliche, in genere di pochi secondi a seconda delle Regioni di replica.

È necessario utilizzare la modalità MREC quando:

- L'applicazione può tollerare la restituzione di dati obsoleti da operazioni a elevata consistenza di lettura se tali dati sono stati aggiornati in un'altra Regione.
- Si dà la priorità a latenze per operazioni di e operazioni a elevata consistenza di lettura inferiori rispetto alla coerenza di lettura multi-Regione.
- La strategia di alta disponibilità multi-Regione può tollerare un RPO superiore a zero.

Le tabelle globali MRSC avranno latenze superiori per operazioni di scrittura e operazioni a elevata consistenza di lettura rispetto alle tabelle globali MREC superiori. Le tabelle globali MRSC supportano un RPO pari a zero.

È necessario utilizzare la modalità MRSC quando:

- Sono necessarie operazioni a elevata consistenza di lettura in più Regioni.
- Si dà la priorità alla coerenza di lettura globale rispetto a una latenza di scrittura inferiore.
- La strategia di alta disponibilità multi-Regione richiede un RPO pari a zero.

Monitoraggio delle tabelle globali

Le tabelle globali configurate per la coerenza finale tra più regioni (MREC) pubblicano la metrica su. `ReplicationLatency` CloudWatch Questa metrica tiene traccia del tempo trascorso tra la scrittura di un elemento in una tabella di replica e la visualizzazione di tale elemento in un'altra replica nella tabella globale. `ReplicationLatency` è espresso in millisecondi e viene emesso per ogni coppia di Regioni di origine e di destinazione in una tabella globale.

`ReplicationLatency`I valori tipici dipendono dalla distanza tra le AWS regioni scelte e da altre variabili come il tipo di carico di lavoro e la velocità effettiva. Ad esempio, una replica di origine nella Regione Stati Uniti occidentali (California settentrionale) (us-west-1) ha un `ReplicationLatency` inferiore rispetto alla Regione Stati Uniti occidentali (Oregon) (us-west-2) rispetto alla Regione Africa (Città del Capo) (af-south-1).

Un valore crescente per `ReplicationLatency` potrebbe indicare che gli aggiornamenti da una replica non si propagano ad altre tabelle di replica in modo tempestivo. In questo caso, puoi reindirizzare temporaneamente l'attività di lettura e scrittura dell'applicazione verso un'altra regione. AWS

Le tabelle globali configurate per l'elevata consistenza multi-Regione (MRSC) non pubblicano una metrica `ReplicationLatency`.

Test di iniezione di guasti

Le tabelle globali MREC si integrano con il AWSAWS Fault Injection Service (FIS) per eseguire esperimenti di failinjection sui carichi di lavoro delle tabelle globali. Ciò consente di testare la risposta dell'applicazione a un isolamento simulato di una Regione sospendendo la replica da e verso una replica selezionata. Per ulteriori informazioni, consulta pausing global table replication.

Time to live (TTL)

Le tabelle globali configurate per MREC supportano la configurazione dell'eliminazione Time to Live (TTL). Le impostazioni del TTL vengono sincronizzate automaticamente per tutte le repliche in una tabella globale. Quando il TTL elimina un elemento da una replica in una Regione, l'eliminazione viene replicata su tutte le altre repliche nella tabella globale. Il TTL non consuma capacità di scrittura, quindi non viene addebitato alcun costo per l'eliminazione TTL nella Regione in cui è avvenuta l'eliminazione. Tuttavia, viene addebitato il costo dell'eliminazione replicata in ogni altra Regione con una replica nella tabella globale.

La replica di eliminazione TTL consuma la capacità di scrittura sulle repliche in cui viene replicata l'eliminazione. Le repliche configurate per la capacità allocata possono limitare le richieste se la combinazione di throughput di scrittura e throughput di eliminazione TTL è superiore alla capacità di scrittura allocata.

Le tabelle globali configurate per MRSC (elevata consistenza multi-Regione) non supportano la configurazione dell'eliminazione TTL.

Streams

Le tabelle globali configurate per coerenza finale multi-Regione (MREC) replicano le modifiche leggendo tali modifiche da un flusso DynamoDB su una tabella di replica e applicandole a tutte le altre tabelle di replica. I flussi sono quindi abilitati per impostazione predefinita su tutte le repliche in una tabella globale MREC e non possono essere disabilitati su tali repliche. Il processo di replica MREC può combinare più modifiche in un breve periodo di tempo in un'unica scrittura replicata, con il risultato che il flusso di ogni replica contiene record leggermente diversi. I record dei flussi sulle repliche MREC vengono sempre ordinati per elemento, ma l'ordinamento tra gli elementi può differire tra le repliche.

Le tabelle globali configurate per elevata consistenza multi-Regione (MRSC) non utilizzano i flussi DynamoDB per la replica, pertanto i flussi non sono abilitati per impostazione predefinita sulle repliche MRSC. È possibile abilitare i flussi su una replica MRSC. I record dei flussi sulle repliche MRSC sono identici per ogni replica, incluso l'ordinamento dei record dei flussi.

Se si desidera scrivere un'applicazione che elabori i record dei flussi per le modifiche avvenute in una particolare Regione ma non in altre Regioni in una tabella globale, è possibile aggiungere un attributo a ogni elemento che definisce in quale Regione è avvenuta la modifica per quell'elemento. È possibile utilizzare questo attributo per filtrare i record dei flussi in base alle modifiche apportate in altre Regioni, incluso l'uso dei filtri di eventi Lambda per invocare le funzioni Lambda solo per le modifiche in una Regione specifica.

Transazioni

In una tabella globale configurata per MREC, le operazioni di transazione DynamoDB (`TransactWriteItems` e `TransactGetItems`) sono atomiche solo all'interno della Regione in cui è stata invocata l'operazione. Le scritture transazionali non vengono replicate come unità tra le Regioni, il che significa che solo alcune delle scritture di una transazione possono essere restituite dalle operazioni di lettura in altre repliche in un determinato momento.

Ad esempio, se è presente una tabella globale con repliche nelle Regioni Stati Uniti orientali (Ohio) e Stati Uniti occidentali (Oregon) e si esegue un'operazione `TransactWriteItems` nella Regione Stati Uniti orientali (Ohio), potrebbe essere possibile osservare transazioni parzialmente completate nella Regione Stati Uniti occidentali (Oregon) man mano che le modifiche vengono replicate. Le modifiche vengono replicate in altre Regioni solo dopo essere state confermate nella Regione di origine.

Le tabelle globali configurate per MRSC (elevata consistenza multi-Regione) non supportano le operazioni di transazione e restituiranno un errore se tali operazioni vengono invocate su una replica MRSC.

Velocità di trasmissione effettiva di lettura e di scrittura

La replica consuma capacità di scrittura. Le repliche configurate per la capacità allocata possono limitare le richieste se la combinazione di throughput di scrittura e throughput di replica è superiore alla capacità di scrittura allocata. La capacità di scrittura in modalità on demand è sincronizzata per tutte le repliche in una tabella globale. Le tabelle globali configurate per la capacità allocata sincronizzano le impostazioni di dimensionamento automatico tra le repliche. L'impostazione effettiva della capacità di scrittura allocata può variare tra le repliche in base al throughput di scrittura utilizzato.

È possibile configurare in modo indipendente le impostazioni della capacità di lettura per ogni replica in una tabella globale. Quando si aggiunge una replica a una tabella globale, la capacità di lettura della tabella o della replica di origine viene utilizzata come valore iniziale, a meno che non venga specificato un valore di sostituzione.

Sincronizzazione delle impostazioni

Le impostazioni nelle tabelle globali di DynamoDB sono parametri di configurazione che controllano vari aspetti del comportamento e della replica delle tabelle. Queste impostazioni sono gestite tramite il APIs piano di controllo di DynamoDB e possono essere configurate durante la creazione o la modifica di tabelle globali. Le tabelle globali sincronizzano automaticamente determinate impostazioni su tutte le repliche per mantenere la coerenza e consentire al contempo la flessibilità per le ottimizzazioni specifiche della Regione. Comprendere quali impostazioni si sincronizzano e come si comportano aiuta a configurare la tabella globale in modo efficace. Le impostazioni rientrano in tre categorie principali in base al modo in cui sono sincronizzate tra le repliche.

Le seguenti impostazioni sono sempre sincronizzate tra le repliche in una tabella globale:

- Modalità di capacità (capacità allocata o on demand)
- Capacità di scrittura allocata alla tabella
- Dimensionamento automatico della tabella
- Definizione degli indici secondari globali (GSI)
- Capacità di scrittura allocata ai GSI

- Dimensionamento automatico in scrittura dei GSI
- Tipo di crittografia lato server (SSE)
- Definizione dei flussi in modalità MREC
- Time to live (TTL)
- Throughput a caldo
- Throughput massimo on demand

Le seguenti impostazioni sono sincronizzate tra le repliche, ma possono essere sostituite per ogni singola replica:

- Capacità di lettura allocata alla tabella
- Dimensionamento automatico in lettura della tabella
- Capacità di lettura allocata ai GSI
- Dimensionamento automatico in lettura dei GSI
- Classe di tabella
- Throughput massimo on demand

> **Note**
>
> I valori delle impostazioni sostituibili vengono modificati se l'impostazione viene modificata su qualsiasi altra replica. Ad esempio, è disponibile una tabella globale MREC con repliche nelle Regioni Stati Uniti orientali (Virginia settentrionale) e Stati Uniti occidentali (Oregon). La replica degli Stati Uniti orientali (Virginia settentrionale) ha impostato un throughput di lettura impostato su 200. RCUs Per la replica negli Stati Uniti occidentali (Oregon), la velocità effettiva di lettura assegnata è impostata su 100. RCUs Se si aggiorna l'impostazione della velocità effettiva di lettura assegnata sulla replica degli Stati Uniti orientali (Virginia settentrionale) da 200 RCUs a 300 RCUs, il nuovo valore della velocità effettiva di lettura assegnata verrà applicato anche alla replica negli Stati Uniti occidentali (Oregon). Ciò modifica l'impostazione della velocità effettiva di lettura assegnata per la replica degli Stati Uniti occidentali (Oregon) dal valore sovrascritto di 100 al nuovo valore di 300. RCUs RCUs

Le seguenti impostazioni non vengono mai sincronizzate tra le repliche:

- Protezione da eliminazione

- Point-in-time Recupero
- Tag
- Attivazione di Table CloudWatch Contributor Insights
- Abilitazione di GSI Contributor Insights CloudWatch
- Definizione del flusso di dati Kinesis
- Policy delle risorse
- Definizione dei flussi in modalità MRSC

Tutte le altre impostazioni non vengono sincronizzate tra le repliche.

DynamoDB Accelerator (DAX)

Le scritture su repliche di tabelle globali ignorano DynamoDB Accelerator (DAX), aggiornando direttamente DynamoDB. Di conseguenza, le cache di DAX possono diventare obsolete poiché le scritture non le aggiornano. Le cache di DAX configurate per le repliche globali delle tabelle verranno aggiornate solo alla scadenza del TTL della cache.

Considerazioni sulla gestione delle tabelle globali

Non è possibile eliminare una tabella utilizzata per aggiungere una nuova replica globale della tabella prima che siano trascorse 24 ore dalla creazione della nuova replica.

Se si disabilita una AWS regione che contiene repliche di tabelle globali, tali repliche vengono convertite in modo permanente in tabelle a regione singola 20 ore dopo la disattivazione della regione.

Tutorial: creazione di tabelle globali

Questa sezione fornisce step-by-step istruzioni per creare tabelle globali DynamoDB configurate per la modalità di consistenza preferita. Seleziona la modalità a coerenza finale multi-Regione (MREC, Multi-Region Eventual Consistency) o elevata consistenza multi-Regione (MRSC, Multi-Region Strong Consistency) in base ai requisiti dell'applicazione.

Le tabelle globali MREC offrono una latenza di scrittura inferiore con una coerenza finale generale Regioni AWS. Le tabelle globali MRSC forniscono operazioni a elevata consistenza di lettura tra le Regioni con latenze di scrittura leggermente superiori rispetto a MREC. Sceglie la modalità

di coerenza che meglio soddisfa le esigenze dell'applicazione in termini di coerenza, latenza e disponibilità dei dati.

Argomenti

- Creazione di una tabella globale configurata per MREC
- Creazione di una tabella globale configurata per MRSC

Creazione di una tabella globale configurata per MREC

Questa sezione mostra come creare una tabella globale con la modalità a coerenza finale multi-Regione (MREC). MREC è la modalità di coerenza predefinita per le tabelle globali e fornisce scritture a bassa latenza con replica asincrona su tutte le Regioni AWS. Le modifiche apportate a un elemento in una Regione vengono in genere replicate in tutte le altre Regioni entro un secondo. Ciò rende la MREC ideale per le applicazioni che danno priorità a una bassa latenza di scrittura e possono tollerare brevi periodi in cui Regioni diverse possono restituire versioni di dati leggermente diverse.

È possibile creare tabelle globali MREC con repliche in qualsiasi regione in AWS cui è disponibile DynamoDB e aggiungere o rimuovere repliche in qualsiasi momento. Gli esempi seguenti mostrano come creare una tabella globale MREC con repliche in più Regioni.

Creazione di una tabella globale MREC con la console DynamoDB

Completa questa procedura per creare una tabella globale utilizzando Console di gestione AWS. Il seguente esempio consente di creare una tabella globale con le tabelle di replica negli Stati Uniti e in Europa.

1. Accedi Console di gestione AWS e apri la console DynamoDB all'indirizzo. https://console.aws.amazon.com/dynamodb/
2. Per questo esempio, seleziona Stati Uniti orientali (Ohio) dal selettore delle Regioni nella barra di navigazione.
3. Nel riquadro di navigazione sul lato sinistro della console scegli Tables (Tabelle).
4. Scegliere Create Table (Crea tabella).
5. Nella pagina Crea tabella:
    a. Nel campo Table name (Nome tabella) immetti **`Music`**.
    b. In Partition key (Chiave di partizione), inserisci **`Artist`**.

c. Per Chiave di ordinamento, immetti **`SongTitle`**.

d. Mantieni le restanti impostazioni predefinite e seleziona Crea tabella.

   Questa nuova tabella funge da prima tabella di replica in una nuova tabella globale. È il prototipo per altre tabelle di replica che verranno aggiunte in seguito.

6. Dopo che la tabella è diventata attiva:

   a. Nell'elenco delle tabelle, seleziona la tabella Musica.

   b. Seleziona la scheda Tabelle globali.

   c. Scegliere Crea replica.

7. Dall'elenco a discesa Regioni di replica disponibili, seleziona Stati Uniti occidentali (Oregon) us-west-2.

   La console verifica che non esista una tabella con lo stesso nome nella Regione selezionata. Se esiste una tabella con lo stesso nome, è necessario eliminare la tabella esistente prima di poter creare una nuova tabella di replica in quella regione.

8. Scegliere Crea replica. Viene avviato il processo di creazione della tabella nella Regione Stati Uniti occidentali (Oregon) us-west-2.

   La scheda Tabelle globali per la tabella Musica (e per qualsiasi altra tabella di replica) mostra che la tabella è stata replicata in più Regioni.

9. Aggiungi un'altra Regione ripetendo le fasi precedenti, ma seleziona Europa (Francoforte) eu-central-1 come Regione.

10. Per testare la replica:

    a. Assicurati di utilizzare la Console di gestione AWS regione Stati Uniti orientali (Ohio).

    b. Scegli Explore table items (Esplora elementi della tabella).

    c. Scegli Crea elemento.

    d. Entra **`item_1`** per Artista e **`Song Value 1`** per SongTitle.

    e. Scegli Crea elemento.

11. Verifica la replica passando alle altre Regioni:

    a. Dal selettore delle Regioni in alto a destra, seleziona Europa (Francoforte).

    b. Verifica che la tabella Musica contenga l'elemento creato.

    c. Ripeti la verifica per Stati Uniti occidentali (Oregon).

Creazione di una tabella globale MREC utilizzando o Java AWS CLI

CLI

Nell'esempio di codice seguente viene mostrato come creare e gestire le tabelle globali DynamoDB con replica multi-Regione con coerenza possibile (MREC).

- Creazione di una tabella con replica multi-Regione (MREC).
- Inserimento e recupero di elementi dalle tabelle di replica.
- Rimuovi le repliche. one-by-one
- Per eseguire la pulizia mediante l'eliminazione delle tabelle.

AWS CLI con lo script Bash

Per creare una tabella con replica multi-Regione.

```
# 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
```

Per descrivere la tabella multi-Regione.

```
# 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}}'
```

Per inserire elementi in una tabella di replica.

```
# 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
```

Per ottenere elementi dalle tabelle di replica.

```
# 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
```

Per rimuovere le repliche.

```
# 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
```

Per eseguire la pulizia mediante l'eliminazione delle tabelle.

```
# Clean up: Delete the primary table
aws dynamodb delete-table --table-name MusicTable --region us-east-2

echo "Global table demonstration complete."
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento dei comandi AWS CLI .
  - CreateTable
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - UpdateTable

Java

L'esempio di codice seguente mostra come creare e gestire tabelle globali DynamoDB con repliche tra più Regioni.

- Crea una tabella con Global Secondary Index e DynamoDB Streams.
- Aggiunta di repliche in Regioni diverse per creare una tabella globale.
- Rimuovi le repliche da una tabella globale.
- Aggiungi elementi di test per verificare la replica tra le regioni.
- Descrivi la configurazione globale della tabella e lo stato della replica.

SDK per Java 2.x

Crea una tabella con Global Secondary Index e DynamoDB Streams utilizzando. 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;
        }
    }
```

Attendi che una tabella diventi attiva utilizzando. 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;
        }
    }
```

Aggiungi una replica per creare o estendere una tabella globale utilizzando 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;
       }
   }
```

Rimuovi una replica da una tabella globale utilizzando. 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;
        }
    }
```

Aggiungi elementi di test per verificare la replica utilizzando. 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;
        }
    }
```

Descrivi la configurazione globale della tabella e le repliche utilizzando. 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;
        }
    }
```

Esempio completo di operazioni su tabelle globali utilizzando 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;
            }
        }
    }
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento delle API AWS SDK for Java 2.x .

- CreateTable
- DescribeTable
- PutItem
- UpdateTable

Creazione di una tabella globale configurata per MRSC

Questa sezione mostra come creare una tabella globale a elevata consistenza multi-Regione (MRSC). Le tabelle globali MRSC replicano in modo sincrono le modifiche agli elementi tra le Regioni, garantendo che le operazioni a elevata consistenza di lettura su qualsiasi replica restituiscano sempre la versione più recente di un elemento. Quando si converte una tabella a Regione singola in una tabella globale MRSC, è necessario assicurarsi che la tabella sia vuota. La conversione di una tabella a Regione singola in una tabella globale MRSC con elementi esistenti non è supportata. Assicurati che non venga scritto alcun dato nella tabella durante il processo di conversione.

È possibile configurare una tabella globale MRSC con tre repliche o con due repliche e un testimone. Quando si crea una tabella globale MRSC, si scelgono le Regioni in cui vengono distribuite le repliche e un testimone opzionale. L'esempio seguente crea una tabella globale MRSC con repliche nelle Regioni Stati Uniti orientali (Virginia settentrionale) e Stati Uniti orientali (Ohio), con un testimone nella Regione Stati Uniti occidentali (Oregon).

> **Note**
>
> Prima di creare una tabella globale, verifica che i limiti di throughput della quota di servizio siano coerenti in tutte le Regioni di destinazione, poiché ciò è necessario per creare una tabella globale. Per ulteriori informazioni sui limiti di throughput delle tabelle globali, consulta Quote delle tabelle globali.

Creazione di una tabella globale MRSC con la console DynamoDB

Completa questa procedura per creare una tabella globale MRSC utilizzando la Console di gestione AWS.

1. Accedi Console di gestione AWS e apri la console DynamoDB all'indirizzo. https://console.aws.amazon.com/dynamodb/

2. Dal selettore delle Regioni nella barra di navigazione, seleziona una Regione in cui sono supportate le tabelle globali con MRSC, ad esempio **`us-east-2`**.
3. Nel pannello di navigazione, seleziona Tabelle.
4. Scegliere Create table (Crea tabella).
5. Nella pagina Crea tabella:
   a. Nel campo Table name (Nome tabella) immetti **`Music`**.
   b. Per Chiave di partizione immetti **`Artist`** e mantieni il tipo di Stringa predefinito.
   c. Per Chiave di ordinamento immetti **`SongTitle`** e mantieni il tipo di Stringa predefinito.
   d. Mantieni le restanti impostazioni predefinite e seleziona Crea tabella.

      Questa nuova tabella funge da prima tabella di replica in una nuova tabella globale. È il prototipo per altre tabelle di replica che verranno aggiunte in seguito.
6. Attendi che la tabella diventi attiva, quindi selezionala dall'elenco delle tabelle.
7. Seleziona la scheda Tabelle globali, quindi seleziona Crea replica.
8. Nella pagina Crea replica:
   a. In Coerenza multi-Regione, seleziona Coerenza forte.
   b. Per Regione di replica 1, seleziona **`US East (N. Virginia) us-east-1`**.
   c. Per Regione di replica 2, seleziona **`US West (Oregon) us-west-2`**.
   d. Seleziona Configura come testimone per la Regione Stati Uniti occidentali (Oregon).
   e. Seleziona Crea repliche.
9. Attendi il completamento della replica e del processo di creazione del testimone. Lo stato della replica verrà visualizzato come Attivo quando la tabella sarà pronta per l'uso.

Creazione di una tabella globale MRSC utilizzando o Java AWS CLI

Prima di iniziare, assicurati che il principale IAM disponga delle autorizzazioni necessarie per creare una tabella globale MRSC con una Regione testimone.

La seguente policy IAM di esempio consente di creare una tabella DynamoDB (`MusicTable`) negli Stati Uniti orientali (Ohio) con una replica nella Regione Stati Uniti orientali (Virginia settentrionale) e una Regione testimone negli Stati Uniti occidentali (Oregon):

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"
                }
            }
        }
    ]
```

```
}
```

Gli esempi di codice seguenti mostrano come creare e gestire tabelle globali DynamoDB con elevata consistenza multi-Regione (MRSC).

- Creazione di una tabella con elevata consistenza multi-Regione.
- Verifica della configurazione MRSC e dello stato della replica.
- Verifica dell'elevata consistenza tra Regioni con letture immediate.
- Esecuzione di scritture condizionali con garanzie MRSC.
- Esecuzione della pulizia delle risorse su una tabella globale MRSC.

Bash

AWS CLI con lo script Bash

Creazione di una tabella con elevata consistenza multi-Regione.

```
# 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
```

Verifica della configurazione MRSC e dello stato della replica.

```
# 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}}'
```

Verifica della coerenza forte multi-Regione con letture immediate.

```
# 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
```

Esecuzione di scritture condizionali con garanzie 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
```

Esecuzione della pulizia delle risorse su una tabella globale 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
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento dei comandi AWS CLI .
  - CreateTable
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - UpdateItem
  - UpdateTable

Java

SDK per Java 2.x

Crea una tabella regionale pronta per la conversione MRSC utilizzando. 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();
        }
    }
```

Converti una tabella regionale in MRSC con repliche e verifica utilizzando. 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();
        }
    }
```

Descrivi una configurazione di tabella globale MRSC utilizzando. 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();
        }
    }
```

Per aggiungere elementi di test per verificare l'elevata consistenza 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();
        }
    }
```

Leggi gli elementi con letture coerenti dalle repliche MRSC utilizzando. 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();
        }
    }
```

Esegui aggiornamenti condizionali con garanzie MRSC utilizzando. 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();
        }
    }
```

Attendi che le repliche e i testimoni MRSC diventino attivi utilizzando. 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;
        }
    }
```

Pulisci le repliche e i testimoni MRSC utilizzando. 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();
        }
    }
```

Dimostrazione completa del flusso di lavoro MRSC utilizzando. 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;
        }
    }
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento delle API AWS SDK for Java 2.x .
  - CreateTable
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - UpdateItem
  - UpdateTable

Sicurezza per tabelle globali DynamoDB

Le repliche delle tabelle globali sono tabelle DynamoDB, quindi si utilizzano gli stessi metodi per controllare l'accesso alle repliche utilizzati per le tabelle a regione singola, AWS Identity and Access Management incluse le politiche di identità (IAM) e le politiche basate sulle risorse.

Questo argomento spiega come proteggere le tabelle globali DynamoDB utilizzando le autorizzazioni AWS Key Management Service IAM e la crittografia ().AWS KMS Scopri i ruoli collegati ai servizi (SLR) che consentono la replica e l'auto-scaling tra regioni, le autorizzazioni IAM necessarie per creare, aggiornare ed eliminare tabelle globali e le differenze tra le tabelle MREC (Multi-region Eventual Consistency) e Multi-region Strong Consistency (MRSC). Scopri AWS KMS anche le chiavi di crittografia per gestire in modo sicuro la replica tra regioni.

Ruoli collegati ai servizi per tabelle globali

Le tabelle globali DynamoDB si basano su ruoli collegati ai servizi SLRs () per gestire la replica tra regioni e le funzionalità di auto-scaling.

È sufficiente configurare questi ruoli una sola volta per account. AWS Una volta creati, gli stessi ruoli servono a tutte le tabelle globali del tuo account. Per ulteriori informazioni sui ruoli collegati al servizio, consulta Utilizzo dei ruoli collegati al servizio nella Guida per l'utente IAM.

Ruolo collegato al servizio

Amazon DynamoDB crea automaticamente il ruolo collegato `AWSServiceRoleForDynamoDBReplication` al servizio (SLR) quando crei la tua prima tabella globale. Questo ruolo gestisce la replica tra regioni per te.

Quando applichi politiche basate sulle risorse alle repliche, assicurati di non negare nessuna delle autorizzazioni definite nella scheda principale SLR, poiché ciò interromperebbe `AWSServiceRoleForDynamoDBReplicationPolicy` la replica. Se si negano le autorizzazioni SLR richieste, la replica da e verso le repliche interessate verrà interrotta e lo stato della tabella di replica cambierà in `REPLICATION_NOT_AUTHORIZED`.

- Per le tabelle globali Multi-region Eventual Consistency (MREC), se una replica rimane nello `REPLICATION_NOT_AUTHORIZED` stato per più di 20 ore, viene convertita in modo irreversibile in una tabella DynamoDB a regione singola.
- Per le tabelle globali MRSC (Multi-region Strong Consistency), la negazione delle autorizzazioni richieste comporta operazioni di scrittura e lettura fortemente coerenti. `AccessDeniedException` Se una replica rimane nello `REPLICATION_NOT_AUTHORIZED` stato per più di sette giorni, diventa definitivamente inaccessibile e le operazioni di scrittura e lettura fortemente coerenti continueranno a fallire con un errore. Alcune operazioni di gestione, come l'eliminazione delle repliche, avranno esito positivo.

Dimensionamento automatico del ruolo collegato al servizio

Quando si configura una tabella globale per la modalità di capacità fornita, è necessario configurare la scalabilità automatica per la tabella globale. La scalabilità automatica di DynamoDB utilizza il servizio Application AWS Auto Scaling per regolare dinamicamente la capacità di throughput assegnata sulle repliche di tabelle globali. Il servizio Application Auto Scaling crea un ruolo collegato al servizio (SLR) denominato. `AWSServiceRoleForApplicationAutoScaling_DynamoDBTable` Questo ruolo collegato al servizio viene creato automaticamente nel tuo AWS account quando configuri per la prima volta la scalabilità automatica per una tabella DynamoDB. Consente ad Application Auto Scaling di gestire la capacità delle tabelle assegnate e di creare allarmi. CloudWatch

Quando applicate politiche basate sulle risorse alle repliche, assicuratevi di non negare alcuna autorizzazione definita nel principio di Application `AWSApplicationAutoscalingDynamoDBTablePolicy`Auto Scaling SLR, poiché ciò interromperà la funzionalità di auto scaling.

Esempi di politiche IAM per ruoli collegati ai servizi

Una policy IAM con la seguente condizione non influisce sulle autorizzazioni richieste per la SLR di replica DynamoDB e la Auto Scaling SLR. AWS Questa condizione può essere aggiunta a policy altrimenti ampiamente restrittive per evitare l'interruzione involontaria della replica o della scalabilità automatica.

Esclusione delle autorizzazioni SLR richieste dalle politiche di negazione

L'esempio seguente mostra come escludere i ruoli principali collegati ai servizi dalle dichiarazioni di negazione:

```
"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"
        ]
    }
}
```

In che modo le tabelle globali utilizzano IAM AWS

Le sezioni seguenti descrivono le autorizzazioni richieste per le diverse operazioni delle tabelle globali e forniscono esempi di policy per aiutarti a configurare l'accesso appropriato per utenti e applicazioni.

> **Note**
>
> Tutte le autorizzazioni descritte devono essere applicate alla risorsa della tabella ARN specifica nelle regioni interessate. La risorsa della tabella ARN segue il formato`arn:aws:dynamodb:region:account-id:table/table-name`, in cui è necessario specificare i valori effettivi della regione, dell'ID dell'account e del nome della tabella.

Argomenti

- Creazione di tabelle globali e aggiunta di repliche
- Aggiornamento delle tabelle globali
- Eliminazione di tabelle globali e rimozione di repliche

Creazione di tabelle globali e aggiunta di repliche

Le tabelle globali DynamoDB supportano due modalità di coerenza: Multi-region Eventual Consistency (MREC) e Multi-region Strong Consistency (MRSC). Le tabelle globali MREC possono avere più repliche in un numero qualsiasi di regioni e garantire la coerenza finale. Le tabelle globali MRSC richiedono esattamente tre regioni (tre repliche o due repliche e un testimone) e forniscono una forte coerenza con zero Recovery Point Objective (RPO).

Le autorizzazioni necessarie per creare tabelle globali dipendono dal fatto che si stia creando una tabella globale con o senza un testimone.

Autorizzazioni per la creazione di tabelle globali

Le seguenti autorizzazioni sono necessarie sia per la creazione iniziale della tabella globale che per l'aggiunta di repliche in un secondo momento. Queste autorizzazioni si applicano alle tabelle globali Multi-Region Eventual Consistency (MREC) e Multi-Region Strong Consistency (MRSC).

- Le tabelle globali richiedono la replica tra regioni, che DynamoDB gestisce tramite il ruolo collegato al servizio (SLR). `AWSServiceRoleForDynamoDBReplication` La seguente autorizzazione

consente a DynamoDB di creare questo ruolo automaticamente quando si crea una tabella globale per la prima volta:

  - `iam:CreateServiceLinkedRole`

- Per creare una tabella globale o aggiungere una replica utilizzando l'`UpdateTable`API, è necessario disporre delle seguenti autorizzazioni sulla risorsa della tabella di origine:
  - `dynamodb:UpdateTable`
- È necessario disporre delle seguenti autorizzazioni sulla risorsa della tabella nelle Regioni per aggiungere le repliche:
  - `dynamodb:CreateTable`
  - `dynamodb:CreateTableReplica`
  - `dynamodb:Query`
  - `dynamodb:Scan`
  - `dynamodb:UpdateItem`
  - `dynamodb:PutItem`
  - `dynamodb:GetItem`
  - `dynamodb:DeleteItem`
  - `dynamodb:BatchWriteItem`

Autorizzazioni aggiuntive per le tabelle globali MRSC che utilizzano un testimone

Quando si crea una tabella globale MRSC (Multi-Region Strong Consistency) con una regione di riferimento, è necessario disporre della seguente autorizzazione sulla risorsa della tabella in tutte le regioni partecipanti (incluse sia le regioni di replica che la regione di riferimento):

- `dynamodb:CreateGlobalTableWitness`

Esempi di politiche IAM per la creazione di tabelle globali

Creazione di una tabella globale MREC o MRSC in tre regioni

La seguente politica basata sull'identità consente di creare una tabella globale MREC o MRSC denominata «users» in tre regioni, inclusa la creazione del ruolo collegato al servizio di replica DynamoDB richiesto.

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"
      ]
    }
  ]
}
```

Limitazione della creazione di tabelle globali MREC o MRSC a regioni specifiche

La seguente policy basata sull'identità consente di creare repliche di tabelle globali DynamoDB in regioni specifiche utilizzando la RequestedRegion chiave aws: condition, inclusa la creazione del ruolo collegato al servizio di replica DynamoDB richiesto.

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"
      ]
    }
  ]
}
```

Creazione di una tabella globale MRSC con testimone

La seguente policy basata sull'identità consente di creare una tabella globale DynamoDB MRSC denominata «users» con repliche in us-east-1 e us-east-2 e un testimone in us-west-2, inclusa la creazione del ruolo collegato al servizio di replica DynamoDB richiesto.

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"
      ]
    }
  ]
}
```

Limitazione della creazione di testimoni MRSC a regioni specifiche

Questa policy basata sull'identità consente di creare una tabella globale MRSC con repliche limitate a regioni specifiche utilizzando la chiave aws: RequestedRegion condition e la creazione di testimoni senza restrizioni in tutte le regioni, inclusa la creazione del ruolo collegato al servizio di replica DynamoDB richiesto.

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"
      ]
    }
  ]
```

```
}
```

Aggiornamento delle tabelle globali

Per modificare le impostazioni di replica per una tabella globale esistente utilizzando l'`UpdateTable`API, è necessaria la seguente autorizzazione sulla risorsa della tabella nella regione in cui stai effettuando la chiamata API:

- `dynamodb:UpdateTable`

È inoltre possibile aggiornare altre configurazioni di tabelle globali, come i criteri di ridimensionamento automatico e le impostazioni Time to Live. Per queste operazioni di aggiornamento aggiuntive sono necessarie le seguenti autorizzazioni:

- Per aggiornare una politica di ridimensionamento automatico delle repliche con l'`UpdateTableReplicaAutoScaling`API, è necessario disporre delle seguenti autorizzazioni sulla risorsa della tabella in tutte le regioni contenenti repliche:
  - `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`
- Per aggiornare le impostazioni Time to Live con l'`UpdateTimeToLive`API, è necessario disporre delle seguenti autorizzazioni sulla risorsa della tabella in tutte le regioni contenenti repliche:
  - `dynamodb:UpdateTimeToLive`

  Tieni presente che Time to Live (TTL) è supportato solo per le tabelle globali configurate con Multi-Region Eventual Consistency (MREC). Per ulteriori informazioni su come funzionano le tabelle globali con TTL, consulta Come funzionano le tabelle globali DynamoDB.

Eliminazione di tabelle globali e rimozione di repliche

Per eliminare una tabella globale, è necessario rimuovere tutte le repliche. Le autorizzazioni richieste per questa operazione variano a seconda che si stia eliminando una tabella globale con o senza una regione di riferimento.

Autorizzazioni per l'eliminazione di tabelle globali e la rimozione di repliche

Le seguenti autorizzazioni sono necessarie sia per rimuovere singole repliche sia per eliminare completamente le tabelle globali. L'eliminazione di una configurazione di tabella globale rimuove solo la relazione di replica tra tabelle in regioni diverse. Non elimina la tabella DynamoDB sottostante nell'ultima regione rimanente. La tabella nell'ultima regione continua a esistere come tabella DynamoDB standard con gli stessi dati e le stesse impostazioni. Queste autorizzazioni si applicano alle tabelle globali Multi-Region Eventual Consistency (MREC) e Multi-Region Strong Consistency (MRSC).

- Per rimuovere le repliche da una tabella globale utilizzando l'`UpdateTable`API, è necessaria la seguente autorizzazione sulla risorsa della tabella nella regione da cui si sta effettuando la chiamata API:
  - `dynamodb:UpdateTable`
- Sono necessarie le seguenti autorizzazioni sulla risorsa della tabella in ogni regione in cui stai rimuovendo una replica:
  - `dynamodb:DeleteTable`
  - `dynamodb:DeleteTableReplica`

Autorizzazioni aggiuntive per le tabelle globali MRSC che utilizzano un testimone

Per eliminare una tabella globale MRSC (Multi-region Strong Consistency) con un testimone, devi disporre della seguente autorizzazione sulla risorsa della tabella in tutte le regioni partecipanti (incluse sia le regioni di replica che la regione di riferimento):

- `dynamodb:DeleteGlobalTableWitness`

Esempi: politiche IAM per eliminare le repliche di una tabella globale

Eliminazione di repliche di tabelle globali

Questa policy basata sull'identità consente di eliminare una tabella globale DynamoDB denominata «users» e le relative repliche in tre regioni:

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"
      ]
    }
  ]
}
```

Eliminazione di una tabella globale MRSC con un testimone

Questa politica basata sull'identità consente di eliminare la replica e il testimone di una tabella globale MRSC denominata «users»:

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"
      ]
    }
  ]
}
```

Come vengono utilizzate le tabelle globali AWS KMS

Come tutte le tabelle DynamoDB, le repliche delle tabelle globali crittografano sempre i dati inattivi utilizzando chiavi di crittografia archiviate AWS in Key Management Service ().AWS KMS

Tutte le repliche in una tabella globale devono essere configurate con lo stesso tipo di chiave KMS (chiave di proprietà di AWS , chiave gestita da AWS o chiave gestita dal cliente).

> **Important**
>
> DynamoDB richiede l'accesso alla chiave di crittografia della replica per eliminare una replica. Se si desidera disabilitare o eliminare una chiave gestita dal cliente utilizzata per crittografare una replica perché si sta eliminando la replica, è necessario prima eliminare la replica, attendere che lo stato della tabella su una delle repliche rimanenti cambi in `ACTIVE`, quindi disabilitare o eliminare la chiave.

Per una tabella globale configurata per coerenza finale multi-Regione (MREC), se si disabilita o si revoca l'accesso di DynamoDB a una chiave gestita dal cliente utilizzata per crittografare una replica, la replica da e verso la replica verrà interrotta e lo stato della replica cambierà in `INACCESSIBLE_ENCRYPTION_CREDENTIALS`. Se una replica in una tabella globale MREC rimane nello stato `INACCESSIBLE_ENCRYPTION_CREDENTIALS` per più di 20 ore, viene convertita in modo irreversibile in una tabella DynamoDB a Regione singola.

Per una tabella globale configurata per elevata consistenza multi-Regione (MRSC), se si disabilita o si revoca l'accesso di DynamoDB a una chiave gestita dal cliente utilizzata per crittografare

una replica, la replica da e verso la replica si interromperà, i tentativi di eseguire scritture oppure operazioni a elevata consistenza di lettura sulla replica restituiranno un errore e lo stato della replica cambierà in `INACCESSIBLE_ENCRYPTION_CREDENTIALS`. Se una replica in una tabella globale MRSC rimane nello stato `INACCESSIBLE_ENCRYPTION_CREDENTIALS` per più di sette giorni, a seconda delle autorizzazioni specifiche revocate, la replica verrà archiviata o diventerà definitivamente inaccessibile.

Informazioni sulla fatturazione di Amazon DynamoDB per le tabelle globali

Questa guida descrive come funziona la fatturazione di DynamoDB per le tabelle globali, identificando i componenti che contribuiscono al costo delle tabelle globali, incluso un esempio pratico.

Le tabelle globali Amazon DynamoDB sono un database completamente gestito, serverless, multi-Regione e multi-attivo. Le tabelle globali sono progettate per una disponibilità del 99,999%, offrendo una maggiore resilienza delle applicazioni e una migliore continuità aziendale. Le tabelle globali replicano automaticamente le tabelle DynamoDB tra le regioni AWS di tua scelta in modo da ottenere prestazioni di lettura e scrittura locali rapide.

Come funziona

Il modello di fatturazione per le tabelle globali è diverso dalle tabelle DynamoDB a Regione singola. Le operazioni di scrittura per le tabelle DynamoDB a Regione singola vengono fatturate utilizzando le seguenti unità:

- Write Request Units (WRUs) per la modalità di capacità su richiesta, in cui viene addebitata una WRU per ogni scrittura fino a 1 KB
- Unità di capacità di scrittura (WCUs) per la modalità di capacità fornita, in cui una WCU fornisce una scrittura al secondo per un massimo di 1 KB

Quando si crea una tabella globale aggiungendo una tabella di replica a una tabella a Regione singola esistente, tale tabella a Regione singola diventa una tabella di replica, il che significa che cambiano anche le unità utilizzate per fatturare le scritture sulla tabella. Le operazioni di scrittura per le tabelle di replica vengono fatturate utilizzando le seguenti unità:

- Unità di richiesta di scrittura replicate (rWRUs) per la modalità di capacità su richiesta, in cui viene addebitata una RWRu per tabella di replica per ogni scrittura fino a 1 KB

- Unità di capacità di scrittura replicate (rWCUs) per la modalità di capacità fornita, in cui una WCU per tabella di replica fornisce una scrittura al secondo per un massimo di 1 KB

Gli aggiornamenti agli indici secondari globali (GSIs) vengono fatturati utilizzando le stesse unità delle tabelle DynamoDB a regione singola, anche se la tabella di base per il GSI è una tabella di replica. Le operazioni di aggiornamento di vengono fatturate utilizzando le seguenti unità: GSIs

- Write Request Units (WRUs) per la modalità di capacità su richiesta, in cui viene addebitata una WRU per ogni scrittura fino a 1 KB
- Unità di capacità di scrittura (WCUs) per la modalità di capacità fornita, in cui una WCU fornisce una scrittura al secondo per un massimo di 1 KB

Le unità di scrittura replicate (r WCUs e rWRUs) hanno lo stesso prezzo delle unità di scrittura a regione singola (and). WCUs WRUs I costi per il trasferimento di dati tra Regioni vengono applicati per le tabelle globali poiché i dati vengono replicati tra Regioni. I costi di scrittura replicata (rWCU o rWRU) sono previsti in ogni Regione che contiene una tabella di replica per la tabella globale.

Le operazioni di lettura dalle tabelle a Regione singola e dalle tabelle di replica utilizzano le seguenti unità:

- Read Request Units (RRUs) per la modalità di capacità on demand, in cui viene addebitata una RRU per ogni lettura altamente coerente fino a 4 KB
- Unità di capacità di lettura (RCUs) per le tabelle assegnate, in cui una RCU fornisce una lettura estremamente coerente al secondo per un massimo di 4 KB

Modalità di coerenza e fatturazione

Le unità di scrittura replicate (r WCUs e rWRUs) utilizzate per la fatturazione delle operazioni di scrittura sono identiche sia per le modalità Multi-region Strong Consistency (MRSC) che Multi-region Eventual Consistency (MREC). Le tabelle globali che utilizzano la modalità MRSC (Multi-region Strong Consistency) configurate con un testimone non comportano costi di unità di scrittura replicati (r WCUs e rWRUs), costi di archiviazione o costi di trasferimento dei dati per la replica al witness.

Esempio di fatturazione per tabelle globali DynamoDB

Si esamina uno scenario di esempio di più giorni per vedere come funziona in pratica la fatturazione globale delle richieste di scrittura su tabelle (si noti che questo esempio considera solo le richieste di

scrittura e non include i costi di ripristino della tabella e di trasferimento dati tra Regioni diverse che sarebbero sostenuti nell'esempio):

Giorno 1 – Tabella a Regione singola: si dispone di una tabella DynamoDB on demand a Regione singola denominata Tabella_A nella Regione us-west-2. Vengono scritti 100 elementi da 1 KB su Tabella_A. Per queste operazioni di scrittura a Regione singola, viene addebitata 1 unità di richiesta di scrittura (WRU) per 1 KB scritto. Le spese del giorno 1 sono:

- 100 WRUs nella regione us-west-2 per le scritture su una singola regione

Il totale delle unità di richiesta addebitate il giorno 1:100. WRUs

Giorno 2 – Creazione di una tabella globale: si crea una tabella globale aggiungendo una replica a Tabella_A nella Regione us-east-2. La Tabella_A è ora una tabella globale con due tabelle di replica: una nella Regione us-west-2 e una nella Regione us-east-2. Si scrivono 150 elementi da 1 KB nella tabella di replica nella Regione us-west-2. Le spese del giorno 2 sono:

- 150 r WRUs nella regione us-west-2 per le scritture replicate
- 150 r WRUs nella regione us-east-2 per le scritture replicate

Il totale delle unità di richiesta addebitate il giorno 2:300 r. WRUs

Giorno 3 – Aggiunta di un indice secondario globale: si aggiunge un GSI alla tabella di replica nella Regione us-east-2 che proietta tutti gli attributi dalla tabella di base (replica). La tabella globale crea automaticamente il GSI sulla tabella di replica nella Regione us-west-2 per l'utente. Si scrivono 200 nuovi record da 1 KB nella tabella di replica nella Regione us-west-2. Le spese del giorno 3 sono:

- • 200 r WRUs nella regione us-west-2 per le scritture replicate
- • 200 WRUs nella regione us-west-2 per gli aggiornamenti GSI
- • 200 r WRUs nella regione us-east-2 per le scritture replicate
- • 200 WRUs nella regione us-east-2 per gli aggiornamenti GSI

Il totale delle unità di richiesta di scrittura addebitate il giorno 3: 400 WRUs e 400 r. WRUs

I costi totali per unità di scrittura per tutti e tre i giorni sono 500 WRUs (100 WRU il giorno 1 + 400 il WRUs giorno 3) e 700 r WRUs (300 r il WRUs giorno 2 + 400 r il WRUs giorno 3).

In sintesi, le operazioni di scrittura delle tabelle di replica vengono fatturate in unità di scrittura replicate in tutte le Regioni che contengono una tabella di replica. Se disponi di indici secondari globali, ti vengono addebitate le unità di scrittura per gli aggiornamenti GSIs in tutte le aree che contengono un GSI (che in una tabella globale sono tutte le regioni che contengono una tabella di replica).

Versioni delle tabelle globali DynamoDB

Sono disponibili due versioni delle tabelle globali DynamoDB: Tabelle globali versione 2019.11.21 (Corrente) e Tabelle globali versione 2017.11.29 (Legacy). Si consiglia di utilizzare la versione 2019.11.21 (Corrente) poiché è più facile da usare, è supportata in più Regioni e ha un costo inferiore per la maggior parte dei carichi di lavoro rispetto alla versione 2017.11.29 (Legacy).

Determinare la versione di una tabella globale

Determinazione della versione utilizzando il AWS CLI

Identificazione di una replica di una tabella globale versione 2019.11.21 (Corrente)

Per determinare se una tabella è una replica delle tabelle globali versione 2019.11.21 (Corrente), invoca il comando `describe-table` per la tabella. Se l'output contiene l'attributo `GlobalTableVersion`con il valore "2019.11.21", allora la tabella è una replica della tabella globale versione 2019.11.21 (Corrente).

Un comando CLI di esempio per `describe-table`:

```
aws dynamodb describe-table \
--table-name users \
--region us-east-2
```

L'output (abbreviato) contiene l'attributo `GlobalTableVersion` con il valore "2019.11.21", quindi questa tabella è una replica della tabella globale versione 2019.11.21 (Corrente).

```
{
    "Table": {
        "AttributeDefinitions": [
            {
                "AttributeName": "id",
                "AttributeType": "S"
            },
            {
```

```
                "AttributeName": "name",
                "AttributeType": "S"
            }
        ],
        "TableName": "users",
        ...
        "GlobalTableVersion": "2019.11.21",
        "Replicas": [
            {
                "RegionName": "us-west-2",
                "ReplicaStatus": "ACTIVE",
            }
        ],
        ...
    }
}
```

Identificazione di una replica di una tabella globale versione 2017.11.29 (Corrente)

La versione 2017.11.29 (Legacy) utilizza un set di comandi dedicato per la gestione delle tabelle globali. Per determinare se una tabella è una replica delle tabelle globali 2017.11.29 (Legacy), invoca il comando `describe-global-table` per la tabella. Se si riceve una risposta positiva, la tabella è una replica di tabella globale versione 2017.11.29 (Legacy). Se il comando `describe-global-table` restituisce un errore `GlobalTableNotFoundException`, la tabella non è una replica della versione 2017.11.29 (Legacy).

Un comando CLI di esempio per `describe-global-table`:

```
aws dynamodb describe-global-table \
--table-name users \
--region us-east-2
```

Il comando restituisce una risposta positiva, pertanto la tabella è una replica di tabella globale versione 2017.11.29 (Legacy).

```
{
    "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"
    }
}
```

Determinare la versione utilizzando la console DynamoDB

Per identificare la versione di una replica di una tabella globale, procedi come segue:

1. Apri la console DynamoDB a casa. https://console.aws.amazon.com/dynamodb/
2. Nel riquadro di navigazione sul lato sinistro della console scegli Tables (Tabelle).
3. Seleziona la tabella per la quale desideri identificare la versione delle tabelle globali.
4. Scegliere la scheda Global Tables (Tabelle globali).

   La sezione Riepilogo mostra la versione delle tabelle globali in uso.

Differenze di comportamento tra le versioni Legacy e Corrente

L'elenco seguente descrive le differenze di comportamento tra le versioni Legacy e Corrente delle tabelle globali.

- La versione 2019.11.21 (Corrente) utilizza meno capacità di scrittura per diverse operazioni DynamoDB rispetto alla versione 2017.11.29 (Legacy) e pertanto è più conveniente per la maggior parte dei clienti. Le differenze per queste operazioni DynamoDB sono le seguenti:
  - PutItemPer richiamare un elemento da 1 KB in una regione e replicarlo in altre regioni sono necessarie 2 r WRUs per regione per la versione 2017.11.29 (versione precedente), ma solo 1 RWRu per la versione 2019.11.21 (versione corrente).
  - L'invocazione UpdateItemdi un elemento da 1 KB richiede 2 r WRUs nella regione di origine e 1 RWRu per regione di destinazione per 2017.11.29 (Legacy), ma solo 1 RWRu per entrambe le regioni di origine e di destinazione per la versione 2019.11.21 (corrente).
  - L'invocazione DeleteItemdi un elemento da 1 KB richiede 1 RWRu nella regione di origine e 2 r WRUs per regione di destinazione per il 2017.11.29 (Legacy), ma solo 1 RWRu sia per la regione di origine che per la regione di destinazione per la versione 2019.11.21 (corrente).

La tabella seguente mostra l'utilizzo di rWRU delle tabelle 2017.11.29 (Legacy) e 2019.11.21 (Corrente) per un elemento da 1 KB in due Regioni.

Operazioni	2017.11.29 (Legacy)	2019.11.21 (Corrente)	Risparmi
PutItem	4 r WRUs	2 r WRUs	50%
UpdateItem	3 r WRUs	2 r WRUs	33%
DeleteItem	3 r WRUs	2 r WRUs	33%

- la versione 2017.11.29 (Legacy) è disponibile solo in 11. Regioni AWS Tuttavia, la versione 2019.11.21 (Corrente) è disponibile in tutte le Regioni AWS.
- È possibile creare tabelle globali nella versione 2017.11.29 (Legacy) creando prima un set di tabelle regionali vuote, quindi richiamando l'API per formare la CreateGlobalTabletabella globale. Puoi creare tabelle globali nella versione 2019.11.21 (Current) richiamando l'UpdateTableAPI per aggiungere una replica a una tabella regionale esistente.
- La versione 2017.11.29 (Legacy) richiede di svuotare tutte le repliche nella tabella prima di aggiungere una replica in una nuova Regione (anche durante la creazione). La versione 2019.11.21 (Corrente) supporta l'aggiunta e la rimozione di repliche nelle Regioni su una tabella che contiene già dati.
- la versione 2017.11.29 (Legacy) utilizza il seguente set di piani di controllo dedicato per la gestione delle repliche: APIs
  - CreateGlobalTable
  - DescribeGlobalTable
  - DescribeGlobalTableSettings
  - ListGlobalTables
  - UpdateGlobalTable
  - UpdateGlobalTableSettings

  la versione 2019.11.21 (Current) utilizza and per gestire le repliche. DescribeTableUpdateTable APIs
- La versione 2017.11.29 (Legacy) pubblica due record dei flussi DynamoDB per ogni scrittura. La versione 2019.11.21 (Corrente) pubblica solo un record dei flussi DynamoDB per ogni scrittura.

- La versione 2017.11.29 (Legacy) compila e aggiorna gli attributi `aws:rep:deleting`, `aws:rep:updateregion` e `aws:rep:updatetime`. La versione 2019.11.21 (Corrente) non compila né aggiorna questi attributi.
- La versione 2017.11.29 (Legacy) non sincronizza le impostazioni Utilizzo del Time to Live in DynamoDB tra le repliche. La versione 2019.11.21 (Corrente) sincronizza le impostazioni del TTL tra le repliche.
- La versione 2017.11.29 (Legacy) non replica le eliminazioni TTL su altre repliche. La versione 2019.11.21 (Corrente) replica le eliminazioni TTL su tutte le repliche.
- La versione 2017.11.29 (Legacy) non sincronizza le impostazioni di dimensionamento automatico tra le repliche. La versione 2019.11.21 (Corrente) sincronizza le impostazioni di dimensionamento automatico tra le repliche.
- La versione 2017.11.29 (Legacy) non sincronizza le impostazioni degli indici secondari globali (GSI) tra le repliche. La versione 2019.11.21 (Corrente) sincronizza le impostazioni degli indici secondari globali tra le repliche.
- La versione 2017.11.29 (Legacy) non sincronizza le impostazioni di crittografia dei dati a riposo tra le repliche. La versione 2019.11.21 (Corrente) sincronizza le impostazioni di crittografia dei dati a riposo tra le repliche.
- La versione 2017.11.29 (Legacy) pubblica la metrica `PendingReplicationCount`. La versione 2019.11.21 (Corrente) non pubblica questa metrica.

Aggiornamento alla versione corrente

Autorizzazioni richieste per l'aggiornamento delle tabelle globali

Per eseguire l'aggiornamento alla versione 2019.11.21 (Corrente), è necessario disporre delle autorizzazioni `dynamodb:UpdateGlobalTableversion` in tutte le Regioni con repliche. Queste autorizzazioni richieste si aggiungono alle autorizzazioni necessarie per accedere alla console DynamoDB e visualizzare le tabelle.

La seguente policy IAM concede le autorizzazioni per aggiornare qualsiasi tabella globale alla versione 2019.11.21 (Corrente).

```
{
    "version": "2012-10-17",
    "Statement": [
        {
```

```
            "Effect": "Allow",
            "Action": "dynamodb:UpdateGlobalTableversion",
            "Resource": "*"
        }
    ]
}
```

La seguente policy IAM concede le autorizzazioni per aggiornare solo la tabella globale `Music`, con repliche in due Regioni, alla versione 2019.11.21 (Corrente).

```
{
    "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"
            ]
        }
    ]
}
```

Cosa aspettarsi durante l'aggiornamento

- Tutte le repliche di tabelle globali continueranno a elaborare il traffico di lettura e scrittura durante l'aggiornamento.
- Il processo di aggiornamento richiede da pochi minuti a diverse ore a seconda delle dimensioni della tabella e del numero di repliche.
- Durante il processo di aggiornamento, il valore di TableStatuscambierà da a. `ACTIVE` `UPDATING` È possibile visualizzare lo stato della tabella richiamando l'DescribeTableAPI o con la vista Tabelle nella console DynamoDB.
- Il dimensionamento automatico non modificherà le impostazioni di capacità allocata per una tabella globale durante l'aggiornamento della tabella. Si consiglia vivamente di impostare la tabella sulla modalità con capacità on demand durante l'aggiornamento.
- Se si sceglie di utilizzare la modalità con capacità allocata con il dimensionamento automatico durante l'aggiornamento, è necessario aumentare il throughput minimo di lettura e scrittura nelle

policy per soddisfare eventuali aumenti previsti di traffico per evitare la limitazione (della larghezza di banda della rete) durante l'aggiornamento.

- La metrica `ReplicationLatency` può segnalare temporaneamente i picchi di latenza o interrompere la segnalazione dei dati delle metriche durante il processo di aggiornamento. Per ulteriori informazioni, consulta the section called "ReplicationLatency".
- Al termine del processo di aggiornamento, lo stato della tabella torna su `ACTIVE`.

Comportamento dei flussi DynamoDB prima, durante e dopo l'aggiornamento

Operazioni	Regione di replica	Comportamento prima dell'aggiornamento	Comportamento durante l'aggiornamento	Comportamento dopo l'aggiornamento
Put o Update	Origine	La compilazione dei timestamp avviene utilizzando. UpdateItem	La popolazione del timestamp avviene utilizzando. PutItem	Non viene generato alcun timestamp visibile al cliente.
		Vengono generati due record di flussi. Il primo record contiene gli attributi scritti dal cliente. Il secondo record contiene gli attributi `aws:rep:*` .	Vengono generati due record di flussi. Il primo record contiene gli attributi scritti dal cliente. Il secondo record contiene gli attributi `aws:rep:*` .	Viene generato un singolo record di flussi contenente gli attributi scritti dal cliente.
		WCUs Vengono consumate due r per ogni scrittura del cliente.	WCUs Vengono consumate due r per ogni scrittura da parte del cliente.	Viene utilizzata una rWCU per ogni scrittura da parte del cliente.

Operazioni	Regione di replica	Comportamento prima dell'aggiornamento	Comportamento durante l'aggiornamento	Comportamento dopo l'aggiornamento
		`ReplicationLatency` e le `PendingReplicationCount` metriche sono pubblicate in CloudWatch.	`ReplicationLatency` e le `PendingReplicationCount` metriche sono pubblicate in. CloudWatch	`ReplicationLatency` la metrica è pubblicata in. CloudWatch
	Destinazione	La replica avviene utilizzando. PutItem	La replica avviene utilizzando. PutItem	La replica avviene utilizzando. PutItem
		Viene generato un singolo record dei flussi, che contiene sia gli attributi scritti dal cliente che gli attributi `aws:rep:*` .	Viene generato un singolo record dei flussi, che contiene sia gli attributi scritti dal cliente che gli attributi `aws:rep:*` .	Viene generato un singolo record dei flussi, che contiene solo gli attributi scritti dal cliente e nessun attributo di replica.

Operazioni	Regione di replica	Comportamento prima dell'aggiornamento	Comportamento durante l'aggiornamento	Comportamento dopo l'aggiornamento
		Viene utilizzata una rWCU se l'elemento esiste nella Regione di destinazione. Due rWCUs vengono consumate se l'articolo non esiste nella regione di destinazione.	Viene utilizzata una rWCU se l'elemento esiste nella Regione di destinazione. Due rWCUs vengono consumati se l'articolo non esiste nella regione di destinazione.	Viene utilizzata una rWCU per ogni scrittura da parte del cliente.
		`ReplicationLatency` e le `PendingReplicationCount` metriche sono pubblicate in CloudWatch.	`ReplicationLatency` e le `PendingReplicationCount` metriche sono pubblicate in. CloudWatch	`ReplicationLatency` la metrica è pubblicata in. CloudWatch
Elimina	Origine	Elimina qualsiasi elemento con un timestamp più piccolo utilizzando. DeleteItem	Elimina qualsiasi elemento con un timestamp più piccolo utilizzando. DeleteItem	Elimina qualsiasi elemento con un timestamp più piccolo utilizzando. DeleteItem

Operazioni	Regione di replica	Comportamento prima dell'aggiornamento	Comportamento durante l'aggiornamento	Comportamento dopo l'aggiornamento
		Viene generato un singolo record dei flussi, che contiene sia gli attributi scritti dal cliente che gli attributi `aws:rep:*` .	Viene generato un singolo record dei flussi, che contiene sia gli attributi scritti dal cliente che gli attributi `aws:rep:*` .	Viene generato un singolo record dei flussi, che contiene gli attributi scritti dal cliente.
		Viene utilizzata una rWCU per ogni eliminazione da parte del cliente.	Viene utilizzata una rWCU per ogni eliminazione da parte del cliente.	Viene utilizzata una rWCU per ogni eliminazione da parte del cliente.
		`ReplicationLatency` e le `PendingReplicationCount` metriche sono pubblicate in. CloudWatch	`ReplicationLatency` e le `PendingReplicationCount` metriche sono pubblicate in. CloudWatch	`ReplicationLatency` la metrica è pubblicata in. CloudWatch

Operazioni	Regione di replica	Comportamento prima dell'aggiornamento	Comportamento durante l'aggiornamento	Comportamento dopo l'aggiornamento
	Destinazione	Le eliminazioni in due fasi avvengono secondo la seguente procedura: • Nella Fase 1, UpdateItem imposta il flag di eliminazione. • Nella Fase 2, DeleteItem elimina l'elemento.	Elimina l'elemento utilizzando. DeleteItem	Elimina l'elemento utilizzando. DeleteItem
		Vengono generati due record di flussi. Il primo record contiene la modifica al campo `aws:rep:deleting` . Il secondo record contiene gli attributi scritti dal cliente e gli attributi `aws:rep:*` .	Viene generato un singolo record dei flussi, che contiene gli attributi scritti dal cliente.	Viene generato un singolo record dei flussi, che contiene gli attributi scritti dal cliente.

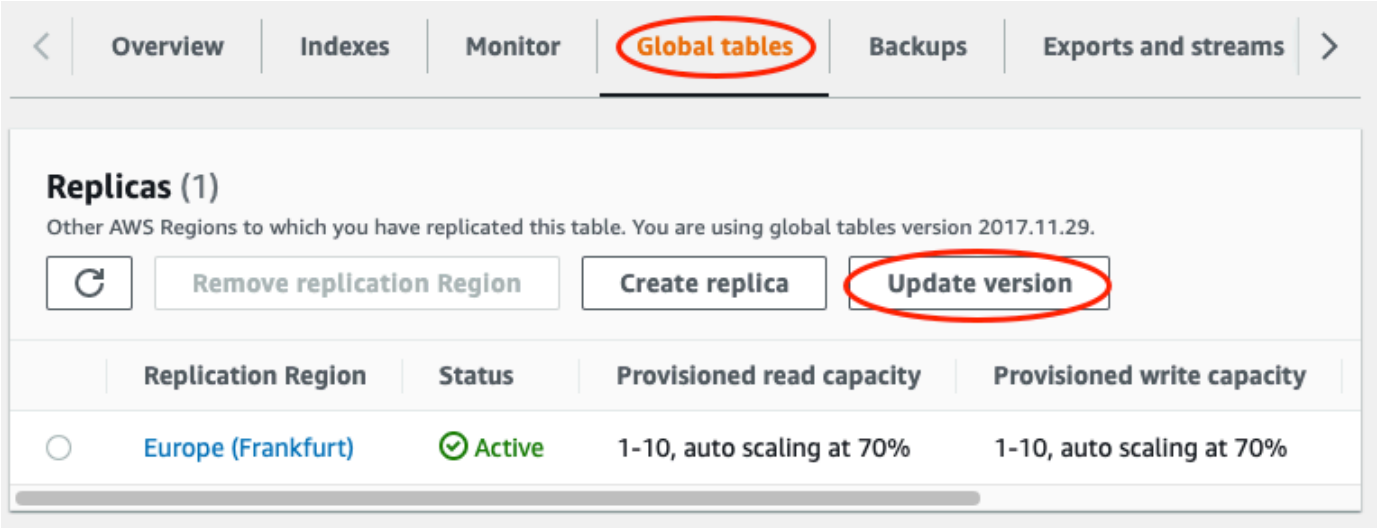
Operazioni	Regione di replica	Comportamento prima dell'aggiornamento	Comportamento durante l'aggiornamento	Comportamento dopo l'aggiornamento
		WCUs Vengono consumate due r per ogni eliminazione da parte del cliente.	Viene utilizzata una rWCU per ogni eliminazione da parte del cliente.	Viene utilizzata una rWCU per ogni eliminazione da parte del cliente.
		`ReplicationLatency` e le `PendingReplicationCount` metriche sono pubblicate in CloudWatch.	`ReplicationLatency` la metrica è pubblicata in. CloudWatch	`ReplicationLatency` la metrica è pubblicata in. CloudWatch

Aggiornamento alla versione 2019.11.21 (Corrente)

Attieniti alla seguente procedura per aggiornare la versione delle tabelle globali DynamoDB utilizzando la Console di gestione AWS.

Come eseguire l'aggiornamento delle tabelle globali alla versione 2019.11.21 (Corrente)

1. Apri la console DynamoDB a casa. https://console.aws.amazon.com/dynamodb/
2. Nel pannello di navigazione sul lato sinistro della console, seleziona Tabelle e quindi seleziona la tabella globale che desideri aggiornare alla versione 2019.11.21 (Corrente).
3. Scegliere la scheda Global Tables (Tabelle globali).
4. Scegli Aggiorna versione.



5. Leggi e accetta i nuovi requisiti, quindi scegli Update version (Aggiorna versione).
6. Al termine del processo di aggiornamento, la versione delle tabelle globali visualizzata sulla console cambia in 2019.11.21.

Best practice per le tabelle globali

Le sezioni seguenti descrivono le best practice per l'implementazione e l'utilizzo di tabelle globali.

Versione

Sono disponibili due versioni delle tabelle globali DynamoDB: versione 2019.11.21 (Corrente) e versione 2017.11.29 (Legacy). Sarebbe opportuno utilizzare la versione 2019.11.21 (Corrente) ogni volta che è possibile.

Deletion protection (Protezione da eliminazione)

È necessario abilitare la protezione dall'eliminazione sulle repliche globali di tabelle che si desidera proteggere dall'eliminazione accidentale. È necessario abilitare la protezione dall'eliminazione su ogni replica.

Uso di AWS CloudFormation

CloudFormation attualmente non supporta il coordinamento di risorse multi-Regione come le tabelle globali tra gli stack. Se si definisce ogni replica di una tabella globale in uno stack Regionale separato, si verificheranno errori dovuti alla deviazione rilevata tra gli stack durante l'esecuzione degli aggiornamenti delle repliche. Per evitare questo problema, è necessario scegliere una Regione come

Regione di riferimento per l'implementazione delle tabelle globali e definire tutte le repliche della tabella globale nello stack di quella Regione.

> **Important**
>
> Non è possibile convertire una risorsa di tipo `AWS::DynamoDB::Table` in una risorsa di tipo `AWS::DynamoDB::GlobalTable` modificandone il tipo nel modello. Il tentativo di convertire una tabella a Regione singola in una tabella globale modificandone il tipo di risorsa CloudFormation può comportare l'eliminazione della tabella DynamoDB.

È possibile utilizzare la risorsa `AWS::DynamoDB::GlobalTable` per creare una tabella in una singola Regione. Questa tabella verrà implementata come qualsiasi altra tabella a Regione singola. Se successivamente si aggiorna lo stack per aggiungere altre Regioni a una risorsa, le repliche verranno aggiunte alla tabella e questa verrà convertita in modo sicuro in una tabella globale.

Se si dispone di una risorsa `AWS::DynamoDB::Table` esistente che desideri convertire in risorsa `AWS::DynamoDB::GlobalTable`, le fasi consigliate per convertire il tipo di risorsa sono:

1. Impostare la policy di eliminazione `AWS::DynamoDB::Table` da mantenere.
2. Rimuovere la tabella dalla definizione degli stack.
3. Aggiungere repliche alla tabella a Regione singola nella console AWS, convertendola in una tabella globale.
4. Importare la nuova tabella globale come una nuova risorsa `AWS::DynamoDB::GlobalTable` nello stack.

Backup e Recupero point-in-time

L'abilitazione di backup automatici e recupero point-in-time (PITR) per una replica in una tabella globale può essere sufficiente per soddisfare gli obiettivi di disaster recovery. I backup di replica creati con AWS-Backup possono essere replicati automaticamente tra le Regioni per una maggiore resilienza. Per la scelta della strategia di backup e abilitazione del PITR, considera gli obiettivi del piano di disaster recovery nel contesto della disponibilità elevata in più Regioni.

Progettazione per un'elevata disponibilità multi-Regione

Per un prontuario sull'implementazione di tabelle globali, consulta Best practice per la progettazione di tabelle DynamoDB globali.

Utilizzo di elementi e attributi in DynamoDB

In Amazon DynamoDB, un elemento è una raccolta di attributi. Ogni attributo ha un nome e un valore. Il valore di un attributo può essere un tipo scalare, un set o un tipo di documento. Per ulteriori informazioni, consulta Amazon DynamoDB: come funziona.

DynamoDB fornisce quattro operazioni per le funzionalità di creazione, lettura, aggiornamento ed eliminazione CRUD (create/read/update/delete) di base. Tutte queste operazioni sono atomiche.

- `PutItem`: crea un elemento.
- `GetItem`: legge un elemento.
- `UpdateItem`: aggiorna un elemento.
- `DeleteItem`: elimina un elemento.

Ognuna di queste operazioni richiede di specificare la chiave primaria dell'elemento da usare. Per leggere, ad esempio, un elemento con `GetItem`, devi specificare la chiave di partizione e la chiave di ordinamento (se applicabile) per l'elemento.

Oltre alle quattro operazioni CRUD di base, DynamoDB fornisce anche le seguenti funzionalità:

- `BatchGetItem`: legge fino a 100 elementi da una o più tabelle.
- `BatchWriteItem`: crea o elimina fino a 25 elementi in una o più tabelle.

Queste operazioni batch combinano diverse operazioni CRUD in una singola richiesta. Le operazioni batch, inoltre, leggono e scrivono gli elementi in parallelo, per ridurre al minimo la latenza delle risposte.

In questa sezione viene descritto come usare queste operazioni e sono inclusi argomenti correlati, come gli aggiornamenti condizionali e i contatori atomici. Questa sezione include anche un codice di esempio che utilizza il AWS SDKs.

Argomenti

- Dimensioni e formati degli elementi di DynamoDB
- Lettura di un elemento
- Scrittura di un elemento
- Valori restituiti

- Operazioni batch
- Contatori atomici
- Scritture condizionali
- Utilizzo di espressioni in DynamoDB
- Utilizzo del Time to Live in DynamoDB
- Esecuzione di query in DynamoDB
- Scansione di tabelle in DynamoDB
- PartiQL: un linguaggio di query compatibile con SQL per Amazon DynamoDB
- Utilizzo degli elementi: Java
- Uso di elementi: .NET

Dimensioni e formati degli elementi di DynamoDB

Le tabelle DynamoDB non hanno uno schema, tranne per la chiave primaria, pertanto gli elementi in una tabella possono avere attributi, dimensioni e tipi di dati diversi.

La dimensione totale di un elemento è data dalla somma delle lunghezze dei nomi e dei valori dei relativi attributi, più eventuali overhead applicabili come descritto di seguito. Per stimare le dimensioni degli attributi, puoi utilizzare le linee guida seguenti:

- Le stringhe sono Unicode con codifica binaria UTF-8. La dimensione di una stringa è (numero di byte con codifica UTF-8 del nome dell'attributo) + (numero di byte con codifica UTF-8).
- I numeri hanno lunghezza variabile, con un massimo di 38 cifre significative. Gli zero iniziali e finali vengono tagliati. La dimensione di un numero è circa (numero di byte con codifica UTF-8 del nome dell'attributo) + (1 byte ogni due cifre significative) + (1 byte).
- Un valore binario deve essere codificato in formato base64 per essere inviato a DynamoDB, ma per il calcolo della dimensione viene utilizzata la lunghezza dei byte in formato RAW del valore. La dimensione di un attributo di tipo binario è (numero di byte con codifica UTF-8 del nome dell'attributo) + (numero di byte in formato RAW).
- La dimensione di un attributo null o booleano è (numero di byte con codifica UTF-8 del nome dell'attributo) + (1 byte).
- Un attributo di tipo `List` o `Map` richiede 3 byte di overhead, indipendentemente dal contenuto. La dimensione di un `List` o di una `Map` è (numero di byte con codifica UTF-8 del nome dell'attributo)

+ somma (dimensione degli elementi annidati) + (3 byte). La dimensione di un `List` o di una `Map` vuoti è (numero di byte con codifica UTF-8 del nome dell'attributo) + (3 byte).

- Ogni elemento `List` o `Map` richiede anche un byte di sovraccarico.

> **Note**
>
> È consigliabile preferire nomi di attributo brevi piuttosto che lunghi. Ciò consente di ridurre la quantità di spazio di archiviazione richiesta, ma può anche ridurre la quantità RCU/WCUs di spazio utilizzato.

Ai fini della fatturazione dell'archiviazione, ogni elemento include un overhead di archiviazione per elemento che dipende dalle funzionalità abilitate.

- Tutti gli elementi in DynamoDB richiedono 100 byte di overhead di archiviazione per l'indicizzazione.
- Alcune funzionalità DynamoDB (tabelle globali, transazioni, acquisizione dei dati di modifica per Kinesis Data Streams con DynamoDB) richiedono un overhead di archiviazione aggiuntivo per tenere conto degli attributi creati dal sistema derivanti dall'abilitazione di tali funzionalità. Ad esempio, le tabelle globali richiedono un overhead aggiuntivo di 48 byte di archiviazione.

Lettura di un elemento

Per leggere un elemento da una tabella DynamoDB, utilizza l'operazione `GetItem`. Devi fornire il nome della tabella e la chiave primaria dell'elemento desiderato.

Example

L' AWS CLI esempio seguente mostra come leggere un elemento dalla `ProductCatalog` tabella.

```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"1"}}'
```

> **Note**
>
> Con `GetItem`, devi specificare l'intera chiave primaria, non una parte di essa. Se, ad esempio, una tabella ha una chiave primaria composita (chiave di partizione e chiave

di ordinamento), devi fornire un valore per la chiave di partizione e uno per la chiave di ordinamento.

Una richiesta `GetItem` esegue una lettura consistente finale per impostazione predefinita. Puoi usare il parametro `ConsistentRead` per richiedere invece una lettura consistente assoluta. (Ciò consuma unità di capacità di lettura aggiuntive, ma restituisce la maggior parte delle up-to-date versioni dell'articolo.)

`GetItem` restituisce tutti gli attributi dell'elemento. Puoi usare un'espressione di proiezione per restituire solo alcuni degli attributi. Per ulteriori informazioni, consulta Utilizzo di espressioni di proiezione in DynamoDB.

Per restituire il numero di unità di capacità di lettura utilizzate da `GetItem`, imposta il parametro `ReturnConsumedCapacity` su `TOTAL`.

Example

L'esempio seguente AWS Command Line Interface (AWS CLI) mostra alcuni `GetItem` parametri opzionali.

```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"1"}}' \
    --consistent-read \
    --projection-expression "Description, Price, RelatedItems" \
    --return-consumed-capacity TOTAL
```

Scrittura di un elemento

Per creare, aggiornare o eliminare un elemento in una tabella DynamoDB, utilizza una delle operazioni seguenti:

- `PutItem`
- `UpdateItem`
- `DeleteItem`

Per ognuna di queste operazioni, devi specificare l'intera chiave primaria e non solo una parte di essa. Se, ad esempio, una tabella ha una chiave primaria composita (chiave di partizione e chiave di ordinamento), devi fornire un valore per la chiave di partizione e uno per la chiave di ordinamento.

Per restituire il numero di unità di capacità di scrittura utilizzate da queste operazioni, imposta il parametro `ReturnConsumedCapacity` su uno dei valori seguenti:

- `TOTAL`: restituisce il numero totale di unità di capacità di scrittura totali consumate.
- `INDEXES`: restituisce il numero totale di unità di capacità di scrittura consumate, con i subtotali per la tabella e gli eventuali indici secondari interessati dall'operazione.
- `NONE`: non vengono restituiti dettagli sulla capacità di scrittura. Questa è l'impostazione predefinita.

PutItem

`PutItem` crea un nuovo elemento. Se nella tabella esiste già un elemento con la stessa chiave, il nuovo elemento sostituisce quello esistente.

Example

Scrivi un nuovo elemento nella tabella `Thread`. La chiave primaria per `Thread` è costituita da `ForumName` (chiave di partizione) e `Subject` (chiave di ordinamento).

```
aws dynamodb put-item \
    --table-name Thread \
    --item file://item.json
```

Gli argomenti per `--item` sono memorizzati nel file `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

Se non esiste un elemento con la chiave specificata, `UpdateItem` crea un nuovo elemento. In caso contrario, modifica gli attributi di un elemento esistente.

Puoi usare un'espressione di aggiornamento per specificare gli attributi che desideri modificare e i loro nuovi valori. Per ulteriori informazioni, consulta Utilizzo di espressioni di aggiornamento in DynamoDB.

Nell'espressione di aggiornamento usi i valori degli attributi di espressione come segnaposto per i valori effettivi. Per ulteriori informazioni, consulta Utilizzo dei valori degli attributi di espressione in DynamoDB.

Example

Modifica i diversi attributi nell'elemento `Thread`. Il parametro `ReturnValues` facoltativo mostra l'elemento dopo l'aggiornamento. Per ulteriori informazioni, consulta Valori restituiti.

```
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
```

Gli argomenti per `--key` sono memorizzati nel file `key.json`:

```
{
    "ForumName": {"S": "Amazon DynamoDB"},
    "Subject": {"S": "New discussion thread"}
}
```

Gli argomenti per `--expression-attribute-values` sono memorizzati nel file `expression-attribute-values.json`:

```
{
    ":zero": {"N":"0"},
    ":lastpostedby": {"S":"barney@example.com"}
}
```

DeleteItem

`DeleteItem` elimina l'elemento con la chiave specificata.

Example

L' AWS CLI esempio seguente mostra come eliminare l'`Thread`elemento.

```
aws dynamodb delete-item \
    --table-name Thread \
    --key file://key.json
```

Valori restituiti

In alcuni casi potrebbe essere necessario che DynamoDB restituisca determinati valori degli attributi nello stato in cui si trovano prima o dopo la modifica. Le operazioni `PutItem`, `UpdateItem` e `DeleteItem` hanno un parametro `ReturnValues` che è possibile usare per restituire i valori degli attributi prima o dopo la modifica.

Il valore predefinito per `ReturnValues` è `NONE`, il che significa che DynamoDB non restituisce informazioni sugli attributi che sono stati modificati.

Di seguito sono illustrate le altre impostazioni valide per `ReturnValues`, organizzate in base all'operazione API di DynamoDB.

PutItem

- `ReturnValues`: `ALL_OLD`
  - Se sovrascrivi un elemento esistente, `ALL_OLD` restituisce l'intero elemento precedente alla sovrascrittura.
  - Se scrivi un elemento che non esiste, `ALL_OLD` non ha alcun effetto.

UpdateItem

L'uso più comune di `UpdateItem` è quello di aggiornare un elemento esistente. `UpdateItem` esegue tuttavia un'operazione upsert, ovvero crea automaticamente l'elemento, se non esiste già.

- `ReturnValues`: `ALL_OLD`
  - Se aggiorni un elemento esistente, `ALL_OLD` restituisce l'intero elemento precedente all'aggiornamento.
  - Se aggiorni un item che non esiste (upsert), `ALL_OLD` non ha alcun effetto.
- `ReturnValues`: `ALL_NEW`

- Se aggiorni un elemento esistente, `ALL_NEW` restituisce l'intero elemento successivo all'aggiornamento.
- Se aggiorni un elemento che non esiste (upsert), `ALL_NEW` restituisce l'intero elemento.

- `ReturnValues`: `UPDATED_OLD`
  - Se aggiorni un elemento esistente, `UPDATED_OLD` restituisce solo gli attributi aggiornati, nello stato precedente all'aggiornamento.
  - Se aggiorni un item che non esiste (upsert), `UPDATED_OLD` non ha alcun effetto.
- `ReturnValues`: `UPDATED_NEW`
  - Se aggiorni un elemento esistente, `UPDATED_NEW` restituisce solo gli attributi interessati, nello stato successivo all'aggiornamento.
  - Se aggiorni un elemento che non esiste (upsert), `UPDATED_NEW` restituisce solo gli attributi aggiornati, nello stato successivo all'aggiornamento.

DeleteItem

- `ReturnValues`: `ALL_OLD`
  - Se elimini un elemento esistente, `ALL_OLD` restituisce l'intero elemento precedente all'eliminazione.
  - Se elimini un elemento che non esiste, `ALL_OLD` non restituisce alcun dato.

Operazioni batch

Per le applicazioni che devono leggere o scrivere più elementi, DynamoDB fornisce le operazioni `BatchGetItem` e `BatchWriteItem`. L'uso di queste operazioni permette di ridurre il numero di viaggi di andata e ritorno di rete dall'applicazione a DynamoDB. Inoltre, DynamoDB esegue le singole operazioni di lettura o scrittura in parallelo. Le applicazioni possono ottenere un vantaggio da questo parallelismo, senza dover gestire la concorrenza o il threading.

Le operazioni batch sono essenzialmente una combinazione di più richieste di lettura o di scrittura. Se, ad esempio, una richiesta `BatchGetItem` contiene cinque elementi, DynamoDB esegue cinque operazioni `GetItem` per conto tuo. Analogamente, se una richiesta `BatchWriteItem` contiene due richieste put e quattro richieste delete, DynamoDB esegue due richieste `PutItem` e quattro richieste `DeleteItem`.

In generale, un'operazione batch non ha esito negativo a meno che tutte le richieste nel batch non abbiano esito negativo. Supponi, ad esempio, di eseguire un'operazione `BatchGetItem`, ma una delle singole richieste `GetItem` nel batch ha esito negativo. In questo caso, `BatchGetItem` restituisce le chiavi e i dati della richiesta `GetItem` che ha avuto esito negativo. Le altre richieste `GetItem` nel batch non sono interessate.

BatchGetItem

Una singola operazione `BatchGetItem` può contenere fino a 100 singole richieste `GetItem` e può recuperare fino a 16 MB di dati. Un'operazione `BatchGetItem` può inoltre recuperare elementi da più tabelle.

Example

Recupera due elementi dalla tabella `Thread` usando un'espressione di proiezione per restituire solo alcuni degli attributi.

```
aws dynamodb batch-get-item \
    --request-items file://request-items.json
```

Gli argomenti per `--request-items` sono memorizzati nel file `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

L'operazione `BatchWriteItem` può contenere fino a 25 singole richieste `PutItem` e `DeleteItem` e può scrivere fino a 16 MB di dati. La dimensione massima di un singolo elemento è 400 KB. Un'operazione `BatchWriteItem` può inoltre inserire o eliminare elementi in più tabelle.

> **Note**
>
> `BatchWriteItem` non supporta le richieste `UpdateItem`.

Example

Scrivi due elementi nella tabella `ProductCatalog`.

```
aws dynamodb batch-write-item \
    --request-items file://request-items.json
```

Gli argomenti per `--request-items` sono memorizzati nel file `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" }
                }
            }
        }
    ]
}
```

Contatori atomici

È possibile utilizzare l'operazione `UpdateItem` per implementare un contatore atomico: un attributo numerico che viene incrementato, incondizionatamente, senza interferire con altre richieste di scrittura. Tutte le richieste di scrittura vengono applicate in base all'ordine di ricezione. Con un contatore atomico, gli aggiornamenti non sono idempotenti. In altre parole, il valore numerico aumenta o diminuisce a ogni chiamata di `UpdateItem`. Se il valore di incremento utilizzato per aggiornare il contatore atomico è positivo, può causare un conteggio in eccesso. Se il valore dell'incremento è negativo, può causare un conteggio in difetto.

Puoi usare un contatore atomico per tenere traccia del numero di visitatori di un sito Web. In questo caso, l'applicazione incrementa un valore numerico, indipendentemente dal valore corrente. Se un'operazione `UpdateItem` non va a buon fine, l'applicazione può semplicemente provare a ripetere l'operazione. Ciò rischierebbe di aggiornare il contatore due volte, anche se probabilmente un lieve discostamento in eccesso o in difetto nel numero di visitatori del sito Web può essere tollerato.

Un contatore atomico non è appropriato quando un conteggio maggiore o minore di quello effettivo non può essere tollerato (ad esempio, in un'applicazione bancaria). In questo caso, è preferibile usare un aggiornamento condizionale al posto di un contatore atomico.

Per ulteriori informazioni, consulta Incremento e decremento di attributi numerici.

Example

L' AWS CLI esempio seguente incrementa il valore `Price` di un prodotto di 5. In questo esempio, l'elemento esisteva già prima dell'aggiornamento del contatore. Poiché `UpdateItem` non è idempotente, il valore di `Price` aumenta ogni volta che si esegue il codice.

```
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
```

Scritture condizionali

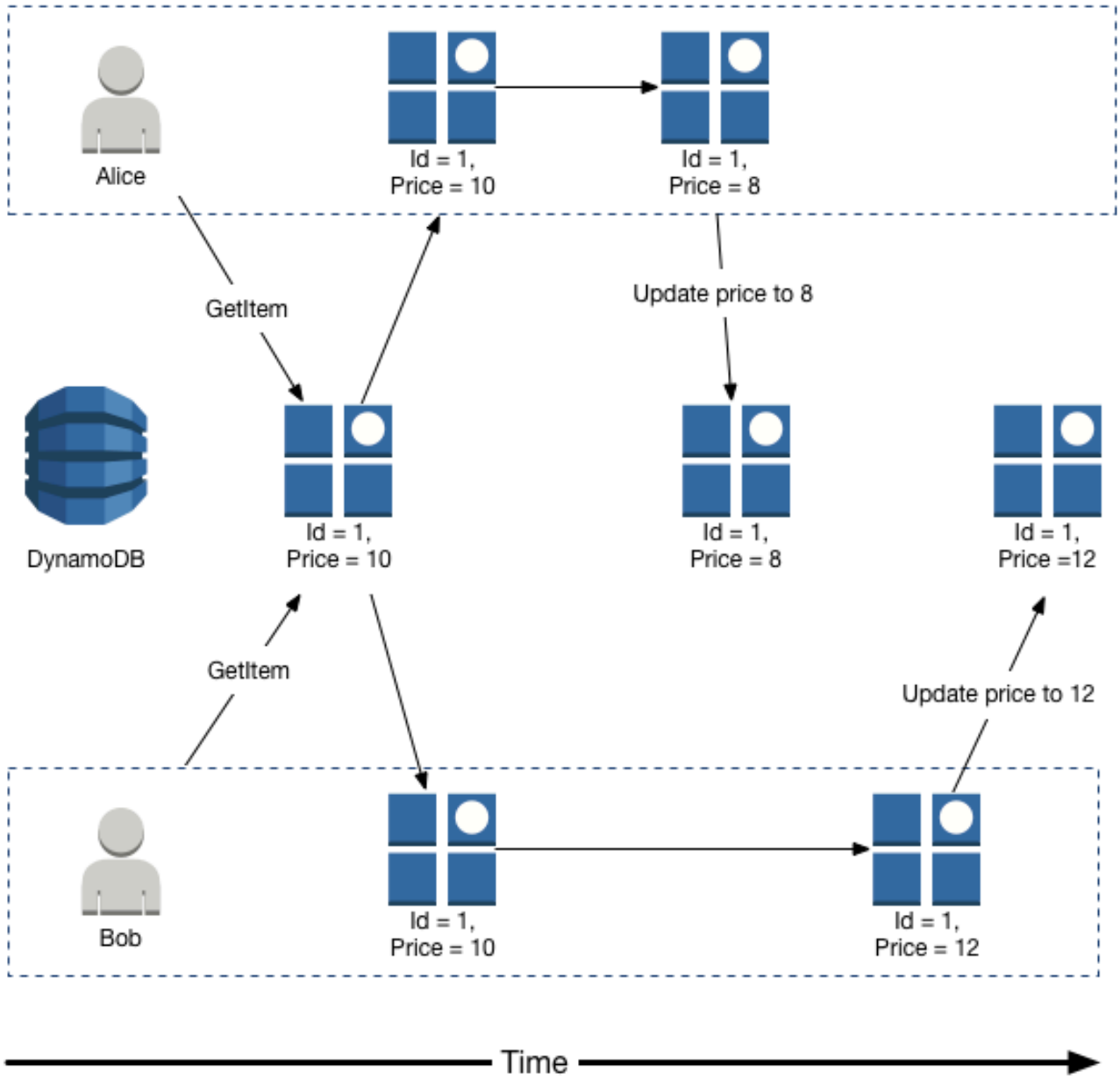
Per impostazione predefinita, le operazioni di scrittura di DynamoDB (`PutItem`, `UpdateItem`, `DeleteItem`) sono non condizionali: ogni operazione sovrascrive un elemento esistente con la chiave primaria specificata.

Facoltativamente, per queste operazioni DynamoDB supporta le scritture condizionali. Una scrittura condizionale ha esito positivo solo se gli attributi dell'elemento soddisfano una o più condizioni. In caso contrario, restituisce un errore.

Le scritture condizionali controllano le condizioni rispetto alla versione aggiornata più recente dell'elemento. Se l'elemento non esisteva in precedenza o se l'ultima operazione completata su quell'elemento è stata un'eliminazione, la scrittura condizionale non troverà alcun elemento precedente.

Le operazioni di scrittura condizionali sono utili in molte situazioni. Potrebbe ad esempio essere necessario fare in modo che un'operazione `PutItem` abbia esito positivo solo se non è già presente un elemento con la stessa chiave primaria. In alternativa, si potrebbe impedire a un'operazione `UpdateItem` di modificare un elemento se uno o più dei relativi attributi ha un determinato valore.

Le scritture condizionali sono utili nei casi in cui più utenti tentano di modificare lo stesso elemento. Si consideri il diagramma seguente, in cui due utenti (Alice e Bob) stanno lavorando con lo stesso elemento in una tabella DynamoDB.



Supponiamo che Alice utilizzi l'attributo to AWS CLI per aggiornare l'`Price`attributo 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
```

Gli argomenti per `--expression-attribute-values` sono memorizzati nel file `expression-attribute-values.json`:

```
{
    ":newval":{"N":"8"}
}
```

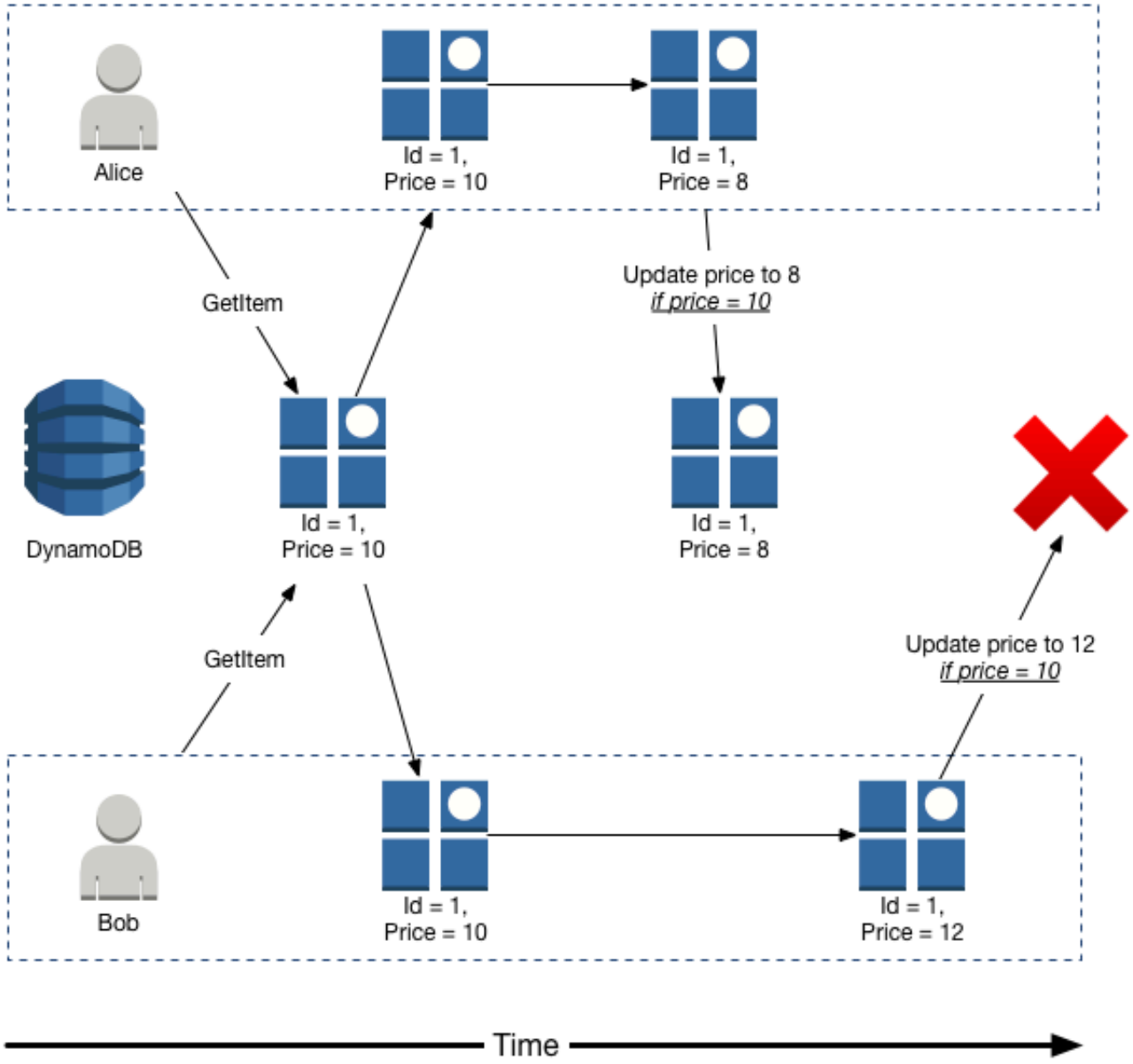
Supponi ora che Bob esegua una richiesta `UpdateItem` simile in un secondo momento, modificando però il valore di `Price` in 12. Per Bob, l'aspetto del parametro `--expression-attribute-values` è simile al seguente.

```
{
    ":newval":{"N":"12"}
}
```

La richiesta di Bob ha esito positivo, ma l'aggiornamento precedente di Alice viene perso.

Per richiedere un'operazione `PutItem`, `DeleteItem` o `UpdateItem` condizionale, specifichi un'espressione di condizione. Un'espressione condizionale è una stringa contenente nomi di attributi, operatori condizionali e funzioni predefinite. L'intera espressione deve essere vera. In caso contrario, l'operazione non va a buon fine.

Considera ora il diagramma seguente, che mostra in che modo le scritture condizionali impedirebbero la sovrascrittura dell'aggiornamento di Alice.



Alice prova innanzitutto ad aggiornare il valore di `Price` a 8, ma solo se il valore di `Price` corrente è 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
```

Gli argomenti per `--expression-attribute-values` sono memorizzati nel file `expression-attribute-values.json`:

```
{
    ":newval":{"N":"8"},
    ":currval":{"N":"10"}
}
```

L'aggiornamento di Alice riesce perché la condizione è vera.

Bob cerca quindi di aggiornare il valore di `Price` a 12, ma solo se il valore di `Price` corrente è 10. Per Bob, l'aspetto del parametro `--expression-attribute-values` è simile al seguente.

```
{
    ":newval":{"N":"12"},
    ":currval":{"N":"10"}
}
```

Poiché in precedenza Alice ha modificato il valore di `Price` in 8, l'espressione condizionale è falsa e l'aggiornamento di Bob non riesce.

Per ulteriori informazioni, consulta Esempio di CLI di espressione condizionale in DynamoDB.

Idempotenza delle scritture condizionali

Le scritture condizionali possono essere idempotenti se la verifica condizionale è sullo stesso attributo in fase di aggiornamento. Ciò significa che DynamoDB esegue una certa richiesta di scrittura solo se determinati valori degli attributi nell'elemento corrispondono a quanto previsto al momento della richiesta.

Supponi, ad esempio, di eseguire una richiesta `UpdateItem` per aumentare il valore di `Price` per un elemento di 3, ma solo se il valore di `Price` corrente è 20. Dopo che hai inviato la richiesta, ma prima di ricevere i risultati, si verifica un errore di rete e non sai se la richiesta ha avuto esito positivo. Poiché la scrittura condizionale è idempotente, è possibile eseguire di nuovo la stessa richiesta `UpdateItem` e DynamoDB aggiornerà l'elemento solo se il valore di `Price` corrente è 20.

Unità di capacità utilizzate dalle scritture condizionali

Se `ConditionExpression` restituisce false durante una scrittura condizionale, DynamoDB utilizza comunque la capacità di scrittura della tabella. La quantità consumata dipende dalla dimensione dell'elemento esistente (o almeno 1). Ad esempio, se un elemento esistente è di 300 KB e il nuovo elemento che si sta cercando di creare o aggiornare è 310 KB, le unità di capacità di scrittura consumate saranno 300 se la condizione restituisce un errore o 310 se la condizione riesce. Se si tratta di un elemento nuovo (non un elemento esistente), le unità di capacità di scrittura consumate saranno 1 se la condizione restituisce un errore e 310 se la condizione riesce.

> **Note**
>
> Le operazioni di scrittura utilizzano solo unità di capacità in scrittura. Non utilizzano mai unità di capacità in lettura.

Una scrittura condizionale non riuscita restituisce `ConditionalCheckFailedException`. In questo caso, non vengono inviate informazioni nella risposta sulla capacità di scrittura utilizzata.

Per restituire il numero di unità di capacità di scrittura utilizzate durante una scrittura condizionale, usa il parametro `ReturnConsumedCapacity`:

- `TOTAL`: restituisce il numero totale di unità di capacità di scrittura totali consumate.
- `INDEXES`: restituisce il numero totale di unità di capacità di scrittura consumate, con i subtotali per la tabella e gli eventuali indici secondari interessati dall'operazione.
- `NONE`: non vengono restituiti dettagli sulla capacità di scrittura. Questa è l'impostazione predefinita.

> **Note**
>
> A differenza di un indice secondario globale, un indice secondario locale condivide la capacità di throughput assegnata con la relativa tabella. L'attività di lettura e scrittura su un indice secondario locale utilizza la capacità di throughput assegnata della tabella.

Utilizzo di espressioni in DynamoDB

In Amazon DynamoDB è possibile utilizzare le espressioni per specificare quali attributi leggere da un elemento, per scrivere dati quando viene soddisfatta una condizione, per specificare come aggiornare un elemento, per definire query e filtrare i risultati di una query.

Questa tabella descrive la grammatica delle espressioni di base e i tipi di espressioni disponibili.

Tipo di espressione	Description
Espressioni di proiezione	Un'espressione di proiezione identifica gli attributi che si desidera recuperare da un elemento quando si utilizzano operazioni come Query o GetItem Scan.
Espressione condizionale	Un'espressione di condizione determina quali elementi devono essere modificati quando si utilizzano le operazioni PutItem UpdateItem, e. DeleteItem
Espressione di aggiornamento	Un'espressione di aggiornamento specifica come UpdateItem verranno modificati gli attributi di un elemento, ad esempio l'imposta zione di un valore scalare o la rimozione di elementi da un elenco o da una mappa.
Espressione di condizione della chiave	Un’espressione di condizione della chiave determina gli elementi che una query leggerà da una tabella o da un indice.
Espressione filtro	Un’espressione di filtro determina quali elementi all’interno dei risultati di Query devono essere restituiti. Tutti gli altri risultati vengono scartati.

Per ulteriori informazioni sulla sintassi delle espressioni e informazioni più dettagliate su ciascun tipo di espressione, consulta le sezioni riportate di seguito.

Argomenti

- Riferimento agli attributi degli elementi quando si utilizzano le espressioni in DynamoDB
- Nomi di attributi di espressione (alias) in DynamoDB
- Utilizzo dei valori degli attributi di espressione in DynamoDB
- Utilizzo di espressioni di proiezione in DynamoDB
- Utilizzo di espressioni di aggiornamento in DynamoDB
- Espressioni, operatori e funzioni di condizione e di filtro in DynamoDB
- Esempio di CLI di espressione condizionale in DynamoDB

> **Note**
>
> Per la compatibilità con le versioni precedenti, DynamoDB supporta anche parametri condizionali che non usano espressioni. Per ulteriori informazioni, consulta Parametri condizionali legacy di DynamoDB.
> Le nuove applicazioni dovrebbero utilizzare le espressioni al posto dei parametri legacy.

Riferimento agli attributi degli elementi quando si utilizzano le espressioni in DynamoDB

In questa sezione viene descritto come fare riferimento agli attributi degli elementi in una espressione in Amazon DynamoDB. Puoi utilizzare qualsiasi attributo, anche se è nidificato all'interno di più elenchi e mappe.

Argomenti

- Attributi di primo livello
- Attributi nidificati
- Percorsi dei documenti

Un articolo di esempio: ProductCatalog

Gli esempi presenti in questa pagina utilizzano il seguente elemento di esempio presente nella tabella `ProductCatalog`. (Questa tabella è descritta in Tabelle e dati di esempio da utilizzare in 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"
 }
```

Tenere presente quanto segue:

- il valore della chiave di partizione (`Id`) è `123`. Non vi è chiave di ordinamento;
- la maggior parte degli attributi dispone di tipi di dati scalari, come `String`, `Number`, `Boolean` e `Null`;
- un attributo (`Color`) è di tipo `String Set`;
- i seguenti attributi sono tipi di dati dei documenti:

- elenco di `RelatedItems`. Ogni elemento è un `Id` per un prodotto correlato;
- una mappa di `Pictures`. Ogni elemento rappresenta una breve descrizione di un'immagine insieme all'URL del file dell'immagine corrispondente;
- una mappa di `ProductReviews`. Ogni elemento rappresenta una classificazione e un elenco delle valutazioni corrispondenti a quella classificazione. Inizialmente, questa mappa viene popolata con valutazioni di cinque stelle e di una stella.

Attributi di primo livello

Un attributo si definisce di primo livello se non è incorporato in un altro attributo. Per gli elementi `ProductCatalog`, gli attributi di primo livello sono:

- `Id`
- `Title`
- `Description`
- `BicycleType`
- `Brand`
- `Price`
- `Color`
- `ProductCategory`
- `InStock`
- `QuantityOnHand`
- `RelatedItems`
- `Pictures`
- `ProductReviews`
- `Comment`
- `Safety.Warning`

Tutti questi attributi di primo livello sono scalari, eccetto `Color` (elenco), `RelatedItems` (elenco), `Pictures` (mappa) e `ProductReviews` (mappa).

Attributi nidificati

Un attributo si definisce nidificato se è incorporato in un altro attributo. Per ottenere l'accesso a un attributo nidificato, usa gli operatori di deferenziazione:

- `[n]`: per gli elementi list
- `.` (punto): per gli elementi map

Accesso agli elementi dell'elenco

L'operatore di deferenziazione per un elemento dell'elenco è [N], dove n è il numero dell'elemento. Gli elementi dell'elenco sono a base zero, ovvero [0] rappresenta il primo elemento nell'elenco, [1] il secondo e così via. Ecco alcuni esempi:

- `MyList[0]`
- `AnotherList[12]`
- `ThisList[5][11]`

L'elemento `ThisList[5]` è esso stesso un elenco nidificato. Pertanto, `ThisList[5][11]` si riferisce al dodicesimo elemento in quell'elenco.

Il numero all'interno delle parentesi quadre deve essere un numero intero non negativo. Pertanto, le espressioni seguenti non sono valide:

- `MyList[-1]`
- `MyList[0.4]`

Accesso agli elementi della mappa

L'operatore di deferenziazione per una mappa è . (un punto). Usa un punto come separatore tra gli elementi in una mappa:

- `MyMap.nestedField`
- `MyMap.nestedField.deeplyNestedField`

Percorsi dei documenti

In un espressione si utilizza un percorso del documento per comunicare a DynamoDB dove trovare un attributo. Per gli attributi di primo livello, il percorso del documento è semplicemente il nome dell'attributo. Per un attributo nidificato, costruisci il percorso del documento usando gli operatori di deferenziazione.

Di seguito sono riportati alcuni esempi di percorsi di documenti. (Fai riferimento all'elemento mostrato in Riferimento agli attributi degli elementi quando si utilizzano le espressioni in DynamoDB).

- Un attributo scalare di primo livello:

  ```
  Description
  ```

- Un attributo dell'elenco di primo livello. (Questo restituisce l'intero elenco, non solo alcuni elementi).

  ```
  RelatedItems
  ```

- Il terzo elemento dall'elenco `RelatedItems`. (Ricorda che gli elementi dell'elenco sono a base zero).

  ```
  RelatedItems[2]
  ```

- L'immagine di vista frontale del prodotto.

  ```
  Pictures.FrontView
  ```

- Tutte le valutazioni da cinque stelle.

  ```
  ProductReviews.FiveStar
  ```

- La prima delle valutazioni da cinque stelle.

  ```
  ProductReviews.FiveStar[0]
  ```

> **Note**
>
> La profondità massima per un percorso del documento è 32. Pertanto, il numero di operatori di deferenziazione in un percorso non può eccedere questo limite.

È possibile utilizzare qualsiasi nome di attributo nel percorso di un documento purché soddisfi i seguenti requisiti:

- Il primo carattere è una lettera (`a-z` o `A-Z`) o un numero (`0-9`)
- Il secondo carattere (se presente) è una lettera (`a-z` o `A-Z`)

> **Note**
>
> Se un nome di attributo non soddisfa questo requisito, devi definire un nome di attributo dell'espressione come un segnaposto.

Per ulteriori informazioni, consulta Nomi di attributi di espressione (alias) in DynamoDB.

Nomi di attributi di espressione (alias) in DynamoDB

Un nome di attributo di espressione è un alias (o segnaposto) utilizzato in un'espressione Amazon DynamoDB in alternativa a un nome di attributo effettivo. Un nome di attributo di espressione deve iniziare con un simbolo di cancelletto (#) ed essere seguito da uno o più caratteri alfanumerici. È consentito anche il carattere trattino basso (_).

In questa sezione vengono descritte diverse situazioni in cui è necessario utilizzare i nomi di attributi di espressione.

> **Note**
>
> Gli esempi in questa sezione utilizzano il AWS Command Line Interface (AWS CLI).

Argomenti

- Parole riservate
- Nomi di attributi contenenti caratteri speciali
- Attributi nidificati
- Riferimento ripetuto ai nomi di attributo

Parole riservate

A volte potrebbe essere necessario scrivere un'espressione contenente un nome di attributo in conflitto con una parola riservata di DynamoDB. Per un elenco completo delle parole riservate, consulta Parole riservate in DynamoDB.)

Ad esempio, l' AWS CLI esempio seguente non riuscirà perché `COMMENT` è una parola riservata.

```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"123"}}' \
    --projection-expression "Comment"
```

Come soluzione alternativa, puoi sostituire `Comment` con un nome di attributo di espressione come `#c`. Il simbolo # (cancelletto) è obbligatorio e indica che si tratta di un segnaposto per un nome di attributo. L' AWS CLI esempio sarebbe ora simile al seguente.

```
aws dynamodb get-item \
     --table-name ProductCatalog \
     --key '{"Id":{"N":"123"}}' \
     --projection-expression "#c" \
     --expression-attribute-names '{"#c":"Comment"}'
```

> **Note**
>
> Se un nome di attributo inizia con un numero, contiene uno spazio o una parola riservata, è necessario utilizzare un nome di attributo di espressione per sostituire il nome di tale attributo nell'espressione.

Nomi di attributi contenenti caratteri speciali

Un punto (".") in un'espressione viene interpretato come carattere separatore in un percorso di un documento. Tuttavia, DynamoDB consente anche di utilizzare un punto e altri caratteri speciali, ad esempio, un trattino ("-") come parte di un nome di attributo. In alcuni casi questo può portare ad ambiguità. A titolo illustrativo, supponi di voler recuperare l'attributo `Safety.Warning` da un elemento `ProductCatalog` (consulta Riferimento agli attributi degli elementi quando si utilizzano le espressioni in DynamoDB).

Supponi di voler accedere a `Safety.Warning` utilizzando un'espressione di proiezione.

```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"123"}}' \
    --projection-expression "Safety.Warning"
```

DynamoDB restituisce un risultato vuoto, anziché la stringa prevista ("`Always wear a helmet`"). Ciò accade poiché DynamoDB interpreta un punto in un'espressione come separatore del percorso di un documento. In questo caso, occorre definire un nome di attributo di espressione (ad esempio `#sw`) come sostituto di `Safety.Warning`. Puoi quindi utilizzare l'espressione di proiezione seguente.

```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"123"}}' \
    --projection-expression "#sw" \
    --expression-attribute-names '{"#sw":"Safety.Warning"}'
```

DynamoDB restituirebbe quindi il risultato corretto.

> **Note**
>
> Se il nome di un attributo contiene un punto (".") o un trattino ("-"), è necessario utilizzare un nome di attributo di espressione per sostituire il nome di tale attributo nell'espressione.

Attributi nidificati

Si supponga di voler accedere all'attributo annidato `ProductReviews.OneStar`. In un nome di attributo di espressione, DynamoDB considera il punto (".") come carattere all'interno di un nome di attributo. Per fare riferimento all'attributo annidato, definisci un nome di attributo di espressione per ogni elemento nel percorso del documento:

- `#pr – ProductReviews`
- `#1star – OneStar`

Puoi quindi utilizzare `#pr.#1star` per l'espressione di proiezione.

```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"123"}}' \
    --projection-expression "#pr.#1star"  \
    --expression-attribute-names '{"#pr":"ProductReviews", "#1star":"OneStar"}'
```

DynamoDB restituirebbe quindi il risultato corretto.

Riferimento ripetuto ai nomi di attributo

I nomi di attributo di espressione sono utili quando si vuole fare riferimento ripetutamente allo stesso nome di attributo. Ad esempio, considera la seguente espressione per recuperare alcune valutazioni da un elemento di `ProductCatalog`.

```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"123"}}' \
    --projection-expression "ProductReviews.FiveStar, ProductReviews.ThreeStar,
 ProductReviews.OneStar"
```

Per renderlo più conciso, puoi sostituire `ProductReviews` con un nome di attributo di espressione come `#pr`. L'aspetto dell'espressione rivista è ora simile al seguente.

- `#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"}'
```

Se definisci un nome di attributo di espressione, è necessario usarlo in maniera uniforme nell'intera espressione. Inoltre, non puoi omettere il simbolo #.

Utilizzo dei valori degli attributi di espressione in DynamoDB

I valori degli attributi di espressione in Amazon DynamoDB agiscono come variabili. Sostituiscono i valori effettivi che si desidera confrontare, valori che potrebbero non essere noti fino al runtime. Un valore dell'attributo dell'espressione deve iniziare con un carattere di due punti (`:`) ed essere seguito da uno o più caratteri alfanumerici.

Per esempio, supponi di voler restituire tutti gli elementi `ProductCatalog` che sono disponibili in `Black` e che costano `500` o meno. Puoi utilizzare un'operazione `Scan` con un'espressione di filtro, come in questo esempio di 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
```

Gli argomenti per `--expression-attribute-values` sono memorizzati nel file `values.json`:

```
{
    ":c": { "S": "Black" },
    ":p": { "N": "500" }
}
```

Se definisci un valore dell'attributo dell'espressione, è necessario usarlo in maniera uniforme nell'intera espressione. Inoltre, non puoi omettere il simbolo `:`.

I valori degli attributi dell'espressione sono usati con espressioni di condizione della chiave, espressioni di condizioni, espressioni di aggiornamento ed espressioni di filtro.

Utilizzo di espressioni di proiezione in DynamoDB

Per leggere i dati da una tabella, vengono utilizzate operazioni quali `GetItem`, `Query` oppure `Scan`. Amazon DynamoDB restituisce tutti gli attributi dell'elemento per impostazione predefinita. Per ottenerne solo alcuni degli attributi e non tutti, usa un'espressione di proiezione.

Un'espressione di proiezione è una stringa che identifica gli attributi che desideri. Per recuperare un singolo attributo, specificane il nome. Per più attributi, i nomi devono essere separati da una virgola.

Di seguito sono riportati alcuni esempi di espressioni di proiezione basate sull'elemento `ProductCatalog` da Riferimento agli attributi degli elementi quando si utilizzano le espressioni in DynamoDB:

- Un singolo attributo di primo livello.

  ```
  Title
  ```

- Tre attributi di livello superiore DynamoDB recupera l'intero set `Color`.

  ```
  Title, Price, Color
  ```

- Quattro attributi di livello superiore. DynamoDB restituisce l'intero contenuto di `RelatedItems` e `ProductReviews`.

  ```
  Title, Description, RelatedItems, ProductReviews
  ```

> **Note**
>
> L'espressione di proiezione non ha alcun effetto sul consumo di throughput allocato. DynamoDB determina le unità di capacità utilizzate in base alla dimensione dell'elemento anziché alla quantità di dati restituiti a un'applicazione.

Parole riservate e caratteri speciali

DynamoDB dispone di parole riservate e caratteri speciali. DynamoDB consente di utilizzare le parole riservate e i caratteri speciali per i nomi, ma si consiglia di evitare questa opzione perché comporta l'utilizzo di alias per i nomi ogni volta che questi vengono utilizzati in un'espressione. Per un elenco completo, consulta Parole riservate in DynamoDB.

Sarà necessario utilizzare i nomi degli attributi di espressione al posto del nome effettivo se:

- Il nome di attributo è nell'elenco di parole riservate in DynamoDB.
- Il nome dell'attributo non soddisfa il requisito che il primo carattere sia `a-z` o `A-Z` e che il secondo carattere (se presente) sia `a-Z`, `A-Z` o`0-9`.
- Il nome dell'attributo contiene un # (cancelletto) o : (due punti).

L' AWS CLI esempio seguente mostra come utilizzare un'espressione di proiezione con un'`GetItem`operazione. Questa espressione di proiezione recupera un attributo scalare di primo livello (`Description`), il primo elemento in un elenco (`RelatedItems[0]`) e un elenco nidificato in una mappa (`ProductReviews.FiveStar`).

```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key '"Id": { "N": "123" } \
    --projection-expression "Description, RelatedItems[0], ProductReviews.FiveStar"
```

Per questo esempio viene restituito il seguente 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"
                }
            ]
        }
    }
}
```

Utilizzo di espressioni di aggiornamento in DynamoDB

L'operazione `UpdateItem` aggiorna una voce esistente o aggiunge una nuova voce alla tabella, se non è già presente. È necessario fornire la chiave dell'elemento che intendi aggiornare. È necessario fornire inoltre un'espressione di aggiornamento che indica gli attributi che intendi modificare e i valori che intendi assegnargli.

Una espressione di aggiornamento specifica come `UpdateItem` modificherà gli attributi di un elemento, ad esempio impostando un valore scalare o rimuovendo elementi da un elenco o da una mappa.

Di seguito è riportato un riepilogo della sintassi delle espressioni di aggiornamento.

```
update-expression ::=
    [ SET action [, action] ... ]
    [ REMOVE action [, action] ...]
    [ ADD action [, action] ... ]
    [ DELETE action [, action] ...]
```

Un'espressione di aggiornamento comprende una o più clausole. Ogni clausola inizia con una parola chiave `SET`, `REMOVE`, `ADD` o `DELETE`. Puoi includere qualsiasi di queste clausole in un'espressione di aggiornamento, in qualsiasi ordine. Tuttavia, ciascuna parola chiave dell'operazione può comparire una sola volta.

All'interno di ogni clausola esistono una o più operazioni separate da virgole. Ciascuna operazione rappresenta una modifica di dati.

Gli esempi in questa sezione si basano sull'elemento `ProductCatalog` illustrato in Utilizzo di espressioni di proiezione in DynamoDB.

Gli argomenti seguenti illustrano diversi casi d'uso dell'operazione `SET`.

Argomenti

- SET: modifica o aggiunta di attributi dell'elemento
- REMOVE: eliminazione degli attributi da un elemento
- ADD: aggiornamento di numeri e set
- DELETE: rimozione di elementi da un set
- Utilizzo di più espressioni di aggiornamento

SET: modifica o aggiunta di attributi dell'elemento

Utilizza l'operazione `SET` in un'espressione di aggiornamento per aggiungere uno o più attributi a un elemento. Se alcuni di questi attributi esistono già, vengono sovrascritti dai nuovi valori. Se intendi evitare la sovrascrittura di un attributo esistente, puoi utilizzare `SET` con la funzione `if_not_exists`. La funzione `if_not_exists` è specifica dell'operazione `SET` e può essere utilizzata solo in un'espressione di aggiornamento.

Quando utilizzi `SET` per aggiornare un elemento dell'elenco, i contenuti dell'elemento vengono sostituiti con i nuovi dati specificati. Se l'elemento non esiste ancora, `SET` aggiunge il nuovo elemento alla fine dell'elenco.

Se aggiungi più elementi in un'unica operazione `SET`, gli elementi verranno ordinati per numero di elemento.

Puoi inoltre utilizzare `SET` per aggiungere o sottrarre un attributo di tipo `Number`. Per eseguire più operazioni `SET`, separarle con virgole.

Nel riepilogo della sintassi seguente:

- L'*path*elemento è il percorso del documento verso l'elemento.
- Un *operand*elemento può essere il percorso del documento verso un elemento o una funzione.

```
set-action ::=
    path = value

value ::=
    operand
    | operand '+' operand
    | operand '-' operand

operand ::=
    path | function

function ::=
    if_not_exists (path, value)
```

Se l'elemento non contiene un attributo nel percorso specificato, `if_not_exists` restituisce `value`. Altrimenti, restituisce `path`.

L'operazione `PutItem` seguente crea un elemento di esempio a cui gli esempi fanno riferimento.

```
aws dynamodb put-item \
    --table-name ProductCatalog \
    --item file://item.json
```

Gli argomenti per `--item` sono memorizzati nel file `item.json`: (Per semplicità, vengono utilizzati solo pochi attributi dell'item).

```
{
    "Id": {"N": "789"},
    "ProductCategory": {"S": "Home Improvement"},
    "Price": {"N": "52"},
    "InStock": {"BOOL": true},
    "Brand": {"S": "Acme"}
}
```

Argomenti

- Modifica degli attributi

- Aggiunta di elenchi e mappe
- Aggiunta di elementi a un elenco
- Aggiunta di attributi di mappa nidificati
- Incremento e decremento di attributi numerici
- Aggiunta di elementi a un elenco
- Prevenzione delle sovrascritture di un attributo esistente

Modifica degli attributi

Example

Aggiorna gli attributi `ProductCategory` e `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
```

Gli argomenti per `--expression-attribute-values` sono memorizzati nel file `values.json`:

```
{
    ":c": { "S": "Hardware" },
    ":p": { "N": "60" }
}
```

> **Note**
>
> Nell'operazione `UpdateItem`, `--return-values ALL_NEW` fa in modo che DynamoDB restituisca l'elemento come compare dopo l'aggiornamento.

Aggiunta di elenchi e mappe

Example

Aggiungi un nuovo elenco e una nuova mappa.

```
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
```

Gli argomenti per `--expression-attribute-values` sono memorizzati nel file `values.json`:

```
{
    ":ri": {
        "L": [
            { "S": "Hammer" }
        ]
    },
    ":pr": {
        "M": {
            "FiveStar": {
                "L": [
                    { "S": "Best product ever!" }
                ]
            }
        }
    }
}
```

Aggiunta di elementi a un elenco

Example

Aggiungi un nuovo attributo all'elenco `RelatedItems`. Ricorda che gli elementi dell'elenco sono a base zero, ovvero [0] rappresenta il primo elemento nell'elenco, [1] il secondo e così via.

```
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
```

Gli argomenti per `--expression-attribute-values` sono memorizzati nel file `values.json`:

```
{
    ":ri": { "S": "Nails" }
}
```

> **Note**
>
> Quando utilizzi SET per aggiornare un elemento dell'elenco, i contenuti dell'elemento vengono sostituiti con i nuovi dati specificati. Se l'elemento non esiste ancora, SET aggiunge il nuovo elemento alla fine dell'elenco.
> Se aggiungi più elementi in un'unica operazione SET, gli elementi verranno ordinati per numero di elemento.

Aggiunta di attributi di mappa nidificati

Example

Aggiungi alcuni attributi di mappa nidificati.

```
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
```

Gli argomenti per `--expression-attribute-names` sono memorizzati nel file `names.json`:

```
{
    "#pr": "ProductReviews",
    "#5star": "FiveStar",
    "#3star": "ThreeStar"
}
```

Gli argomenti per `--expression-attribute-values` sono memorizzati nel file `values.json`:

```
{
    ":r5": { "S": "Very happy with my purchase" },
```

```
    ":r3": {
        "L": [
            { "S": "Just OK - not that great" }
        ]
    }
}
```

Incremento e decremento di attributi numerici

Puoi eseguire aggiunte o sottrazioni da un attributo numerico esistente. A questo scopo, utilizza gli operatori + (più) e - (meno).

Example

Riduci il valore `Price` di 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
```

Per aumentare il valore `Price`, puoi utilizzare l'operatore + nell'espressione di aggiornamento.

Aggiunta di elementi a un elenco

Puoi aggiungere elementi alla fine di un elenco. A tale scopo, utilizza `SET` con la funzione `list_append`. Il nome funzione rileva la distinzione tra maiuscole e minuscole. La funzione `list_append` è specifica dell'operazione `SET` e può essere utilizzata solo in un'espressione di aggiornamento. La sintassi è esposta di seguito.

- `list_append (`*`list1`*`, `*`list2`*`)`

La funzione prende due liste come input e aggiunge tutti gli elementi da *`list2`* a *`list1`*.

Example

In Aggiunta di elementi a un elenco, viene creato l'elenco `RelatedItems` e popolato con due elementi: `Hammer` e `Nails`. Vengono quindi aggiunti due altri elementi alla fine di `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
```

Gli argomenti per `--expression-attribute-values` sono memorizzati nel file `values.json`:

```
{
    ":vals": {
        "L": [
            { "S": "Screwdriver" },
            {"S": "Hacksaw" }
        ]
    }
}
```

Infine, viene aggiunto un ulteriore elemento all'inizio di `RelatedItems`. A questo scopo, scambia l'ordine degli elementi `list_append`. Ricorda che `list_append` accetta due elenchi come input e aggiunge il secondo elenco al primo.

```
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
```

L'attributo `RelatedItems` risultante contiene ora cinque elementi nell'ordine seguente: `Chisel`, `Hammer`, `Nails`, `Screwdriver`, `Hacksaw`.

Prevenzione delle sovrascritture di un attributo esistente

Example

Imposta l'attributo `Price` di un elemento, ma solo se l'elemento non dispone già di un attributo `Price`. Se l'attributo `Price` esiste già, non accade nulla.

```
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: eliminazione degli attributi da un elemento

Utilizza l'operazione `REMOVE` in una espressione di aggiornamento per rimuovere uno o più attributi di un elemento in Amazon DynamoDB. Per eseguire più operazioni `REMOVE`, separarle con virgole.

Di seguito è riportato un riepilogo della sintassi di `REMOVE` in un'espressione di aggiornamento. L'unico operando è il percorso di documento dell'attributo che intendi rimuovere.

```
remove-action ::=
    path
```

Example

Rimuovi alcuni attributi di un elemento. Se gli attributi non esistono, non accade nulla.

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "REMOVE Brand, InStock, QuantityOnHand" \
    --return-values ALL_NEW
```

Rimozione di elementi da un elenco

Puoi utilizzare `REMOVE` per eliminare singoli elementi da un elenco.

Example

In Aggiunta di elementi a un elenco, modifica un attributo di elenco (`RelatedItems`) in modo che contenga cinque elementi:

- `[0]`—`Chisel`
- `[1]`—`Hammer`

- `[2]—Nails`
- `[3]—Screwdriver`
- `[4]—Hacksaw`

L'esempio seguente AWS Command Line Interface (AWS CLI) elimina `Hammer` e `Nails` dall'elenco.

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "REMOVE RelatedItems[1], RelatedItems[2]" \
    --return-values ALL_NEW
```

Dopo che `Hammer` e `Nails` sono stati rimossi, gli elementi rimanenti vengono spostati. Ora l'elenco contiene quanto segue:

- `[0]—Chisel`
- `[1]—Screwdriver`
- `[2]—Hacksaw`

ADD: aggiornamento di numeri e set

> **Note**
>
> In generale, consigliamo di utilizzare `SET` anziché `ADD`.

Utilizza l'operazione `ADD` in un'espressione di aggiornamento per aggiungere un nuovo attributo e i suoi valori a un elemento.

Se l'attributo esiste già, il comportamento di `ADD` dipende dal tipo di dati dell'attributo:

- Se l'attributo è un numero e anche il valore da aggiungere è un numero, il valore viene aggiunto matematicamente all'attributo esistente. Se il valore è un numero negativo, viene sottratto dall'attributo esistente.
- Se l'attributo è un set e anche il valore da aggiungere è un set, il valore viene aggiunto matematicamente al set esistente.

> **Note**
>
> L'operazione ADD supporta solo i tipi di dati Number e Set.

Per eseguire più operazioni ADD, separarle con virgole.

Nel riepilogo della sintassi seguente:

- L'*path*elemento è il percorso del documento verso un attributo. L'attributo deve essere un tipo di dati `Number` o Set.
- L'*value*elemento è un numero che si desidera aggiungere all'attributo (per `Number` i tipi di dati) o un set da aggiungere all'attributo (per i tipi di set).

```
add-action ::=
    path value
```

Gli argomenti seguenti illustrano diversi casi d'uso dell'operazione ADD.

Argomenti

- Aggiunta di un numero
- Aggiunta di elementi a un set

Aggiunta di un numero

Supponiamo che l'attributo `QuantityOnHand` non esista. L' AWS CLI esempio seguente è impostato `QuantityOnHand` su 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
```

Ora che `QuantityOnHand` esiste, puoi eseguire nuovamente l'esempio per incrementare `QuantityOnHand` ogni volta di 5.

Aggiunta di elementi a un set

Supponiamo che l'attributo `Color` non esista. L'esempio AWS CLI seguente imposta `Color` su un set di stringhe con due elementi.

```
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
```

Ora che `Color` esiste, puoi aggiungergli più elementi:

```
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: rimozione di elementi da un set

> **Important**
>
> L'operazione `DELETE` supporta solo i tipi di dati `Set`.

Utilizza l'operazione `DELETE` in un'espressione di aggiornamento per rimuovere uno o più elementi da un set. Per eseguire più operazioni `DELETE`, separarle con virgole.

Nel riepilogo della sintassi seguente:

- L'*`path`*elemento è il percorso del documento verso un attributo. L'attributo deve essere un tipo di dati Set.
- *`subset`*È uno o più elementi da cui si desidera eliminare*`path`*. È necessario specificare *`subset`* come tipo di set.

```
delete-action ::=
```

```
    path subset
```

Example

In Aggiunta di elementi a un set, crea lo String Set `Color`. In questo esempio vengono rimossi alcuni elementi di tale set.

```
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
```

Utilizzo di più espressioni di aggiornamento

Puoi utilizzare più espressioni di aggiornamento in una singola istruzione.

Example

Se vuoi modificare il valore di un attributo e rimuovere completamente un altro attributo, puoi utilizzare le operazioni SET e REMOVE in una singola istruzione. Questa operazione imposta il valore di `Price` su 15 e rimuove l'attributo `InStock` dall'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

Se desideri aggiungere una voce a un elenco modificando anche il valore di un attributo, puoi utilizzare due operazioni SET in una singola istruzione. Questa operazione aggiunge "Nails" all'attributo di elenco `RelatedItems` e imposta il valore di `Price` su 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
```

Espressioni, operatori e funzioni di condizione e di filtro in DynamoDB

Per manipolare i dati in una tabella DynamoDB, è possibile utilizzare le operazioni `PutItem`, `UpdateItem` e `DeleteItem`. Per le operazioni di manipolazione dei dati, puoi specificare un'espressione di condizione per determinare quale elemento deve essere modificato. Se l'espressione condizionale restituisce true, l'operazione ha esito positivo. In caso contrario, l'operazione non va a buon fine.

In questa sezione sono descritte le funzioni e le parole chiave integrate per scrivere espressioni di condizione ed espressioni di filtro in Amazon DynamoDB. Per informazioni più dettagliate sulle funzioni e sulla programmazione con DynamoDB, consulta Programmazione con DynamoDB e SDK AWS e DynamoDB API Reference.

Argomenti

- Sintassi per le espressioni di filtro e condizioni
- Realizzazione di confronti
- Funzioni
- Valutazioni logiche
- Parentesi
- Priorità nelle condizioni

Sintassi per le espressioni di filtro e condizioni

Nel seguente riepilogo della sintassi, an *operand* può essere il seguente:

- Un nome di attributo di primo livello, ad esempio `Id`, `Title`, `Description` o `ProductCategory`
- Un percorso di documento che fa riferimento a un attributo nidificato

```
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)
```

Realizzazione di confronti

Utilizza questi comparatori per confrontare un operando con un singolo valore:

- *a* = *b*— Vero se *a* è uguale a. *b*
- *a* <> *b*— Vero se non *a* è uguale a*b*.
- *a* < *b*— Vero se *a* è minore di*b*.
- *a* <= *b*— Vero se *a* è minore o uguale a*b*.
- *a* > *b*— Vero se *a* è maggiore di*b*.
- *a* >= *b*— Vero se *a* è maggiore o uguale a*b*.

Utilizza le parole chiave `BETWEEN` e `IN` per confrontare un operando con un intervallo di valori o con un elenco enumerato di valori:

- *a* `BETWEEN` *b* `AND` *c*— Vero se *a* è maggiore o uguale a *b* e minore o uguale a*c*.
- *a* `IN (`*b*`,` *c*`,` *d*`)` — Vero se *a* è uguale a qualsiasi valore nell'elenco, ad esempio qualsiasi di *bc*, o. *d* L'elenco può contenere fino a 100 valori separati da virgole.

Funzioni

Utilizza le seguenti funzioni per determinare se un attributo è presente in un elemento o per valutare il valore di un attributo. I nomi di funzione rispettano la distinzione tra lettere maiuscole e minuscole. Per un attributo nidificato, è necessario fornire l'intero percorso del documento.

Funzione	Description
`attribute_exists ( path)`	True se l'elemento contiene l'attributo specificato da `path`. Esempio: Verifica se un elemento nella tabella `Product` ha un'immagine di vista laterale. • `attribute_exists (#Pictures.#SideView)`
`attribute_not_exists ( path)`	True se l'attributo specificato da `path` non è presente nell'elemento. Esempio: Verifica se un elemento ha un attributo `Manufacturer` . • `attribute_not_exists (Manufacturer)`
`attribute_type ( path, type)`	True se l'attributo del percorso specificato è di un particolare tipo di dato. Il parametro `type` deve essere uno dei seguenti: • `S`: String • `SS`: set di stringhe • `N`: Number • `NS`: set numerico

Funzione	Description
	• `B`: binario • `BS`: set binario • `BOOL`: Boolean • `NULL`: null • `L`: list • `M`: Map È necessario utilizzare un valore di attributo di espressione per il parametro `type`. Esempio: Verifica se l'attributo `Quantity0nHand` è del tipo List. In questo esempio, `:v_sub` è un segnaposto per la stringa `L`. • `attribute_type (ProductReviews.FiveStar, :v_sub)` È necessario utilizzare un valore di attributo di espressione per il parametro `type`.

Funzione	Description
`begins_with ( path, substr)`	True se l'attributo specificato da `path` inizia con una particolare sottostringa. Esempio: Verifica se i primi caratteri dell'URL dell'immagine di vista frontale sono `http://`. • `begins_with (Pictures.FrontView, :v_sub)` Il valore di attributo di espressione `:v_sub` è un segnaposto per `http://`.

Funzione	Description
`contains (path, operand)`	True se l'attributo specificato da `path` è uno dei seguenti: • un valore di tipo `String` che contiene una particolare sottostringa. • un valore di tipo `Set` che contiene un particolare elemento appartenente all'insieme. • Valore di tipo `List` che contiene un particolare elemento appartenente all'insieme. Se l'attributo specificato da `path` è `String`, `operand` deve essere `String`. Se l'attributo specificato da `path` è un `Set`, l'`operand` deve essere il tipo di elemento del set. Il percorso e l'operando devono essere distinti. Vale a dire, `contains (a, a)` restituisce un errore. Esempio: Verifica se l'attributo `Brand` contiene la sottostringa `Company`. • `contains (Brand, :v_sub)` Il valore di attributo di espressione `:v_sub` è un segnaposto per `Company`. Esempio: Verifica se il prodotto è disponibile in rosso. • `contains (Color, :v_sub)`

Funzione	Description
	Il valore di attributo di espressione `:v_sub` è un segnaposto per `Red`.

Funzione	Description
`size (path)`	Restituisce un numero che rappresenta le dimensioni di un attributo. I seguenti sono tipi di dati validi per l'utilizzo con `size`. Se l'attributo è di tipo `String`, `size`restituisce la lunghezza della stringa. Esempio: Verifica se la stringa `Brand` è inferiore o uguale a 20 caratteri. Il valore di attributo di espressione `:v_sub` è un segnaposto per `20`. • `size (Brand) <= :v_sub` Se l'attributo è di tipo `Binary`, `size` restituisce il numero di byte nel valore attributo. Esempio: supponi che l'elemento `ProductCatalog` abbia un attributo di tipo binario denominato `VideoClip` , che contiene un breve video del prodotto in uso. L'espressione seguente verifica se `VideoClip` supera i 64.000 byte. Il valore di attributo di espressione `:v_sub` è un segnaposto per `64000`. • `size(VideoClip) > :v_sub` Se il tipo di dati dell'attributo è `Set`, `size` restituisce il numero di elementi nell'insieme.

Funzione	Description
	Esempio: Verifica se il prodotto è disponibile in più di un colore. Il valore di attributo di espressione `:v_sub` è un segnaposto per `1`. • `size (Color) < :v_sub` Se il tipo dell'attributo è `List` o `Map`, `size` restituisce il numero di elementi figlio. Esempio: Verifica se il numero di revisioni `OneStar` ha superato una certa soglia. Il valore di attributo di espressione `:v_sub` è un segnaposto per `3`. • `size(ProductReviews.OneStar) > :v_sub`

Valutazioni logiche

Utilizza le parole chiave `AND`, `OR` e `NOT` per eseguire valutazioni logiche. Nell'elenco seguente, *a* e *b* rappresentano le condizioni da valutare.

- *a* `AND` *b*— Vero se *a* e *b* sono entrambi veri.
- *a* `OR` *b*— Vero se una delle due *a* o *b* (o entrambe) sono vere.
- `NOT` *a*— Vero se *a* è falso. Falso se *a* è vero.

Di seguito è riportato un esempio di codice di AND in un'operazione.

```
dynamodb-local (*)> select * from exprtest where a > 3 and a < 5;
```

Parentesi

Utilizza le parentesi per modificare la priorità di una valutazione logica. Ad esempio, supponiamo che le condizioni *a* e *b* siano vere e che tale condizione *c* sia falsa. La seguente espressione restituisce true:

- *a* OR *b* AND *c*

Tuttavia, se racchiudi una condizione tra parentesi, verrà valutata per prima. Ad esempio, la seguente espressione restituisce false:

- (*a* OR *b*) AND *c*

> **Note**
>
> Puoi annidare le parentesi in un'espressione: quelle più interne saranno valutate per prime.

Di seguito è riportato un esempio di codice con parentesi in una valutazione logica.

```
dynamodb-local (*)> select * from exprtest where attribute_type(b, string)
or ( a = 5 and c = “coffee”);
```

Priorità nelle condizioni

DynamoDB valuta le condizioni da sinistra a destra utilizzando le seguenti regole di precedenza:

- = <> < <= > >=
- IN
- BETWEEN
- attribute_exists attribute_not_exists begins_with contains
- Parentesi
- NOT
- AND
- OR

Esempio di CLI di espressione condizionale in DynamoDB

Di seguito sono riportati alcuni AWS Command Line Interface (AWS CLI) esempi di utilizzo delle espressioni condizionali. Questo esempi sono basati sulla tabella `ProductCatalog`, che è stata introdotta in Riferimento agli attributi degli elementi quando si utilizzano le espressioni in DynamoDB. La chiave di partizione di questa tabella è `Id`; non è presente una chiave di ordinamento. L'operazione `PutItem` seguente crea un elemento `ProductCatalog` di esempio a cui gli esempi fanno riferimento.

```
aws dynamodb put-item \
    --table-name ProductCatalog \
    --item file://item.json
```

Gli argomenti per `--item` sono memorizzati nel file `item.json`: (Per semplicità, vengono utilizzati solo pochi attributi dell'item).

```
{
    "Id": {"N": "456" },
    "ProductCategory": {"S": "Sporting Goods" },
    "Price": {"N": "650" }
}
```

Argomenti

- Put condizionale
- Eliminazioni condizionali
- Aggiornamenti condizionali
- Esempi di espressioni condizionali

Put condizionale

L'operazione `PutItem` sovrascrive un elemento con la stessa chiave primaria (se esiste). Se vuoi evitare che ciò accada, utilizza un'espressione di condizione. Ciò consente alla scrittura di procedere solo se l'elemento in questione non possiede già la stessa chiave primaria.

L'esempio seguente utilizza `attribute_not_exists()` per verificare se la chiave primaria esiste nella tabella prima di tentare l'operazione di scrittura.

> **Note**
>
> Se la chiave primaria è composta sia da una chiave di partizione (PK) che da una chiave di ordinamento (SK), il parametro controllerà se `attribute_not_exists(pk)` E `attribute_not_exists(sk)` vengono considerati true o false come intera istruzione prima di tentare l'operazione di scrittura.

```
aws dynamodb put-item \
    --table-name ProductCatalog \
    --item file://item.json \
    --condition-expression "attribute_not_exists(Id)"
```

Se l'espressione di condizione viene valutata false, D restituisce il seguente messaggio di errore: The conditional request failed (La richiesta condizionale ha avuto esito negativo).

> **Note**
>
> Per ulteriori informazioni su `attribute_not_exists` e altre funzioni, consulta Espressioni, operatori e funzioni di condizione e di filtro in DynamoDB.

Eliminazioni condizionali

Per eseguire un'eliminazione condizionale, utilizza un'operazione `DeleteItem` con un'espressione di condizione. L'espressione di condizione deve restituire true affinché l'operazione abbia esito positivo; in caso contrario, ha esito negativo.

Si consideri l'elemento sopra definito.

Supponiamo che tu voglia eliminare l'elemento, ma solo alle seguenti condizioni:

- il valore `ProductCategory` è "Sporting Goods" o "Gardening Supplies";
- il valore `Price` è compreso tra 500 e 600.

Nell'esempio seguente si tenta di eliminare l'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
```

Gli argomenti per `--expression-attribute-values` sono memorizzati nel file `values.json`:

```
{
    ":cat1": {"S": "Sporting Goods"},
    ":cat2": {"S": "Gardening Supplies"},
    ":lo": {"N": "500"},
    ":hi": {"N": "600"}
}
```

> **Note**
>
> Nell'espressione di condizione, `:` (due punti) indica un valore di attributo dell'espressione, ovvero un segnaposto per un valore effettivo. Per ulteriori informazioni, consulta Utilizzo dei valori degli attributi di espressione in DynamoDB.
> Per ulteriori informazioni su `IN`, `AND` e altre parole chiave, consulta Espressioni, operatori e funzioni di condizione e di filtro in DynamoDB.

In questo esempio, il confronto `ProductCategory` viene valutato come true, ma il confronto `Price` viene valutato come false. Ciò fa sì che l'espressione di condizione venga valutata come false e che l'operazione `DeleteItem` abbia esito negativo.

Aggiornamenti condizionali

Per eseguire un aggiornamento condizionale, utilizza un'operazione `UpdateItem` con un'espressione di condizione. L'espressione di condizione deve restituire true affinché l'operazione abbia esito positivo; in caso contrario, ha esito negativo.

> **Note**
>
> `UpdateItem` supporta anche espressioni di aggiornamento, in cui si specificano le modifiche che intendi apportare a una voce. Per ulteriori informazioni, consulta Utilizzo di espressioni di aggiornamento in DynamoDB.

Si supponga di iniziare con l'elemento mostrato sopra.

Nel seguente esempio viene eseguita un'operazione `UpdateItem`. Prova a ridurre il `Price` di un prodotto di 75, ma l'espressione della condizione impedisce l'aggiornamento se `Price` è minore o uguale a 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
```

Gli argomenti per `--expression-attribute-values` sono memorizzati nel file `values.json`:

```
{
    ":discount": { "N": "75"},
    ":limit": {"N": "500"}
}
```

Se l'attributo `Price` iniziale è 650, l'operazione `UpdateItem` riduce `Price` a 575. Se esegui nuovamente l'operazione `UpdateItem`, il valore `Price` viene ridotto a 500. Se esegui l'operazione una terza volta, l'espressione di condizione restituisce false e l'aggiornamento ha esito negativo.

> **Note**
>
> Nell'espressione di condizione, `:` (due punti) indica un valore di attributo dell'espressione, ovvero un segnaposto per un valore effettivo. Per ulteriori informazioni, consulta Utilizzo dei valori degli attributi di espressione in DynamoDB.
> Per ulteriori informazioni su "**>**" e altri operatori, consulta Espressioni, operatori e funzioni di condizione e di filtro in DynamoDB.

Esempi di espressioni condizionali

Per ulteriori informazioni sulle funzioni utilizzate negli esempi seguenti, consulta Espressioni, operatori e funzioni di condizione e di filtro in DynamoDB. Per ulteriori informazioni su come specificare diversi tipi di attributo in un'espressione, consulta Riferimento agli attributi degli elementi quando si utilizzano le espressioni in DynamoDB.

Verifica degli attributi in un elemento

Puoi verificare la presenza o l'assenza di qualsiasi attributo. Se l'espressione condizionale viene valutata a true, l'operazione ha esito positivo, altrimenti ha esito negativo.

Nel seguente esempio si utilizza `attribute_not_exists` per eliminare un prodotto solo se non possiede un attributo `Price`.

```
aws dynamodb delete-item \
    --table-name ProductCatalog \
    --key '{"Id": {"N": "456"}}' \
    --condition-expression "attribute_not_exists(Price)"
```

DynamoDB fornisce anche una funzione `attribute_exists`. Nel seguente esempio un prodotto viene eliminato solo se ha ricevuto recensioni negative.

```
aws dynamodb delete-item \
    --table-name ProductCatalog \
    --key '{"Id": {"N": "456"}}' \
    --condition-expression "attribute_exists(ProductReviews.OneStar)"
```

Verifica del tipo di attributo

Puoi controllare il tipo di dati di un valore di attributo utilizzando la funzione `attribute_type`. Se l'espressione condizionale viene valutata a true, l'operazione ha esito positivo, altrimenti ha esito negativo.

Nell'esempio seguente viene utilizzato `attribute_type` per eliminare un prodotto solo se ha un attributo `Color` di 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
```

Gli argomenti per `--expression-attribute-values` sono memorizzati nel expression-attribute-values file.json.

```
{
    ":v_sub":{"S":"SS"}
```

```
}
```

Verifica del valore iniziale della stringa

Puoi verificare se un valore di attributo String inizia con una particolare sottostringa utilizzando la funzione `begins_with`. Se l'espressione condizionale viene valutata a true, l'operazione ha esito positivo, altrimenti ha esito negativo.

Nell'esempio seguente viene utilizzato `begins_with` per eliminare un prodotto solo se l'elemento `FrontView` della mappa `Pictures` inizia con un valore specifico.

```
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
```

Gli argomenti per `--expression-attribute-values` sono memorizzati nel file expression-attribute-values .json.

```
{
    ":v_sub":{"S":"http://"}
}
```

Controllo di un elemento in un set

Puoi verificare la presenza di un elemento in un set o cercare una sottostringa all'interno di una stringa utilizzando la funzione `contains`. Se l'espressione condizionale viene valutata a true, l'operazione ha esito positivo, altrimenti ha esito negativo.

Nell'esempio seguente viene utilizzato `contains` per eliminare un prodotto solo se il String Set `Color` ha un elemento con un valore specifico.

```
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
```

Gli argomenti per `--expression-attribute-values` sono memorizzati nel file expression-attribute-values .json.

```
{
    ":v_sub":{"S":"Red"}
}
```

Controllo della dimensione di un valore di attributo

Puoi verificare la dimensione di un valore di attributo utilizzando la funzione `size`. Se l'espressione condizionale viene valutata a true, l'operazione ha esito positivo, altrimenti ha esito negativo.

Nell'esempio seguente viene utilizzato `size` per eliminare un prodotto solo se la dimensione dell'attributo `VideoClip` Binary è maggiore di `64000` byte.

```
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
```

Gli argomenti per `--expression-attribute-values` sono memorizzati nel file expression-attribute-values .json.

```
{
    ":v_sub":{"N":"64000"}
}
```

Utilizzo del Time to Live in DynamoDB

Il Time to Live (TTL) per DynamoDB è un metodo conveniente per eliminare elementi che non sono più pertinenti. Il TTL consente di definire un timestamp per elemento in modo da indicare quando un elemento non è più necessario. DynamoDB elimina automaticamente gli elementi scaduti entro pochi giorni dalla data di scadenza, senza utilizzare il throughput di scrittura.

Per utilizzare il TTL, è necessario prima abilitarlo su una tabella e poi definire un attributo specifico per archiviare il timestamp di scadenza del TTL. Il timestamp deve essere archiviato nel formato ora epoch Unix con la granularità dei secondi. Ogni volta che un elemento viene creato o aggiornato, è possibile calcolare l'ora di scadenza e salvarla nell'attributo TTL.

Gli articoli con attributi TTL validi e scaduti possono essere eliminati dal sistema in qualsiasi momento, in genere entro pochi giorni dalla scadenza. È comunque possibile aggiornare gli elementi

scaduti in attesa di eliminazione, come anche modificare o rimuovere i relativi attributi TTL. Durante l'aggiornamento di un elemento scaduto, si consiglia di utilizzare un'espressione condizionale per assicurarsi che l'elemento non sia stato successivamente eliminato. Utilizza le espressioni di filtro per rimuovere gli elementi scaduti dai risultati di Scan e Query.

Gli elementi eliminati funzionano in modo simile a quelli eliminati tramite le tipiche operazioni di eliminazione. Una volta eliminati, gli elementi entrano nei flussi DynamoDB come eliminazioni da parte del servizio anziché degli utenti e vengono rimossi dagli indici secondari locali e dagli indici secondari globali proprio come con le altre operazioni di eliminazione.

Se si utilizzano le Tabelle globali versione 2019.11.21 (Corrente) e si utilizza anche la funzionalità TTL, DynamoDB replica le eliminazioni TTL in tutte le tabelle di replica. L'eliminazione TTL iniziale non consuma unità di capacità di scrittura (WCU, Write Capacity Units) nella Regione in cui si verifica la scadenza del TTL. Tuttavia, l'eliminazione TTL replicata nelle tabelle di replica consuma una WCU replicata quando si utilizza la capacità con provisioning o la scrittura replicata quando si utilizza la modalità con capacità on demand in ciascuna delle Regioni di replica e verranno addebitati i costi applicabili.

Per ulteriori informazioni su TTL, consulta i seguenti argomenti:

Argomenti

- Abilitazione del Time to Live in DynamoDB
- Calcolo del Time to Live (TTL) in DynamoDB
- Utilizzo di elementi scaduti e Time to Live

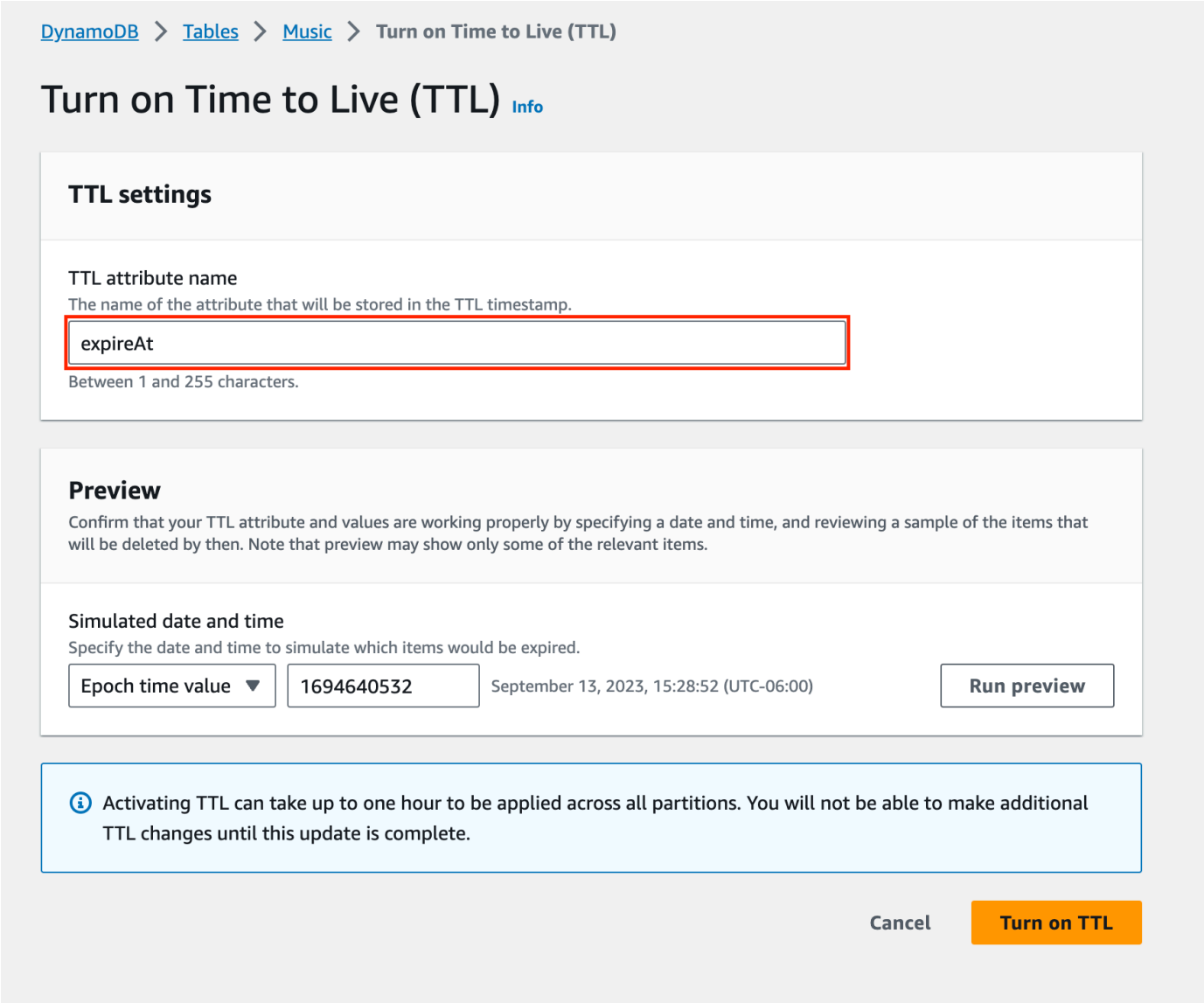
Abilitazione del Time to Live in DynamoDB

> **Note**
>
> Per facilitare il debug e la verifica del corretto funzionamento della funzionalità del TTL, i valori forniti per l'elemento TTL vengono registrati in testo semplice nei log di diagnostica di DynamoDB.

È possibile abilitare il TTL nella console Amazon DynamoDB, nella AWS Command Line Interface (AWS CLI) o utilizzando la documentazione di riferimento dell'API DynamoDB con uno qualsiasi degli SDK AWS previsti. È necessaria circa un'ora per abilitare il TTL su tutte le partizioni.

Abilitazione del TTL di DynamoDB tramite la console AWS

1. Accedi alla console Console di gestione AWS e apri la console DynamoDB all'indirizzo https://console.aws.amazon.com/elasticache/.
2. Scegli Tables (Tabelle) e quindi seleziona la tabella da modificare.
3. Nella scheda Impostazioni aggiuntive, nella sezione Time to Live (TTL) seleziona Attiva.
4. Quando abiliti TTL su una tabella, DynamoDB richiede di indicare un determinato nome attributo ricercato dal servizio per determinare se un item è idoneo per la scadenza. Il nome dell'attributo TTL, mostrato di seguito, fa distinzione tra maiuscole e minuscole e deve corrispondere all'attributo definito nelle operazioni di lettura e scrittura. In caso di mancata corrispondenza, gli elementi scaduti non verranno eliminati. La ridenominazione dell'attributo TTL richiede di disabilitare il TTL, per poi riabilitarlo con il nuovo attributo. Una volta disabilitato, il TTL continuerà a elaborare le eliminazioni per circa 30 minuti. Il TTL deve essere riconfigurato sulle tabelle ripristinate.



5. (Facoltativo) È possibile eseguire un test simulando la data e l'ora della scadenza con la corrispondenza di alcuni elementi. Questo fornisce un elenco di esempio di elementi e conferma che esistono elementi contenenti il nome dell'attributo TTL fornito insieme alla scadenza.

Una volta abilitato il TTL, l'attributo TTL viene contrassegnato come TTL quando si visualizzano gli elementi nella console DynamoDB. Puoi visualizzare la data e l'ora di scadenza di un item passando il puntatore del mouse sull'attributo.

Abilitazione del TTL di DynamoDB tramite l'API

Python

È possibile abilitare il TTL attraverso il codice, utilizzando l'operazione 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')
```

È possibile verificare che il TTL sia abilitato utilizzando l'operazione DescribeTimeToLive, che descrive lo stato del TTL su una tabella. Lo stato di `TimeToLive` è `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

È possibile abilitare il TTL attraverso il codice, utilizzando l'operazione 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');
```

Abilitazione del Time to Live tramite la AWS CLI

1. Abilita TTL nella tabella `TTLExample`.

```
aws dynamodb update-time-to-live --table-name TTLExample --time-to-live-
specification "Enabled=true, AttributeName=ttl"
```

2. Descrivi TTL nella tabella `TTLExample`.

```
aws dynamodb describe-time-to-live --table-name TTLExample
{
    "TimeToLiveDescription": {
        "AttributeName": "ttl",
        "TimeToLiveStatus": "ENABLED"
    }
}
```

3. Aggiungi un elemento alla tabella `TTLExample` con l'attributo Time to Live (TTL) impostato usando la shell BASH e la AWS CLI.

```
EXP=`date -d '+5 days' +%s`
aws dynamodb put-item --table-name "TTLExample" --item '{"id": {"N": "1"}, "ttl":
 {"N": "'$EXP'"}}'
```

Questo esempio inizia con la data corrente e aggiunge 5 giorni per creare un periodo di scadenza. Il periodo di scadenza viene quindi convertito nel formato temporale epoch Unix per aggiungere infine un item nella tabella `TTLExample`.

> **Note**
>
> Un modo per impostare i valori di scadenza per il Time to Live (TTL) consiste nel calcolare il numero di secondi da aggiungere al periodo di scadenza. Ad esempio, 5 giorni sono 432.000 secondi. È tuttavia in genere preferibile scegliere come punto di partenza una data.

Ottenere l'ora corrente in formato epoch Unix è piuttosto semplice, come negli esempi seguenti.

- Terminale Linux: `date +%s`
- Python: `import time; int(time.time())`
- Java: `System.currentTimeMillis() / 1000L`
- JavaScript: `Math.floor(Date.now() / 1000)`

Abilitazione del TTL di DynamoDB tramite 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
```

Ulteriori dettagli sull'utilizzo del TTL all'interno dei modelli CloudFormation sono disponibili qui.

Calcolo del Time to Live (TTL) in DynamoDB

Un modo comune per implementare il TTL consiste nell'impostare una scadenza per gli elementi in base a quando sono stati creati o aggiornati l'ultima volta. Questa operazione è fattibile aggiungendo la scadenza ai timestamp `createdAt` e `updatedAt`. Ad esempio, il TTL per gli elementi appena creati può essere impostato su `createdAt` + 90 giorni. Quando l'elemento viene aggiornato, il TTL può essere ricalcolato a `updatedAt` + 90 giorni.

La scadenza calcolata deve essere in formato epoch, in secondi. Per essere considerato valido per la scadenza e l'eliminazione, il TTL non può essere passato da più di cinque anni. Se si utilizza un altro formato, i processi TTL ignorano l'item. Se si imposta la data di scadenza su un periodo futuro in cui si desidera che l'elemento scada, l'elemento scadrà dopo tale periodo. Ad esempio, si supponga di aver impostato la data di scadenza su 1724241326 (ovvero lunedì 21 agosto 2024 11:55:26 (GMT)). L'elemento scade dopo il periodo specificato.

Argomenti

- Creazione di un oggetto e impostazione del Time to Live
- Aggiornamento di un elemento e del Time to Live

Creazione di un oggetto e impostazione del Time to Live

L'esempio seguente mostra come calcolare la scadenza durante la creazione di un nuovo elemento, utilizzando `expireAt` come nome dell'attributo TTL. Un'istruzione di assegnazione ottiene il momento corrente come variabile. Nell'esempio, la scadenza viene calcolata come a 90 giorni dal momento corrente. La scadenza viene quindi convertita in formato epoch e salvata come tipo di dati intero nell'attributo TTL.

Gli esempi di codice seguenti mostrano come creare un elemento con un valore TTL.

Java

SDK per 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;
        }
    }
}
```

- Per i dettagli sull'API, PutItemconsulta AWS SDK for Java 2.x API Reference.

JavaScript

SDK per 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');
```

- Per i dettagli sull'API, consulta la sezione AWS SDK per JavaScript API PutItemReference.

Python

SDK per 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"
)
```

- Per i dettagli sull'API, consulta PutItem AWSSDK for Python (Boto3) API Reference.

Aggiornamento di un elemento e del Time to Live

Questo esempio è una continuazione di quello della sezione precedente. La scadenza può essere ricalcolata se l'elemento viene aggiornato. L'esempio seguente ricalcola il timestamp `expireAt` in modo che corrisponda a 90 giorni dal momento corrente.

Gli esempi di codice seguenti mostrano come aggiornare il valore TTL di un elemento.

Java

SDK per Java 2.x

Per aggiornare il TTL su un elemento DynamoDB esistente in una tabella.

```
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;
        }
    }
```

- Per i dettagli sulle API, consulta UpdateItemla sezione API Reference.AWS SDK for Java 2.x

JavaScript

SDK per 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');
```

- Per i dettagli sull'API, consulta la sezione AWS SDK per JavaScript API UpdateItemReference.

Python

SDK per 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"
```

```
)
```

- Per i dettagli sull'API, consulta UpdateItem AWSSDK for Python (Boto3) API Reference.

Gli esempi del TTL discussi in questa introduzione dimostrano un metodo per garantire che in una tabella vengano conservati solo gli elementi aggiornati di recente. Gli elementi aggiornati hanno una durata prolungata, mentre gli elementi non aggiornati dopo la creazione scadono e vengono eliminati gratuitamente, liberando spazio di archiviazione e mantenendo le tabelle pulite.

Utilizzo di elementi scaduti e Time to Live

Gli elementi scaduti in attesa di eliminazione possono essere filtrati dalle operazioni di lettura e scrittura. Ciò è utile negli scenari in cui i dati scaduti non sono più validi e non devono essere utilizzati. Se non vengono filtrati, continueranno a essere visualizzati nelle operazioni di lettura e scrittura finché non verranno eliminati dal processo in background.

> **Note**
>
> Questi elementi continuano a essere conteggiati ai fini dei costi di archiviazione e lettura fino a quando non vengono eliminati.

Le eliminazioni TTL possono essere identificate nei flussi DynamoDB, ma solo nella Regione in cui è avvenuta l'eliminazione. Le eliminazioni TTL che vengono replicate nelle aree della tabella globale non sono identificabili nei flussi DynamoDB nelle aree in cui viene replicata l'eliminazione.

Filtraggio di elementi scaduti dalle operazioni di lettura

Per operazioni di lettura come Scan e Query, un'espressione di filtro può filtrare gli elementi scaduti in attesa di eliminazione. Come illustrato nel seguente frammento di codice, l'espressione di filtro può filtrare gli elementi in cui il TTL è uguale o inferiore al momento corrente. Ad esempio, il codice dell'SDK per Python include un'istruzione di assegnazione che ottiene il momento corrente come variabile (`now`) e la converte in `int` per il formato ora epoch.

Gli esempi di codice seguenti mostrano come eseguire query su elementi TTL.

Java

SDK per Java 2.x

Interroga l'espressione filtrata per raccogliere elementi TTL in una tabella DynamoDB utilizzando. 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;
        }
```

- Per ulteriori informazioni sulle API, consulta Query nella Documentazione di riferimento delle API AWS SDK for Java 2.x .

JavaScript

SDK per (v3) JavaScript

Interroga l'espressione filtrata per raccogliere elementi TTL in una tabella DynamoDB utilizzando. AWS SDK per 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');
```

- Per ulteriori informazioni sulle API, consulta Query nella Documentazione di riferimento delle API AWS SDK per JavaScript .

Python

SDK per Python (Boto3)

Interroga l'espressione filtrata per raccogliere elementi TTL in una tabella DynamoDB utilizzando. 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")
```

- Per informazioni dettagliate sulle API, consulta Query nella Documentazione di riferimento per l'API SDK for Python (Boto3)AWS .

Scrittura in base a una condizione su elementi scaduti

È possibile utilizzare un'espressione condizionale per evitare scritture su elementi scaduti. Il frammento di codice riportato di seguito è un aggiornamento condizionale che verifica se la scadenza è posteriore al momento corrente. Se true, l'operazione di scrittura continuerà.

Gli esempi di codice seguenti mostrano come aggiornare in modo condizionale il valore TTL di un elemento.

Java

SDK per Java 2.x

Per aggiornare il TTL di un elemento DynamoDB esistente in una tabella con una condizione.

```
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;
        }
    }
}
```

- Per i dettagli sulle API, consulta UpdateItemla sezione API Reference.AWS SDK for Java 2.x

JavaScript

SDK per JavaScript (v3)

Per aggiornare il TTL di un elemento DynamoDB esistente in una tabella con una condizione.

```
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');
```

- Per i dettagli sull'API, consulta la sezione AWS SDK per JavaScript API UpdateItemReference.

Python

SDK per Python (Boto3)

Per aggiornare il TTL di un elemento DynamoDB esistente in una tabella con una condizione.

```
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",
)
```

- Per i dettagli sull'API, consulta UpdateItem AWSSDK for Python (Boto3) API Reference.

Identificazione degli elementi eliminati nei flussi DynamoDB

Il record Streams contiene un campo di identità utente `Records[<index>].userIdentity`. Gli elementi eliminati dal processo TTL hanno i seguenti campi:

```
Records[<index>].userIdentity.type
"Service"

Records[<index>].userIdentity.principalId
```

```
"dynamodb.amazonaws.com"
```

Il JSON seguente mostra la porzione rilevante di un singolo record di flussi:

```
"Records": [
  {
 ...
  "userIdentity": {
  "type": "Service",
       "principalId": "dynamodb.amazonaws.com"
    }
   ...
 }
]
```

Esecuzione di query in DynamoDB

Puoi usare l'operazione API `Query` in Amazon DynamoDB per trovare gli elementi in base ai valori delle chiavi primarie.

È necessario fornire il nome dell'attributo della chiave di partizione e un singolo valore per tale attributo. `Query` restituisce tutti gli elementi con lo stesso valore della chiave di partizione. Facoltativamente, puoi fornire un attributo della chiave di ordinamento e utilizzare un operatore di confronto per perfezionare i risultati della ricerca.

Per ulteriori informazioni su come usare `Query`, come la sintassi della richiesta, i parametri di risposta e altri esempi, vedere Query nella Documentazione di riferimento dell'API Amazon DynamoDB.

Argomenti

- Espressioni di condizione della chiave per l’operazione Query in DynamoDB
- Espressioni di filtro per l’operazione Query in DynamoDB
- Impaginazione dei risultati delle query sulle tabelle in DynamoDB
- Altri aspetti dell’utilizzo dell’operazione Query in DynamoDB

Espressioni di condizione della chiave per l’operazione Query in DynamoDB

Puoi utilizzare qualsiasi nome di attributo in un'espressione condizionale della chiave, a condizione che il primo carattere sia `a-z` o `A-Z` e che il resto dei caratteri (a iniziare dal secondo carattere, se

presente) sia `a-z`, `A-Z` o `0-9`. Inoltre, il nome dell'attributo non deve essere una parola riservata di DynamoDB. Per l'elenco completo delle parole riservate, consulta Parole riservate in DynamoDB. Se un nome di attributo non soddisfa questi requisiti, devi definire un nome di attributo dell'espressione come un segnaposto. Per ulteriori informazioni, consulta Nomi di attributi di espressione (alias) in DynamoDB.

Per gli elementi con un determinato valore di chiave di partizione, DynamoDB li memorizza tutti insieme, ordinati in base al valore della chiave di ordinamento. In un'operazione `Query`, DynamoDB recupera gli elementi nell'ordine stabilito e quindi li elabora utilizzando `KeyConditionExpression` e qualsiasi `FilterExpression` eventualmente presente. Solo allora i risultati della `Query` vengono restituiti al client.

Un'operazione `Query` restituisce sempre un set di risultati. Se non vengono trovati item corrispondenti, il set di risultati è vuoto.

I risultati della `Query` sono sempre ordinati per il valore della chiave di ordinamento. Se il tipo di dati della chiave di ordinamento è `Number`, i risultati vengono restituiti in ordine numerico. In caso contrario, i risultati vengono restituiti nell'ordine di byte UTF-8. Per impostazione predefinita, l'ordinamento è crescente. Per invertire l'ordine, imposta il parametro `ScanIndexForward` su `false`.

Una singola operazione `Query` può recuperare un massimo di 1 MB di dati. Questo limite si applica prima che ai risultati venga applicato `FilterExpression` o `ProjectionExpression`. Se `LastEvaluatedKey` è presente nella risposta ed è non null, devi eseguire la paginazione del set di risultati (consulta Impaginazione dei risultati delle query sulle tabelle in DynamoDB).

Esempi di espressioni di condizione della chiave

Per specificare i criteri di ricerca, è possibile utilizzare una espressione di condizione chiave, ovvero una stringa che determina gli elementi da leggere dalla tabella o dall'indice.

Devi specificare il nome e il valore della chiave di partizione come condizione di uguaglianza. Non puoi utilizzare un attributo non chiave in un'espressione di condizione chiave.

Facoltativamente puoi fornire una seconda condizione per la chiave di ordinamento (se presente). La condizione della chiave di ordinamento deve utilizzare uno dei seguenti operatori di confronto:

- *a* = *b*: true se l'attributo *a* è uguale al valore *b*
- *a* < *b*: true se *a* è minore di *b*
- *a* <= *b*: true se *a* è minore o uguale a *b*

- *a* > *b*: true se *a* è maggiore di *b*
- *a* >= *b*: true se *a* è maggiore o uguale a *b*
- *a* BETWEEN *b* AND *c*: true se *a* è maggiore o uguale a *b* e minore o uguale a *c*.

È supportata anche la seguente funzione:

- begins_with (*a*, *substr*): true se il valore dell'attributo *a* inizia con una determinata sottostringa.

I seguenti esempi di AWS Command Line Interface (AWS CLI) dimostrano l'uso delle espressioni di condizione della chiave. Queste espressioni usano segnaposti (come ad esempio `:name` e `:sub`) anziché i valori effettivi. Per ulteriori informazioni, consultare Nomi di attributi di espressione (alias) in DynamoDB e Utilizzo dei valori degli attributi di espressione in DynamoDB.

Example

Esegui la query sulla tabella `Thread` per un particolare `ForumName` (chiave di partizione). Tutti gli elementi con quel valore di `ForumName` vengono letti dalla query, perché la chiave di ordinamento (`Subject`) non è inclusa in `KeyConditionExpression`.

```
aws dynamodb query \
    --table-name Thread \
    --key-condition-expression "ForumName = :name" \
    --expression-attribute-values  '{":name":{"S":"Amazon DynamoDB"}}'
```

Example

Esegui la query sulla tabella `Thread` per un particolare `ForumName` (chiave di partizione); questa volta vengono restituiti solo gli elementi con uno specifico `Subject` (chiave di ordinamento).

```
aws dynamodb query \
    --table-name Thread \
    --key-condition-expression "ForumName = :name and Subject = :sub" \
    --expression-attribute-values  file://values.json
```

Gli argomenti per `--expression-attribute-values` sono memorizzati nel file `values.json`:

```
{
```

```
    ":name":{"S":"Amazon DynamoDB"},
    ":sub":{"S":"DynamoDB Thread 1"}
}
```

Example

Esegui la query sulla tabella `Reply` per un particolare `Id` (chiave di partizione); vengono restituiti solo gli elementi il cui `ReplyDateTime` (chiave di ordinamento) inizia con determinati caratteri.

```
aws dynamodb query \
    --table-name Reply \
    --key-condition-expression "Id = :id and begins_with(ReplyDateTime, :dt)" \
    --expression-attribute-values  file://values.json
```

Gli argomenti per `--expression-attribute-values` sono memorizzati nel file `values.json`:

```
{
    ":id":{"S":"Amazon DynamoDB#DynamoDB Thread 1"},
    ":dt":{"S":"2015-09"}
}
```

Espressioni di filtro per l'operazione Query in DynamoDB

Se devi perfezionare ulteriormente i risultati di `Query`, opzionalmente puoi fornire un'espressione filtro. Un'espressione di filtro determina quali item all'interno dei risultati di `Query` devono essere restituiti. Tutti gli altri risultati vengono scartati.

Un'espressione di filtro viene applicata al termine di un'operazione `Query`, ma prima che i risultati vengano restituiti. Pertanto, un'operazione `Query` consuma la stessa quantità di capacità in lettura, a prescindere dal fatto che sia presente un'espressione di filtro.

Un'operazione `Query` può recuperare un massimo di 1 MB di dati. Questo limite si applica prima che l'espressione di filtro venga valutata.

Un'espressione di filtro non può contenere attributi di chiave di partizione o chiave di ordinamento. Devi specificare quegli attributi nell'espressione della condizione di chiave e non nell'espressione di filtro.

La sintassi per un'espressione di filtro è simile a quella di un'espressione di condizione chiave. Le espressioni di filtro possono utilizzare gli stessi comparatori, funzioni e operatori logici come

espressione di condizione chiave. Inoltre, le espressioni di filtro possono utilizzare l'operatore non uguale (<>), l'operatore `OR`, l'operatore `CONTAINS`, l'operatore `IN`, l'operatore `BEGINS_WITH`, l'operatore `BETWEEN`, l'operatore `EXISTS` e l'operatore `SIZE`. Per ulteriori informazioni, consultare Espressioni di condizione della chiave per l'operazione Query in DynamoDB e Sintassi per le espressioni di filtro e condizioni.

Example

Il seguente esempio di AWS CLI consente di eseguire query sulla tabella `Thread` per parametri `ForumName` (chiave di partizione) e `Subject` (chiave di ordinamento) specifici. Tra gli elementi trovati, vengono restituiti solo i thread di discussione più popolari, in altre parole solo i thread con più di un certo numero di `Views`.

```
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
```

Gli argomenti per `--expression-attribute-values` sono memorizzati nel file `values.json`:

```
{
    ":fn":{"S":"Amazon DynamoDB"},
    ":sub":{"S":"DynamoDB Thread 1"},
    ":num":{"N":"3"}
}
```

Tenere presente che `Views` è una parola riservata in DynamoDB (vedere Parole riservate in DynamoDB), quindi in questo esempio si usa #v come segnaposto. Per ulteriori informazioni, consulta Nomi di attributi di espressione (alias) in DynamoDB.

> **Note**
>
> Un'espressione filtro rimuove gli elementi dal set di risultati della `Query`. Se possibile, evita di utilizzare `Query` dove ti aspetti di recuperare un gran numero di item, ma sai che devi anche scartare la maggior parte di questi item.

Impaginazione dei risultati delle query sulle tabelle in DynamoDB

DynamoDB esegue la paginazione dei risultati delle operazioni `Query`. Con la paginazione, i risultati della `Query` vengono divisi in "pagine" di dati la cui dimensione è al massimo 1 MB. Un'applicazione può elaborare la prima pagina dei risultati, quindi la seconda pagina e così via.

Una singola operazione `Query` restituisce solo un set di risultati che rientra nel limite di dimensione di 1 MB. Per determinare se ci sono più risultati e recuperarli una pagina alla volta, le applicazioni devono fare quanto segue:

1. Esamina i risultati della `Query` di livello inferiore:
   - Se il risultato contiene un elemento `LastEvaluatedKey` che non è null, passa alla fase 2.
   - Se nel risultato non è presente un `LastEvaluatedKey`, allora non ci sono altri elementi da recuperare.
2. Costruzione di una `Query` con la stessa `KeyConditionExpression`. Tuttavia, questa volta, accettare il valore `LastEvaluatedKey` della fase 1 e usarlo come parametro `ExclusiveStartKey` nella nuova richiesta di `Query`.
3. Eseguire la nuova richiesta di `Query`.
4. Passa alla fase 1.

In altre parole, l'item `LastEvaluatedKey` della risposta di uno `Query` deve essere usato come item `ExclusiveStartKey` per la successiva richiesta di `Query`. Se non è presente un elemento `LastEvaluatedKey` nella risposta di uno `Query`, vuol dire che hai recuperato la pagina finale dei risultati. Se `LastEvaluatedKey` non è vuoto, non significa necessariamente che esistano più dati nel set di risultati. L'unico modo per sapere che hai raggiunto la fine del set di risultati è quando `LastEvaluatedKey` è vuoto.

Puoi usare AWS CLI per osservare questo comportamento. La AWS CLI invia ripetutamente richieste `Query` di basso livello a DynamoDB, finché `LastEvaluatedKey` non è più presente nei risultati. Considera il seguente esempio di AWS CLI che recupera i titoli di film di un particolare anno:

```
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
```

In genere, AWS CLI gestisce la paginazione automaticamente. Tuttavia, in questo esempio, il parametro `--page-size` di AWS CLI limita il numero di item per pagina. Il parametro `--debug` consente di stampare le informazioni di livello inferiore relative alle richieste e alle risposte.

Se si esegue l'esempio, l'aspetto della prima risposta da DynamoDB sarà simile al seguente.

```
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}'
```

L'item `LastEvaluatedKey` nella risposta indica che non tutti gli elementi sono stati recuperati. La AWS CLI invia quindi un'altra richiesta `Query` a DynamoDB. Questo modello di richiesta e risposta continua fino alla risposta finale.

```
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}'
```

L'assenza di `LastEvaluatedKey` indica che non ci sono più item da recuperare.

> **Note**
>
> Gli SDK AWS gestiscono le risposte di basso livello di DynamoDB (inclusa la presenza o l'assenza di `LastEvaluatedKey`) e forniscono varie astrazioni per la paginazione dei risultati di `Query` Ad esempio, l'interfaccia del documento SDK per Java fornisce il supporto `java.util.Iterator` per poter esaminare i risultati uno alla volta.
> Per esempi di codice in vari linguaggi di programmazione, consulta la Guida alle operazioni di base di Amazon DynamoDB e la documentazione dell'SDK AWS per il linguaggio in uso.

È inoltre possibile ridurre la dimensione della pagina limitando il numero di elementi nel set di risultati, con il parametro `Limit` dell'operazione `Query`.

Per ulteriori informazioni sull'esecuzione di query con DynamoDB, consulta Esecuzione di query in DynamoDB.

Altri aspetti dell'utilizzo dell'operazione Query in DynamoDB

Questa sezione tratta aspetti aggiuntivi dell'operazione Query in DynamoDB, tra cui la limitazione delle dimensioni dei risultati, il conteggio degli elementi scansionati rispetto a quelli restituiti, il monitoraggio dell'utilizzo della capacità di lettura e il controllo della coerenza di lettura.

Limitazione del numero di elementi nel set di risultati

Con l'operazione `Query` puoi limitare il numero di elementi che vengono letti. A tale scopo, imposta il parametro `Limit` sul numero massimo di item desiderati.

Ad esempio, supponi di eseguire un'operazione `Query` su una tabella, con un valore `Limit` di 6 e senza un'espressione di filtro. Il risultato dell'operazione `Query` contiene i primi sei item della tabella che corrispondono all'espressione di condizione della chiave della richiesta.

Supponi ora di aggiungere un'espressione di filtro a `Query`. In questo caso, DynamoDB legge fino a sei elementi e restituisce solo quelli che corrispondono all'espressione di filtro. Il risultato `Query` finale contiene sei elementi o meno, anche se, se DynamoDB avesse continuato la lettura, più elementi avrebbero trovato la corrispondenza con l'espressione del filtro.

Conteggio degli elementi nei risultati

Oltre agli elementi che corrispondono ai tuoi criteri, la risposta `Query` contiene i seguenti elementi:

- `ScannedCount`: il numero di voci corrispondenti all'espressione di condizione della query prima dell'applicazione di un'espressione di filtro (se presente).
- `Count`: il numero di elementi che rimangono dopo aver applicato un'espressione di filtro.

> **Note**
>
> Se non si utilizza un'espressione di filtro `ScannedCount` e `Count` hanno lo stesso valore.

Se la dimensione del set di risultati di `Query` è maggiore di 1 MB, `ScannedCount` e `Count` rappresentano solo un conteggio parziale degli elementi totali. Devi eseguire più operazioni `Query` per recuperare tutti i risultati (consulta Impaginazione dei risultati delle query sulle tabelle in DynamoDB).

Ogni risposta di `Query` contiene `ScannedCount` e `Count` per gli elementi che sono stati elaborati da quella particolare richiesta `Query`. Per ottenere i totali generali per tutte le richieste di `Query`, puoi mantenere in esecuzione il conteggio per entrambi gli elementi `ScannedCount` e `Count`.

Unità di capacità utilizzate dalla query

Puoi eseguire `Query` su qualsiasi tabella o indice secondario, purché tu fornisca il nome dell'attributo della chiave di partizione e un singolo valore per tale attributo. `Query` restituisce tutti gli elementi con tale valore di chiave di partizione. Facoltativamente, puoi fornire un attributo della chiave di ordinamento e utilizzare un operatore di confronto per perfezionare i risultati della ricerca. `Query` Le operazioni API consumano unità di capacità in lettura, come segue.

Se si esegue **`Query`** per un...	DynamoDB utilizza le unità di capacità in lettura da...
Tabella	La capacità di lettura assegnata della tabella.
Indice secondario globale	La capacità di lettura assegnata dell'indice.
Indice secondario locale	La capacità di lettura assegnata della tabella di base.

Per impostazione predefinita, un'operazione `Query` non restituisce alcun dato sulla capacità di lettura che consuma. Tuttavia, puoi specificare il parametro `ReturnConsumedCapacity` in una richiesta di `Query` per ottenere queste informazioni. Le seguenti sono le impostazioni valide per `ReturnConsumedCapacity`:

- `NONE`: non vengono restituiti dati relativi alla capacità utilizzata. Questa è l'impostazione predefinita.
- `TOTAL`: la risposta include il numero aggregato di unità di capacità di lettura utilizzate.
- `INDEXES`: la risposta mostra il numero aggregato di unità di capacità di lettura utilizzate, insieme alla capacità utilizzata per ogni tabella e indice a cui è stato effettuato l'accesso.

DynamoDB calcola il numero di unità di capacità di lettura utilizzate in base al numero di elementi e alla dimensione di quegli elementi, non alla quantità di dati restituiti a un'applicazione. Per questo motivo, il numero di unità di capacità consumate è lo stesso sia che tu richieda tutti gli attributi (il comportamento predefinito) o solo alcuni di essi (usando un'espressione di proiezione). Il numero è lo stesso anche indipendentemente dal fatto che si utilizzi o meno un'espressione di filtro. `Query`

utilizza un'unità di capacità di lettura minima per eseguire un'operazione a elevata consistenza di lettura al secondo o due letture a coerenza finale al secondo per un elemento di un massimo di 4 KB. Se è necessario leggere un elemento che è più grande di 4 KB, DynamoDB necessità di unità di richiesta di lettura aggiuntive. Le tabelle vuote e le tabelle molto grandi con un numero limitato di chiavi di partizione potrebbero comportare un addebito di alcune RCU aggiuntive superiore alla quantità di dati sottoposti a query. Ciò copre il costo di evasione della richiesta `Query` anche in assenza di dati.

Consistenza di lettura per la query

Un'operazione `Query` esegue letture consistenti finali per impostazione predefinita. Ciò significa che i risultati di `Query` potrebbero non riflettere le modifiche dovute alle operazioni `PutItem` o `UpdateItem` completate di recente. Per ulteriori informazioni, consulta Coerenza di lettura di DynamoDB.

Se hai bisogno di elevata coerenza di lettura, imposta il parametro `ConsistentRead` su `true` nella richiesta di `Query`.

Scansione di tabelle in DynamoDB

Un'operazione `Scan` in Amazon DynamoDB legge ogni elemento in una tabella o in un indice secondario. Per impostazione predefinita, un'operazione `Scan` restituisce tutti gli attributi dei dati per ogni item nella tabella o nell'indice. Puoi usare il parametro `ProjectionExpression` in modo che `Scan` restituisca solo alcuni attributi, piuttosto che tutti.

`Scan` restituisce sempre un set di risultati. Se non vengono trovati item corrispondenti, il set di risultati è vuoto.

Una singola richiesta `Scan` può recuperare un massimo di 1 MB di dati. Facoltativamente, DynamoDB può applicare un'espressione di filtro a questi dati, limitando i risultati prima che vengano restituiti all'utente.

Argomenti

- Espressioni di filtro per la scansione
- Limitazione del numero di elementi nel set di risultati
- Paginazione dei risultati
- Conteggio degli elementi nei risultati
- Unità di capacità utilizzate dalla scansione

- Consistenza di lettura per la scansione
- Scansione parallela

Espressioni di filtro per la scansione

Se devi perfezionare ulteriormente i risultati di `Scan`, opzionalmente puoi fornire un'espressione filtro. Un'espressione di filtro determina quali item all'interno dei risultati di `Scan` devono essere restituiti. Tutti gli altri risultati vengono scartati.

Un'espressione di filtro viene applicata al termine di un'operazione `Scan`, ma prima che i risultati vengano restituiti. Pertanto, un'operazione `Scan` consuma la stessa quantità di capacità in lettura, a prescindere dal fatto che sia presente un'espressione di filtro.

Un'operazione `Scan` può recuperare un massimo di 1 MB di dati. Questo limite si applica prima che l'espressione di filtro venga valutata.

Con `Scan`, è possibile specificare qualsiasi attributo in un'espressione filtro, inclusi gli attributi di chiave di partizione e chiave di ordinamento.

La sintassi per un'espressione di filtro è identica a quella di un'espressione di condizione. Le espressioni di filtro possono utilizzare gli stessi comparatori, funzioni e operatori logici come espressione di condizione. Per ulteriori informazioni sugli operatori logici, consulta Espressioni, operatori e funzioni di condizione e di filtro in DynamoDB.

Example

L'esempio seguente AWS Command Line Interface (AWS CLI) analizza la `Thread` tabella e restituisce solo gli ultimi elementi inviati da un determinato utente.

```
aws dynamodb scan \
     --table-name Thread \
     --filter-expression "LastPostedBy = :name" \
     --expression-attribute-values '{":name":{"S":"User A"}}'
```

Limitazione del numero di elementi nel set di risultati

L'operazione `Scan` consente di limitare il numero di item restituiti nel risultato. A questo scopo, imposta il parametro `Limit` sul numero massimo di item che devono essere restituiti dall'operazione `Scan`, prima di filtrare la valutazione dell'espressione.

Ad esempio, supponi di eseguire un'operazione `Scan` su una tabella con un valore `Limit` di 6 e senza un'espressione di filtro. Il risultato `Scan` contiene i primi sei elementi della tabella.

Supponi ora di aggiungere un'espressione di filtro a `Scan`. In questo caso, DynamoDB applica l'espressione di filtro ai sei elementi che sono stati restituiti, eliminando quelli che non corrispondono. Il risultato finale dell'operazione `Scan` contiene al massimo 6 item, a seconda del numero di quelli che sono stati filtrati.

Paginazione dei risultati

DynamoDB esegue la paginazione dei risultati delle operazioni `Scan`. Con la paginazione, i risultati della `Scan` vengono divisi in "pagine" di dati la cui dimensione è al massimo 1 MB. Un'applicazione può elaborare la prima pagina dei risultati, quindi la seconda pagina e così via.

Una singola operazione `Scan` restituisce solo un set di risultati che rientra nel limite di dimensione di 1 MB.

Per determinare se ci sono più risultati e recuperarli una pagina alla volta, le applicazioni devono fare quanto segue:

1. Esamina i risultati della `Scan` di livello inferiore:
   - Se il risultato contiene un elemento `LastEvaluatedKey`, passa alla fase 2.
   - Se non è presente un item `LastEvaluatedKey` nel risultato non ci sono altri item da recuperare.
2. Costruire una nuova richiesta `Scan`, con gli stessi parametri della precedente. Tuttavia, questa volta, accettare il valore `LastEvaluatedKey` della fase 1 e usarlo come parametro `ExclusiveStartKey` nella nuova richiesta di `Scan`.
3. Eseguire la nuova richiesta di `Scan`.
4. Passa alla fase 1.

In altre parole, l'item `LastEvaluatedKey` della risposta di uno `Scan` deve essere usato come item `ExclusiveStartKey` per la successiva richiesta di `Scan`. Se non è presente un elemento `LastEvaluatedKey` nella risposta `Scan`, è stata recuperata la pagina finale dei risultati. L'assenza di `LastEvaluatedKey` è l'unico modo per sapere che hai raggiunto la fine del set di risultati.

È possibile utilizzare il AWS CLI per visualizzare questo comportamento. AWS CLI Invia `Scan` richieste di basso livello a DynamoDB, ripetutamente, `LastEvaluatedKey` fino a quando non è più

presente nei risultati. Considerate l' AWS CLI esempio seguente che analizza l'intera `Movies` tabella ma restituisce solo i film di un genere particolare.

```
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, AWS CLI gestisce l'impaginazione automaticamente. Tuttavia, in questo esempio, il AWS CLI `--page-size` parametro limita il numero di elementi per pagina. Il parametro `--debug` consente di stampare le informazioni di livello inferiore relative alle richieste e alle risposte.

> **Note**
>
> I risultati dell'impaginazione varieranno anche in base ai parametri di input che vengono passati.
>
> - L'utilizzo di `aws dynamodb scan --table-name Prices --max-items 1` restituisce un `NextToken`
> - L'utilizzo di `aws dynamodb scan --table-name Prices --limit 1` restituisce un `LastEvaluatedKey`.
>
> Inoltre, tieni presente che l'uso di `--starting-token` in particolare richiede il valore `NextToken`.

Se si esegue l'esempio, l'aspetto della prima risposta da DynamoDB sarà simile al seguente.

```
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}'
```

L'item `LastEvaluatedKey` nella risposta indica che non tutti gli elementi sono stati recuperati. AWS CLI Quindi invia un'altra `Scan` richiesta a DynamoDB. Questo modello di richiesta e risposta continua fino alla risposta finale.

```
2017-07-07 12:19:17,830 - MainThread - botocore.parsers - DEBUG - Response body:
b'{"Count":1,"Items":[{"title":{"S":"WarGames"}}],"ScannedCount":6}'
```

L'assenza di `LastEvaluatedKey` indica che non ci sono più item da recuperare.

> **Note**
>
> AWS SDKs Gestiscono le risposte DynamoDB di basso livello (inclusa la presenza o l'assenza di) e forniscono varie astrazioni per l'impaginazione `LastEvaluatedKey` dei risultati. `Scan` Ad esempio, l'interfaccia del documento SDK per Java fornisce il supporto `java.util.Iterator` per poter esaminare i risultati uno alla volta.
> Per esempi di codice in vari linguaggi di programmazione, consulta la Guida alle operazioni di base di Amazon DynamoDB e la documentazione dell'SDK AWS per il linguaggio in uso.

Conteggio degli elementi nei risultati

Oltre agli elementi che corrispondono ai tuoi criteri, la risposta `Scan` contiene i seguenti elementi:

- `ScannedCount`: il numero di elementi valutati, prima di qualsiasi `ScanFilter`applicato. Un valore `ScannedCount` elevato con pochi o nessun risultato `Count` indica un'operazione `Scan` inefficiente. Se non hai utilizzato un filtro nella richiesta, allora `ScannedCount` e `Count` avranno lo stesso valore.
- `Count`: il numero di elementi che rimangono dopo aver applicato un'espressione di filtro (se presente).

> **Note**
>
> Se non si utilizza un'espressione di filtro, `ScannedCount` e `Count` hanno lo stesso valore.

Se la dimensione del set di risultati di `Scan` è maggiore di 1 MB, `ScannedCount` e `Count` rappresentano solo un conteggio parziale degli elementi totali. Devi eseguire più operazioni `Scan` per recuperare tutti i risultati (consulta Paginazione dei risultati).

Ogni risposta di `Scan` contiene `ScannedCount` e `Count` per gli elementi che sono stati elaborati da quella particolare richiesta `Scan`. Per ottenere i totali generali per tutte le richieste `Scan`, puoi mantenere in esecuzione il conteggio per entrambi gli elementi `ScannedCount` e `Count`.

Unità di capacità utilizzate dalla scansione

È possibile effettuare la `Scan` di qualsiasi tabella o indice secondario. Le operazioni `Scan` consumano le unità di capacità di lettura come riportato di seguito:

Se si esegue **`Scan`** per un...	DynamoDB utilizza le unità di capacità in lettura da...
Tabella	La capacità di lettura assegnata della tabella.
Indice secondario globale	La capacità di lettura assegnata dell'indice.
Indice secondario locale	La capacità di lettura assegnata della tabella di base.

> **Note**
>
> L’accesso multi-account per le operazioni di scansione degli indici secondari non è attualmente supportato dalle policy basate su risorse.

Per impostazione predefinita, un'operazione `Scan` non restituisce alcun dato sulla capacità di lettura che consuma. Tuttavia, puoi specificare il parametro `ReturnConsumedCapacity` in una richiesta di `Scan` per ottenere queste informazioni. Le seguenti sono le impostazioni valide per `ReturnConsumedCapacity`:

- `NONE`: non vengono restituiti dati relativi alla capacità utilizzata. Questa è l'impostazione predefinita.
- `TOTAL`: la risposta include il numero aggregato di unità di capacità di lettura utilizzate.
- `INDEXES`: la risposta mostra il numero aggregato di unità di capacità di lettura utilizzate, insieme alla capacità utilizzata per ogni tabella e indice a cui è stato effettuato l'accesso.

DynamoDB calcola il numero di unità di capacità di lettura utilizzate in base al numero di elementi e alla dimensione di quegli elementi, non alla quantità di dati restituiti a un’applicazione. Per questo

motivo, il numero di unità di capacità consumate è lo stesso sia che tu richieda tutti gli attributi (il comportamento predefinito) o solo alcuni di essi (usando un'espressione di proiezione). Il numero è lo stesso anche indipendentemente dal fatto che si utilizzi o meno un'espressione di filtro. `Scan` utilizza un'unità di capacità di lettura minima per eseguire un'operazione a elevata consistenza di lettura al secondo o due letture a coerenza finale al secondo per un elemento di un massimo di 4 KB. Se è necessario leggere un elemento che è più grande di 4 KB, DynamoDB necessità di unità di richiesta di lettura aggiuntive. Le tabelle vuote e le tabelle molto grandi che hanno una quantità limitata di chiavi di partizione potrebbero comportare costi aggiuntivi RCUs oltre alla quantità di dati scansionati. Ciò copre il costo di evasione della richiesta `Scan` anche in assenza di dati.

Consistenza di lettura per la scansione

Un'operazione `Scan` esegue letture consistenti finali per impostazione predefinita. Ciò significa che i risultati di `Scan` potrebbero non riflettere le modifiche dovute alle operazioni `PutItem` o `UpdateItem` completate di recente. Per ulteriori informazioni, consulta Coerenza di lettura di DynamoDB.

Se hai bisogno di letture fortemente consistenti, dal momento in cui inizia lo `Scan`, imposta il parametro `ConsistentRead` su `true` nella richiesta di `Scan`. Ciò garantisce che tutte le operazioni di scrittura completate prima dell'inizio dell'operazione `Scan` sono incluse nella risposta `Scan`.

L'impostazione di `ConsistentRead` su `true` può essere utile negli scenari di backup o replica delle tabelle, insieme a DynamoDB Streams. Per prima cosa, usa `Scan` con `ConsistentRead` impostato su true, per ottenere una copia coerente dei dati nella tabella. Durante l'operazione `Scan`, DynamoDB Streams registra qualsiasi attività di scrittura aggiuntiva che si verifica sulla tabella. Al termine dell'operazione `Scan`, puoi applicare l'attività di scrittura dal flusso alla tabella.

> **Note**
>
> Un'operazione `Scan` con `ConsistentRead` impostato su `true` consuma il doppio delle unità di capacità in lettura, rispetto a lasciare `ConsistentRead` sul valore predefinito (`false`).

Scansione parallela

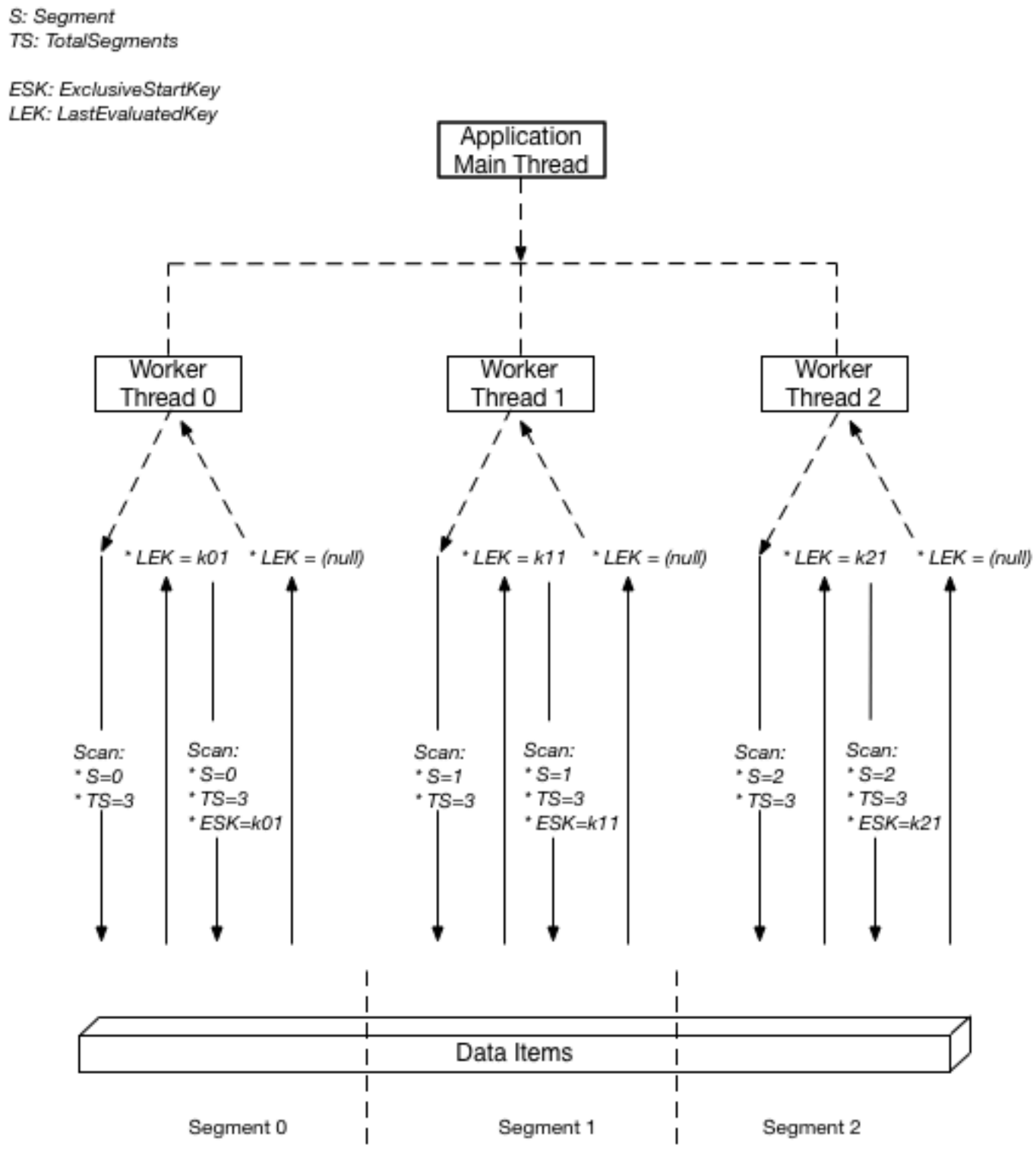
Per impostazione predefinita, l'operazione `Scan` elabora i dati in sequenza. Amazon DynamoDB restituisce i dati all'applicazione in incrementi di 1 MB e un'applicazione esegue ulteriori operazioni `Scan` per recuperare il successivo 1 MB di dati.

Più grande è la tabella o l'indice da sottoporre a scansione, più tempo richiederà `Scan` per il completamento. Inoltre, un'operazione `Scan` sequenziale potrebbe non essere sempre in grado di utilizzare completamente la capacità effettiva di trasmissione di lettura assegnata: anche se DynamoDB distribuisce i dati di una tabella di grandi dimensioni su più partizioni fisiche, un'operazione `Scan` può leggere solo una partizione alla volta. Per questo motivo, la velocità effettiva di una `Scan` è vincolata dalla velocità effettiva massima di una singola partizione.

Per risolvere questi problemi, l'operazione `Scan` può dividere in modo logico una tabella o un indice secondario in più segmenti, con più worker dell'applicazione che eseguono la scansione dei segmenti in parallelo. Ogni worker può essere un thread (in linguaggi di programmazione che supportano il multithreading) o un processo del sistema operativo. Per eseguire una scansione parallela, ogni worker emette la propria richiesta `Scan` con i seguenti parametri:

- `Segment`: un segmento che deve essere sottoposto a scansione da un particolare worker. Ogni worker deve utilizzare un valore diverso per `Segment`.
- `TotalSegments`: il numero totale dei segmenti per la scansione parallela. Il valore deve essere identico al numero di worker che saranno utilizzati dall'applicazione.

Nel diagramma riportato di seguito viene illustrato come un'applicazione multithread esegue una `Scan` parallela con tre gradi di parallelismo.



In questo diagramma, l'applicazione genera tre thread e assegna a ogni thread un numero. (Il primo numero dei segmenti è sempre 0.) Ogni thread emette una richiesta `Scan`, impostando `Segment`

sul suo numero designato e `TotalSegments` su 3. Ogni thread esegue la scansione del segmento designato, recuperando i dati 1 MB alla volta e restituisce i dati al thread principale dell'applicazione.

DynamoDB assegna gli elementi ai segmenti applicando una funzione hash alla chiave di partizione di ciascun elemento. Per un determinato `TotalSegments` valore, tutti gli elementi con la stessa chiave di partizione vengono sempre assegnati allo stesso valore. `Segment` Ciò significa che in una tabella in cui Item 1, Item 2 e Item 3 condividono tutti `pk="account#123"` (ma hanno chiavi di ordinamento diverse), questi elementi verranno elaborati dallo stesso lavoratore, indipendentemente dai valori delle chiavi di ordinamento o dalla dimensione della raccolta di elementi.

Poiché l'assegnazione dei segmenti si basa esclusivamente sull'hash della chiave di partizione, i segmenti possono essere distribuiti in modo non uniforme. Alcuni segmenti potrebbero non contenere elementi, mentre altri potrebbero contenere molte chiavi di partizione con raccolte di elementi di grandi dimensioni. Di conseguenza, l'aumento del numero totale di segmenti non garantisce prestazioni di scansione più rapide, in particolare quando le chiavi di partizione non sono distribuite uniformemente nello spazio delle chiavi.

I valori per `Segment` e `TotalSegments` si applicano alle singole richieste `Scan` ed è possibile usare valori diversi in qualsiasi momento. Potrebbe essere necessario fare diverse prove con questi valori e il numero di worker utilizzati fino a quando l'applicazione non raggiunge le prestazioni migliori.

> **Note**
>
> Una scansione parallela con un numero elevato di worker può facilmente utilizzare tutta la velocità effettiva assegnata per la tabella o l'indice da sottoporre a scansione. È preferibile evitare tali scansioni se la tabella o l'indice subiscono anche un'intensa attività di lettura o scrittura da altre applicazioni.
>
> Per controllare la quantità di dati restituiti per ogni richiesta, utilizza il parametro `Limit`. Ciò consente di evitare situazioni in cui un worker utilizza tutta la velocità effettiva assegnata a spese di tutti gli altri worker.

PartiQL: un linguaggio di query compatibile con SQL per Amazon DynamoDB

Amazon DynamoDB supporta PartiQL, un linguaggio di query compatibile con SQL, per selezionare, inserire, aggiornare ed eliminare i dati in Amazon DynamoDB. Utilizzando PartiQL, è possibile

interagire facilmente con le tabelle DynamoDB ed eseguire query ad hoc utilizzando NoSQL Workbench e Console di gestione AWS DynamoDB for PartiQL. AWS Command Line Interface APIs

Le operazioni PartiQL forniscono la stessa disponibilità, latenza e prestazioni delle altre operazioni del piano dati di DynamoDB.

Nelle sezioni seguenti viene descritta l'implementazione DynamoDB di PartiQL.

Argomenti

- Che cos'è PartiQL?
- PartiQL in Amazon DynamoDB
- Nozioni di base su PartiQL per DynamoDB
- Tipi di dati PartiQL per DynamoDB
- Istruzioni PartiQL per DynamoDB
- Utilizzo delle funzioni PartiQL con DynamoDB
- Operatori PartiQL aritmetici, di confronto e logici per DynamoDB
- Esecuzione di transazioni con PartiQL per DynamoDB
- Esecuzione di operazioni in batch con PartiQL per DynamoDB
- Policy di sicurezza IAM con PartiQL per DynamoDB

Che cos'è PartiQL?

PartiQL consente l'accesso alle query compatibile con SQL su più archivi dati contenenti dati strutturati, semistrutturati e nidificati. È ampiamente utilizzato in Amazon ed è ora disponibile come parte di molti AWS servizi, incluso DynamoDB.

Per la specifica PartiQL e un tutorial sul linguaggio delle query di base, consulta la Documentazione di PartiQL.

> **Note**
>
> - Amazon DynamoDB supporta un sottoinsieme del linguaggio di query PartiQL.
> - Amazon DynamoDB non supporta il formato dati Amazon Ion o i letterali Amazon Ion.

PartiQL in Amazon DynamoDB

Per eseguire query PartiQL in DynamoDB, è possibile utilizzare:

- La console DynamoDB
- NoSQL Workbench
- Il AWS Command Line Interface ()AWS CLI
- Il DynamoDB APIs

Per informazioni sull'utilizzo di questi metodi per accedere a DynamoDB, consulta Accesso a DynamoDB.

Nozioni di base su PartiQL per DynamoDB

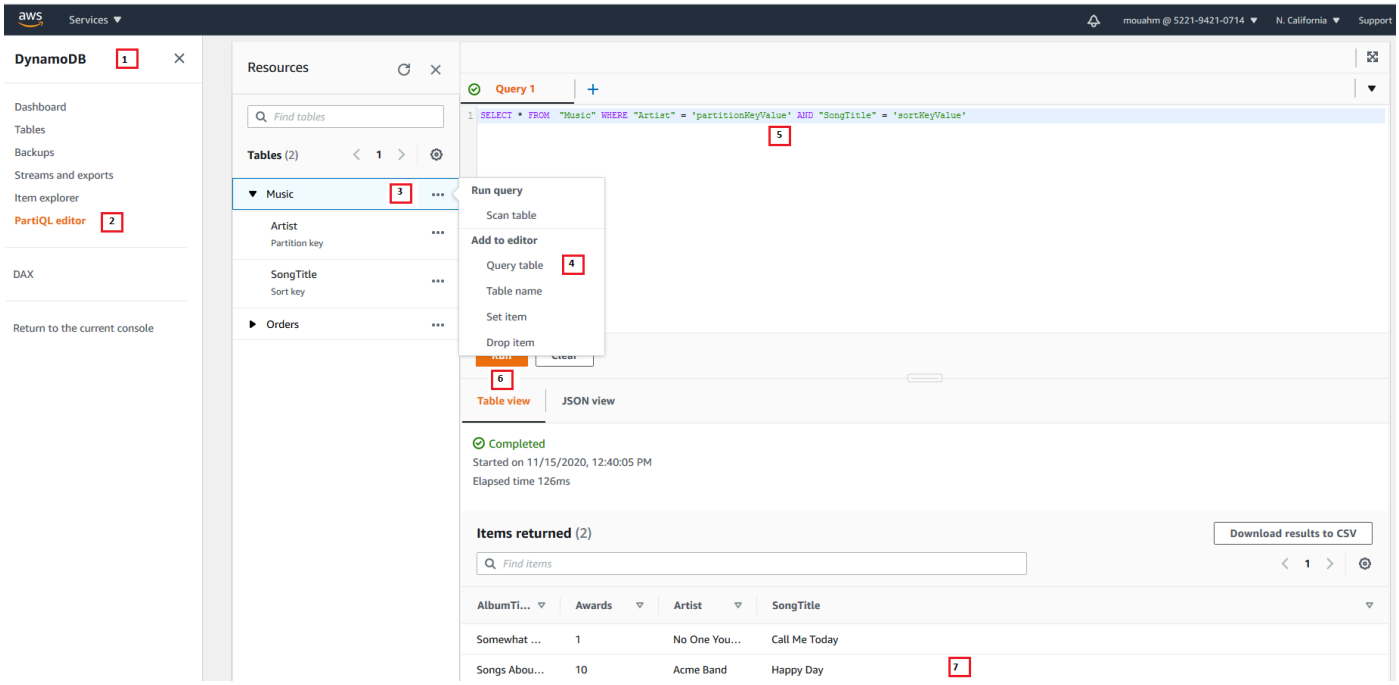
In questa sezione viene descritto come utilizzare PartiQL per DynamoDB dalla console Amazon DynamoDB, da AWS Command Line Interface (AWS CLI) e dalle API DynamoDB.

Nei seguenti esempi, la tabella DynamoDB definita nel tutorial Nozioni di base su DynamoDB è un prerequisito.

Per informazioni sull'utilizzo della console DynamoDB, della AWS Command Line Interface o delle API DynamoDB per accedere a DynamoDB, consulta Accesso a DynamoDB.

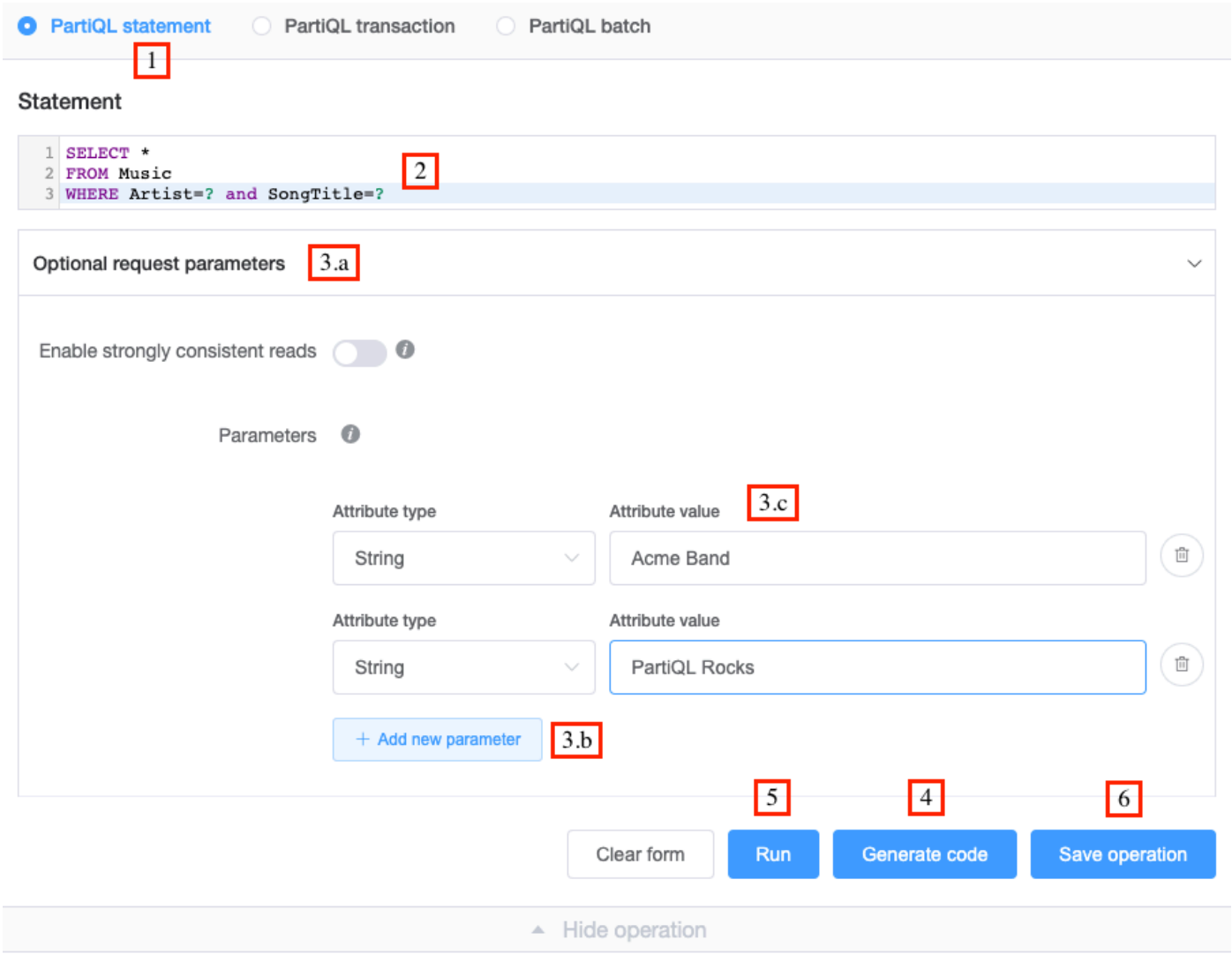
Per scaricare e utilizzare NoSQL Workbench per creare istruzioni PartiQL per DynamoDB, seleziona Operazioni PartiQL nell'angolo in alto a destra dell'Operation builder di NoSQL Workbench per DynamoDB.

Console



1. Accedi alla console Console di gestione AWS e apri la console DynamoDB all'indirizzo https://console.aws.amazon.com/elasticache/.
2. Nel pannello di navigazione sul lato sinistro della console seleziona Editor PartiQL.
3. Seleziona la tabella Music.
4. Scegli Esegui query sulla tabella. Questa azione genera una query che non determinerà una scansione completa della tabella.
5. Sostituisci `partitionKeyValue` con il valore stringa `Acme Band`. Sostituisci `sortKeyValue` con il valore stringa `Happy Day`.
6. Scegli il pulsante Esegui.
7. È possibile visualizzare i risultati della query scegliendo i pulsanti Vista tabella o Vista JSON.

NoSQL workbench



1. Scegli Istruzione PartiQL.
2. Specifica la seguente istruzione SELECT (SELEZIONA) PartiQL.

```
SELECT *
FROM Music
WHERE Artist=? and SongTitle=?
```

3. Per specificare un valore per i parametri `Artist` e `SongTitle`:
   a. Scegli Parametri di richiesta facoltativi.
   b. Scegli Aggiungi nuovi parametri.
   c. Scegli il tipo di attributo string e il valore `Acme Band`.

d. Ripeti i passaggi b e c, quindi scegli il tipo string e il valore `PartiQL Rocks`.

4. Per generare codice, selezionare Generate code (Genera codice).

   Selezionare la lingua desiderata dalle schede visualizzate. È possibile copiare questo codice e utilizzarlo nell'applicazione.

5. Per eseguire l'operazione immediatamente, seleziona Esegui.

AWS CLI

1. Crea un elemento nella tabella `Music` utilizzando l'istruzione INSERT (INSERISCI) PartiQL.

```
aws dynamodb execute-statement --statement "INSERT INTO Music  \
         VALUE  \
         {'Artist':'Acme Band','SongTitle':'PartiQL Rocks'}"
```

2. Recupera un elemento dalla tabella Music utilizzando l'istruzione SELECT (SELEZIONA) PartiQL.

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

3. Aggiorna un elemento nella tabella `Music` utilizzando l'istruzione UPDATE (AGGIORNA) PartiQL.

```
aws dynamodb execute-statement --statement "UPDATE Music  \
                                            SET AwardsWon=1  \
                                            SET AwardDetail={'Grammys':[2020,
 2018]}  \
                                            WHERE Artist='Acme Band' AND
 SongTitle='PartiQL Rocks'"
```

   Aggiungi un valore di elenco per un elemento nella tabella `Music`.

```
aws dynamodb execute-statement --statement "UPDATE Music  \
                                            SET AwardDetail.Grammys
 =list_append(AwardDetail.Grammys,[2016])  \
                                            WHERE Artist='Acme Band' AND
 SongTitle='PartiQL Rocks'"
```

Rimuovi un valore di elenco per un elemento nella tabella `Music`.

```
aws dynamodb execute-statement --statement "UPDATE Music  \
                                            REMOVE AwardDetail.Grammys[2]  \
                                            WHERE Artist='Acme Band' AND
 SongTitle='PartiQL Rocks'"
```

Aggiungi un nuovo membro della mappa per un elemento nella tabella `Music`.

```
aws dynamodb execute-statement --statement "UPDATE Music  \
                                            SET AwardDetail.BillBoard=[2020]  \
                                            WHERE Artist='Acme Band' AND
 SongTitle='PartiQL Rocks'"
```

Aggiungi un nuovo attributo del set di stringhe per un elemento nella tabella `Music`.

```
aws dynamodb execute-statement --statement "UPDATE Music  \
                                            SET BandMembers =<<'member1',
 'member2'>>  \
                                            WHERE Artist='Acme Band' AND
 SongTitle='PartiQL Rocks'"
```

Aggiungi un attributo del set di stringhe per un elemento nella tabella `Music`.

```
aws dynamodb execute-statement --statement "UPDATE Music  \
                                            SET BandMembers
 =set_add(BandMembers, <<'newmember'>>)  \
                                            WHERE Artist='Acme Band' AND
 SongTitle='PartiQL Rocks'"
```

4. Elimina un elemento dalla tabella `Music` utilizzando l'istruzione DELETE (ELIMINA) 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());
        }
    }

}
```

Tipi di dati PartiQL per DynamoDB

Nella tabella seguente sono elencati i tipi di dati che è possibile utilizzare con PartiQL per DynamoDB.

Tipo di dati DynamoDB	Rappresentazione PartiQL	Note
`Boolean`	TRUE | FALSE	Non prevede una distinzione tra lettere maiuscole e minuscole.
`Binary`	N/D	Supportato solo tramite codice.
`List`	[value1, value2,...]	Non vi sono limitazioni sui tipi di dati che possono essere memorizzati in un tipo List e gli elementi in un List non devono essere dello stesso tipo.
`Map`	{ 'name' : value }	Non vi sono limitazioni sui tipi di dati che possono essere memorizzati in un elemento Map e gli elementi in un Map non devono essere dello stesso tipo.
`Null`	NULL	Non prevede una distinzione tra lettere maiuscole e minuscole.
`Number`	1, 1.0, 1e0	I numeri possono essere positivi, negativi o zero. I numeri possono avere una precisione fino a 38 cifre.

Tipo di dati DynamoDB	Rappresentazione PartiQL	Note
`Number Set`	<<number1, number2>>	Gli elementi di un set di numeri devono essere di tipo Number.
`String Set`	<<'string1', 'string2'>>	Gli elementi di un set di stringhe devono essere di tipo String.
`String`	'string value'	Per specificare i valori String devono essere utilizzate le virgolette singole.

Esempi

Nell'istruzione seguente viene illustrato come inserire i seguenti tipi di dati: `String`, `Number`, `Map`, `List`, `Number Set` e `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 seguente istruzione illustra come inserire nuovi elementi nei tipi `Map`, `List`, `Number Set` e `String Set` e come modificare il valore di 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 seguente istruzione illustra come rimuovere gli elementi dai tipi `Map`, `List`, `Number Set` e `String Set` e come modificare il valore di 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'
```

Per ulteriori informazioni, consulta Tipi di dati di DynamoDB.

Istruzioni PartiQL per DynamoDB

Amazon DynamoDB supporta le seguenti istruzioni PartiQL.

> **Note**
>
> DynamoDB non supporta tutte le istruzioni PartiQL.
> Questo riferimento fornisce esempi di sintassi di base ed esempi di utilizzo delle istruzioni PartiQL eseguite manualmente utilizzando AWS CLI o. APIs

Il linguaggio di manipolazione dei dati (DML, Data Manipulation Language) è l'insieme di istruzioni PartiQL utilizzate per gestire i dati nelle tabelle DynamoDB. Le istruzioni DML vengono utilizzate per aggiungere, modificare o eliminare i dati in una tabella.

Sono supportate le seguenti istruzioni DML e del linguaggio di query:

- Istruzioni SELECT PartiQL per DynamoDB
- Istruzioni UPDATE PartiQL per DynamoDB
- Istruzioni INSERT PartiQL per DynamoDB
- Istruzioni DELETE PartiQL per DynamoDB

Anche Esecuzione di transazioni con PartiQL per DynamoDB e Esecuzione di operazioni in batch con PartiQL per DynamoDB sono supportati da PartiQL per DynamoDB.

Istruzioni SELECT PartiQL per DynamoDB

Utilizza l'istruzione `SELECT` per recuperare i dati da una tabella in Amazon DynamoDB.

L'uso di un'istruzione `SELECT` può comportare una scansione completa della tabella se non si fornisce una condizione di uguaglianza o IN con una chiave di partizione nella clausola WHERE. L'operazione di scansione esamina ogni elemento per i valori richiesti e può utilizzare la velocità effettiva assegnata per una tabella o un indice di grandi dimensioni in un'unica operazione.

Se desideri evitare la scansione completa della tabella in PartiQL, è possibile:

- Crea le tue istruzioni `SELECT` per non provocare scansioni complete della tabella assicurandoti che la condizione della clausola WHERE (DOVE) sia configurata di conseguenza.
- Disabilitare le scansioni di tabelle complete utilizzando la policy IAM specificata in Esempio: Consentire le istruzioni Select e rifiutare le istruzioni di scansione completa della tabella in PartiQL per DynamoDB, nella Guida per gli sviluppatori di DynamoDB.

Per ulteriori informazioni, consulta la sezione Best practice per eseguire query e scansioni di dati nella Guida per gli sviluppatori di DynamoDB.

Argomenti

- Sintassi
- Parameters
- Esempi

Sintassi

```
SELECT expression  [, ...]
FROM table[.index]
[ WHERE condition ] [ [ORDER BY key [DESC|ASC] , ...]
```

Parameters

expression

(Obbligatorio) Una proiezione formata dal carattere jolly * o un elenco di proiezione di uno o più nomi di attributi o percorsi di documento dal set di risultati. Un'espressione può essere costituita

da chiamate a Utilizzo delle funzioni PartiQL con DynamoDB o da campi modificati da Operatori PartiQL aritmetici, di confronto e logici per DynamoDB .

table

(Obbligatorio) Il nome della tabella per cui eseguire la query.

index

(Facoltativo) Il nome dell'indice su cui eseguire una query.

> **Note**
>
> È necessario aggiungere virgolette doppie al nome della tabella e al nome dell'indice quando si esegue una query su un indice.
>
> ```
> SELECT *
> FROM "TableName"."IndexName"
> ```

condition

(Facoltativo) I criteri di selezione per la query.

> **Important**
>
> Per garantire che una istruzione `SELECT` non comporti una scansione completa della tabella, la condizione della clausola `WHERE` deve specificare una chiave di partizione. Utilizza l'operatore di uguaglianza o IN.
> Ad esempio, in presenza di una tabella `Orders` con una chiave di partizione `OrderID` e altri attributi non chiave, tra cui `Address`, le istruzioni seguenti non comportano una scansione completa della tabella:
>
> ```
> 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]
```

Le seguenti istruzioni SELECT, tuttavia, provocheranno una scansione completa della tabella:

```
SELECT *
FROM "Orders"
WHERE OrderID > 1

SELECT *
FROM "Orders"
WHERE Address='some address'

SELECT *
FROM "Orders"
WHERE OrderID = 100 OR Address='some address'
```

key

(Facoltativo) Una chiave hash o una chiave di ordinamento da utilizzare per ordinare i risultati restituiti. L'ordine di default è crescente (ASC); specifica DESC se desideri che i risultati vengano rigenerati in ordine decrescente.

> **Note**
>
> Se si omette la clausola WHERE, saranno recuperati tutti gli elementi della tabella.

Esempi

La seguente query restituisce un elemento, se esistente, dalla tabella Orders specificando la chiave di partizione, OrderID, e utilizzando l'operatore di uguaglianza.

```
SELECT OrderID, Total
```

```
FROM "Orders"
WHERE OrderID = 1
```

La seguente query restituisce tutti gli elementi nella tabella `Orders` che hanno una chiave di partizione specifica, `OrderID`, i valori che utilizzano l'operatore OR.

```
SELECT OrderID, Total
FROM "Orders"
WHERE OrderID = 1 OR OrderID = 2
```

La seguente query restituisce tutti gli elementi nella tabella `Orders` che hanno una chiave di partizione specifica, `OrderID`, i valori che utilizzano l'operatore IN. I risultati restituiti sono in ordine decrescente, in base al valore dell'attributo della chiave `OrderID`.

```
SELECT OrderID, Total
FROM "Orders"
WHERE OrderID IN [1, 2, 3] ORDER BY OrderID DESC
```

La seguente query mostra una scansione completa della tabella che restituisce tutti gli elementi dalla tabella `Orders` che hanno un `Total` maggiore di 500, dove `Total` è un attributo non chiave.

```
SELECT OrderID, Total
FROM "Orders"
WHERE Total > 500
```

La seguente query mostra una scansione completa della tabella che restituisce tutti gli elementi dalla tabella `Orders` in un intervallo di ordinamento `Total` specifico, utilizzando l'operatore IN e un attributo non chiave `Total`.

```
SELECT OrderID, Total
FROM "Orders"
WHERE Total IN [500, 600]
```

La seguente query mostra una scansione completa della tabella che restituisce tutti gli elementi dalla tabella `Orders` in un intervallo di ordinamento `Total` specifico, utilizzando l'operatore BETWEEN e un attributo non chiave `Total`.

```
SELECT OrderID, Total
```

```
FROM "Orders"
WHERE Total BETWEEN 500 AND 600
```

La seguente query restituisce la prima data in cui è stato utilizzato un dispositivo firestick specificando la chiave di partizione `CustomerID` e la chiave di ordinamento `MovieID` nella condizione clausola WHERE (DOVE) e utilizzando percorsi documento nella clausola SELECT (SELEZIONA).

```
SELECT Devices.FireStick.DateWatched[0]
FROM WatchList
WHERE CustomerID= 'C1' AND MovieID= 'M1'
```

La seguente query mostra una scansione completa della tabella che restituisce l'elenco degli elementi in cui un dispositivo firestick è stato utilizzato per la prima volta dopo il 12/12/19 utilizzando percorsi di documento nella condizione della clausola WHERE (DOVE).

```
SELECT Devices
FROM WatchList
WHERE Devices.FireStick.DateWatched[0] >= '12/24/19'
```

Istruzioni UPDATE PartiQL per DynamoDB

Utilizza l'istruzione `UPDATE` per modificare il valore di uno o più attributi all'interno di un elemento in una tabella Amazon DynamoDB.

> **Note**
>
> È possibile aggiornare solo un elemento alla volta; non è possibile emettere una singola istruzione PartiQL DynamoDB che aggiorna più elementi. Per informazioni sull'aggiornamento di più elementi, consulta Esecuzione di transazioni con PartiQL per DynamoDB o Esecuzione di operazioni in batch con PartiQL per DynamoDB.

Argomenti

- Sintassi
- Parameters
- Valore restituito
- Esempi

Sintassi

```
UPDATE  table
[SET | REMOVE]  path  [=  data] […]
WHERE condition [RETURNING returnvalues]
<returnvalues>  ::= [ALL OLD | MODIFIED OLD | ALL NEW | MODIFIED NEW] *
```

Parameters

table

(Obbligatorio) La tabella contenente i dati da modificare.

path

(Obbligatorio) Un nome attributo o un percorso di documento da creare o modificare.

data

(Obbligatorio) Un valore di attributo o il risultato di un'operazione.

Le operazioni supportate da utilizzare con SET sono:

- LIST_APPEND: aggiunge un valore a un tipo di elenco.
- SET_ADD: aggiunge un valore a un numero o un set di stringhe.
- SET_DELETE: rimuove un valore da un numero o un set di stringhe.

condition

(Obbligatorio) I criteri di selezione per l'elemento da modificare. Questa condizione deve essere risolta in un singolo valore di chiave primaria.

returnvalues

(Facoltativo) Utilizza `returnvalues` se desideri ottenere gli attributi dell'elemento come appaiono prima o dopo l'aggiornamento. I valori validi sono:

- `ALL OLD *`: restituisce tutti gli attributi dell'elemento come apparivano prima dell'operazione di aggiornamento.
- `MODIFIED OLD *`: restituisce solo gli attributi aggiornati come apparivano prima dell'operazione di aggiornamento.
- `ALL NEW *`: restituisce tutti gli attributi dell'elemento come appaiono dopo l'operazione di aggiornamento.

- `MODIFIED NEW *`: restituisce solo gli attributi aggiornati come appaiono dopo l'operazione `UpdateItem`.

Valore restituito

Questa istruzione non restituisce un valore a meno che non sia stato specificato il parametro `returnvalues`.

> **Note**
>
> Se la clausola WHERE (DOVE) dell'istruzione UPDATE (AGGIORNA) non restituisce true per alcun elemento nella tabella DynamoDB, viene restituito `ConditionalCheckFailedException`.

Esempi

Aggiorna un valore dell'attributo in un elemento esistente. Se l'attributo non esiste, ne verrà creato uno.

La seguente query aggiorna un elemento nella tabella `"Music"` aggiungendo un attributo di tipo number (`AwardsWon`) e un attributo di tipo map (`AwardDetail`).

```
UPDATE "Music"
SET AwardsWon=1
SET AwardDetail={'Grammys':[2020, 2018]}
WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'
```

È possibile aggiungere `RETURNING ALL OLD *` per riportare gli attributi così come apparivano prima dell'operazione `Update`.

```
UPDATE "Music"
SET AwardsWon=1
SET AwardDetail={'Grammys':[2020, 2018]}
WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'
RETURNING ALL OLD *
```

Restituisce quanto segue:

```
{
```

```
    "Items": [
        {
            "Artist": {
                "S": "Acme Band"
            },
            "SongTitle": {
                "S": "PartiQL Rocks"
            }
        }
    ]
}
```

È possibile aggiungere `RETURNING ALL NEW *` per riportare gli attributi così come apparivano dopo l'operazione `Update`.

```
UPDATE "Music"
SET AwardsWon=1
SET AwardDetail={'Grammys':[2020, 2018]}
WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'
RETURNING ALL NEW *
```

Restituisce quanto segue:

```
{
    "Items": [
        {
            "AwardDetail": {
                "M": {
                    "Grammys": {
                        "L": [
                            {
                                "N": "2020"
                            },
                            {
                                "N": "2018"
                            }
                        ]
                    }
                }
            },
            "AwardsWon": {
                "N": "1"
            }
```

```
        }
    ]
}
```

La seguente query aggiorna un elemento nella tabella "Music" aggiungendolo a un elenco AwardDetail.Grammys.

```
UPDATE "Music"
SET AwardDetail.Grammys =list_append(AwardDetail.Grammys,[2016])
WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'
```

La seguente query aggiorna un elemento nella tabella "Music" rimuovendolo da un elenco AwardDetail.Grammys.

```
UPDATE "Music"
REMOVE AwardDetail.Grammys[2]
WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'
```

La seguente query aggiorna un elemento nella tabella "Music" aggiungendo BillBoard alla mappa AwardDetail.

```
UPDATE "Music"
SET AwardDetail.BillBoard=[2020]
WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'
```

La seguente query aggiorna un elemento nella tabella "Music" aggiungendo l'attributo del set di stringhe BandMembers.

```
UPDATE "Music"
SET BandMembers =<<'member1', 'member2'>>
WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'
```

La seguente query aggiorna un elemento nella tabella "Music" aggiungendo newbandmember al set di stringhe BandMembers.

```
UPDATE "Music"
SET BandMembers =set_add(BandMembers, <<'newbandmember'>>)
WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'
```

Istruzioni DELETE PartiQL per DynamoDB

Utilizza l'istruzione `DELETE` per eliminare un elemento esistente dalla tabella Amazon DynamoDB.

> **Note**
>
> È possibile eliminare un solo elemento alla volta. Non è possibile emettere una singola istruzione DynamoDB PartiQL che elimina più elementi. Per informazioni sull'aggiornamento di più elementi, consulta Esecuzione di transazioni con PartiQL per DynamoDB o Esecuzione di operazioni in batch con PartiQL per DynamoDB.

Argomenti

- Sintassi
- Parameters
- Valore restituito
- Esempi

Sintassi

```
DELETE FROM table
 WHERE condition [RETURNING returnvalues]
 <returnvalues>  ::= ALL OLD *
```

Parameters

`table`

(Obbligatorio) La tabella DynamoDB contenente l'elemento da eliminare.

`condition`

(Obbligatorio) I criteri di selezione per l'elemento da eliminare. Questa condizione deve essere risolta in un singolo valore di chiave primaria.

`returnvalues`

(Facoltativo) Utilizza `returnvalues` se desideri ottenere gli attributi dell'elemento come apparivano prima dell'eliminazione. I valori validi sono:

- `ALL OLD *`: viene restituito il contenuto del vecchio elemento.

Valore restituito

Questa istruzione non restituisce un valore a meno che non sia stato specificato il parametro `returnvalues`.

> **Note**
>
> Se la tabella DynamoDB non dispone di alcun elemento con la stessa chiave primaria di quella dell'elemento per il quale viene emessa l'istruzione DELETE (ELIMINA), viene restituito SUCCESS (ESITO POSITIVO) con 0 elementi eliminati. Se la tabella ha un elemento con la stessa chiave primaria, ma la condizione nella clausola WHERE (DOVE) dell'istruzione DELETE (ELIMINA) restituisce valore false, viene restituito il parametro `ConditionalCheckFailedException`.

Esempi

La query seguente elimina un elemento nella tabella `"Music"`.

```
DELETE FROM "Music" WHERE "Artist" = 'Acme Band' AND "SongTitle" = 'PartiQL Rocks'
```

È possibile aggiungere il parametro `RETURNING ALL OLD *` per restituire i dati che sono stati eliminati.

```
DELETE FROM "Music" WHERE "Artist" = 'Acme Band' AND "SongTitle" = 'PartiQL Rocks'
 RETURNING ALL OLD *
```

L'istruzione `Delete` ora restituisce quanto segue:

```
{
    "Items": [
        {
            "Artist": {
                "S": "Acme Band"
            },
            "SongTitle": {
                "S": "PartiQL Rocks"
            }
        }
    ]
```

```
}
```

Istruzioni INSERT PartiQL per DynamoDB

Utilizza l'istruzione `INSERT` per aggiungere un elemento a una tabella in Amazon DynamoDB.

> **Note**
>
> È possibile inserire un solo elemento alla volta; non è possibile emettere una singola istruzione PartiQL DynamoDB che inserisce più elementi. Per informazioni sull'inserimento di più elementi, consulta Esecuzione di transazioni con PartiQL per DynamoDB o Esecuzione di operazioni in batch con PartiQL per DynamoDB.

Argomenti

- Sintassi
- Parameters
- Valore restituito
- Esempi

Sintassi

Inserisci un singolo elemento.

```
INSERT INTO table VALUE item
```

Parameters

table

(Obbligatorio) La tabella in cui si desidera inserire i dati. La tabella deve già essere presente.

item

(Obbligatorio) Un elemento DynamoDB valido rappresentato come tupla PartiQL. È necessario specificare solo un elemento e tenere presente che ciascun nome di attributo nell'elemento fa distinzione tra maiuscole e minuscole e può essere riportato con virgolette singole (`'...'`) in PartiQL.

Anche i valori stringa sono riportati con virgolette singole (`'...'`) in PartiQL.

Valore restituito

Questa istruzione non restituisce alcun valore.

> **Note**
>
> Se nella tabella DynamoDB è già presente un elemento con la stessa chiave primaria della chiave primaria dell'elemento da inserire, viene restituito `DuplicateItemException`.

Esempi

```
INSERT INTO "Music" value {'Artist' : 'Acme Band','SongTitle' : 'PartiQL Rocks'}
```

Utilizzo delle funzioni PartiQL con DynamoDB

PartiQL in Amazon DynamoDB supporta le seguenti varianti incorporate di funzioni standard SQL.

> **Note**
>
> Tutte le funzioni SQL non incluse in questo elenco non sono attualmente supportate in DynamoDB.

Funzioni di aggregazione

- Utilizzo della funzione SIZE con PartiQL per Amazon DynamoDB

Funzioni condizionali

- Utilizzo della funzione EXISTS con PartiQL per DynamoDB
- Utilizzo della funzione ATTRIBUTE_TYPE con PartiQL per DynamoDB
- Utilizzo della funzione BEGINS_WITH con PartiQL per DynamoDB
- Utilizzo della funzione CONTAINS con PartiQL per DynamoDB
- Utilizzo della funzione MISSING con PartiQL per DynamoDB

Utilizzo della funzione EXISTS con PartiQL per DynamoDB

È possibile utilizzare EXISTS (ESISTE) per eseguire la stessa funzione che `ConditionCheck` esegue nell'API TransactWriteItems. La funzione EXISTS (ESISTE) può essere utilizzata solo nelle transazioni.

Dato un valore, restituisce `TRUE` se il valore è una raccolta non vuota. In caso contrario, restituisce `FALSE`.

> **Note**
>
> Questa funzione può essere utilizzata solo nelle operazioni transazionali.

Sintassi

```
EXISTS ( statement )
```

Argomenti

Istruzione

(Obbligatorio) L'istruzione SELECT (SELEZIONA) valutata dalla funzione.

> **Note**
>
> L'istruzione SELECT (SELEZIONA) deve specificare una chiave primaria completa e un'altra condizione.

Tipo restituito

`bool`

Esempi

```
EXISTS(
    SELECT * FROM "Music"
    WHERE "Artist" = 'Acme Band' AND "SongTitle" = 'PartiQL Rocks')
```

Utilizzo della funzione BEGINS_WITH con PartiQL per DynamoDB

Restituisce TRUE se l'attributo specificato inizia con una particolare sottostringa.

Sintassi

```
begins_with(path, value )
```

Argomenti

path

(Obbligatorio) Il nome attributo o il percorso del documento da utilizzare.

value

(Obbligatorio) La stringa da cercare.

Tipo restituito

bool

Esempi

```
SELECT * FROM "Orders" WHERE "OrderID"=1 AND begins_with("Address", '7834 24th')
```

Utilizzo della funzione MISSING con PartiQL per DynamoDB

Restituisce TRUE se l'elemento non contiene l'attributo specificato. Con questa funzione possono essere utilizzati solo gli operatori di uguaglianza e disuguaglianza.

Sintassi

```
 attributename IS | IS NOT  MISSING
```

Argomenti

attributename

(Obbligatorio) Il nome dell'attributo da utilizzare.

Tipo restituito

`bool`

Esempi

```
SELECT * FROM Music WHERE "Awards" is MISSING
```

Utilizzo della funzione ATTRIBUTE_TYPE con PartiQL per DynamoDB

Restituisce TRUE se l'attributo del percorso specificato è di un particolare tipo di dati.

Sintassi

```
attribute_type( attributename, type )
```

Argomenti

attributename

(Obbligatorio) Il nome dell'attributo da utilizzare.

tipo

(Obbligatorio) Il tipo di attributo da controllare. Per visualizzare un elenco di valori validi, consulta attribute_type di DynamoDB.

Tipo restituito

`bool`

Esempi

```
SELECT * FROM "Music" WHERE attribute_type("Artist", 'S')
```

Utilizzo della funzione CONTAINS con PartiQL per DynamoDB

Restituisce TRUE se l'attributo specificato dal percorso è uno dei seguenti:

- Una stringa che contiene una particolare sottostringa.

- Un set che contiene un particolare determinato elemento.

Per ulteriori informazioni, consulta la funzione contains di DynamoDB.

Sintassi

```
contains( path, substring )
```

Argomenti

path

(Obbligatorio) Il nome attributo o il percorso del documento da utilizzare.

sottostringa

(Obbligatorio) La sottostringa dell'attributo o il membro del set da controllare. Per ulteriori informazioni, consulta la funzione contains di DynamoDB.

Tipo restituito

`bool`

Esempi

```
SELECT * FROM "Orders" WHERE "OrderID"=1 AND contains("Address", 'Kirkland')
```

Utilizzo della funzione SIZE con PartiQL per Amazon DynamoDB

Restituisce un numero che rappresenta le dimensioni di un attributo (in byte). I seguenti sono tipi di dati validi per l'utilizzo con size. Per ulteriori informazioni, consulta la funzione size di DynamoDB.

Sintassi

```
size( path)
```

Argomenti

path

(Obbligatorio) Il nome attributo o il percorso del documento.

Per i tipi supportati, consulta la funzione size di DynamoDB.

Tipo restituito

`int`

Esempi

```
SELECT * FROM "Orders" WHERE "OrderID"=1 AND size("Image") >300
```

Operatori PartiQL aritmetici, di confronto e logici per DynamoDB

PartiQL in Amazon DynamoDB supporta i seguenti operatori SQL standard.

> **Note**
>
> Tutti gli operatori SQL che non sono inclusi in questo elenco non sono attualmente supportati in DynamoDB.

Operatori aritmetici

Operatore	Descrizione
+	Add (Aggiungi)
-	Subtract

Operatori di confronto

Operatore	Descrizione
=	Equal to
<>	Not Equal to
!=	Not Equal to
>	Greater than

Operatore	Descrizione
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

Operatori logici

Operatore	Descrizione
`AND`	`TRUE` se tutte le condizioni separate da `AND` sono `TRUE`
`BETWEEN`	`TRUE` se l'operando è all'interno dell'intervallo di confronti. Questo operatore include il limite inferiore e superiore degli operandi su cui viene applicato.
`IN`	`TRUE` se l'operando è uguale a uno di un elenco di espressioni (con un massimo di 50 valori di attributo hash o a un massimo di 100 valori di attributo non chiave)
`IS`	`TRUE` se l'operando è un dato tipo di dati PartiQL, incluso`NULL` o `MISSING`
`NOT`	Inverte il valore di una determinata espressione booleana
`OR`	`TRUE` se una delle condizioni separate da `OR` è `TRUE`

Per ulteriori informazioni sull'utilizzo degli operatori logici, consulta Realizzazione di confronti e Valutazioni logiche.

Esecuzione di transazioni con PartiQL per DynamoDB

In questa sezione viene descritto come utilizzare le transazioni con PartiQL per DynamoDB. Le transazioni PartiQL sono limitate a 100 istruzioni totali (operazioni).

Per ulteriori informazioni sulle transazioni DynamoDB, consulta Gestione di flussi di lavoro complessi con transazioni DynamoDB.

> **Note**
>
> L'intera transazione deve essere costituita da istruzioni di lettura o di scrittura. Non puoi combinare entrambe in un'unica transazione. La funzione EXISTS è un'eccezione. È possibile utilizzarlo per verificare la condizione degli attributi specifici dell'elemento in modo simile al funzionamento `ConditionCheck` dell'TransactWriteItemsAPI.

Argomenti

- Sintassi
- Parameters
- Valori restituiti
- Esempi

Sintassi

```
[
   {
      "Statement":" statement ",
      "Parameters":[
         {
            " parametertype " : " parametervalue "
         }, ...]
   } , ...
]
```

Parameters

statement

(Obbligatorio) Un'istruzione supportata PartiQL per DynamoDB.

> **Note**
>
> L'intera transazione deve essere costituita da istruzioni di lettura o di scrittura. Non puoi combinare entrambe in un'unica transazione.

parametertype

(Facoltativo) Un tipo DynamoDB, se durante la specifica dell'istruzione PartiQL sono stati utilizzati dei parametri.

parametervalue

(Facoltativo) Il valore di un parametro se durante la specifica dell'istruzione PartiQL sono stati utilizzati dei parametri.

Valori restituiti

Questa istruzione non restituisce alcun valore per le operazioni di scrittura (INSERT, UPDATE o DELETE). Tuttavia, restituisce valori diversi per le operazioni di lettura (SELECT) in base alle condizioni specificate nella clausola WHERE.

> **Note**
>
> Se una qualsiasi delle singole operazioni INSERT, UPDATE o DELETE restituisce un errore, le transazioni verranno annullate con l'eccezione `TransactionCanceledException` e il codice motivo dell'annullamento includerà gli errori delle singole operazioni.

Esempi

L'esempio seguente esegue più istruzioni come transazione.

AWS CLI

1. Salva il codice JSON seguente in un file chiamato 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. Esegui il comando seguente al prompt dei comandi.

```
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());
        }
    }

}
```

L'esempio seguente mostra i diversi valori restituiti quando DynamoDB legge elementi con condizioni diverse specificate nella clausola WHERE.

AWS CLI

1. Salva il codice JSON seguente in un file chiamato 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. Esegui il comando seguente in un prompt dei comandi.

```
aws dynamodb execute-transaction --transact-statements  file://partiql.json
```

3. Viene restituita la risposta seguente:

```
{
    "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
        {}
    ]
}
```

Esecuzione di operazioni in batch con PartiQL per DynamoDB

In questa sezione viene descritto come utilizzare le istruzioni in batch con PartiQL per DynamoDB.

> **Note**
>
> - L'intera transazione deve essere costituita da istruzioni di lettura o di scrittura; non è possibile combinare entrambe le istruzioni in un batch.
> - `BatchExecuteStatement` e `BatchWriteItem` possono eseguire al massimo 25 istruzioni per batch.
> - `BatchExecuteStatement` utilizza `BatchGetItem`, che richiede un elenco di chiavi primarie in istruzioni separate.

Argomenti

- Sintassi
- Parameters
- Esempi

Sintassi

```
[
  {
    "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

(Obbligatorio) Un'istruzione supportata PartiQL per DynamoDB.

> **Note**
>
> - L'intera transazione deve essere costituita da istruzioni di lettura o di scrittura; non è possibile combinare entrambe le istruzioni in un batch.
> - `BatchExecuteStatement` e `BatchWriteItem` possono eseguire al massimo 25 istruzioni per batch.

parametertype

(Facoltativo) Un tipo DynamoDB, se durante la specifica dell'istruzione PartiQL sono stati utilizzati dei parametri.

parametervalue

(Facoltativo) Il valore di un parametro se durante la specifica dell'istruzione PartiQL sono stati utilizzati dei parametri.

Esempi

AWS CLI

1. Salvataggio del JSON seguente in un file chiamato 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. Esegui il comando seguente al prompt dei comandi.

```
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());
        }
```

```
    }

}
```

Policy di sicurezza IAM con PartiQL per DynamoDB

Sono richieste le seguenti autorizzazioni:

- Per leggere gli elementi utilizzando PartiQL per DynamoDB, è necessario disporre dell'autorizzazione `dynamodb:PartiQLSelect` sulla tabella o sull'indice.
- Per inserire gli elementi utilizzando PartiQL per DynamoDB, è necessario disporre dell'autorizzazione `dynamodb:PartiQLInsert` sulla tabella o sull'indice.
- Per aggiornare gli elementi utilizzando PartiQL per DynamoDB, è necessario disporre dell'autorizzazione `dynamodb:PartiQLUpdate` sulla tabella o sull'indice.
- Per inserire gli elementi utilizzando PartiQL per DynamoDB, è necessario disporre dell'autorizzazione `dynamodb:PartiQLDelete` sulla tabella o sull'indice.

Esempio: consentire tutte le istruzioni PartiQL for DynamoDB () su una tabella Select/Insert/Update/Delete

La seguente policy IAM concede le autorizzazioni per eseguire tutte le istruzioni PartiQL per DynamoDB su una tabella specifica.

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"
```

```
         ]
      }
   ]
}
```

Esempio: Consentire le istruzioni SELECT (SELEZIONA) PartiQL per DynamoDB su una tabella

La seguente policy IAM concede le autorizzazioni per eseguire l'istruzione `select` su una tabella specifica.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "dynamodb:PartiQLSelect"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/Music"
         ]
      }
   ]
}
```

Esempio: Consentire le istruzioni INSERT PartiQL per DynamoDB su un indice

La seguente policy IAM concede le autorizzazioni per eseguire l'istruzione `insert` su un indice specifico.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
```

```
         "Action":[
            "dynamodb:PartiQLInsert"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/Music/index/index1"
         ]
      }
   ]
}
```

Esempio: Consentire le istruzioni transazionali PartiQL per DynamoDB solo su una tabella

La seguente policy IAM concede le autorizzazioni per eseguire le istruzioni transazionali su una tabella specifica.

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"
               ]
            }
         }
      }
   ]
}
```

Esempio: Consentire letture e scritture non transazionali di PartiQL per DynamoDB e bloccare le istruzioni transazionali di letture e scritture transazionali di PartiQL su una tabella.

La seguente policy IAM concede le autorizzazioni per eseguire letture e le scritture non transazionali PartiQL per DynamoDB bloccando le letture e le scritture transazionali PartiQL per 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"
         ]
      }
   ]
```

```
}
```

Esempio: Consentire le istruzioni Select e rifiutare le istruzioni di scansione completa della tabella in PartiQL per DynamoDB

La seguente policy IAM concede le autorizzazioni per eseguire l'istruzione `select` su una tabella specifica durante il blocco delle istruzioni `select` che determinano una scansione completa della tabella.

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"
         ]
      }
   ]
}
```

Utilizzo degli elementi: Java

È possibile utilizzare l'API documento AWS SDK per Java per eseguire operazioni tipiche di creazione, lettura, aggiornamento ed eliminazione (CRUD) sugli elementi Amazon DynamoDB in una tabella.

> **Note**
>
> L'SDK per Java offre inoltre un modello di persistenza degli oggetti che permette di mappare le classi lato client alle tabelle DynamoDB. Questo approccio può ridurre la quantità di codice da scrivere. Per ulteriori informazioni, consulta Java 1.x: DynamoDBMapper.

Questa sezione contiene esempi Java per eseguire diverse operazioni operazioni sugli elementi dell'API documento Java e diversi esempi di lavoro completi.

Argomenti

- Collocazione di un elemento
- Ottenimento di un elemento
- Scrittura in batch: collocazione ed eliminazione di più elementi
- Ricezione in batch: ricezione di più elementi
- Aggiornamento di un elemento
- Eliminazione di un elemento
- Esempio: operazioni CRUD utilizzando l'API del documento AWS SDK per Java
- Esempio: operazioni in batch utilizzando l'API del documento AWS SDK per Java
- Esempio: gestione degli attributi di tipo binario utilizzando l'API del documento AWS SDK per Java

Collocazione di un elemento

Il metodo `putItem` memorizza un item nella tabella. Se l'item esiste, lo sostituisce per intero. Se invece di sostituire l'intero item vuoi aggiornare solo attributi specifici, puoi utilizzare il metodo `updateItem`. Per ulteriori informazioni, consulta Aggiornamento di 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

Completare la procedura riportata di seguito.

1. Creare un'istanza della classe `DynamoDB`.
2. Crea un'istanza della classe `Table` per rappresentare la tabella con cui vuoi lavorare.
3. Crea un'istanza della classe `Item` per rappresentare il nuovo item. Devi specificare la chiave primaria del nuovo item e gli attributi.
4. Chiama il metodo `putItem` dell'oggetto `Table` usando l'elemento `Item` creato nella fase precedente.

Il seguente esempio di codice Java mostra le attività precedenti. Il codice scrive un nuovo item nella tabella `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);
```

Nell'esempio precedente, l'item ha attributi scalari (`String`, `Number`, `Boolean`, `Null`), set (`String Set`) e tipi di documento (`List`, `Map`).

Specifica dei parametri facoltativi

Insieme ai parametri richiesti, puoi anche specificare parametri opzionali per il metodo `putItem`. Ad esempio, il seguente esempio di codice Java utilizza un parametro facoltativo per specificare una condizione per il caricamento dell'item. Se la condizione specificata non viene soddisfatta, AWS SDK per Java genera un'eccezione `ConditionalCheckFailedException`. L'esempio di codice specifica i seguenti parametri facoltativi nel metodo `putItem`:

- un item `ConditionExpression` che definisce le condizioni della richiesta. Il codice definisce la condizione in cui l'item esistente che ha la stessa chiave primaria viene sostituito solo se ha un attributo ISBN uguale a un valore specifico.
- Una mappa per `ExpressionAttributeValues` che viene usata nella condizione. In questo caso, è necessaria una sola sostituzione: il segnaposto `:val` nell'espressione della condizione viene sostituito al runtime con il valore ISBN effettivo da verificare.

Nell'esempio seguente viene aggiunto un nuovo item del libro utilizzando questi parametri facoltativi.

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 e documenti JSON

È possibile memorizzare un documento JSON come attributo in una tabella DynamoDB. Per eseguire questa operazione, usa il metodo `withJSON` di `Item`. Questo metodo analizza il documento JSON e mappa ciascun elemento a un tipo di dati DynamoDB nativo.

Supponiamo di voler memorizzare il seguente documento JSON, contenente i fornitori in grado di soddisfare gli ordini di un determinato prodotto.

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"
        ]
    }
}
```

Puoi utilizzare il metodo `withJSON` per memorizzarlo nella tabella `ProductCatalog`, in un attributo `Map` denominato `VendorInfo`. Il seguente codice di esempio Java dimostra come eseguire questa operazione.

```
// 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);
```

Ottenimento di un elemento

Per recuperare un singolo item usa il metodo `getItem` di un oggetto `Table`. Completare la procedura riportata di seguito.

1. Creare un'istanza della classe `DynamoDB`.
2. Crea un'istanza della classe `Table` per rappresentare la tabella con cui vuoi lavorare.
3. Chiama il metodo `getItem` dell'istanza di `Table`. Devi specificare la chiave primaria dell'item che intendi recuperare.

Il seguente esempio di codice Java mostra le fasi precedenti. Il codice ottiene l'item che ha la chiave di partizione specificata.

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);

Table table = dynamoDB.getTable("ProductCatalog");
```

```
Item item = table.getItem("Id", 210);
```

Specifica dei parametri facoltativi

Insieme ai parametri richiesti, puoi anche specificare parametri opzionali per il metodo `getItem`. Ad esempio, il seguente esempio di codice Java utilizza un metodo facoltativo per recuperare un solo elenco specifico di attributi e per specificare letture fortemente consistenti. Per ulteriori informazioni sulla lettura consistente, consulta Coerenza di lettura di DynamoDB.

Puoi usare un `ProjectionExpression` per recuperare solo attributi o elementi specifici, piuttosto che un intero item. Un `ProjectionExpression` può specificare attributi nidificati o di primo livello usando i percorsi dei documenti. Per ulteriori informazioni, consulta Utilizzo di espressioni di proiezione in DynamoDB.

I parametri del metodo `getItem` non consentono di specificare lettura coerente. Tuttavia, puoi creare un `GetItemSpec`, che fornisce l'accesso completo a tutti gli input all'operazione `GetItem` di basso livello. Il seguente esempio di codice consente di creare un `GetItemSpec` e utilizzare tale specifica come input per il metodo `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());
```

Per stampare un `Item` in un formato leggibile, utilizza il metodo `toJSONPretty`. L'aspetto dell'output dall'esempio precedente è simile al seguente.

```
{
  "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 e documenti JSON

Nella sezione PutItem e documenti JSON, viene archiviato un documento JSON in un attributo `Map` denominato `VendorInfo`. Puoi utilizzare il metodo `getItem` per recuperare l'intero documento nel formato JSON. Oppure puoi utilizzare la notazione del percorso del documento per recuperare solo gli stessi elementi nel documento. Il seguente esempio di codice Java mostra queste tecniche.

```
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());
```

L'aspetto dell'output dall'esempio precedente è simile al seguente.

```
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**
>
> Puoi usare il metodo `toJSON` per convertire qualsiasi item (o gli attributi) in una stringa in formato JSON. Il seguente codice recupera diversi attributi di primo livello e nidificati e stampa i risultati come JSON.
>
> ```
> GetItemSpec spec = new GetItemSpec()
>     .withPrimaryKey("Id", 210)
>     .withProjectionExpression("VendorInfo.V01, Title, Price");
>
> Item item = table.getItem(spec);
> System.out.println(item.toJSON());
> ```
>
> L'output sarà simile al seguente.
>
> ```
> {"VendorInfo":{"V01":{"Name":"Acme Books","Offices":
> ["Seattle"]}},"Price":30,"Title":"Book 210 Title"}
> ```

Scrittura in batch: collocazione ed eliminazione di più elementi

La scrittura in batch fa riferimento alla collocazione e all'eliminazione di più item in un batch. Il metodo `batchWriteItem` ti consente di inserire ed eliminare più item da una o più tabelle con un'unica chiamata. Di seguito sono riportate le fasi per inserire o eliminare più item utilizzando l'API documento AWS SDK per Java.

1. Creare un'istanza della classe `DynamoDB`.
2. Crea un'istanza della classe `TableWriteItems` che descrive tutte le operazioni di inserimento ed eliminazione per una tabella. Se vuoi scrivere su più tabelle in una sola operazione di scrittura in batch, devi creare un'istanza di `TableWriteItems` per tabella.
3. Chiama il metodo `batchWriteItem` fornendo gli oggetti `TableWriteItems` creati nella fase precedente.
4. Elabora la risposta. Dovresti controllare se erano presenti item di richiesta non elaborati restituiti nella risposta. Questo potrebbe accadere se raggiungi la quota di throughput assegnata o si verifica un altro errore transitorio. Inoltre, DynamoDB limita la dimensione della richiesta e il numero di operazioni che possono essere specificate in una richiesta. Se si superano questi

limiti, DynamoDB rifiuta la richiesta. Per ulteriori informazioni, consulta Quote in Amazon DynamoDB.

Il seguente esempio di codice Java mostra le fasi precedenti. L'esempio esegue un'operazione `batchWriteItem` su due tabelle: `Forum` e `Thread`. Gli oggetti `TableWriteItems` corrispondenti definiscono le operazioni seguenti:

- inserimento di un item nella tabella `Forum`.
- Inserimento ed eliminazione di un item nella tabella `Thread`.

Il codice quindi chiama `batchWriteItem` per eseguire l'operazione.

```
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
```

Per un esempio di utilizzo, consulta Esempio: operazione di scrittura in batch utilizzando l'API del documento AWS SDK per Java.

Ricezione in batch: ricezione di più elementi

Il metodo `batchGetItem` ti consente di recuperare più item da una o più tabelle. Per recuperare un singolo item puoi usare il metodo `getItem`.

Completare la procedura riportata di seguito.

1. Creare un'istanza della classe `DynamoDB`.
2. Crea un'istanza della classe `TableKeysAndAttributes` e descrivi un elenco di valori delle chiavi primarie da recuperare da una tabella. Se vuoi leggere da più tabelle in una sola operazione Get in batch, devi creare un'istanza di `TableKeysAndAttributes` per tabella.
3. Chiama il metodo `batchGetItem` fornendo gli oggetti `TableKeysAndAttributes` creati nella fase precedente.

Il seguente esempio di codice Java mostra le fasi precedenti. L'esempio consente di recuperare due item dalla tabella `Forum` e tre item dalla tabella `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);
    }
}
```

Specifica dei parametri facoltativi

Insieme ai parametri richiesti, puoi anche specificare parametri opzionali usando `batchGetItem`. Puoi ad esempio fornire un `ProjectionExpression` con ogni `TableKeysAndAttributes` che definisci. Ciò ti consente di specificare gli attributi che vuoi recuperare dalla tabella.

L'esempio di codice seguente recupera due item dalla tabella `Forum`. Il parametro `withProjectionExpression` specifica che deve essere recuperato solo l'attributo `Threads`.

Example

```
TableKeysAndAttributes forumTableKeysAndAttributes = new
 TableKeysAndAttributes("Forum")
    .withProjectionExpression("Threads");

forumTableKeysAndAttributes.addHashOnlyPrimaryKeys("Name",
    "Amazon S3",
    "Amazon DynamoDB");

BatchGetItemOutcome outcome = dynamoDB.batchGetItem(forumTableKeysAndAttributes);
```

Aggiornamento di un elemento

Il metodo `updateItem` di un oggetto `Table` può aggiornare i valori degli attributi esistenti, aggiungere nuovi attributi o eliminare attributi da un item esistente.

Il metodo `updateItem` si comporta nel modo seguente:

- Se un item non esiste (nessun item nella tabella con la chiave primaria specificata), `updateItem` aggiunge un nuovo item alla tabella.
- Se un item esiste, `updateItem` esegue l'aggiornamento come specificato dal parametro `UpdateExpression`.

> **Note**
>
> È anche possibile "aggiornare" un item usando `putItem`. Ad esempio, se chiami `putItem` per aggiungere un item alla tabella ma esiste già un item con la chiave primaria specificata, `putItem` sostituisce l'intero item. Se nell'item esistente sono presenti attributi che non sono specificati nell'input, `putItem` rimuove quegli attributi dall'item.

> Come regola generale, ti consigliamo di usare `updateItem` ogni volta che desideri modificare qualsiasi attributo dell'item. Il metodo `updateItem` modifica solo gli attributi specificati nell'input e gli altri attributi dell'item rimangono invariati.

Completare la procedura riportata di seguito.

1. Crea un'istanza della classe `Table` per rappresentare la tabella da utilizzare.
2. Chiama il metodo `updateTable` dell'istanza di `Table`. Dovrai specificare la chiave primaria dell'item che vuoi recuperare, insieme a una `UpdateExpression` che descrive gli attributi da modificare e come modificarli.

Il seguente esempio di codice Java mostra le attività precedenti. Il codice aggiorna un item libro nella tabella `ProductCatalog`. Aggiunge un nuovo autore al set di `Authors` ed elimina l'attributo `ISBN` esistente. Inoltre, riduce il prezzo di una unità.

Una mappa `ExpressionAttributeValues` viene usata in `UpdateExpression`. I segnaposto `:val1` e `:val2` vengono sostituiti al runtime con i valori effettivi per `Authors` e `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);
```

Specifica dei parametri facoltativi

Insieme ai parametri obbligatori puoi specificare anche dei parametri facoltativi per il metodo `updateItem`, inclusa una condizione che deve essere soddisfatta affinché l'aggiornamento possa avere luogo. Se la condizione specificata non viene soddisfatta, AWS SDK per Java genera un'eccezione `ConditionalCheckFailedException`. Ad esempio, il seguente esempio di codice Java aggiorna in base a condizioni il prezzo di un libro a 25. Specifica una `ConditionExpression` che afferma che il prezzo deve essere aggiornato solo se il prezzo esistente è 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);
```

Contatore atomico

Puoi usare il metodo `updateItem` per implementare un contatore atomico e incrementare o decrementare il valore di un attributo esistente senza interferire con altre richieste di scrittura. Per incrementare un contatore atomico, usa un `UpdateExpression` con un'operazione `set` per aggiungere un valore numerico a un attributo esistente di tipo `Number`.

Il seguente esempio ne dimostra l'uso, incrementando l'attributo `Quantity` di uno. Dimostra anche l'uso del parametro `ExpressionAttributeNames` in una `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);
```

Eliminazione di un elemento

Il metodo `deleteItem` elimina un item da una tabella. È necessario fornire la chiave primaria dell'item che intendi eliminare.

Completare la procedura riportata di seguito.

1. Crea un'istanza del client `DynamoDB`.
2. Chiama il metodo `deleteItem` fornendo la chiave dell'item che desideri eliminare.

Il seguente esempio Java mostra queste attività.

Example

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);

Table table = dynamoDB.getTable("ProductCatalog");

DeleteItemOutcome outcome = table.deleteItem("Id", 101);
```

Specifica dei parametri facoltativi

Puoi specificare i parametri facoltativi per `deleteItem`. Ad esempio, il seguente esempio di codice Java specifica una `ConditionExpression`, in cui si afferma che un item libro in `InPublication` può essere eliminato solo se il libro non è più in pubblicazione (l'attributo `ProductCatalog` è 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);
```

Esempio: operazioni CRUD utilizzando l'API del documento AWS SDK per Java

Il seguente esempio di codice illustra le operazioni CRUD su un elemento di Amazon DynamoDB. L'esempio consente di creare un item, recuperarlo, eseguire vari aggiornamenti e infine eliminare l'item.

> **Note**
>
> L'SDK per Java offre inoltre un modello di persistenza degli oggetti che permette di mappare le classi lato client alle tabelle DynamoDB. Questo approccio può ridurre la quantità di codice da scrivere. Per ulteriori informazioni, consulta Java 1.x: DynamoDBMapper.

> **Note**
>
> In questo esempio di codice si presuppone che i dati siano già stati caricati in DynamoDB per l'account seguendo le istruzioni riportate nella sezione Creazione di tabelle e caricamento di dati per esempi di codice in DynamoDB.
> Per le istruzioni dettagliate per eseguire il seguente esempio, consultare Esempi di codice 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());
        }
    }
}
```

Esempio: operazioni in batch utilizzando l'API del documento AWS SDK per Java

In questa sezione vengono forniti esempi di operazioni get in batch e scrittura in batch in Amazon DynamoDB tramite l'API documento AWS SDK per Java.

> **Note**
>
> L'SDK per Java offre inoltre un modello di persistenza degli oggetti che permette di mappare le classi lato client alle tabelle DynamoDB. Questo approccio può ridurre la quantità di codice da scrivere. Per ulteriori informazioni, consulta Java 1.x: DynamoDBMapper.

Argomenti

- Esempio: operazione di scrittura in batch utilizzando l'API del documento AWS SDK per Java
- Esempio: operazione di ottenimento in batch utilizzando l'API del documento AWS SDK per Java

Esempio: operazione di scrittura in batch utilizzando l'API del documento AWS SDK per Java

Nel seguente esempio di codice Java viene utilizzato il metodo `batchWriteItem` per eseguire le seguenti operazioni di eliminazione e inserimento:

- inserimento di un item nella tabella `Forum`;
- Inserimento di un item ed eliminazione di un item dalla tabella `Thread`.

Quando crei la tua risposta di scrittura in batch, puoi specificare qualsiasi quantità di richieste di inserimento ed eliminazione in una o più tabelle. Tuttavia, `batchWriteItem` limita la dimensione di una richiesta di scrittura in batch e il numero di operazioni di inserimento ed eliminazione in una singola operazione di scrittura in batch. Se la tua richiesta eccede questi limiti, essa verrà rigettata. Se la tua tabella non dispone di sufficiente throughput assegnato per soddisfare questa richiesta, gli elementi di richiesta non eseguiti vengono restituiti nella risposta.

Il seguente esempio controlla la risposta nel caso in cui vi siano degli elementi di richiesta non eseguiti. Se sono presenti, esegue il loopback e invia nuovamente la richiesta `batchWriteItem` con gli elementi non elaborati della richiesta. Seguendo gli esempi presenti in questa guida dovrebbe essere già avvenuta la creazione delle tabelle `Forum` e `Thread`. Puoi anche creare queste tabelle e caricare i dati di esempio a livello di programmazione. Per ulteriori informazioni, consulta Creazione di tabelle di esempio e caricamento di dati utilizzando AWS SDK per Java.

Per le istruzioni dettagliate per eseguire il test del seguente esempio, consulta Esempi di codice 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);
        }

    }

}
```

Esempio: operazione di ottenimento in batch utilizzando l'API del documento AWS SDK per Java

Nel seguente esempio di codice Java si utilizza il metodo `batchGetItem` per recuperare più item dalle tabelle `Forum` e `Thread`. L'item `BatchGetItemRequest` specifica i nomi delle tabelle e un elenco di chiavi per ogni item da ottenere. Nell'esempio si esegue la risposta stampando gli elementi recuperati.

> **Note**
>
> In questo esempio di codice si presuppone che i dati siano già stati caricati in DynamoDB per l'account seguendo le istruzioni riportate nella sezione Creazione di tabelle e caricamento di dati per esempi di codice in DynamoDB.
> Per le istruzioni dettagliate per eseguire il seguente esempio, consultare Esempi di codice 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());
        }

    }

}
```

Esempio: gestione degli attributi di tipo binario utilizzando l'API del documento AWS SDK per Java

Il seguente esempio di codice Java illustra come gestire gli attributi di tipo binario. L'esempio aggiunge un item alla tabella `Reply`. L'item include un attributo di tipo binario (`ExtendedMessage`) che memorizza i dati compressi. L'esempio recupera quindi l'item e stampa tutti valori degli attributi. Nell'esempio si utilizza la classe `GZIPOutputStream` per comprimere un flusso di esempio e assegnarlo all'attributo `ExtendedMessage`. Quando l'attributo binario viene recuperato, viene decompresso utilizzando la classe `GZIPInputStream`.

> **Note**
>
> L'SDK per Java offre inoltre un modello di persistenza degli oggetti che permette di mappare le classi lato client alle tabelle DynamoDB. Questo approccio può ridurre la quantità di codice da scrivere. Per ulteriori informazioni, consulta Java 1.x: DynamoDBMapper.

Se hai seguito la sezione Creazione di tabelle e caricamento di dati per esempi di codice in DynamoDB dovresti aver già creato la tabella `Reply`. Puoi anche creare questa tabella a livello di programmazione. Per ulteriori informazioni, consulta Creazione di tabelle di esempio e caricamento di dati utilizzando AWS SDK per Java.

Per le istruzioni dettagliate per eseguire il test del seguente esempio, consulta Esempi di codice 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 di elementi: .NET

Puoi usare l'API di basso livello di AWS SDK per .NET per eseguire tipiche operazioni di creazione, lettura, aggiornamento ed eliminazione (CRUD) su un item in una tabella. Di seguito sono riportate le fasi comuni per eseguire le operazione CRUD sui dati, usando l'API di basso livello .NET:

1. Crea un'istanza della classe `AmazonDynamoDBClient` (client).
2. Fornisci i parametri obbligatori e specifici per l'operazione in un oggetto di richiesta corrispondente.

   Per esempio, usa l'oggetto di richiesta `PutItemRequest` durante il caricamento di un item e l'oggetto di richiesta `GetItemRequest` quando recuperi un item esistente.

   Puoi usare l'oggetto di richiesta per fornire sia i parametri obbligatori sia quelli facoltativi;
3. Eseguire il metodo appropriato fornito dal client inviando l'oggetto di richiesta creato nella fase precedente.

   Il client `AmazonDynamoDBClient` fornisce i metodi `PutItem`, `GetItem`, `UpdateItem` e `DeleteItem` per le operazioni CRUD.

Argomenti

- Collocazione di un elemento
- Ottenimento di un elemento
- Aggiornamento di un elemento
- Contatore atomico
- Eliminazione di un elemento
- Scrittura in batch: collocazione ed eliminazione di più elementi
- Ricezione in batch: ricezione di più elementi
- Esempio: operazioni CRUD utilizzando l'API di basso livello AWS SDK per .NET
- Esempio: operazioni batch utilizzando l'API di basso livello AWS SDK per .NET
- Esempio: gestione degli attributi di tipo binario utilizzando l'API di basso livello AWS SDK per .NET

Collocazione di un elemento

Il metodo `PutItem` carica un item nella tabella. Se l'item esiste, lo sostituisce per intero.

> **Note**
>
> Se invece di sostituire l'intero item vuoi aggiornare solo attributi specifici, puoi utilizzare il metodo `UpdateItem`. Per ulteriori informazioni, consulta Aggiornamento di un elemento.

Di seguito sono riportate le fasi per caricare un item usando l'API di basso livello dell'SDK per .NET:

1. Creare un'istanza della classe `AmazonDynamoDBClient`.
2. fornisci i parametri obbligatori creando un istanza della classe `PutItemRequest`.

   Per inserire un item, è necessario fornire il nome della tabella e l'item;
3. Eseguire il metodo `PutItem` fornendo l'oggetto `PutItemRequest` creato nella fase precedente.

Il seguente esempio C# mostra le fasi precedenti. L'esempio carica un item nella tabella `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);
```

Nell'esempio precedente, carica un item libro che abbia gli attributi `Id`, `Title`, `ISBN` e `Authors`. Nota che `Id` è un attributo di tipo numerico, mentre tutti gli altri sono tipo stringa. Authors è un set di `String`.

Specifica dei parametri facoltativi

Puoi anche fornire parametri facoltativi usando l'oggetto `PutItemRequest`, come mostrato nel seguente esempio C#. L'esempio specifica i parametri facoltativi seguenti:

- `ExpressionAttributeNames`, `ExpressionAttributeValues` e `ConditionExpression` specificano che l'item può essere sostituito solo se l'item esistente ha l'attributo ISBN con un valore specifico.
- Il parametro `ReturnValues` per richiedere il vecchio item nella risposta.

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);
```

Per ulteriori informazioni, consulta PutItem.

Ottenimento di un elemento

Il metodo `GetItem` recupera un item.

> **Note**
>
> Per recuperare più item puoi usare il metodo `BatchGetItem`. Per ulteriori informazioni, consulta Ricezione in batch: ricezione di più elementi.

Di seguito sono riportate le fasi per recuperare un item esistente usando l'API di basso livello di AWS SDK per .NET.

1. Creare un'istanza della classe `AmazonDynamoDBClient`.
2. fornisci i parametri obbligatori creando un istanza della classe `GetItemRequest`.

   Per ottenere un item, è necessario fornire il nome della tabella e la chiave primaria dell'item.
3. Eseguire il metodo `GetItem` fornendo l'oggetto `GetItemRequest` creato nella fase precedente.

Il seguente esempio C# mostra le fasi precedenti. L'esempio recupera un item dalla tabella `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.
```

Specifica dei parametri facoltativi

Puoi anche fornire parametri facoltativi usando l'oggetto `GetItemRequest`, come mostrato nel seguente esempio C#. L'esempio specifica i parametri facoltativi seguenti:

- Il parametro `ProjectionExpression` per specificare gli attributi da recuperare.
- Il parametro `ConsistentRead` per eseguire una lettura estremamente coerente. Per ulteriori informazioni sulla lettura coerente, consulta Coerenza di lettura di 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;
```

Per ulteriori informazioni, consulta GetItem.

Aggiornamento di un elemento

Il metodo `UpdateItem` aggiorna un item esistente se presente. Puoi usare l'operazione `UpdateItem` per aggiornare i valori degli attributi esistenti, aggiungere nuovi attributi o eliminare

attributi dalla raccolta esistente. Se non viene trovato l'item che dispone della chiave primaria specificata, viene aggiunto un nuovo item.

L'operazione `UpdateItem` usa le seguenti linee guida:

- Se l'item non esiste, `UpdateItem` aggiunge un nuovo item usando la chiave primaria specificata nell'input.
- Se l'item esiste, `UpdateItem` applica gli aggiornamenti come segue:
  - sostituisce il valore dell'attributo esistente attraverso i valori dell'aggiornamento;
  - se l'attributo che fornisci nell'input non esiste, viene aggiunto un nuovo attributo all'item;
  - se l'attributo di input è nullo, viene eliminato l'attributo qualora presente;
  - se utilizzi `ADD` per `Action`, puoi aggiungere valori a un set esistente (di stringhe o numeri) o aggiungere (se usi un numero positivo) o sottrarre (se usi un numero negativo) matematicamente dal valore dell'attributo numerico esistente.

> **Note**
>
> L'operazione `PutItem` può anche eseguire un aggiornamento. Per ulteriori informazioni, consulta Collocazione di un elemento. Ad esempio, se chiami `PutItem` per caricare un item e la chiave primaria esiste, l'operazione `PutItem` sostituisce l'intero item. Se vi sono attributi nell'item esistente e quegli attributi non sono specificati nell'input, l'operazione `PutItem` li elimina. Tuttavia, `UpdateItem` aggiorna solo gli attributi di input specificati. Qualsiasi altro attributo esistente di quell'item rimarrà invariato.

Di seguito sono riportate le fasi per aggiornare un item esistente usando l'API di basso livello dell'SDK per .NET:

1. Creare un'istanza della classe `AmazonDynamoDBClient`.
2. fornisci i parametri obbligatori creando un istanza della classe `UpdateItemRequest`.

   Questo è l'oggetto di richiesta in cui descrivi tutti gli aggiornamenti, come l'aggiunta di attributi, l'aggiornamento di quelli esistenti o l'eliminazione di attributi. Per rimuovere un attributo esistente, specifica il nome dell'attributo con valore nullo;
3. Eseguire il metodo `UpdateItem` fornendo l'oggetto `UpdateItemRequest` creato nella fase precedente.

Il seguente esempio di codice C# mostra le fasi precedenti. L'esempio aggiorna un item libro nella tabella `ProductCatalog`. Aggiunge un nuovo autore alla raccolta `Authors` ed elimina l'attributo `ISBN` esistente. Inoltre, riduce il prezzo di una unità.

```
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);
```

Specifica dei parametri facoltativi

Puoi anche fornire parametri facoltativi usando l'oggetto `UpdateItemRequest`, come mostrato nel seguente esempio C#. Esso specifica i seguenti parametri opzionali:

- `ExpressionAttributeValues` e `ConditionExpression` per specificare che il prezzo può essere aggiornato solo se il prezzo esistente è 20,00;

- il parametro `ReturnValues` per richiedere l'item aggiornato nella risposta.

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);
```

Per ulteriori informazioni, vedi UpdateItem.

Contatore atomico

Puoi usare il metodo `updateItem` per implementare un contatore atomico e incrementare o decrementare il valore di un attributo esistente senza interferire con altre richieste di scrittura. Per aggiornare un contatore atomico, usa `updateItem` con un attributo di tipo `Number` nel parametro `UpdateExpression` e ADD come `Action`.

Il seguente esempio ne dimostra l'uso, incrementando l'attributo `Quantity` di uno.

```
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);
```

Eliminazione di un elemento

Il metodo `DeleteItem` elimina un item da una tabella.

Di seguito sono riportate le fasi per eliminare un item usando l'API dell'SDK per .NET di basso livello.

1. Creare un'istanza della classe `AmazonDynamoDBClient`.
2. fornisci i parametri obbligatori creando un istanza della classe `DeleteItemRequest`.

   Per eliminare un item, sono necessari il nome della tabella e la chiave primaria dell'item;
3. Eseguire il metodo `DeleteItem` fornendo l'oggetto `DeleteItemRequest` creato nella fase precedente.

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);
```

Specifica dei parametri facoltativi

Puoi anche fornire parametri facoltativi usando l'oggetto `DeleteItemRequest`, come mostrato nel seguente esempio di codice C#. Esso specifica i seguenti parametri opzionali:

- `ExpressionAttributeValues` e `ConditionExpression` per specificare che l'item libro può essere eliminato solo se non è più in pubblicazione (il valore dell'attributo InPublication è falso);
- il parametro `ReturnValues` per richiedere l'item eliminato nella risposta.

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);
```

Per ulteriori informazioni, consulta DeleteItem.

Scrittura in batch: collocazione ed eliminazione di più elementi

La scrittura in batch fa riferimento alla collocazione e all'eliminazione di più item in un batch. Il metodo `BatchWriteItem` ti consente di inserire ed eliminare più item da una o più tabelle con un'unica chiamata. Di seguito sono riportate le fasi per recuperare più item usando l'API di basso livello dell'SDK per .NET.

1. Creare un'istanza della classe `AmazonDynamoDBClient`.
2. descrivi tutte le operazioni di inserimento ed eliminazione creando un istanza della classe `BatchWriteItemRequest`;
3. Eseguire il metodo `BatchWriteItem` fornendo l'oggetto `BatchWriteItemRequest` creato nella fase precedente.
4. Elabora la risposta. Dovresti controllare se erano presenti item di richiesta non elaborati restituiti nella risposta. Questo potrebbe accadere se raggiungi la quota di throughput assegnata o si verifica un altro errore transitorio. Inoltre, DynamoDB limita la dimensione della richiesta e il numero di operazioni che possono essere specificate in una richiesta. Se si superano questi limiti, DynamoDB rifiuta la richiesta. Per ulteriori informazioni, consulta BatchWriteItem.

Il seguente esempio di codice C# mostra le fasi precedenti. L'esempio crea un `BatchWriteItemRequest` per eseguire le seguenti operazioni di scrittura:

- inserisci un item nella tabella `Forum`;
- inserisce ed elimina un item dalla tabella `Thread`.

Il codice quindi esegue `BatchWriteItem` per l'operazione in batch.

```
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);
```

Per un esempio di utilizzo, consulta Esempio: operazioni batch utilizzando l'API di basso livello AWS SDK per .NET.

Ricezione in batch: ricezione di più elementi

Il metodo `BatchGetItem` ti consente di recuperare più item da una o più tabelle.

> **Note**
>
> Per recuperare un singolo item puoi usare il metodo `GetItem`.

Di seguito sono riportate le fasi per recuperare più item usando l'API di basso livello di AWS SDK per .NET.

1. Creare un'istanza della classe `AmazonDynamoDBClient`.
2. fornisci i parametri obbligatori creando un istanza della classe `BatchGetItemRequest`.

   Per recuperare più item, sono necessari il nome della tabella e un elenco dei valori della chiave primaria.
3. Eseguire il metodo `BatchGetItem` fornendo l'oggetto `BatchGetItemRequest` creato nella fase precedente.
4. Elabora la risposta. Dovresti controllare se erano presenti chiavi non elaborate. Questo potrebbe accadere se hai raggiunto la quota di throughput assegnata o alcuni errori di transizione.

Il seguente esempio di codice C# mostra le fasi precedenti. L'esempio recupera gli elementi da due tabelle, `Forum` e `Thread`. La richiesta specifica due item nella tabella `Forum` e tre nella tabella `Thread`. La risposta include item da entrambe le tabelle. Il codice mostra come puoi elaborare la risposta.

```
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);
}
```

Specifica dei parametri facoltativi

Puoi anche fornire parametri facoltativi usando l'oggetto `BatchGetItemRequest`, come mostrato nel seguente esempio di codice C#. L'esempio recupera un item dalla tabella `Forum`. Esso specifica il seguente parametro opzionale:

- Il parametro `ProjectionExpression` per specificare gli attributi da recuperare.

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);
```

Per ulteriori informazioni, consulta BatchGetItem.

Esempio: operazioni CRUD utilizzando l'API di basso livello AWS SDK per .NET

Il seguente esempio di codice C# illustra le operazioni CRUD su un elemento di Amazon DynamoDB. L'esempio consente di aggiungere un item alla tabella `ProductCatalog`, di recuperarlo, eseguire diversi aggiornamenti e, infine, eliminare l'item. Se questa tabella non è stata creata, è possibile crearla anche tramite programmazione. Per ulteriori informazioni, consulta Creazione di tabelle di esempio e caricamento di dati utilizzando AWS SDK per .NET.

Per le istruzioni dettagliate per eseguire il test del seguente esempio, consulta Esempi di codice .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("************************************************");
        }
    }
}
```

Esempio: operazioni batch utilizzando l'API di basso livello AWS SDK per .NET

Argomenti

- Esempio: operazione di scrittura in batch utilizzando l'API di basso livello AWS SDK per .NET
- Esempio: operazione di ottenimento in batch utilizzando l'API di basso livello AWS SDK per .NET

In questa sezione vengono forniti esempi di operazioni in batch, scrittura in batch e get in batch supportate da Amazon DynamoDB.

Esempio: operazione di scrittura in batch utilizzando l'API di basso livello AWS SDK per .NET

Nel seguente esempio di codice C# viene utilizzato il metodo `BatchWriteItem` per eseguire le seguenti operazioni di eliminazione e inserimento:

- inserimento di un item nella tabella `Forum`;
- Inserimento di un item ed eliminazione di un item dalla tabella `Thread`.

Quando crei la tua risposta di scrittura in batch, puoi specificare qualsiasi quantità di richieste di inserimento ed eliminazione in una o più tabelle. Tuttavia, `BatchWriteItem` limita la dimensione di una richiesta di scrittura in batch e il numero di operazioni put e delete in una singola operazione di scrittura in batch. Per ulteriori informazioni, consulta BatchWriteItem. Se la tua richiesta eccede

questi limiti, essa verrà rigettata. Se la tua tabella non dispone di sufficiente throughput assegnato per soddisfare questa richiesta, gli elementi di richiesta non eseguiti vengono restituiti nella risposta.

Il seguente esempio controlla la risposta nel caso in cui vi siano degli elementi di richiesta non eseguiti. Se sono presenti, esegue il loopback e invia nuovamente la richiesta `BatchWriteItem` con gli elementi non elaborati della richiesta. Puoi anche creare queste tabelle di esempio e caricare i dati di esempio in modo programmatico. Per ulteriori informazioni, consulta Creazione di tabelle di esempio e caricamento di dati utilizzando AWS SDK per .NET.

Per le istruzioni dettagliate per eseguire il test del seguente esempio, consulta Esempi di codice .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);
        }
    }
}
```

Esempio: operazione di ottenimento in batch utilizzando l'API di basso livello AWS SDK per .NET

Il seguente esempio di codice C# utilizza il metodo `BatchGetItem` per recuperare più elementi dalle tabelle `Forum` e `Thread` in Amazon DynamoDB. L'item `BatchGetItemRequest` specifica i nomi delle tabelle e un elenco di chiavi primarie per ogni tabella. Nell'esempio si esegue la risposta stampando gli elementi recuperati.

Per le istruzioni dettagliate per eseguire il test del seguente esempio, consulta Esempi di codice .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("*************************************************");
        }
    }
}
```

Esempio: gestione degli attributi di tipo binario utilizzando l'API di basso livello AWS SDK per .NET

Il seguente esempio di codice C# illustra come gestire gli attributi di tipo binario. L'esempio aggiunge un item alla tabella `Reply`. L'item include un attributo di tipo binario (`ExtendedMessage`) che memorizza i dati compressi. L'esempio recupera quindi l'item e stampa tutti valori degli attributi. A titolo illustrativo, l'esempio usa la classe `GZipStream` per comprimere un flusso di esempio, assegnarlo all'attributo `ExtendedMessage` e per poi decomprimerlo durante la stampa del valore dell'attributo.

Per le istruzioni dettagliate per eseguire il test del seguente esempio, consulta Esempi di codice .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();
            }
        }
    }
}
```

Miglioramento dell’accesso ai dati con gli indici secondari in DynamoDB

Amazon DynamoDB offre l'accesso rapido agli elementi in una tabella specificando i valori della chiave primaria. Tuttavia, molte applicazioni potrebbero trarre vantaggio dalla presenza di una o più chiavi secondarie (o alternative) disponibili, per permettere un accesso ai dati efficiente con altri attributi rispetto alla chiave primaria. A tale scopo, puoi creare uno o più indici secondari in una tabella ed emettere delle richieste `Query` o `Scan` rispetto a tali indici.

Un indice secondario è una struttura di dati contente un sottoinsieme di attributi di una tabella, oltre a una chiave alternativa per il supporto delle operazioni `Query`. Puoi recuperare dati dall'indice tramite una `Query`, nello stesso modo in cui utilizzi `Query` con una tabella. Una tabella può presentare più indici secondari, che consentono alle applicazioni di accedere a molti modelli di query diversi.

> **Note**
>
> Puoi inoltre eseguire un'operazione `Scan` su un indice nello stesso modo in cui esegui eseguiresti una `Scan` su una tabella.
> L’accesso multi-account per le operazioni di scansione degli indici secondari non è attualmente supportato dalle policy basate su risorse.

Ciascun indice secondario è associato esattamente a una tabella della quale ottiene i dati. Questa è denominata tabella di base dell'indice. Quando crei un indice, è necessario definire una chiave alternativa dell'indice (chiave di partizione e chiave di ordinamento). Vengono definiti anche gli attributi che si desidera siano proiettati, o copiati, dalla tabella di base nell'indice. DynamoDB copia questi attributi dell'indice insieme agli attributi della chiave primaria dalla tabella di base. Puoi scansionare o eseguire una query sull'indice nello stesso modo in cui lo faresti su una tabella.

Ciascun indice secondario viene gestito automaticamente da DynamoDB. Quando aggiungi, modifichi o elimini item nella tabella di base, verranno aggiornati tutti gli indici della tabella sulla base di tali modifiche.

DynamoDB supporta due tipi di indici secondari:

- Indice secondario globale: un indice con una chiave di partizione e una chiave di ordinamento che possono essere differenti da quelle presenti sulla tabella di base. Un indice secondario viene considerato globale perché le query possono riferirsi a tutti i dati di una tabella di base, in tutte le partizioni. Un indice secondario locale è memorizzato nel suo spazio di partizione lontano dalla tabella di base e viene dimensionato separatamente dalla tabella di base.
- Indice secondario locale: un indice con la stessa chiave di partizione della tabella di base ma con una chiave di ordinamento diversa. Un indice secondario è locale nel senso che l'ambito di ogni partizione di un indice secondario locale è rappresentato da una partizione di tabella di base con lo stesso valore di chiave di partizione.

Per un confronto tra gli indici secondari globali e gli indici secondari locali, guarda questo video.

Making the right choice between GSI and LSI

Argomenti

- Utilizzo degli indici secondari globali in DynamoDB
- Indici secondari locali

Quando determini il tipo di indice da utilizzare, è consigliabile prendere in considerazione i requisiti della tua applicazione. Nella tabella riportata di seguito sono riportate le differenze principali tra un indice secondario globale e un indice secondario locale.

Caratteristica	Indice secondario globale	Indice secondario locale
Schema della chiave	La chiave primaria di un indice secondario globale può essere sia semplice (chiave di partizione) che composita (chiave di partizione e chiave di ordinamento).	La chiave primaria di un indice secondario locale deve essere composita (chiave di partizion e e chiave di ordinamento).

Caratteristica	Indice secondario globale	Indice secondario locale
Attributi della chiave	La chiave di partizione e la chiave di ordinamento (se presente) dell'indice possono essere costituite da qualsiasi attributo della tabella di base di tipo stringa, numero o binario.	La chiave di partizione dell'indice è lo stesso attributo della chiave di partizione della tabella di base. La chiave di partizione può essere qualsiasi attributo della tabella di base di tipo stringa, numero o binario.
Limitazioni della dimensione per valore di chiave di partizione	Non sono previste limitazioni sulle dimensioni per gli indici secondari globali.	Per ciascun valore della chiave di partizione, la dimensione totale tutti gli elementi indicizzati deve essere minore o uguale a 10 GB.
Operazioni di indici online	Gli indici secondari globali possono essere creati al momento della creazione di una tabella. È inoltre possibile aggiungere un nuovo indice secondario globale a una tabella esistente oppure eliminare un indice secondari o globale esistente. Per ulteriori informazioni, consulta Gestione degli indici secondari globali in DynamoDB.	È possibile creare indici secondari locali al momento della creazione di una tabella. Non è possibile aggiungere un indice secondario locale a una tabella esistente né eliminare gli indici secondari locali presenti correntemente.
Query e partizioni	Un indice secondario globale consente di eseguire una query sull'intera tabella, su tutte le partizioni.	Un indice secondario locale consente di eseguire una query su una singola partizione, come specificato dal valore della chiave di partizione nella query.

Caratteristica	Indice secondario globale	Indice secondario locale
Consistenza di lettura	Tuttavia, le query sugli indici secondari globali supportano solo la consistenza finale.	Quando si esegue una query su un indice secondario locale, è possibile scegliere tra consistenza finale o forte consistenza.
Utilizzo del throughput assegnato	Ciascun indice secondari o globale ha le proprie impostazioni di velocità effettiva assegnata per l'attivit à di lettura e scrittura. Le query o le scansioni su un indice secondario globale utilizzano unità di capacità dall'indice, non della tabella di base. Lo stesso vale per gli aggiornamenti dell'indice secondario globale a causa delle scritture nella tabella. Un indice secondario globale associato a tabelle globali consuma unità di capacità di scrittura.	Le query o le scansioni su un indice secondario locale utilizzano unità di capacità di lettura della tabella di base. Quando si scrive su una tabella, vengono aggiornati anche i relativi indici secondari locali; tali aggiornamenti utilizzano unità di capacità di scrittura della tabella di base. Un indice secondario globale associato a tabelle globali consuma unità di capacità di scrittura replicate.
Attributi proiettati	Con le query o le scansioni dell'indice secondario globale, è possibile richiedere solo gli attributi proiettati nell'indi ce. DynamoDB non recupera alcun attributo dalla tabella.	Se si esegue una query o una scansione sull'indice secondario globale, è possibile richiedere gli attributi che non sono proiettati sull'indi ce. DynamoDB recupera automaticamente tali attributi dalla tabella.

Se si intende creare una o più tabelle con gli indici secondari, è necessario farlo in sequenza. Ad esempio, sarebbe necessario creare la prima tabella e attendere che il relativo stato divenga `ACTIVE` per poter creare la tabella successiva, dunque attendere che il suo stato divenga `ACTIVE`e così via. Se si prova a creare più tabelle contemporaneamente con un indice secondario, DynamoDB restituisce una `LimitExceededException`.

Ogni indice secondario utilizza la stessa classe di tabella e modalità di capacità della tabella di base a cui è associato. Per ogni indice secondario, è necessario specificare quanto segue:

- Tipo di indice da creare, ovvero un indice secondario globale o un indice secondario locale.
- Un nome dell'indice. Le regole di denominazione per gli indici sono le stesse regole per le tabelle, come elencate in Quote in Amazon DynamoDB. Il nome deve essere univoco per la tabella di base a cui è associato, ma puoi utilizzare lo stesso nome per gli indici associati a tabelle di base diverse.
- Lo schema delle chiavi dell'indice. Ciascun attributo dello schema della chiave di indicizzazione deve essere un attributo di tipo `String`, `Number` o `Binary`. Non sono consentiti altri tipi di dati, tra cui documenti e set. Altri requisiti della schema della chiave dipendono dal tipo di indice:
  - Per un indice secondario globale, la chiave di partizione può essere qualsiasi attributo scalare della tabella di base. È facoltativa una chiave di ordinamento, che può essere anch'essa qualsiasi attributo scalare della tabella di base.
  - Per un indice secondario locale, la chiave di partizione deve essere uguale alla chiave di partizione della tabella di base; la chiave di ordinamento deve essere un attributo non chiave della tabella di base.
- Gli attributi aggiuntivi, se presenti, da proiettare dalla tabella di base nell'indice. Questi attributi si aggiungono agli attributi di chiave della tabella, proiettati automaticamente in ciascun indice. Puoi proiettare attributi di qualsiasi tipo di dati, tra cui scalari, documenti e set.
- Le impostazioni di throughput assegnato dell'indice, se necessario:
  - Per un indice secondario globale, è necessario specificare le impostazioni delle unità di capacità di lettura e scrittura. Queste impostazioni di throughput assegnato sono indipendenti dalle impostazioni della tabella di base.
  - Per un indice secondario locale, non è necessario specificare le impostazioni delle unità di capacità di lettura e scrittura. Tutte le operazioni di lettura e scrittura su un indice secondario locale utilizzano le impostazioni di velocità effettiva assegnata della relativa tabella di base.

Per la massima flessibilità delle query, è possibile creare fino a 20 indici secondari globali (quota predefinita) e fino a 5 indici secondari locali per tabella.

Per ottenere un elenco dettagliato di indici secondari su una tabella, utilizza l'operazione `DescribeTable`. `DescribeTable` restituisce il nome, la dimensione dell'archiviazione e il numero di elementi per ogni indice secondario nella tabella. Questi valori non vengono aggiornati in tempo reale, bensì ogni circa sei ore.

È possibile accedere ai dati in un indice secondario utilizzando l'operazione `Query` o `Scan`. È necessario specificare il nome della tabella di base e il nome dell'indice che si desidera utilizzare, gli attributi da restituire nei risultati e qualsiasi espressione di condizione o filtro da applicare. DynamoDB può restituire i risultati in ordine crescente o decrescente.

Quando elimini una tabella, verranno eliminati anche tutti gli indici associati alla tabella.

Per le best practice, consulta Best practice per l'uso di indici secondari in DynamoDB..

Utilizzo degli indici secondari globali in DynamoDB

Alcune applicazioni devono poter eseguire molti tipi di query, usando un'ampia gamma di attributi diversi come criteri di query. Per supportare questi requisiti, è possibile creare uno o più indici secondari globali ed emettere richieste `Query` rispetto a questi indici in Amazon DynamoDB.

Argomenti

- Scenario: Utilizzo di un indice secondario globale
- Proiezioni di attributi
- Schema chiave multiattributo
- Lettura di dati da un indice secondario globale
- Sincronizzazione dei dati tra tabelle e indici secondari globali
- Classi di tabella con indici secondari globali
- Considerazioni sulla velocità di trasmissione effettiva assegnata per indici secondari globali
- Considerazioni sullo storage per indici secondari globali
- Modelli di progettazione
- Gestione degli indici secondari globali in DynamoDB
- Rilevamento e correzione delle violazioni delle chiavi in DynamoDB
- Utilizzo di indici secondari globali: Java
- Utilizzo di indici secondari globali: .NET
- Utilizzo di indici secondari in DynamoDB con la AWS CLI

Scenario: Utilizzo di un indice secondario globale

A titolo illustrativo, considera una tabella denominata `GameScores` che tiene traccia di utenti e punteggi per un'applicazione di gioco per dispositivi mobili. Ogni item in `GameScores` è identificato da una chiave di partizione (`UserId`) e una chiave di ordinamento (`GameTitle`). Il seguente diagramma mostra in che modo verrebbero organizzarti gli elementi nella tabella. Non tutti gli attributi vengono visualizzati.

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	...
...	...	...	...	...	...	

Supponi ora di voler scrivere un'applicazione di tipo classifica per visualizzare i punteggi più alti per ogni gioco. Una query che specifica gli attributi chiave (`UserId` e `GameTitle`) sarebbe molto efficiente. Tuttavia, se l'applicazione deve recuperare i dati da `GameScores` solo in base a `GameTitle`, occorre utilizzare un'operazione `Scan`. Con l'aggiunta di altri item alla tabella, le scansioni di tutti i dati rallenterebbero e diventerebbero meno pratiche, rendendo difficile rispondere a domande come le seguenti:

- Qual è il punteggio più alto mai registrato per il gioco Meteor Blasters?
- Quale utente ha il punteggio più alto per Galaxy Invaders?
- Qual è il rapporto più elevato tra vittorie e sconfitte?

Per accelerare le query su attributi non chiave, è possibile creare un indice secondario globale. Un indice secondario globale contiene una selezione di attributi dalla tabella di base organizzati tramite una chiave primaria diversa da quella della tabella. La chiave dell'indice non deve avere alcuno degli attributi di chiave della tabella, né lo stesso schema della chiave di una tabella.

Ad esempio, è possibile creare un indice secondario globale denominato `GameTitleIndex` con una chiave di partizione `GameTitle` e una chiave di ordinamento `TopScore`. Gli attributi della chiave primaria della tabella di base vengono sempre proiettati in un indice, pertanto è presente anche l'attributo `UserId`. Il diagramma seguente mostra l'aspetto di un indice `GameTitleIndex`.

GameTitleIndex

GameTitle	*TopScore*	*UserId*
"Alien Adventure"	*192*	*"102"*
"Attack Ships"	*3*	*"103"*
"Galaxy Invaders"	*0*	*"102"*
"Galaxy Invaders"	*2317*	*"103"*
"Galaxy Invaders"	*5842*	*"101"*
"Meteor Blasters"	*723*	*"103"*
"Meteor Blasters"	*1000*	*"101"*
"Starship X"	*24*	*"101"*
"Starship X"	*42*	*"103"*
...	...	...

A questo punto, puoi eseguire una query su `GameTitleIndex` e ottenere facilmente i punteggi per Meteor Blasters. I risultati sono ordinati in base ai valori della chiave di ordinamento, ovvero `TopScore`. Se imposti il parametro `ScanIndexForward` su false, i risultati vengono restituiti in ordine decrescente e di conseguenza il punteggio più alto viene restituito per primo.

Ogni indice secondario globale deve avere una chiave di partizione e può avere anche una chiave di ordinamento facoltativa. Lo schema della chiave di indicizzazione può essere diverso dallo schema della tabella di base. È possibile avere una tabella con una chiave primaria semplice (chiave di

partizione) e creare un indice secondario globale con una chiave primaria composita (chiave di partizione e chiave di ordinamento) o viceversa. Gli attributi della chiave dell'indice possono essere costituiti da qualsiasi attributo `String`, `Number` o `Binary` di primo livello della tabella di base. Altri tipi scalari, documento e set non sono consentiti.

Se vuoi, puoi proiettare altri attributi della tabella di base nell'indice. Quando si esegue una query sull'indice, DynamoDB può recuperare questi attributi proiettati in modo efficiente. Tuttavia, le query su un indice secondario globale non possono recuperare gli attributi dalla tabella di base. Ad esempio, se esegui una query `GameTitleIndex` come mostrato nel diagramma precedente, la query non è in grado di accedere ad alcun attributo non di chiave diverso da `TopScore` (se vengono automaticamente proiettati gli attributi della chiave `GameTitle` e `UserId`).

In una tabella DynamoDB ogni valore di chiave deve essere univoco. Tuttavia, i valori delle chiavi in un indice secondario globale non devono essere univoci. A titolo illustrativo, supponi che un gioco denominato Comet Quest sia particolarmente difficile, con molti nuovi utenti che provano ma non riescono a ottenere un punteggio maggiore di zero. Di seguito sono illustrati alcuni dati che possono rappresentare questa situazione.

UserId	GameTitle	TopScore
123	Comet Quest	0
201	Comet Quest	0
301	Comet Quest	0

Quando questi dati vengono aggiunti alla tabella `GameScores`, vengono propagati da DynamoDB a `GameTitleIndex`. Se quindi eseguiamo una query sull'indice utilizzando Comet Quest per `GameTitle` e 0 per `TopScore`, vengono restituiti i dati seguenti.

GameTitle	*TopScore*	*UserId*
"Comet Quest"	*0*	*"123"*
"Comet Quest"	*0*	*"201"*
"Comet Quest"	*0*	*"301"*

Nella risposta vengono visualizzati solo gli elementi con i valori di chiave specificati. All'interno di questo set di dati, gli elementi non hanno un ordine specifico.

Un indice secondario globale tiene traccia solo degli elementi di dati in cui esistono effettivamente i suoi attributi di chiave. Ad esempio, supponi di aver aggiunto un nuovo item alla tabella `GameScores`, ma di aver fornito solo gli attributi della chiave primaria obbligatori.

UserId	GameTitle
400	Comet Quest

Poiché non è stato specificato l'attributo `TopScore`, DynamoDB non propaga questo elemento a `GameTitleIndex`. Di conseguenza, se ha eseguito una query su `GameScores` per tutti gli elementi di Comet Quest, ottieni i quattro item seguenti:

UserId	GameTitle	TopScore
"123"	"Comet Quest"	0
"201"	"Comet Quest"	0
"301"	"Comet Quest"	0
"400"	"Comet Quest"	

Una query simile su `GameTitleIndex` continua a restituire tre item anziché quattro. Il motivo è che l'item con `TopScore` inesistente non viene propagato nell'indice.

GameTitle	*TopScore*	*UserId*
"Comet Quest"	*0*	*"123"*
"Comet Quest"	*0*	*"201"*
"Comet Quest"	*0*	*"301"*

Proiezioni di attributi

Una proiezione è l'insieme di attributi copiato da una tabella in un indice secondario. La chiave di partizione e la chiave di ordinamento della tabella vengono sempre proiettati nell'indice; è possibile proiettare altri attributi per supportare i requisiti di query dell'applicazione. Quando si esegue una

query su un indice, Amazon DynamoDB può accedere a qualsiasi attributo nella proiezione come se tali attributi fossero in una propria tabella.

Quando si crea un indice secondario, è necessario specificare gli attributi che saranno proiettati nell'indice. DynamoDB fornisce tre diverse opzioni per questo:

- KEYS_ONLY: ogni elemento dell'indice è costituito solo dalla chiave di partizione della tabella e dai valori della chiave di ordinamento, oltre ai valori della chiave di indice. L'opzione `KEYS_ONLY` si traduce nell'indice secondario più piccolo possibile.
- INCLUDE: oltre agli attributi descritti in `KEYS_ONLY`, l'indice secondario includerà gli altri attributi non chiave che sono stati specificati.
- ALL: l'indice secondario include tutti gli attributi della tabella di origine. Poiché tutti i dati della tabella sono duplicati nell'indice, una proiezione `ALL` restituisce il più grande indice secondario possibile.

Nel diagramma precedente, `GameTitleIndex` dispone di un solo attributo proiettato: `UserId`. Pertanto, sebbene un'applicazione possa determinare in maniera efficiente il valore `UserId` dei punteggi più alti per ogni partita utilizzando `GameTitle` e `TopScore` nelle query, non può determinare in maniera efficiente il rapporto più elevato tra vittorie e sconfitte per i punteggi più alti. A questo scopo, è necessario eseguire una query aggiuntiva sulla tabella di base per recuperare le vittorie e le sconfitte per ciascuno dei punteggi più alti. Un modo più efficiente per supportare le query su questi dati consiste nel proiettare gli attributi dalla tabella di base nell'indice secondario globale, come mostrato in questo diagramma.

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
...	...	...	...	...

Poiché gli attributi non di chiave `Wins` e `Losses` vengono proiettati nell'indice, un'applicazione può determinare il rapporto tra vittorie e sconfitte per qualsiasi gioco o per qualsiasi combinazione di gioco e ID utente.

Quando si scelgono gli attributi da proiettare in un indice secondario globale, è necessario considerare il compromesso tra i costi correlati alla velocità effettiva assegnata e i costi di archiviazione:

- Se è necessario accedere a pochi attributi con la latenza più bassa possibile, considerare la possibilità di proiettare solo quegli attributi in un indice secondario globale. Più piccolo è l'indice, minore è il costo per memorizzarlo e minori sono i costi di scrittura.
- Se l'applicazione accede frequentemente ad alcuni attributi non chiave, è necessario considerare di proiettare quegli attributi in un indice secondario globale. I costi di archiviazione aggiuntivi per l'indice secondario globale compensano il costo di esecuzione di scansioni frequenti delle tabelle.
- Se è necessario accedere alla maggior parte degli attributi non chiave su base frequente, è possibile proiettare questi attributi, o anche l'intera tabella di base, in un indice secondario

globale. Questo offre massima flessibilità. Tuttavia, i costi di storage aumenteranno o addirittura raddoppieranno.

- Se l'applicazione esegue di rado le query sulla tabella, ma esegue molte scritture o aggiornamenti dei dati nella tabella, considera di proiettare `KEYS_ONLY`. L'indice secondario globale avrebbe dimensioni minime e sarebbe comunque disponibile se necessario per l'attività di query.

Schema chiave multiattributo

Gli indici secondari globali supportano chiavi con più attributi, che consentono di comporre chiavi di partizione e ordinare le chiavi a partire da più attributi. Con le chiavi multiattributo, è possibile creare una chiave di partizione da un massimo di quattro attributi e una chiave di ordinamento da un massimo di quattro attributi, per un totale di un massimo di otto attributi per schema di chiavi.

Le chiavi multiattributo semplificano il modello di dati eliminando la necessità di concatenare manualmente gli attributi in chiavi sintetiche. Invece di creare stringhe composite, ad esempio`TOURNAMENT#WINTER2024#REGION#NA-EAST`, puoi utilizzare direttamente gli attributi naturali del tuo modello di dominio. DynamoDB gestisce automaticamente la logica delle chiavi composite, eseguendo l'hashing di più attributi chiave di partizione per la distribuzione dei dati e mantenendo l'ordinamento gerarchico tra più attributi delle chiavi di ordinamento.

Ad esempio, prendi in considerazione un sistema di tornei di gioco in cui desideri organizzare le partite per torneo e regione. Con le chiavi multiattributo, è possibile definire la chiave di partizione come due attributi separati: `tournamentId` e. `region` Allo stesso modo, è possibile definire la chiave di ordinamento utilizzando più attributi come `roundbracket`, e `matchId` creare una gerarchia naturale. Questo approccio mantiene i dati digitati e il codice pulito, senza manipolazione o analisi delle stringhe.

Quando si esegue una query su un indice secondario globale con chiavi multiattributo, è necessario specificare tutti gli attributi delle chiavi di partizione utilizzando condizioni di uguaglianza. Per gli attributi chiave di ordinamento, è possibile interrogarli left-to-right nell'ordine in cui sono definiti nello schema chiave. Ciò significa che è possibile interrogare solo il primo attributo chiave di ordinamento, i primi due attributi insieme o tutti gli attributi insieme, ma non è possibile saltare gli attributi intermedi. Condizioni di disuguaglianza come>, <`BETWEEN`, o `begins_with()` devono essere l'ultima condizione della query.

Le chiavi con più attributi funzionano particolarmente bene quando si creano indici secondari globali su tabelle esistenti. È possibile utilizzare gli attributi già presenti nella tabella senza inserire le chiavi

sintetiche tra i dati. Ciò semplifica l'aggiunta di nuovi modelli di query all'applicazione creando indici che riorganizzano i dati utilizzando diverse combinazioni di attributi.

Ogni attributo in una chiave con più attributi può avere il proprio tipo di dati: `String` (S), (N) o `Number` (B). `Binary` Quando scegliete i tipi di dati, tenete presente che `Number` gli attributi vengono ordinati numericamente senza richiedere il riempimento zero, mentre gli attributi vengono ordinati lessicograficamente. `String` Ad esempio, se utilizzi un `Number` tipo per un attributo score, i valori 5, 50, 500 e 1000 vengono ordinati in ordine numerico naturale. Gli stessi valori `String` di type verrebbero ordinati come «1000", «5", «50", «500" a meno che non vengano riempiti con zeri iniziali.

Quando progettate chiavi con più attributi, ordinate gli attributi dal più generale al più specifico. Per le chiavi di partizione, combinate gli attributi che vengono sempre interrogati insieme e che garantiscono una buona distribuzione dei dati. Per quanto riguarda le chiavi di ordinamento, posiziona gli attributi più richiesti al primo posto nella gerarchia per massimizzare la flessibilità delle query. Questo ordinamento consente di eseguire query a qualsiasi livello di granularità che corrisponda ai modelli di accesso.

Vedi gli esempi di Chiavi con più attributi implementazione.

Lettura di dati da un indice secondario globale

È possibile recuperare gli elementi da un indice secondario globale utilizzando le operazioni `Query` e `Scan`. Le operazioni `GetItem` e `BatchGetItem` non possono essere utilizzate su un indice secondario globale.

Esecuzione di query su un indice secondario globale

È possibile utilizzare l'operazione `Query` per accedere a uno o più elementi in un indice secondario globale. La query deve specificare il nome della tabella di base e il nome dell'indice che si desidera utilizzare, gli attributi da restituire nei risultati della query e qualsiasi condizione di query da applicare. DynamoDB può restituire i risultati in ordine crescente o decrescente.

Considera i dati seguenti restituiti da un'operazione `Query` che richiede dati di gioco per un'applicazione di tipo classifica.

```
{
    "TableName": "GameScores",
    "IndexName": "GameTitleIndex",
    "KeyConditionExpression": "GameTitle = :v_title",
    "ExpressionAttributeValues": {
        ":v_title": {"S": "Meteor Blasters"}
```

```
    },
    "ProjectionExpression": "UserId, TopScore",
    "ScanIndexForward": false
}
```

In questa query:

- DynamoDB GameTitleIndexaccede, utilizzando la chiave di partizione per individuare gli elementi GameTitledell'indice per Meteor Blasters. Tutti gli elementi dell'indice con questa chiave sono memorizzati l'uno accanto all'altro per il recupero rapido.
- All'interno di questo gioco, DynamoDB utilizza l'indice per accedere a tutti gli IDs utenti e ai punteggi più alti di questo gioco.
- Vengono restituiti i risultati in base all'ordine decrescente, perché il parametro `ScanIndexForward` è impostato su false.

Scansione di un indice secondario globale

È possibile utilizzare l'operazione `Scan` per recuperare tutti i dati da un indice secondario globale. Devi fornire il nome della tabella di base e il nome dell'indice nella richiesta. Con un'operazione `Scan`, DynamoDB legge tutti i dati nell'indice e li restituisce all'applicazione. Inoltre puoi richiedere che vengano restituiti solo alcuni dati e che quelli rimanenti vengano eliminati. A questo scopo, usa il parametro `FilterExpression` dell'operazione `Scan`. Per ulteriori informazioni, consulta Espressioni di filtro per la scansione.

Sincronizzazione dei dati tra tabelle e indici secondari globali

DynamoDB sincronizza automaticamente ogni indice secondario globale con la relativa tabella di base. Quando un'applicazione scrive o elimina elementi in una tabella, ogni indice secondario globale nella tabella viene aggiornato in modo asincrono, usando un modello a consistenza finale. Le applicazioni non scrivono mai direttamente in un indice. Tuttavia, è importante comprendere le implicazioni di come DynamoDB mantiene questi indici.

Gli indici secondari globali ereditano la modalità di read/write capacità dalla tabella base. Per ulteriori informazioni, consulta Considerazioni sul passaggio tra modalità di capacità in DynamoDB.

Quando si crea un indice secondario globale, viene specificato uno o più attributi della chiave di indice e i rispettivi tipi di dati. Questo significa che ogni volta che scrivi un item nella tabella di base, i tipi di dati per questi attributi devono corrispondere ai tipi di dati dello schema della chiave dell'indice. Nel caso di `GameTitleIndex`, la chiave di partizione `GameTitle` nell'indice è definita

come un tipo di dati `String`. La chiave di ordinamento `TopScore` nell'indice è di tipo `Number`. Se si prova ad aggiungere un elemento alla tabella `GameScores` e si specifica un tipo di dati diverso per `GameTitle` o `TopScore`, DynamoDB restituisce un `ValidationException` perché il tipo di dati non corrisponde.

Quando inserisci o elimini item in una tabella, ogni indice secondario globale nella tabella viene aggiornato in base a un modello eventualmente consistente. Le modifiche apportate ai dati della tabella vengono propagate in ogni indice secondario globale entro una frazione di secondo in condizioni normali. Tuttavia, in alcuni improbabili scenari di errore, possono verificarsi ritardi di propagazione prolungati. Per questo motivo, le applicazioni devono poter prevedere e gestire le situazioni in cui una query su un indice secondario globale restituisce risultati non aggiornati.

Se si scrive un elemento in una tabella, non è necessario specificare gli attributi per la chiave di ordinamento di alcun indice secondario globale. Utilizzando `GameTitleIndex` come un esempio, non devi specificare un valore per l'attributo `TopScore` per scrivere un nuovo item nella tabella `GameScores`. In questo caso, DynamoDB non scrive alcun dato nell'indice per questo particolare elemento.

Una tabella con molti indici secondari globali comporta costi maggiori per l'attività di scrittura rispetto alle tabelle con meno indici. Per ulteriori informazioni, consulta Considerazioni sulla velocità di trasmissione effettiva assegnata per indici secondari globali.

Classi di tabella con indici secondari globali

Un indice secondario globale utilizzerà sempre la stessa classe di tabella della tabella di base. Ogni volta che viene aggiunto un nuovo indice secondario globale per una tabella, il nuovo indice utilizzerà la stessa classe di tabella della tabella di base. Quando la classe di tabella di una tabella viene aggiornata, vengono aggiornati anche tutti gli indici secondari globali associati.

Considerazioni sulla velocità di trasmissione effettiva assegnata per indici secondari globali

Quando si crea un indice secondario globale su una tabella con modalità assegnata, è necessario specificare le unità di capacità di lettura e scrittura per il carico di lavoro previsto sull'indice. Le impostazioni correlate alla velocità effettiva assegnata di un indice secondario globale sono separate da quelle della relativa tabella di base. Un'operazione `Query` su un indice secondario globale utilizza unità di capacità di lettura dell'indice e non della tabella di base. Quando inserisci, aggiorni o elimini item in una tabella, anche gli indici secondari globali nella tabella vengono aggiornati. Questi aggiornamenti dell'indice utilizzano unità di capacità in scrittura dell'indice, non della tabella.

Ad esempio, se si esegui un'operazione `Query` su un indice secondario globale e se ne supera la capacità di lettura assegnata, la richiesta sarà sottoposta a limitazione. Se si esegue un'attività di scrittura pesante sulla tabella, ma un indice secondario globale su quella tabella ha una capacità di scrittura insufficiente, l'attività di scrittura sulla tabella verrà limitata.

> **Important**
>
> Per evitare il possibile throttling, la capacità in scrittura assegnata per un indice secondario globale deve essere maggiore o uguale alla capacità in scrittura della tabella di base poiché nuovi aggiornamenti scrivono nella tabella di base e nell'indice secondario globale.

Per visualizzare le impostazioni di velocità effettiva assegnata per un indice secondario globale, utilizza l'operazione `DescribeTable`. Vengono restituite informazioni dettagliare relative a tutti gli indici secondari della tabella.

Unità di capacità in lettura

Gli indici secondari globali supportano letture consistenti finali, ognuna delle quali utilizza metà di un'unità di capacità in lettura. Questo significa che una singola query su un indice secondario globale può recuperare fino a 2 × 4 KB = 8 KB per ogni unità di capacità di lettura.

Per le query su un indice secondario globale, DynamoDB calcola l'attività di lettura assegnata come avviene per le query sulle tabelle. L'unica differenza è che il calcolo si basa sulle dimensioni delle voci dell'indice, piuttosto che sulla dimensione dell'item nella tabella di base. Il numero di unità di capacità in lettura è la somma di tutte le dimensioni degli attributi proiettati di tutti gli elementi restituiti. Il risultato viene arrotondato al limite di 4 KB successivo. Per ulteriori informazioni sul modo in cui DynamoDB calcola l'uso della velocità effettiva assegnata, consulta Modalità con capacità allocata di DynamoDB.

La dimensione massima dei risultati restituiti da un'operazione `Query` è 1 MB. Sono incluse le dimensioni di tutti i nomi e i valori degli attributi in tutti gli elementi restituiti.

Ad esempio, si consideri un indice secondario globale in cui ogni elemento contiene 2.000 byte di dati. Si supponga ora di eseguire un'operazione `Query` su questo indice e che `KeyConditionExpression` della query restituisca otto elementi. La dimensione totale degli elementi corrispondenti è 2.000 byte 8 elementi = 16.000 byte. Questo risultato viene quindi arrotondato al limite da 4 KB più vicino. Poiché le query su un indice secondario globale sono a consistenza finale, il costo totale equivale a 0,5 × (16 KB / 4 KB) o 2 unità di capacità di lettura.

Unità di capacità in scrittura

Quando un elemento in una tabella viene aggiunto, aggiornato o eliminato e questa operazione interessa un indice secondario globale, l'indice utilizzerà le unità di capacità di scrittura assegnate per l'operazione. Il costo totale del throughput assegnato per una scrittura è la somma delle unità di capacità in scrittura utilizzate dalla scrittura nella tabella di base e quelle utilizzate dall'aggiornamento degli indici secondari globali. Se una scrittura in una tabella non richiede l'aggiornamento di un indice secondario globale, non viene utilizzata capacità di scrittura dell'indice.

Perché la scrittura in una tabella riesca, le impostazioni correlate al throughput assegnato per la tabella e tutti i suoi indici secondari globali devono avere capacità in scrittura sufficiente per la scrittura. In caso contrario, la scrittura nella tabella viene limitata.

> **Important**
>
> Quando si crea un indice secondario globale (GSI), le operazioni di scrittura sulla tabella di base possono essere sottoposte a limitazione (della larghezza di banda della rete) se l'attività dei GSI risultante dalle scritture sulla tabella di base supera la capacità di scrittura allocata del GSI. Questa limitazione (della larghezza di banda della rete) influisce su tutte le operazioni di scrittura, dal processo di indicizzazione alla potenziale interruzione dei carichi di lavoro di produzione. Per ulteriori informazioni, consulta Risoluzione dei problemi di limitazione (della larghezza di banda della rete) in Amazon DynamoDB.

Il costo della scrittura di un elemento in un indice secondario globale dipende da diversi fattori:

- Se scrivi un nuovo item nella tabella che definisce un attributo indicizzato o aggiorni un item esistente per definire un attributo indicizzato in precedenza non definito, è necessario eseguire un'operazione di scrittura per inserire l'item nell'indice.
- Se un aggiornamento della tabella modifica il valore di un attributo chiave indicizzato (da A a B), sono necessarie due scritture, una per eliminare l'item precedente dall'indice e un'altra per inserire il nuovo item nell'indice.
- Se un item era presente nell'indice, ma una scrittura nella tabella ha causato la cancellazione dell'attributo indicizzato, è necessaria una scrittura per eliminare la proiezione dell'item precedente dall'indice.
- Se un item non è presente nell'indice prima o dopo l'aggiornamento dell'item, non vi è alcun costo di scrittura aggiuntivo per l'indice.

- Se un aggiornamento della tabella modifica solo il valore degli attributi proiettati nello schema della chiave dell'indice, ma non modifica il valore di alcun attributo di chiave indicizzato, è necessaria una scrittura per aggiornare i valori degli attributi proiettati nell'indice.

Tutti questi fattori presuppongono che la dimensione di ciascun elemento nell'indice sia minore o uguale alla dimensione dell'elemento di 1 KB per il calcolo delle unità di capacità di scrittura. Le voci di indice più grandi richiedono unità di capacità in scrittura aggiuntive. Puoi contenere al minimo i costi di scrittura considerando gli attributi che le query devono restituire e proiettando solo tali attributi nell'indice.

Considerazioni sullo storage per indici secondari globali

Quando un'applicazione scrive un elemento in una tabella, DynamoDB copia automaticamente il sottoinsieme di attributi corretto in qualsiasi indice secondario globale in cui devono essere presenti gli attributi. All' AWS account vengono addebitati i costi per la memorizzazione dell'articolo nella tabella di base e anche per la memorizzazione degli attributi in tutti gli indici secondari globali di tale tabella.

La quantità di spazio utilizzata da un item dell'indice è la somma di quanto segue:

- La dimensione in byte della chiave primaria della tabella di base (chiave di partizione e chiave di ordinamento)
- La dimensione in byte dell'attributo della chiave di indicizzazione
- La dimensione in byte degli attributi proiettati (se presenti)
- 100 byte di sovraccarico per elemento dell'indice

Per stimare i requisiti di archiviazione per un indice secondario globale, è possibile stimare la dimensione media di un elemento nell'indice e quindi moltiplicarla per il numero di elementi nella tabella di base che hanno attributi di chiave dell'indice secondario globale.

Se una tabella contiene un elemento in cui uno o più attributi particolari non sono definiti, ma tale attributo è definito come chiave di partizione dell'indice o chiave di ordinamento, DynamoDB non scrive alcun dato per quell'elemento nell'indice.

Modelli di progettazione

I modelli di progettazione forniscono soluzioni comprovate alle sfide più comuni quando si lavora con indici secondari globali. Questi modelli consentono di creare applicazioni efficienti e scalabili mostrando come strutturare gli indici per casi d'uso specifici.

Ogni modello include una guida all'implementazione completa con esempi di codice, best practice e casi d'uso reali per aiutarvi ad applicare il pattern alle vostre applicazioni.

Argomenti

- Schema di chiavi multiattributo

Schema di chiavi multiattributo

Panoramica

Le chiavi multi-attributo consentono di creare chiavi di partizione e ordinamento del Global Secondary Index (GSI) composte da un massimo di quattro attributi ciascuna. Ciò riduce il codice lato client e semplifica la modellazione iniziale dei dati e l'aggiunta di nuovi modelli di accesso in un secondo momento.

Consideriamo uno scenario comune: per creare un GSI che interroghi gli elementi in base a più attributi gerarchici, tradizionalmente è necessario creare chiavi sintetiche concatenando i valori. Ad esempio, in un'app di gioco, per interrogare le partite dei tornei per torneo, regione e round, potresti creare una chiave di partizione GSI sintetica come TOURNAMENT# WINTER2 024 #REGION #NA -EAST e una chiave di ordinamento sintetica come ROUND #SEMIFINALS #BRACKET #UPPER. Questo approccio funziona, ma richiede la concatenazione di stringhe durante la scrittura dei dati, l'analisi durante la lettura e il riempimento delle chiavi sintetiche su tutti gli elementi esistenti se si aggiunge il GSI a una tabella esistente. Ciò rende il codice più disordinato e difficile da mantenere la sicurezza dei tipi sui singoli componenti chiave.

Le chiavi multiattributo risolvono questo problema per. GSIs Definisci la tua chiave di partizione GSI utilizzando più attributi esistenti come TournamentID e region. DynamoDB gestisce automaticamente la logica delle chiavi composite, eseguendo l'hashing delle stesse per la distribuzione dei dati. Gli elementi vengono scritti utilizzando gli attributi naturali del modello di dominio e il GSI li indicizza automaticamente. Nessuna concatenazione, nessuna analisi, nessun riempimento. Il codice rimane pulito, i dati rimangono digitati e le query rimangono semplici. Questo approccio è particolarmente utile quando si hanno dati gerarchici con raggruppamenti di attributi naturali (come torneo → regione → round o organizzazione → dipartimento → squadra).

Esempio di applicazione

Questa guida illustra la creazione di un sistema di tracciamento delle partite di torneo per una piattaforma di eSport. La piattaforma deve interrogare in modo efficiente le partite su più fronti: per torneo e regione per la gestione dei gironi, per giocatore per la cronologia delle partite e per data di programmazione.

Modello di dati

In questa procedura dettagliata, il sistema di tracciamento delle partite dei tornei supporta tre modelli di accesso principali, ognuno dei quali richiede una struttura chiave diversa:

Schema di accesso 1: cerca una partita specifica in base al relativo ID univoco

- Soluzione: tabella di base con `matchId` chiave di partizione

Schema di accesso 2: interroga tutte le partite di un torneo e di una regione specifici, filtrandole facoltativamente per round, tabellone o partita

- Soluzione: Indice secondario globale con chiave di partizione multiattributo (`tournamentId+region`) e chiave di ordinamento multiattributo (+) `round` `bracket` `matchId`
- Query di esempio: «Tutte le partite WINTER2 024 nella regione NA-EST» o «Tutte le SEMIFINALI corrispondono nel girone SUPERIORE per 024/NA-EST» WINTER2

Schema di accesso 3: interroga la cronologia delle partite di un giocatore, filtrandola facoltativamente per intervallo di date o round del torneo

- Soluzione: Indice secondario globale con chiave di partizione singola (`player1Id`) e chiave di ordinamento multiattributo (+) `matchDate` `round`
- Domande di esempio: «Tutte le partite del giocatore 101" o «Le partite del giocatore 101 a gennaio 2024"

La differenza fondamentale tra gli approcci tradizionali e quelli con più attributi diventa evidente quando si esamina la struttura degli elementi:

Approccio tradizionale all'indice secondario globale (chiavi concatenate):

```
// 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
};
```

Approccio all'indice secondario globale multiattributo (chiavi native):

```
// 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 le chiavi multiattributo, gli elementi vengono scritti una sola volta con attributi di dominio naturali. DynamoDB li indicizza automaticamente su GSIs più pagine senza richiedere chiavi sintetiche concatenate.

Schema della tabella di base:

- Chiave di partizione: `matchId` (1 attributo)

Schema dell'indice secondario globale (TournamentRegionIndex con chiavi multiattributo):

- Chiave di partizione:`tournamentId`, `region` (2 attributi)
- Chiave di ordinamento: `roundbracket`,, `matchId` (3 attributi)

Schema dell'indice secondario globale (PlayerMatchHistoryIndex con chiavi multiattributo):

- Chiave di partizione: `player1Id` (1 attributo)
- Chiave di ordinamento:`matchDate`, `round` (2 attributi)

Tabella base: TournamentMatches

MatchID (PK)	ID del torneo	region	round	staffa	ID giocatore 1	ID giocatore 2	Data della partita	vincitore	punteggio
partita-001	WINTER4	NA-EST	FINALE	CAMPIONO	101	103	2024-01-0	101	3-1
inquadrao-002	WINTER4	NA-EST	SEMIFINI	UPPER	101	105	2024-01-8	101	3-2
inquadrao-003	WINTER4	NA-EST	SEMIFINI	UPPER	103	107	2024-01-8	103	3-0
inquadrao-004	WINTER4	NA-EST	QUARTI DI FINALE	UPPER	101	109	2024-01-5	101	3-1
inquadrao-005	WINTER4	NORD-OVEST	FINALE	CAMPIONO	102	104	2024-01-0	102	3-2
inquadrao-006	WINTER4	NORD-OVEST	SEMIFINI	UPPER	102	106	2024-01-8	102	3-1
partita-007	SPRING4	NA-EST	QUARTI DI FINALE	UPPER	101	108	2024-03-5	101	3-0
inquadrao-008	SPRING4	NA-EST	QUARTI DI FINALE	LOWER	103	110	2024-03-5	103	3-2

GSI: TournamentRegionIndex (chiavi multiattributo)

ID torneo (PK)	regione (PK)	rotondo (SK)	staffa (SK)	MatchID (SK)	ID giocatore 1	ID giocatore 2	Data della partita	vincitore	punteggio
WINTER 4	NA-EST	FINALE	CAMPIO O	partita-0 01	101	103	2024-01-0	101	3-1
WINTER 4	NA-EST	QUARTI DI FINALE	UPPER	match-0(	101	109	2024-01-5	101	3-1
WINTER 4	NA-EST	SEMIFIN I	UPPER	partita-0 02	101	105	2024-01-8	101	3-2
WINTER 4	NA-EST	SEMIFIN I	UPPER	match-0(	103	107	2024-01-8	103	3-0
WINTER 4	NORD-OVEST	FINALE	CAMPIO O	partita-0 05	102	104	2024-01-0	102	3-2
WINTER 4	NORD-OVEST	SEMIFIN I	UPPER	partita-0 06	102	106	2024-01-8	102	3-1
SPRING 4	NA-EST	QUARTI DI FINALE	LOWER	partita-0 08	103	110	2024-03-5	103	3-2
SPRING 4	NA-EST	QUARTI DI FINALE	UPPER	match-0(	101	108	2024-03-5	101	3-0

GSI: PlayerMatchHistoryIndex (chiavi multiattributo)

Player1ID (PK)	MatchDa (SK)	rotondo (SK)	ID del torneo	region	staffa	MatchID	ID giocatore 2	vincitore	punteggio
101	2024-01-5	QUARTI DI FINALE	WINTER4	NA-EST	UPPER	partita-004	109	101	3-1
101	2024-01-8	SEMIFIN I	WINTER4	NA-EST	UPPER	partita-002	105	101	3-2
101	2024-01-0	FINALE	WINTER4	NA-EST	CAMPIO O	partita-001	103	101	3-1
101	2024-03-5	QUARTI DI FINALE	SPRING4	NA-EST	UPPER	partita-007	108	101	3-0
102	2024-01-8	SEMIFIN I	WINTER4	NORD-OVEST	UPPER	incontro006	106	102	3-1
102	2024-01-0	FINALE	WINTER4	NORD-OVEST	CAMPIO O	partita-005	104	102	3-2
103	2024-01-8	SEMIFIN I	WINTER4	NA-EST	UPPER	partita-003	107	103	3-0
103	2024-03-5	QUARTI DI FINALE	SPRING4	NA-EST	LOWER	partita-008	110	103	3-2

Prerequisiti

Prima di iniziare, assicurati di disporre dei seguenti elementi:

Account e autorizzazioni

- Un AWS account attivo (creane uno qui se necessario)
- Autorizzazioni IAM per le operazioni DynamoDB:
  - `dynamodb:CreateTable`
  - `dynamodb:DeleteTable`
  - `dynamodb:DescribeTable`
  - `dynamodb:PutItem`
  - `dynamodb:Query`
  - `dynamodb:BatchWriteItem`

> **Note**
>
> Nota di sicurezza: per l'uso in produzione, crea una policy IAM personalizzata con solo le autorizzazioni necessarie. Per questo tutorial, puoi utilizzare la policy AWS `AmazonDynamoDBFullAccessV2` gestita.

Ambiente di sviluppo

- Node.js installato sul tuo computer
- AWS credenziali configurate utilizzando uno di questi metodi:

Opzione 1: AWS CLI

```
aws configure
```

Opzione 2: variabili di ambiente

```
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
```

Installazione dei pacchetti richiesti

```
npm install @aws-sdk/client-dynamodb @aws-sdk/lib-dynamodb
```

Implementazione

Passaggio 1: creare una tabella GSIs utilizzando chiavi multiattributo

Crea una tabella con una struttura chiave di base semplice e GSIs che utilizzi chiavi multiattributo.

esempio di codice

```
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");
```

Principali decisioni di progettazione:

Tabella base: la tabella base utilizza una semplice chiave di `matchId` partizione per le ricerche dirette, mantenendo la struttura della tabella di base semplice e GSIs fornendo al contempo modelli di interrogazione complessi.

TournamentRegionIndex Indice secondario globale: l'indice secondario `TournamentRegionIndex` globale utilizza `tournamentId` + `region` come chiave di partizione multiattributo, creando un isolamento tra regione e torneo in cui i dati vengono distribuiti combinando l'hash di entrambi gli attributi, permettendo di eseguire query efficienti all'interno di un contesto specifico tra torneo e regione. La chiave di ordinamento multiattributo (`round`+ `bracket` +`matchId`) fornisce un ordinamento gerarchico che supporta le query a qualsiasi livello della gerarchia con un ordinamento naturale da generale (round) a specifico (match ID).

PlayerMatchHistoryIndex Indice secondario globale: il `PlayerMatchHistoryIndex` Global Secondary Index riorganizza i dati per giocatore utilizzandoli `player1Id` come chiave di partizione, abilitando le interrogazioni tra tornei per un giocatore specifico. La chiave di ordinamento

multiattributo (`matchDate+round`) fornisce un ordine cronologico con la possibilità di filtrare per intervalli di date o turni di torneo specifici.

Fase 2: Inserimento di dati con attributi nativi

Aggiungi i dati delle partite del torneo utilizzando attributi naturali. Il GSI indicizzerà automaticamente questi attributi senza richiedere chiavi sintetiche.

esempio di codice

```
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");
```

Struttura dei dati spiegata:

Utilizzo degli attributi naturali: ogni attributo rappresenta un vero concetto di torneo senza necessità di concatenazione o analisi di stringhe, e fornisce una mappatura diretta al modello di dominio.

Indicizzazione automatica dell'indice secondario globale: indicizza GSIs automaticamente gli elementi utilizzando gli attributi esistenti (`tournamentId`,,,, `matchId` for TournamentRegionIndex e `region` `round` `bracket` `player1Id``matchDate`, `round` for PlayerMatchHistoryIndex) senza richiedere chiavi concatenate sintetiche.

Non è necessario il backfill: quando aggiungi un nuovo indice secondario globale con chiavi multiattributo a una tabella esistente, DynamoDB indicizza automaticamente tutti gli elementi esistenti utilizzando i loro attributi naturali, senza bisogno di aggiornare gli elementi con chiavi sintetiche.

Fase 3: Interroga l'indice secondario globale con tutti gli attributi delle chiavi di partizione TournamentRegionIndex

Questo esempio interroga il TournamentRegionIndex Global Secondary Index che ha una chiave di partizione multiattributo (+). `tournamentId` `region` Tutti gli attributi delle chiavi di partizione devono essere specificati con condizioni di uguaglianza nelle query: non è possibile eseguire query `tournamentId` solo con o utilizzare operatori di disuguaglianza sugli attributi delle chiavi di partizione.

esempio di codice

```
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`);
});
```

Output 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
```

Interrogazioni non valide:

```
// Missing region attribute
KeyConditionExpression: 'tournamentId = :tournament'

// Using inequality on partition key attribute
KeyConditionExpression: 'tournamentId = :tournament AND #region > :region'
```

Prestazioni: le chiavi di partizione con più attributi vengono codificate insieme, fornendo le stesse prestazioni di ricerca O (1) delle chiavi ad attributo singolo.

Fase 4: Interroga le chiavi di ordinamento dell'indice secondario globale left-to-right

Gli attributi delle chiavi di ordinamento devono essere interrogati left-to-right nell'ordine in cui sono definiti nell'indice secondario globale. Questo esempio dimostra l'esecuzione di interrogazioni TournamentRegionIndex a diversi livelli gerarchici: filtraggio per just`round`, per `round` + `bracket` o per tutti e tre gli attributi chiave di ordinamento. Non è possibile saltare gli attributi al centro, ad esempio non è possibile eseguire una query per e mentre si salta. `round` `matchId` `bracket`

esempio di codice

```
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`);
}
```

Output 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
```

Left-to-right regole di interrogazione: è necessario interrogare gli attributi in ordine da sinistra a destra, senza saltarne nessuno.

Modelli validi:

- Solo primo attributo: `round = 'SEMIFINALS'`
- Primi due attributi: `round = 'SEMIFINALS' AND bracket = 'UPPER'`
- Tutti e tre gli attributi: `round = 'SEMIFINALS' AND bracket = 'UPPER' AND matchId = 'match-002'`

Schemi non validi:

- Il primo attributo viene ignorato: `bracket = 'UPPER'` (salta il giro)
- Interrogazione non corretta: `matchId = 'match-002' AND round = 'SEMIFINALS'`

- Lasciare spazi vuoti: `round = 'SEMIFINALS' AND matchId = 'match-002'` (salta la parentesi)

> **Note**
>
> Suggerimento di progettazione: ordina gli attributi chiave di ordinamento dal più generale al più specifico per massimizzare la flessibilità delle query.

Fase 5: Utilizza le condizioni di disuguaglianza nelle chiavi di ordinamento dell'Indice secondario globale

Le condizioni di disuguaglianza devono essere l'ultima condizione della ricerca. Questo esempio dimostra l'utilizzo degli operatori di confronto (>=,`BETWEEN`) e del prefisso matching (`begins_with()`) sugli attributi delle chiavi di ordinamento. Una volta utilizzato un operatore di disuguaglianza, non è possibile aggiungere ulteriori condizioni di chiave di ordinamento dopo di esso: la disuguaglianza deve essere l'ultima condizione nell'espressione della condizione chiave.

esempio di codice

```
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`);
}
```

Regole degli operatori di disuguaglianza: è possibile utilizzare gli operatori di confronto (>,,,<=) >=<, BETWEEN per le interrogazioni sugli intervalli e per la corrispondenza dei prefissi. `begins_with()` La disuguaglianza deve essere l'ultima condizione della query.

Schemi validi:

- Condizioni di uguaglianza seguite dalla disuguaglianza: `round = 'SEMIFINALS' AND bracket = 'UPPER' AND matchId > 'match-001'`
- Disuguaglianza sul primo attributo: `round BETWEEN 'QUARTERFINALS' AND 'SEMIFINALS'`
- Corrispondenza del prefisso come condizione finale: `round = 'SEMIFINALS' AND begins_with(bracket, 'U')`

Schemi non validi:

- Aggiungere condizioni dopo una disuguaglianza: `round > 'QUARTERFINALS' AND bracket = 'UPPER'`
- Utilizzo di più disuguaglianze: `round > 'QUARTERFINALS' AND bracket > 'L'`

> **Important**
>
> `begins_with()`viene considerata una condizione di disuguaglianza, quindi non può essere seguita da alcuna condizione di chiave di ordinamento aggiuntiva.

Fase 6: Interrogazione dell'indice secondario PlayerMatchHistoryIndex globale con una chiave di ordinamento multiattributo

Questo esempio interroga il PlayerMatchHistoryIndex che ha una chiave di partizione singola (`player1Id`) e una chiave di ordinamento con più attributi (+). `matchDate round` Ciò consente l'analisi tra tornei interrogando tutte le partite di un giocatore specifico senza conoscere il torneo, IDs mentre la tabella base richiederebbe interrogazioni separate per combinazione torneo-regione.

esempio di codice

```
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`);
```

Variazioni del modello

Dati di serie temporali con chiavi multiattributo

Ottimizzazione per le query su serie temporali con attributi temporali gerarchici

esempio di codice

```
{
    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
```

Vantaggi: la gerarchia temporale naturale (anno → mese → giorno → timestamp) consente una granularità delle query efficienti in qualsiasi momento senza analisi o manipolazione della data. L'indice secondario globale indicizza automaticamente tutte le letture utilizzando i loro attributi temporali naturali.

Ordini di e-commerce con chiavi multiattributo

Tieni traccia degli ordini con più dimensioni

esempio di codice

```
{
    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
```

Dati organizzativi gerarchici

Gerarchie organizzative modello

esempio di codice

```
{
    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
```

Chiavi sparse con più attributi

Combina chiavi multiattributo per creare un GSI sparso

esempio di codice

```
{
    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
```

Multi-tenancy SaaS

Piattaforma SaaS multi-tenant con isolamento del cliente

esempio di codice

```
// 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'
    }
}));
```

Vantaggi: interrogazioni efficienti nel contesto tra inquilino e cliente e organizzazione naturale dei dati.

Transazioni finanziarie

Sistema bancario che monitora le transazioni del conto utilizzando GSIs

esempio di codice

```
// 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'
    }
}));
```

Esempio completo

L'esempio seguente illustra le chiavi con più attributi dalla configurazione alla pulizia:

esempio di codice

```
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);
```

Scaffold di codice minimo

esempio di codice

```
// 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' }
```

Risorse aggiuntive

- Best practice per DynamoDB
- Lavorare con tabelle e dati
- Indici secondari globali
- Operazioni di interrogazione e scansione

Gestione degli indici secondari globali in DynamoDB

In questa sezione viene descritto come creare, modificare ed eliminare indici secondari globali in Amazon DynamoDB.

Argomenti

- Creazione di una tabella con indici secondari globali
- Descrizione degli indici secondari globali su una tabella
- Aggiunta di un indice secondario globale a una tabella esistente
- Eliminazione di un indice secondario globale
- Modifica di un indice secondario globale durante la creazione

Creazione di una tabella con indici secondari globali

Per creare una tabella con uno o più indici secondari globali, utilizza l'operazione `CreateTable` con il parametro `GlobalSecondaryIndexes`. Per una massima flessibilità delle query, è possibile creare fino a 20 indici secondari globali (quota predefinita) per tabella.

Devi specificare un attributo che funga da chiave di partizione dell'indice. Facoltativamente, puoi specificare un altro attributo per la chiave di ordinamento dell'indice. Non è necessario che questi attributi di chiave coincidano con quelli della tabella. Ad esempio, nella tabella GameScores (consulta Utilizzo degli indici secondari globali in DynamoDB), né `TopScore` né `TopScoreDateTime` sono attributi chiave. È possibile creare un indice secondario globale con una chiave di partizione `TopScore` e una chiave di ordinamento `TopScoreDateTime`. Puoi utilizzare questo indice per determinare se esiste una correlazione tra punteggi elevati e il periodo del giorno in cui si svolge un gioco.

Ogni attributo di chiave dell'indice deve essere uno scalare di tipo `String`, `Number` o `Binary` (non può essere un tipo documento o set). È possibile proiettare gli attributi di qualsiasi tipo di dati in un

indice secondario globale. Questo include scalari, documenti e set. Per l'elenco completo dei tipi di dati, consulta Tipi di dati.

Se utilizzi la modalità assegnata, devi specificare le impostazioni `ProvisionedThroughput` per l'indice, consistenti di `ReadCapacityUnits` e `WriteCapacityUnits`. Queste impostazioni di throughput assegnato sono distinte da quelle della tabella, ma il funzionamento è analogo. Per ulteriori informazioni, consulta Considerazioni sulla velocità di trasmissione effettiva assegnata per indici secondari globali.

Gli indici secondari globali ereditano la modalità di capacità in lettura/scrittura dalla tabella di base. Per ulteriori informazioni, consulta Considerazioni sul passaggio tra modalità di capacità in DynamoDB.

> **Note**
>
> Le operazioni di backfill e le operazioni di scrittura in corso condividono la velocità effettiva di scrittura all'interno dell'indice secondario globale. Quando si crea un nuovo GSI, può essere importante verificare se la chiave di partizione scelta sta producendo una distribuzione irregolare o ristretta di dati o traffico attraverso i valori chiave della partizione del nuovo indice. In questo caso, è possibile che si verifichino operazioni di backfill e scrittura contemporaneamente e limitando le scritture nella tabella di base. Il servizio adotta misure per ridurre al minimo il potenziale di questo scenario, ma non ha alcuna visione della forma dei dati dei clienti in relazione alla chiave di partizione dell'indice, alla proiezione scelta o alla scarsità della chiave primaria dell'indice.
> Se si sospetta che il nuovo indice secondario globale possa avere dati limitati o disallineati o distribuzione del traffico tra i valori chiave delle partizioni, considerare quanto segue prima di aggiungere nuovi indici alle tabelle di importanza operativa.
>
> - Potrebbe essere più sicuro aggiungere l'indice in un momento in cui l'applicazione guida la quantità minima di traffico.
> - Prendere in considerazione l'abilitazione di CloudWatch Contributor Insights sulla tabella di base e sugli indici. Questo ti darà informazioni preziose sulla distribuzione del traffico.
> - Per tabelle e indici di base in modalità capacità con provisioning, impostare la capacità di scrittura con provisioning del nuovo indice su almeno il doppio di quella della tabella di base. Guarda `WriteThrottleEvents`, `ThrottledRequests`, `OnlineIndexPercentageProgress`, `OnlineIndexConsumedWriteCapacity` e i parametri CloudWatch `OnlineIndexThrottleEvents` in tutto il processo. Regola la capacità di scrittura fornita in base alle esigenze per completare il backfill in un tempo

ragionevole senza effetti significativi di limitazione (della larghezza di banda della rete) sulle operazioni in corso.

- Prepararsi ad annullare la creazione dell'indice se si verifica un impatto operativo dovuto alla limitazione (della larghezza di banda della rete) della scrittura e l'aumento della capacità di scrittura con provisioning nel nuovo GSI non lo risolve.

Descrizione degli indici secondari globali su una tabella

Per visualizzare lo stato di tutti gli indici secondari globali su una tabella, utilizza l'operazione `DescribeTable`. La parte `GlobalSecondaryIndexes` della risposta mostra tutti gli indici della tabella insieme allo stato corrente di ciascuno (`IndexStatus`).

`IndexStatus` per un indice secondario globale sarà uno dei seguenti:

- `CREATING`: l'indice è in fase di creazione e non è ancora disponibile per l'uso.
- `ACTIVE`: l'indice è pronto per l'uso e le applicazioni possono eseguire operazioni `Query` sull'indice.
- `UPDATING`: vengono modificate le impostazioni di velocità effettiva assegnata dell'indice.
- `DELETING`: l'indice è attualmente in fase di eliminazione e non può essere più utilizzato.

Quando DynamoDB ha terminato la creazione di un indice secondario globale, lo stato dell'indice cambia da `CREATING` a `ACTIVE`.

Aggiunta di un indice secondario globale a una tabella esistente

Per aggiungere un indice secondario globale a una tabella esistente, utilizza l'operazione `UpdateTable` con il parametro `GlobalSecondaryIndexUpdates`. Devi specificare quanto segue:

- Un nome per l'indice. Il nome deve essere univoco tra tutti gli indici della tabella.
- Lo schema delle chiavi dell'indice. Devi specificare un attributo per la chiave di partizione dell'indice; opzionalmente, puoi specificare un altro attributo per la chiave di ordinamento dell'indice. Non è necessario che questi attributi di chiave coincidano con quelli della tabella. I tipi di dati per ogni attributo dello schema devono essere scalari: `String`, `Number` o `Binary`.
- Gli attributi da proiettare dalla tabella all'indice:
  - `KEYS_ONLY`: ogni elemento dell'indice è costituito solo dalla chiave di partizione della tabella e dai valori della chiave di ordinamento, oltre ai valori della chiave di indice.

- `INCLUDE`: oltre agli attributi descritti in `KEYS_ONLY`, l'indice secondario include gli altri attributi non chiave che sono stati specificati.
- `ALL`: l'indice include tutti gli attributi della tabella di origine.

- Le impostazioni di throughput assegnato dell'indice, consistenti di `ReadCapacityUnits` e `WriteCapacityUnits`. Queste impostazioni di throughput assegnato sono distinte da quelle della tabella.

È possibile creare un solo indice secondario globale per operazione `UpdateTable`.

> **Important**
>
> Quando si crea un indice secondario globale (GSI), le operazioni di scrittura sulla tabella di base possono essere sottoposte a limitazione (della larghezza di banda della rete) se l'attività dei GSI risultante dalle scritture sulla tabella di base supera la capacità di scrittura allocata del GSI. Questa limitazione (della larghezza di banda della rete) influisce su tutte le operazioni di scrittura, dal processo di indicizzazione alla potenziale interruzione dei carichi di lavoro di produzione. Per ulteriori informazioni, consulta Risoluzione dei problemi di limitazione (della larghezza di banda della rete) in Amazon DynamoDB.

Fasi della creazione di un indice

Quando si aggiunge un nuovo indice secondario locale a una tabella esistente, mentre l'indice viene creato la tabella continua a essere disponibile. Tuttavia, il nuovo indice non è disponibile per le operazioni Query finché il suo stato non cambia da `CREATING` ad `ACTIVE`.

> **Note**
>
> La creazione di un indice secondario globale non utilizza Application Auto Scaling. L'aumento della capacità di `MIN` Application Auto Scaling non diminuisce il tempo di creazione dell'indice secondario globale.

Dietro le quinte, DynamoDB crea l'indice in due fasi:

Allocazione delle risorse

DynamoDB alloca le risorse di calcolo e archiviazione necessarie per la creazione dell'indice.

Durante la fase di allocazione delle risorse, l'attributo `IndexStatus` è `CREATING` e l'attributo `Backfilling` è false. Utilizza l'operazione `DescribeTable` per recuperare lo stato di una tabella e di tutti i suoi indici secondari.

Mentre l'indice si trova nella fase di allocazione delle risorse, non puoi eliminare l'indice o la sua tabella padre. Inoltre, non puoi modificare il throughput assegnato dell'indice o della tabella. Non puoi aggiungere o eliminare altri indici nella tabella. Tuttavia, puoi modificare il throughput assegnato di questi altri indici.

Compilazione

Per ogni elemento nella tabella, DynamoDB determina quale set di attributi scrivere sull'indice in base alla relativa proiezione (`KEYS_ONLY`, `INCLUDE` o `ALL`). Procede quindi a scrivere questi attributi nell'indice. Durante la fase di backfill, DynamoDB tiene traccia degli elementi che vengono aggiunti, eliminati o aggiornati nella tabella. Anche gli attributi di questi item vengono aggiunti, eliminati o aggiornati nell'indice secondo il caso.

Durante la fase di compilazione, l'attributo `IndexStatus` è impostato su `CREATING` e l'attributo `Backfilling` è true. Utilizza l'operazione `DescribeTable` per recuperare lo stato di una tabella e di tutti i suoi indici secondari.

Mentre l'indice è in fase di compilazione, non puoi eliminare la tabella padre. Tuttavia, puoi comunque eliminare l'indice o modificare il throughput assegnato della tabella e di qualsiasi suoi indici secondari globali.

> **Note**
>
> Nella fase di compilazione, alcune scritture di item in violazione possono riuscire mentre altre vengono rifiutate. Dopo la compilazione, tutte le scritture negli elementi che violano lo schema delle chiavi del nuovo indice vengono rifiutate. È consigliabile eseguire lo strumento Rilevamento violazioni alla fine della fase di backfill per rilevare e risolvere le eventuali violazioni delle chiavi che possono essersi verificate. Per ulteriori informazioni, consulta Rilevamento e correzione delle violazioni delle chiavi in DynamoDB.

Mentre le fasi di allocazione delle risorse e compilazione sono in corso, l'indice si trova nello stato `CREATING`. Durante questo periodo, DynamoDB esegue operazioni di lettura sulla tabella. Le operazioni di lettura dalla tabella di base per popolare l'indice secondario globale non vengono addebitate. Tuttavia, vengono addebitate le operazioni di scrittura per popolare l'indice secondario globale appena creato.

Quando la creazione è terminata, lo stato dell'indice diventa `ACTIVE`. Non puoi eseguire operazioni di `Query` o `Scan` dell'indice finché è `ACTIVE`.

> **Note**
>
> In alcuni casi DynamoDB non è in grado di scrivere dati dalla tabella all'indice a causa di violazioni delle chiavi di indice. Ciò può verificarsi se:
>
> - Il tipo di dati di un valore di attributo non corrisponde al tipo di dati di un tipo di dati dello schema della chiave dell'indice.
> - La dimensione di un attributo supera la lunghezza massima di un attributo della chiave dell'indice.
> - Un attributo della chiave dell'indice ha un valore di attributo String vuoto o Binary vuoto.
>
> Le violazioni delle chiavi dell'indice non interferiscono con la creazione di un indice secondario. Tuttavia, quando l'indice diventa `ACTIVE`, le chiavi in violazione non sono presenti nell'indice.
> DynamoDB fornisce uno strumento autonomo per individuare e risolvere questi problemi. Per ulteriori informazioni, consulta Rilevamento e correzione delle violazioni delle chiavi in DynamoDB.

Aggiunta di un indice secondario globale a una tabella di grandi dimensioni

Il tempo necessario per creare un indice secondario globale dipende da diversi fattori, tra cui:

- Le dimensioni della tabella
- Il numero di item nella tabella idonei per essere inclusi nell'indice
- Il numero di attributi proiettati nell'indice
- La capacità di scrittura assegnata dell'indice
- L'attività di scrittura sulla tabella principale durante la creazione dell'indice

Se si aggiunge un indice secondario globale a una tabella molto grande, il processo di creazione potrebbe richiedere molto tempo. Per monitorare lo stato di avanzamento e determinare se l'indice dispone di capacità di scrittura sufficiente, consulta i seguenti parametri Amazon CloudWatch:

- `OnlineIndexPercentageProgress`

- `OnlineIndexConsumedWriteCapacity`
- `OnlineIndexThrottleEvents`

Per ulteriori informazioni sui parametri CloudWatch correlati a DynamoDB, consulta Parametri di DynamoDB.

> **Important**
>
> Potrebbe essere necessario consentire l'inserimento di tabelle molto grandi prima di creare o aggiornare un indice secondario globale. Contatta il Supporto AWS per consentire l'inserimento delle tabelle.

> **Note**
>
> In linea generale, è consigliabile impostare la capacità di scrittura assegnata all'indice su 1,5 volte la capacità di scrittura della tabella. Questa impostazione è valida per molti casi d'uso. I requisiti effettivi tuttavia possono essere maggiori o minori.

Durante il backfill di un indice, DynamoDB utilizza la capacità interna del sistema per leggere dalla tabella. Ciò ha lo scopo di ridurre al minimo l'impatto della creazione dell'indice e di assicurare che la tabella non esaurisca la capacità di lettura.

Tuttavia è possibile che il volume dell'attività di scrittura in entrata superi la capacità assegnata dell'indice. Questo è un caso di collo di bottiglia, in cui la creazione dell'indice richiede più tempo a causa del throttling dell'attività di scrittura nell'indice. Durante la creazione dell'indice è consigliabile monitorare i parametri Amazon CloudWatch dell'indice per verificare se la capacità di scrittura utilizzata stia superando quella assegnata. In uno scenario di collo di bottiglia, devi aumentare la capacità di scrittura assegnata dell'indice per evitare il throttling della scrittura in fase di compilazione.

Dopo la creazione dell'indice, devi impostare la sua capacità di scrittura assegnata in modo da riflettere il normale utilizzo dell'applicazione.

Eliminazione di un indice secondario globale

Se l'indice secondario globale non è più necessario, è possibile eliminarlo utilizzando l'operazione `UpdateTable`.

È possibile eliminare un solo indice secondario globale per operazione `UpdateTable`.

Mentre l'indice secondario globale viene eliminato, non vi sono effetti sull'attività di lettura o scrittura nella tabella padre. Durante l'eliminazione è comunque possibile modificare il throughput assegnato di altri indici.

> **Note**
>
> - Quando elimini una tabella utilizzando l'operazione `DeleteTable`, vengono eliminati anche tutti gli indici secondari globali della tabella.
> - L'operazione di eliminazione dell'indice secondario globale non verrà addebitata sul tuo account.

Modifica di un indice secondario globale durante la creazione

Nel corso della creazione di un indice puoi utilizzare l'operazione `DescribeTable` per determinare in quale fase si trova. La descrizione dell'indice include un attributo booleano, `Backfilling`, che indica se DynamoDB sta attualmente caricando l'indice con elementi della tabella. Se `Backfilling` è true, la fase di allocazione delle risorse è terminata ed è in corso la compilazione dell'indice.

Mentre la compilazione è in corso, puoi aggiornare i parametri di throughput assegnato dell'indice. Questa scelta può essere determinata dalla volontà di accelerare la creazione. Puoi incrementare la capacità di scrittura dell'indice mentre viene creato, per ridurla successivamente. Per modificare le impostazioni di throughput assegnato dell'indice, utilizza l'operazione `UpdateTable`. Lo stato dell'indice cambia in `UPDATING` e `Backfilling` è true finché l'indice non è pronto per l'uso.

Durante la fase di compilazione, puoi eliminare l'indice in corso di creazione. Durante questa fase, non puoi aggiungere o eliminare altri indici nella tabella.

> **Note**
>
> Per gli indici creati nell'ambito di un'operazione `CreateTable`, l'attributo `Backfilling` non compare nell'output di `DescribeTable`. Per ulteriori informazioni, consulta Fasi della creazione di un indice.

Rilevamento e correzione delle violazioni delle chiavi in DynamoDB

Durante la fase di backfill della creazione dell'indice secondario globale, Amazon DynamoDB esamina ogni elemento nella tabella per determinare se è idoneo all'inclusione nell'indice. Alcuni elementi potrebbero non essere idonei perché causerebbero violazioni della chiave dell'indice. In questi casi, gli elementi rimangono nella tabella, ma l'indice non ha una voce corrispondente per tale elemento.

Una violazione della chiave di indice si verifica nelle seguenti situazioni:

- È presente una mancata corrispondenza del tipo di dati tra un valore di attributo e il tipo di dati dello schema della chiave dell'indice. Si supponga, ad esempio, che uno degli elementi nella tabella `GameScores` aveva un valore `TopScore` di tipo `String`. Se è stato aggiunto un indice secondario globale con una chiave di partizione di `TopScore`, di tipo `Number`, l'elemento della tabella violerebbe la chiave di indice.
- Il valore di un attributo supera la lunghezza massima di un attributo della chiave dell'indice. La lunghezza massima di una chiave di partizione è 2048 byte e la lunghezza massima di una chiave di ordinamento è 1024 byte. Se uno qualsiasi dei valori di attributo corrispondenti nella tabella supera questi limiti, l'elemento della tabella violerebbe la chiave di indice.

> **Note**
>
> Se un valore di attributo String o Binary è impostato per un attributo utilizzato come chiave di indice, il valore dell'attributo deve avere una lunghezza maggiore di zero; in caso contrario, l'elemento della tabella violerebbe la chiave di indice.
> Questo strumento non contrassegna questa violazione della chiave di indice, al momento.

Se si verifica una violazione della chiave di indice, la fase di backfill continua senza interruzioni. Tuttavia, gli elementi che violano non saranno inclusi nell'indice. Una volta completata la fase di backfill, tutte le scritture sugli elementi che violano lo schema delle chiavi del nuovo indice saranno rifiutate.

Per identificare e correggere i valori degli attributi in una tabella che violano una chiave di indice, utilizza lo strumento Rilevatore di violazioni. Per eseguire Rilevatore di violazioni, è necessario creare un file di configurazione che specifica il nome di una tabella da sottoporre a scansione, i nomi e i tipi di dati della chiave di partizione dell'indice secondario globale e della chiave di ordinamento e quali

azioni intraprendere se vengono rilevate violazioni della chiave di indice. Rilevatore di violazioni può essere eseguito in una delle due diverse modalità:

- Modalità di rilevamento: rileva le violazioni della chiave dell'indice. Utilizza la modalità di rilevamento per segnalare gli elementi della tabella che causerebbero violazioni chiave in un indice secondario globale. Facoltativamente, è possibile richiedere che questi elementi di tabella che violano vengano eliminati immediatamente quando vengono rilevati. L'output dalla modalità di rilevamento viene scritto in un file, che è possibile utilizzare per ulteriori analisi.
- Modalità di correzione: correggere le violazioni della chiave di indice. In modalità di correzione, Rilevatore violazioni legge un file di input con lo stesso formato del file di output dalla modalità di rilevamento. La modalità di correzione legge i record dal file di input e, per ogni record, elimina o aggiorna gli elementi corrispondenti nella tabella. Se si sceglie di aggiornare gli elementi, è necessario modificare il file di input e impostare i valori appropriati per questi aggiornamenti.

Download ed esecuzione di Rilevatore violazioni

Rilevatore violazioni è disponibile come file eseguibile Java Archive (`.jar`) e viene eseguito su computer Windows, macOS o Linux. Rilevatore violazioni richiede Java 1.7 (o versioni successive) e Apache Maven.

- Download di Rilevatore violazioni da GitHub

Seguire le istruzioni riportate nel file `README.md` per scaricare e installare Rilevatore violazioni tramite Maven.

Per avviare Rilevatore violazioni, passare alla directory dove è stato creato `ViolationDetector.java` ed immettere il comando seguente.

```
java -jar ViolationDetector.jar [options]
```

La riga di comando di Rilevatore violazioni accetta le seguenti opzioni:

- `-h | --help`: stampa un riepilogo di utilizzo e le opzioni per Rilevatore violazioni.
- `-p | --configFilePath value`: il nome completo di un file di configurazione di Rilevatore violazioni. Per ulteriori informazioni, consulta File di configurazione di Rilevatore violazioni.
- `-t | --detect value`: rileva le violazioni della chiave di indice nella tabella e le scrive nel file di output di Rilevatore violazioni. Se il valore di questo parametro è impostato su `keep`, gli elementi

con violazioni della chiave non vengono modificati. Se il valore è impostato su `delete`, gli elementi con violazioni della chiave vengono eliminati dalla tabella.

- `-c | --correct value`: legge le violazioni della chiave di indice da un file di input ed esegue azioni correttive sugli elementi della tabella. Se il valore di questo parametro è impostato su `update`, gli elementi con violazioni della chiave vengono aggiornati con valori nuovi e che non violano. Se il valore è impostato su `delete`, gli elementi con violazioni della chiave vengono eliminati dalla tabella.

File di configurazione di Rilevatore violazioni

Durante il runtime, lo strumento Rilevatore violazioni richiede un file di configurazione. I parametri di questo file determinano a quali risorse DynamoDB può accedere Rilevatore violazioni e la velocità effettiva assegnata che può utilizzare. Nella tabella seguente vengono descritti questi parametri.

Nome del parametro	Descrizione	Obbligatorio?

`awsCredentialsFile`	Il nome completo di un file contenente le credenziali AWS. Il file delle credenziali deve avere il seguente formato:
	`accessKey = access_key_id_goes_here`
	`secretKey = secret_key_goes_here`
		Sì

`dynamoDBRegion`	La regione AWS in cui si trova la tabella. Ad esempio: `us-west-2` .	Sì
`tableName`	Il nome della tabella DynamoDB da sottoporre a scansione.	Sì
`gsiHashKeyName`	Il nome della chiave della partizione dell'indice.	Sì

Nome del parametro	Descrizione	Obbligatorio?
`gsiHashKeyType`	Il tipo di dati della chiave di partizione dell'indice, `String`, `Number` o `Binary`: `S | N | B`	Sì
`gsiRangeKeyName`	Il nome della chiave di ordinamento dell'indice. Non specificare questo parametro se l'indice dispone solo di una chiave primaria semplice (chiave di partizione).	No
`gsiRangeKeyType`	Il tipo di dati della chiave di ordinamento dell'indice, `String`, `Number` o `Binary`: `S | N | B` Non specificare questo parametro se l'indice dispone solo di una chiave primaria semplice (chiave di partizion e).	No
`recordDetails`	Indica se scrivere i dettagli completi delle violazioni della chiave di indice nel file di output. Se impostato su `true` (impostazione predefinita), vengono riportate tutte le informazioni sugli elementi che violano. Se impostato su `false`, viene riportato solo il numero di violazioni.	No

Nome del parametro	Descrizione	Obbligatorio?
`recordGsiValueInViolationRecord`	Indica se scrivere i valori delle chiavi di indice che violano nel file di output. Se impostato su `true` (impostazione predefinita), vengono riportati i valori chiave. Se impostato su `false` (impostazione predefinita), i valori chiave non vengono riportati.	No

Nome del parametro	Descrizione	Obbligatorio?

`detectionOutputPath`	Il percorso completo del file di output di Rilevatore violazioni. Questo parametro supporta la scrittura in una directory locale o in Amazon Simple Storage Service (Amazon S3). Di seguito vengono mostrati gli esempi:
	
	`detectionOutputPath = //local/path/filename.csv`
	
	`detectionOutputPath = s3://bucket/filename.csv`
	
	Le informazioni nel file di output vengono visualizzate in formato CSV (valori separati da virgola). Se non si imposta `detectionOutputPath`, il file di output è denominato `violation_detection.csv` e viene scritto nella directory di lavoro corrente.
		No

Nome del parametro	Descrizione	Obbligatorio?
`numOfSegments`	Il numero di segmenti di scansione parallela da utilizzare quando Rilevatore violazioni esegue la scansione della tabella. Il valore predefinito è 1, che indica che la tabella viene scansionata in modo sequenziale. Se il valore è 2 o superiore, Rilevatore violazioni divide la tabella in tanti segmenti logici e un numero uguale di thread di scansione. L'impostazione massima per `numOfSegments` è 4.096. Per le tabelle più grandi, una scansione parallela è generalmente più veloce di una scansione sequenziale. Inoltre, se la tabella è abbastanza grande da estendersi su più partizioni, una scansione parallela distribuisce l'attività di lettura in modo uniforme su più partizioni. Per ulteriori informazioni sulle scansioni parallele in DynamoDB, consulta Scansione parallela.	No

Nome del parametro	Descrizione	Obbligatorio?
`numOfViolations`	Il limite superiore delle violazioni della chiave di indice da scrivere nel file di output. Se impostato su `-1` (impostazione predefinita), viene scansionata l'intera tabella. Se impostato su un numero intero positivo, Rilevatore violazioni si interrompe dopo il rilevamento di tale numero di violazioni.	No
`numOfRecords`	Il numero di elementi nella tabella da sottoporre a scansione. Se impostato su -1 (impostazione predefinita), viene scansionata l'intera tabella. Se impostato su un numero intero positivo, Rilevatore violazioni si interrompe dopo la scansione di molti elementi nella tabella.	No
`readWriteIOPSPercent`	Regola la percentuale di unità di capacità di lettura assegnata utilizzate durante la scansione della tabella. I valori validi sono compresi tra `1` e `100`. Il valore predefinito (`25`) significa che Rilevatore violazioni non consumerà più del 25% della velocità effettiva di lettura assegnata della tabella.	No

Nome del parametro	Descrizione	Obbligatorio?
`correctionInputPath`	Il percorso completo del file di output di correzione di Rilevatore violazioni. Se si esegue Rilevatore violazioni in modalità di correzione, il contenuto di questo file viene utilizzato per modificare o eliminare elementi di dati nella tabella che violano l'indice secondario globale. Il formato del file `correctionInputPath` è uguale a quello del file `detectionOutputPath`. Ciò consente di elaborare l'output dalla modalità di rilevamento come input in modalità di correzione.	No

Nome del parametro	Descrizione	Obbligatorio?

`correctionOutputPath`	Il percorso completo del file di output di correzione di Rilevatore violazioni. Questo file viene creato solo se ci sono errori di aggiornamento.
	
	Questo parametro supporta la scrittura in una directory locale o su Amazon S3. Di seguito vengono mostrati gli esempi:
	
	`correctionOutputPath = //local/path/filename.csv`
	
	`correctionOutputPath = s3://bucket/filename.csv`
	
	Le informazioni nel file di output vengono visualizzate in formato CSV. Se non si imposta `correctionOutputPath`, il file di output è denominato `violation_update_errors.csv` e viene scritto nella directory di lavoro corrente.
		No

Rilevamento

Per rilevare le violazioni della chiave di indice, utilizza Rilevatore violazioni con l'opzione `--detect` della riga di comando. Per mostrare come funziona questa opzione, si consideri la tabella

`ProductCatalog`. Di seguito è riportato un elenco di elementi nella tabella. Sono visualizzati solo la chiave primaria (`Id`) e l'attributo `Price`.

ID (chiave primaria)	Prezzo
101	5
102	20
103	200
201	100
202	200
203	300
204	400
205	500

Tutti i valori per `Price` sono di tipo `Number`. Tuttavia, poiché DynamoDB è senza schemi, è possibile aggiungere un elemento con un `Price` non numerico. Ad esempio, si supponga di aggiungere un altro elemento alla tabella `ProductCatalog`.

ID (chiave primaria)	Prezzo
999	"Hello"

La tabella ha ora un totale di nove elementi.

A questo punto aggiungere un nuovo indice secondario globale alla tabella: `PriceIndex`. La chiave primaria di questo indice è una chiave di partizione, `Price`, che è di tipo `Number`. Dopo che l'indice è stato creato, conterrà otto elementi, ma la tabella `ProductCatalog` ne ha nove. La ragione di questa discrepanza è che il valore `"Hello"` è di tipo `String`, ma `PriceIndex` ha una chiave primaria di tipo `Number`. Il valore `String` viola la chiave dell'indice secondario globale, quindi non è presente nell'indice.

Per utilizzare Rilevatore violazioni in questo scenario, è innanzitutto necessario creare un file di configurazione come il seguente.

```
# 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
```

Quindi, è possibile eseguire Rilevatore violazioni come nell'esempio seguente.

```
$  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
```

Se il parametro di configurazione `recordDetails` è impostato su `true`, Rilevatore violazioni scrive i dettagli di ogni violazione nel file di output, come nell'esempio seguente.

```
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,,
```

Il file di output è in formato CSV. La prima riga del file è un'intestazione, seguita da un record per elemento che viola la chiave di indice. I campi di questi record di violazione sono i seguenti:

- Chiave hash tabella: il valore della chiave di partizione dell'elemento nella tabella.
- Chiave di intervallo tabella: il valore della chiave di ordinamento dell'elemento nella tabella.
- Valore chiave hash GSI: il valore della chiave di partizione dell'indice secondario globale.
- Tipo di violazione della chiave hash GSI: `Type Violation` o `Size Violation`.
- Descrizione della violazione della chiave hash GSI: la causa della violazione.
- Valore di aggiornamento della chiave hash GSI (PER UTENTE): in modalità di correzione, un nuovo valore fornito dall'utente per l'attributo.
- Valore chiave hash GSI: il valore della chiave di ordinamento dell'indice secondario globale.
- Tipo di violazione della chiave di intervallo GSI: `Type Violation` o `Size Violation`.
- Descrizione della violazione della chiave di intervallo GSI: la causa della violazione.
- Valore di aggiornamento della chiave di intervallo GSI (PER UTENTE): in modalità di correzione, un nuovo valore fornito dall'utente per l'attributo.
- Elimina attributo vuoto durante l'aggiornamento (S/N): in modalità di correzione, determina se eliminare (S) o mantenere (N) l'elemento di violazione nella tabella, ma solo se uno dei seguenti campi è vuoto:
  - `GSI Hash Key Update Value(FOR USER)`
  - `GSI Range Key Update Value(FOR USER)`

  Se uno di questi campi non è vuoto, allora `Delete Blank Attribute When Updating(Y/N)` non ha effetto.

> **Note**
>
> Il formato dell'output può variare a seconda del file di configurazione e delle opzioni della riga di comando. Ad esempio, se la tabella ha una chiave primaria semplice (senza una chiave di ordinamento), nell'output non saranno presenti campi della chiave di ordinamento.
> I record di violazione nel file potrebbero non essere ordinati.

Correzione

Per correggere le violazioni della chiave di indice, utilizza Rilevatore violazioni con l'opzione `--correct` della riga di comando. In modalità di correzione, Rilevatore violazioni legge il file di input specificato dal parametro `correctionInputPath`. Il file ha lo stesso formato del file

`detectionOutputPath`, in modo da poter utilizzare l'output del rilevamento come input per la correzione.

Rilevatore violazioni fornisce due modi diversi per correggere le violazioni della chiave di indice:

- Elimina le violazioni: eliminare gli elementi della tabella che hanno valori di attributi che violano.
- Aggiorna violazioni: aggiornare gli elementi della tabella, sostituendo gli attributi che violano con valori che non violano.

In entrambi i casi, è possibile utilizzare il file di output dalla modalità di rilevamento come input per la modalità di correzione.

Continuando con l'esempio `ProductCatalog`, si supponga di voler eliminare l'elemento che viola dalla tabella. A tale scopo, utilizza la seguente riga di comando.

```
$  java -jar ViolationDetector.jar --configFilePath config.txt --correct delete
```

A questo punto, verrà richiesto di confermare se desideri eliminare gli elementi che violano.

```
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
```

Ora sia `ProductCatalog` che `PriceIndex` hanno lo stesso numero di elementi.

Utilizzo di indici secondari globali: Java

È possibile utilizzare l'API documento AWS SDK per Java per creare una tabella Amazon DynamoDB con uno o più indici secondari globali, descrivere gli indici sulla tabella ed eseguire query utilizzando gli indici.

Di seguito sono riportate le fasi comuni per le operazioni di tabella.

1. Creare un'istanza della classe `DynamoDB`.
2. Fornisci i parametri obbligatori e facoltativi per l'operazione creando gli oggetti di richiesta corrispondenti.
3. Chiama il metodo appropriato fornito dal client creato nella fase precedente.

Argomenti

- Creazione di una tabella con un indice secondario globale
- Descrizione di una tabella con un indice secondario globale
- Esecuzione di query su un indice secondario globale
- Esempio: indici secondari globali che utilizzano l'API del documento AWS SDK per Java

Creazione di una tabella con un indice secondario globale

Puoi creare indici secondari globali al momento della creazione di una tabella. A tale scopo, utilizza `CreateTable` e fornisci le specifiche per uno o più indici secondari globali. Il seguente esempio di codice Java crea una tabella per contenere le informazioni sui dati meteo. La chiave di partizione è `Location` e la chiave di ordinamento è `Date`. Un indice secondario globale denominato `PrecipIndex` consente di accedere rapidamente ai dati delle precipitazioni di vari luoghi.

Di seguito sono riportate le fasi per creare una tabella con un indice secondario globale, utilizzando l'API documento di DynamoDB.

1. Creare un'istanza della classe `DynamoDB`.
2. Crea un'istanza della classe `CreateTableRequest` per fornire le informazioni della richiesta.

   È necessario fornire il nome della tabella, la sua chiave primaria e i valori del throughput assegnato. Per l'indice secondario globale, è necessario fornire il nome dell'indice, le impostazioni di velocità effettiva assegnata, le definizioni degli attributi per la chiave di ordinamento dell'indice, lo schema della chiave per l'indice e la proiezione degli attributi.
3. Chiama il metodo `createTable` fornendo l'oggetto richiesta come parametro.

Il seguente esempio di codice Java mostra le fasi precedenti. Il codice crea una tabella (`WeatherData`) con un indice secondario globale (`PrecipIndex`). La chiave di partizione dell'indice è `Date` e la sua chiave di ordinamento è `Precipitation`. Tutti gli attributi della tabella vengono proiettati nell'indice. Gli utenti possono eseguire query su questo indice per ottenere dati sul meteo di una data specifica, ordinando facoltativamente i dati per quantità di precipitazioni.

Poiché `Precipitation` non è un attributo chiave per la tabella, non è obbligatorio. Tuttavia, item `WeatherData` senza `Precipitation` non vengono visualizzati in `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());
```

È necessario attendere fino a quando DynamoDB crea la tabella e imposta lo stato su `ACTIVE`. Dopodiché, puoi iniziare a inserire item di dati nella tabella.

Descrizione di una tabella con un indice secondario globale

Per ottenere informazioni sugli indici secondari globali in una tabella, utilizza `DescribeTable`. Puoi accedere al nome, allo schema della chiave e agli attributi proiettati di ciascun indice.

Di seguito sono riportate le fasi per accedere alle informazioni dell'indice secondario globale su una tabella.

1. Creare un'istanza della classe `DynamoDB`.
2. Crea un'istanza della classe `Table` per rappresentare l'indice con cui intendi lavorare.
3. Chiama il metodo `describe` per l'oggetto `Table`.

Il seguente esempio di codice Java mostra le fasi precedenti.

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());
    }
}
```

Esecuzione di query su un indice secondario globale

È possibile utilizzare l'operazione `Query` su un indice secondario globale quasi nello stesso modo in cui si esegue una `Query` su una tabella. È necessario specificare il nome dell'indice, i criteri di query per la chiave di partizione e di ordinamento dell'indice (se presente) e gli attributi da restituire. In questo esempio, l'indice è `PrecipIndex`, avente una chiave di partizione `Date` e una chiave di ordinamento `Precipitation`. La query di indice restituisce tutti i dati sul meteo di una data specifica in cui le precipitazioni sono maggiori di zero.

Di seguito sono riportate le fasi per eseguire una query su un indice secondario globale utilizzando l'API documento di AWS SDK per Java.

1. Creare un'istanza della classe `DynamoDB`.
2. Crea un'istanza della classe `Table` per rappresentare l'indice con cui intendi lavorare.
3. Crea un'istanza della classe `Index` per l'indice su cui intendi eseguire una query.
4. Chiama il metodo `query` per l'oggetto `Index`.

Il nome di attributo `Date` è una parola riservata in DynamoDB. Pertanto, devi utilizzare un nome di attributo di espressione come segnaposto in `KeyConditionExpression`.

Il seguente esempio di codice Java mostra le fasi precedenti.

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());
}
```

Esempio: indici secondari globali che utilizzano l'API del documento AWS SDK per Java

Il seguente esempio di codice Java mostra come utilizzare gli indici secondari globali. L'esempio crea una tabella denominata `Issues`, che può essere utilizzata in un semplice sistema di tracciamento di bug per lo sviluppo di software. La chiave di partizione è `IssueId` e la chiave di ordinamento è `Title`. In questa tabella esistono tre indici secondari globali:

- `CreateDateIndex`: la chiave di partizione è `CreateDate` e la chiave di ordinamento è `IssueId`. Oltre alle chiavi di tabella, vengono proiettati nell'indice gli attributi `Description` e `Status`.
- `TitleIndex`: la chiave di partizione è `Title` e la chiave di ordinamento è `IssueId`. Non vengono proiettati nell'indice attributi diversi dalle chiavi di tabella.
- `DueDateIndex`: la chiave di partizione è `DueDate`; non è presente una chiave di ordinamento. Tutti gli attributi della tabella vengono proiettati nell'indice.

Una volta creata la tabella `Issues`, il programma carica la tabella con i dati che rappresentano i report sui bug del software. Quindi esegue una query sui dati utilizzando gli indici secondari globali. Infine, il programma elimina la tabella `Issues`.

Per le istruzioni dettagliate per eseguire il test del seguente esempio, consulta Esempi di codice 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);
    }

}
```

Utilizzo di indici secondari globali: .NET

È possibile utilizzare l'API di basso livello AWS SDK per .NET per creare una tabella Amazon DynamoDB con uno o più indici secondari globali, descrivere gli indici sulla tabella ed eseguire query utilizzando gli indici. Queste operazioni vengono mappate alle operazioni DynamoDB corrispondenti. Per ulteriori informazioni, consulta la Documentazione di riferimento delle API di Amazon DynamoDB.

Di seguito sono riportate le operazioni comuni per le operazioni delle tabelle che utilizzano l'API di basso livello .NET.

1. Creare un'istanza della classe `AmazonDynamoDBClient`.
2. Fornisci i parametri obbligatori e facoltativi per l'operazione creando gli oggetti di richiesta corrispondenti.

   Ad esempio, creare un oggetto `CreateTableRequest` per creare una tabella e un oggetto `QueryRequest` per eseguire una query su una tabella o un indice.
3. Eseguire il metodo appropriato fornito dal client creato nella fase precedente.

Argomenti

- Creazione di una tabella con un indice secondario globale
- Descrizione di una tabella con un indice secondario globale
- Esecuzione di query su un indice secondario globale
- Esempio: indici secondari globali che utilizzano l'API di basso livello AWS SDK per .NET

Creazione di una tabella con un indice secondario globale

Puoi creare indici secondari globali al momento della creazione di una tabella. A tale scopo, utilizza `CreateTable` e fornisci le specifiche per uno o più indici secondari globali. Il seguente esempio di codice C# crea una tabella per contenere le informazioni sui dati meteo. La chiave di partizione è `Location` e la chiave di ordinamento è `Date`. Un indice secondario globale denominato `PrecipIndex` consente di accedere rapidamente ai dati delle precipitazioni di vari luoghi.

Di seguito sono riportate le fasi per creare una tabella con un indice secondario globale, utilizzando l'API di basso livello di DynamoDB.

1. Creare un'istanza della classe `AmazonDynamoDBClient`.
2. Crea un'istanza della classe `CreateTableRequest` per fornire le informazioni della richiesta.

È necessario fornire il nome della tabella, la sua chiave primaria e i valori del throughput assegnato. Per l'indice secondario globale, è necessario fornire il nome dell'indice, le impostazioni di velocità effettiva assegnata, le definizioni degli attributi per la chiave di ordinamento dell'indice, lo schema della chiave per l'indice e la proiezione degli attributi.

3. Eseguire il metodo `CreateTable` fornendo l'oggetto della richiesta come parametro.

Il seguente esempio di codice C# mostra le fasi precedenti. Il codice crea una tabella (`WeatherData`) con un indice secondario globale (`PrecipIndex`). La chiave di partizione dell'indice è `Date` e la sua chiave di ordinamento è `Precipitation`. Tutti gli attributi della tabella vengono proiettati nell'indice. Gli utenti possono eseguire query su questo indice per ottenere dati sul meteo di una data specifica, ordinando facoltativamente i dati per quantità di precipitazioni.

Poiché `Precipitation` non è un attributo chiave per la tabella, non è obbligatorio. Tuttavia, item `WeatherData` senza `Precipitation` non vengono visualizzati in `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);
```

È necessario attendere fino a quando DynamoDB crea la tabella e imposta lo stato su `ACTIVE`. Dopodiché, puoi iniziare a inserire item di dati nella tabella.

Descrizione di una tabella con un indice secondario globale

Per ottenere informazioni sugli indici secondari globali in una tabella, utilizza `DescribeTable`. Puoi accedere al nome, allo schema della chiave e agli attributi proiettati di ciascun indice.

Di seguito sono riportate le fasi per accedere alle informazioni dell'indice secondario globale per una tabella utilizzando l'API di basso livello .NET.

1. Creare un'istanza della classe `AmazonDynamoDBClient`.
2. Eseguire il metodo `describeTable` fornendo l'oggetto della richiesta come parametro.

   Crea un'istanza della classe `DescribeTableRequest` per fornire le informazioni della richiesta. Devi specificare il nome della tabella.
3.

Il seguente esempio di codice C# mostra le fasi precedenti.

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);
      }
}
```

Esecuzione di query su un indice secondario globale

È possibile utilizzare l'operazione `Query` su un indice secondario globale quasi nello stesso modo in cui si esegue una `Query` su una tabella. È necessario specificare il nome dell'indice, i criteri di query per la chiave di partizione e di ordinamento dell'indice (se presente) e gli attributi da restituire. In questo esempio, l'indice è `PrecipIndex`, avente una chiave di partizione `Date` e una chiave di ordinamento `Precipitation`. La query di indice restituisce tutti i dati sul meteo di una data specifica in cui le precipitazioni sono maggiori di zero.

Di seguito sono riportate le fasi per eseguire una query su un indice secondario globale utilizzando l'API di basso livello .NET.

1. Creare un'istanza della classe `AmazonDynamoDBClient`.
2. Crea un'istanza della classe `QueryRequest` per fornire le informazioni della richiesta.
3. Eseguire il metodo `query` fornendo l'oggetto della richiesta come parametro.

Il nome di attributo `Date` è una parola riservata in DynamoDB. Pertanto, devi utilizzare un nome di attributo di espressione come segnaposto in `KeyConditionExpression`.

Il seguente esempio di codice C# mostra le fasi precedenti.

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();
}
```

Esempio: indici secondari globali che utilizzano l'API di basso livello AWS SDK per .NET

Il seguente esempio di codice C# mostra come utilizzare gli indici secondari globali. L'esempio crea una tabella denominata `Issues`, che può essere utilizzata in un semplice sistema di tracciamento di bug per lo sviluppo di software. La chiave di partizione è `IssueId` e la chiave di ordinamento è `Title`. In questa tabella esistono tre indici secondari globali:

- `CreateDateIndex`: la chiave di partizione è `CreateDate` e la chiave di ordinamento è `IssueId`. Oltre alle chiavi di tabella, vengono proiettati nell'indice gli attributi `Description` e `Status`.
- `TitleIndex`: la chiave di partizione è `Title` e la chiave di ordinamento è `IssueId`. Non vengono proiettati nell'indice attributi diversi dalle chiavi di tabella.
- `DueDateIndex`: la chiave di partizione è `DueDate`; non è presente una chiave di ordinamento. Tutti gli attributi della tabella vengono proiettati nell'indice.

Una volta creata la tabella `Issues`, il programma carica la tabella con i dati che rappresentano i report sui bug del software. Quindi esegue una query sui dati utilizzando gli indici secondari globali. Infine, il programma elimina la tabella `Issues`.

Per le istruzioni dettagliate per eseguire il test del seguente esempio, consulta Esempi di codice .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;
                }
            }
        }
    }
}
```

Utilizzo di indici secondari in DynamoDB con la AWS CLI

È possibile utilizzare la AWS CLI per creare una tabella Amazon DynamoDB con uno o più indici secondari globali, descrivere gli indici sulla tabella ed eseguire query utilizzando gli indici.

Argomenti

- Creazione di una tabella con un indice secondario globale
- Aggiunta di un indice secondario globale a una tabella esistente
- Descrizione di una tabella con un indice secondario globale
- Esecuzione di query su un indice secondario globale

Creazione di una tabella con un indice secondario globale

Gli indici secondari globali possono essere creati al momento della creazione di una tabella. A tale scopo, utilizza il parametro `create-table` e fornire le specifiche per uno o più indici secondari globali. Nell'esempio seguente viene creata una tabella denominata `GameScores` con un indice secondario globale denominato `GameTitleIndex`. La tabella di base ha una chiave di partizione di `UserId` e una chiave di ordinamento di `GameTitle`, permettendo di trovare il miglior punteggio di un singolo utente per un gioco specifico in modo efficiente, mentre il GSI ha una chiave di partizione di `GameTitle` e una chiave di ordinamento di `TopScore`, permettendo di trovare rapidamente il punteggio più alto complessivo per un determinato gioco.

```
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
                }
            }
        ]"
```

È necessario attendere fino a quando DynamoDB crea la tabella e imposta lo stato su `ACTIVE`. Dopodiché, puoi iniziare a inserire item di dati nella tabella. È possibile utilizzare describe-table per determinare lo stato di avanzamento della creazione della tabella.

Aggiunta di un indice secondario globale a una tabella esistente

Gli indici secondari globali possono essere aggiunti o modificati anche dopo la creazione della tabella. A tale scopo, utilizza il parametro `update-table` e fornire le specifiche per uno o più indici secondari globali. Nell'esempio seguente viene utilizzato lo stesso schema dell'esempio precedente, ma si presuppone che la tabella sia già stata creata e che la GSI verrà aggiunta in seguito.

```
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\"]
                    }
                }
            }
        ]"
```

Descrizione di una tabella con un indice secondario globale

Per ottenere informazioni sugli indici secondari globali in una tabella, utilizza il parametro `describe-table`. Puoi accedere al nome, allo schema della chiave e agli attributi proiettati di ciascun indice.

```
aws dynamodb describe-table --table-name GameScores
```

Esecuzione di query su un indice secondario globale

È possibile utilizzare l'operazione `query` su un indice secondario globale nello stesso modo in cui si esegue una `query` su una tabella. È necessario specificare il nome dell'indice, i criteri della query per la chiave di ordinamento dell'indice e gli attributi da restituire. In questo esempio, l'indice è `GameTitleIndex` e la chiave di ordinamento dell'indice è `GameTitle`.

Gli unici attributi restituiti sono quelli proiettati nell'indice. È possibile modificare questa query per selezionare anche attributi non chiave, ma ciò richiederebbe un'attività di recupero della tabella relativamente costosa. Per ulteriori informazioni sul recupero delle tabelle, consulta Proiezioni di attributi.

```
aws dynamodb query --table-name GameScores\
    --index-name GameTitleIndex \
    --key-condition-expression "GameTitle = :v_game" \
    --expression-attribute-values '{":v_game":{"S":"Alien Adventure"} }'
```

Indici secondari locali

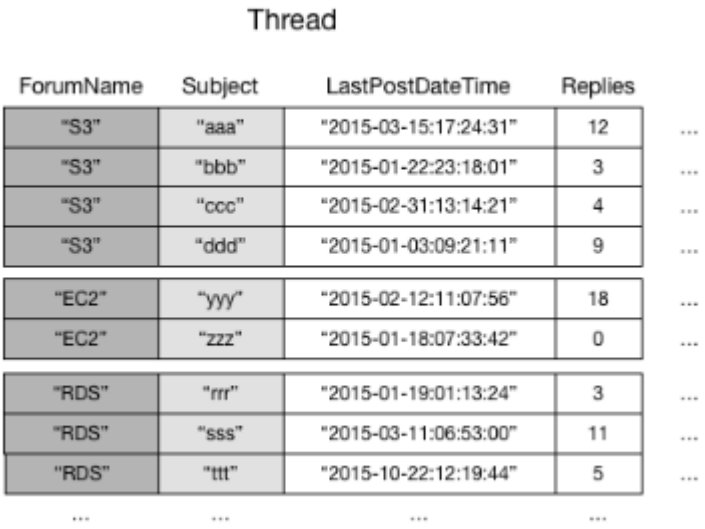
Alcune applicazioni devono eseguire query sui dati soltanto utilizzando la chiave primaria della tabella di base. Tuttavia, potrebbero esserci situazioni in cui una chiave di ordinamento alternativa sarebbe utile. Per dare all'applicazione una scelta di chiavi di ordinamento, è possibile creare uno o più indici secondari locali su una tabella Amazon DynamoDB ed emettere le richieste `Query` o `Scan` rispetto a questi indici.

Argomenti

- Scenario: Utilizzo di un indice secondario locale
- Proiezioni di attributi
- Creazione di un indice secondario locale
- Lettura di dati da un indice secondario locale
- Scritture di elementi e indici secondari locali
- Considerazioni sulla velocità di trasmissione effettiva assegnata per indici secondari locali
- Considerazioni sull'archiviazione per indici secondari locali
- Raccolte di elementi negli indici secondari locali
- Utilizzo di indici secondari locali: Java
- Utilizzo di indici secondari locali: .NET
- Utilizzo di indici secondari locali nella AWS CLI di DynamoDB

Scenario: Utilizzo di un indice secondario locale

Ad esempio, si consideri la tabella `Thread`. Questa tabella è utile per un'applicazione come Forum di discussione AWS. Il seguente diagramma mostra in che modo verrebbero organizzarti gli elementi nella tabella. Non tutti gli attributi vengono visualizzati.



DynamoDB archivia tutti gli elementi con lo stesso valore della chiave di partizione in modo continuo. In questo esempio, dato un particolare `ForumName`, un'operazione `Query` potrebbe individuare immediatamente tutti i thread per quel forum. All'interno di un gruppo di elementi con lo stesso valore della chiave di partizione, gli elementi vengono ordinati in base al valore della chiave di ordinamento. Se la chiave di ordinamento (`Subject`) viene fornita anche nella query, DynamoDB può restringere i risultati restituiti, ad esempio restituendo tutti i thread nel forum "S3" che hanno un `Subject` che inizia con la lettera "a".

Alcune richieste potrebbero richiedere modelli di accesso ai dati più complessi. Ad esempio:

- Quali thread del forum ottengono il maggior numero di visualizzazioni e risposte?
- Quale thread in un particolare forum ha il maggior numero di messaggi?
- Quanti thread sono stati pubblicati in un particolare forum in un determinato periodo di tempo?

Per rispondere a queste domande, l'operazione `Query` non sarebbe sufficiente. Con questa procedura, sarà necessario eseguire una `Scan` di tutta la tabella. Per una tabella con milioni di elementi, questa operazione consumerebbe una grande quantità di velocità effettiva di lettura assegnata e richiederebbe molto tempo per il completamento.

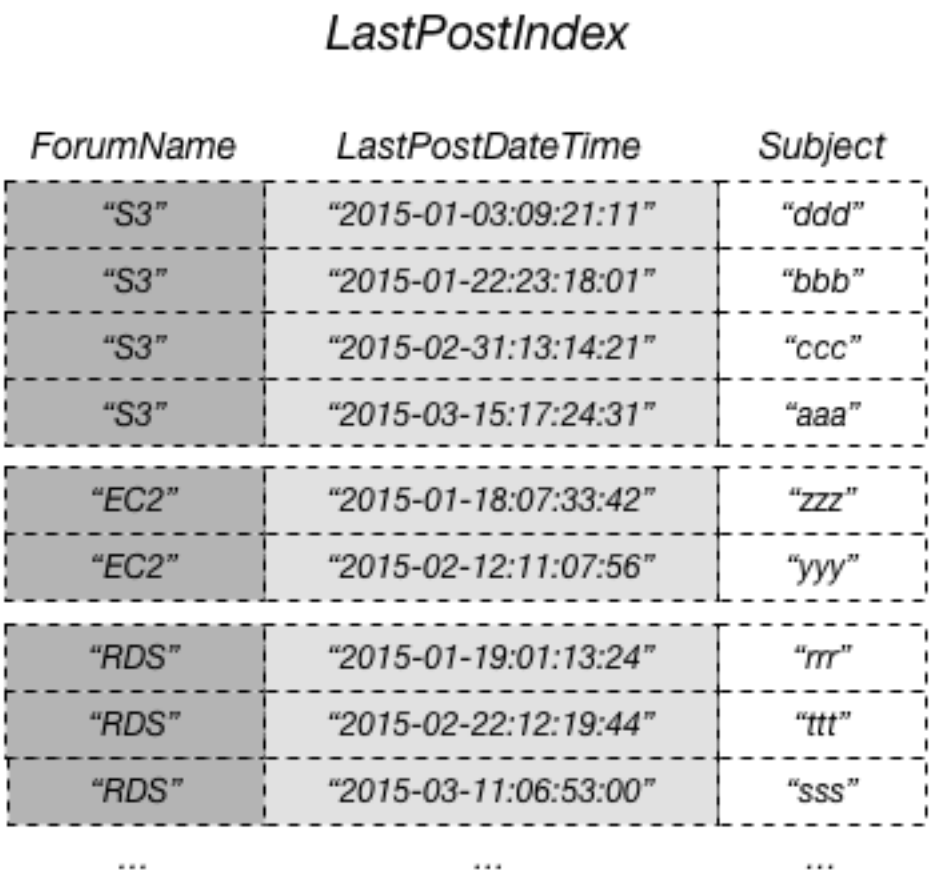
Tuttavia, è possibile specificare uno o più indici secondari locali su attributi non chiave, ad esempio `Replies` o `LastPostDateTime`.

Un indice secondario locale gestisce una chiave di ordinamento alternativa per un determinato valore della chiave di partizione. Un indice secondario locale contiene anche una copia di alcuni o

tutti gli attributi della relativa tabella di base. È possibile specificare gli attributi proiettati nell'indice secondario locale quando si crea la tabella. I dati in un indice secondario locale sono organizzati in base alla stessa chiave di partizione della tabella di base ma con una chiave di ordinamento diversa. Ciò consente di accedere in modo efficiente agli elementi di dati in questa diversa dimensione. Per una maggiore flessibilità di query o scansione, è possibile creare un massimo di cinque indici secondari locali per tabella.

Supponiamo che un'applicazione debba trovare tutti i thread che sono stati pubblicati negli ultimi tre mesi in un particolare forum. Senza un indice secondario locale, l'applicazione dovrebbe eseguire la `Scan` dell'intera tabella `Thread` ed eliminare tutti i post che non erano nel periodo di tempo specificato. Con un indice secondario locale, un'operazione `Query`potrebbe usare `LastPostDateTime` come chiave di ordinamento e trovare rapidamente i dati.

Il seguente diagramma mostra un indice secondario locale denominato `LastPostIndex`. Si noti che la chiave di partizione è la stessa di quella della tabella `Thread`, ma la chiave di ordinamento è`LastPostDateTime`.



Ciascun indice secondario locale deve soddisfare le seguenti condizioni:

- La chiave di partizione deve essere la stessa della tabella di base.
- La chiave di ordinamento deve essere costituita esattamente da un attributo scalare.
- La chiave di ordinamento della tabella di base deve essere proiettata nell'indice, dove funzionerà da attributo non chiave.

In questo esempio, la chiave di partizione è `ForumName` e la chiave di ordinamento dell'indice secondario locale è `LastPostDateTime`. Inoltre, il valore della chiave di ordinamento dalla tabella di base (in questo esempio, `Subject`) viene proiettato nell'indice ma non fa parte della chiave di indice. Se un'applicazione ha bisogno di un elenco basato su `ForumName` e `LastPostDateTime`, può emettere una richiesta `Query` rispetto a `LastPostIndex`. I risultati della query sono ordinati per `LastPostDateTime` e possono essere restituiti in ordine crescente o decrescente. La query può applicare anche condizioni chiave, ad esempio restituendo solo gli elementi che dispongono di un `LastPostDateTime` entro un determinato lasso di tempo.

Ogni indice secondario locale contiene automaticamente le chiavi di partizione e ordinamento dalla relativa tabella di base; facoltativamente, è possibile proiettare gli attributi non chiave nell'indice. Quando si esegue una query sull'indice, DynamoDB può recuperare questi attributi proiettati in modo efficiente. Se si esegue una query sull'indice secondario locale, è possibile richiamare gli attributi che non sono proiettati sull'indice. DynamoDB recupera automaticamente questi attributi dalla tabella di base, ma a una latenza maggiore e con costi di velocità effettiva assegnata più elevati.

Per qualsiasi indice secondario locale, è possibile memorizzare fino a 10 GB di dati per ogni valore di chiave di partizione distinto. Questa figura include tutti gli elementi della tabella di base più tutti gli elementi degli indici, che hanno lo stesso valore della chiave di partizione. Per ulteriori informazioni, consulta Raccolte di elementi negli indici secondari locali.

Proiezioni di attributi

Con `LastPostIndex`, un'applicazione potrebbe usare `ForumName` e `LastPostDateTime` come criteri di query. Tuttavia, per richiamare eventuali altri attributi, DynamoDB deve eseguire ulteriori operazioni di lettura rispetto alla tabella `Thread`. Queste letture extra sono conosciute come recuperi e possono aumentare la quantità totale di velocità effettiva assegnata necessaria per una query.

Supponiamo di voler compilare una pagina web con un elenco di tutte le discussioni in «S3" e il numero di risposte per ogni thread, ordinate in base all'ultima risposta che inizia con la risposta più recente. date/time Per popolare questo elenco, sono necessari i seguenti attributi:

- `Subject`

- `Replies`
- `LastPostDateTime`

Il modo più efficiente per interrogare questi dati e per evitare operazioni di recupero sarebbe quello di proiettare l'attributo `Replies` dalla tabella nell'indice secondario locale, come mostrato in questo diagramma.

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 proiezione è l'insieme di attributi copiato da una tabella in un indice secondario. La chiave di partizione e la chiave di ordinamento della tabella vengono sempre proiettati nell'indice; è possibile proiettare altri attributi per supportare i requisiti di query dell'applicazione. Quando si esegue una query su un indice, Amazon DynamoDB può accedere a qualsiasi attributo nella proiezione come se tali attributi fossero in una propria tabella.

Quando si crea un indice secondario, è necessario specificare gli attributi che saranno proiettati nell'indice. DynamoDB fornisce tre diverse opzioni per questo:

- KEYS_ONLY: ogni elemento dell'indice è costituito solo dalla chiave di partizione della tabella e dai valori della chiave di ordinamento, oltre ai valori della chiave di indice. L'opzione `KEYS_ONLY` si traduce nell'indice secondario più piccolo possibile.
- INCLUDE: oltre agli attributi descritti in `KEYS_ONLY`, l'indice secondario includerà gli altri attributi non chiave che sono stati specificati.
- ALL: l'indice secondario include tutti gli attributi della tabella di origine. Poiché tutti i dati della tabella sono duplicati nell'indice, una proiezione `ALL` restituisce il più grande indice secondario possibile.

Nel diagramma precedente, l'attributo non chiave `Replies` è proiettato in `LastPostIndex`. Un'applicazione può interrogare `LastPostIndex` al posto della tabella `Thread` completa per popolare una pagina Web con`Subject`, `Replies` e `LastPostDateTime`. Se vengono richiesti altri attributi non chiave, DynamoDB dovrebbe recuperare tali attributi dalla tabella `Thread`.

Dal punto di vista di un'applicazione, il recupero di attributi aggiuntivi dalla tabella di base è automatico e trasparente, quindi non è necessario riscrivere alcuna logica dell'applicazione. Tuttavia, tale recupero può ridurre notevolmente il vantaggio in termini di prestazioni dell'utilizzo di un indice secondario locale.

Quando si scelgono gli attributi da proiettare in un indice secondario locale, è necessario considerare il compromesso tra i costi correlati alla velocità effettiva assegnata e costi di archiviazione:

- Se è necessario accedere a pochi attributi con la latenza più bassa possibile, considerare la possibilità di proiettare solo quegli attributi in un indice secondario locale. Più piccolo è l'indice, minore è il costo per memorizzarlo e minori sono i costi di scrittura. Se sono presenti attributi che occasionalmente è necessario recuperare, il costo per la velocità effettiva assegnata potrebbe superare il costo a lungo termine della memorizzazione di tali attributi.
- Se l'applicazione accede frequentemente ad alcuni attributi non chiave, è necessario considerare di proiettare quegli attributi in un indice secondario locale. I costi di archiviazione aggiuntivi per l'indice secondario locale compensano il costo di esecuzione di scansioni frequenti delle tabelle.
- Se è necessario accedere alla maggior parte degli attributi non chiave su base frequente, è possibile proiettare questi attributi, o anche l'intera tabella di base, in un indice secondario locale. Ciò offre la massima flessibilità e il minor consumo di velocità effettiva assegnata, in quanto non sarebbe necessario eseguire alcun recupero. Tuttavia, i costi di archiviazione aumenteranno o addirittura raddoppieranno se verranno proiettati tutti gli attributi.

- Se l'applicazione esegue di rado le query sulla tabella ma esegue molte scritture o aggiornamenti dei dati, considerare la possibilità di proiettare KEYS_ONLY. L'indice secondario locale avrebbe dimensioni minime e sarebbe comunque disponibile, se necessario, per l'attività di query.

Creazione di un indice secondario locale

Per creare una tabella con uno o più indici secondari locali su una tabella, utilizza il parametro `LocalSecondaryIndexes` con l'operazione `CreateTable`. Gli indici secondari locali in una tabella vengono creati al momento della creazione della tabella. Quando si elimina una tabella, vengono eliminati anche tutti gli indici secondari locali della tabella.

È necessario specificare un attributo non chiave che funzioni da chiave di ordinamento dell'indice secondario locale. L'attributo scelto deve essere uno `String`, `Number` oppure `Binary` scalare. Altri tipi scalari, documento e set non sono consentiti. Per l'elenco completo dei tipi di dati, consulta Tipi di dati.

> **Important**
>
> Per le tabelle con gli indici secondari locali, esiste un limite di 10 GB per valore di chiave di partizione. Una tabella con indici secondari locali può memorizzare qualsiasi numero di elementi, a condizione che la dimensione totale per qualsiasi valore di una chiave di partizione non superi 10 GB. Per ulteriori informazioni, consulta Limite delle dimensioni delle raccolte di elementi.

È possibile proiettare gli attributi di qualsiasi tipo di dati in un indice secondario locale. Questo include scalari, documenti e set. Per l'elenco completo dei tipi di dati, consulta Tipi di dati.

Lettura di dati da un indice secondario locale

È possibile richiamare gli elementi da un indice secondario locale utilizzando le operazioni `Query` e `Scan`. Le operazioni `GetItem` e `BatchGetItem` non possono essere utilizzate su un indice secondario locale.

Esecuzione di una query su un indice secondario locale

In una tabella DynamoDB, il valore combinato della chiave di partizione e della chiave di ordinamento per ogni elemento deve essere univoco. Tuttavia, in un indice secondario locale, il valore della

chiave di ordinamento non deve essere univoco per un determinato valore di chiave di partizione. Se nell'indice secondario locale sono presenti più elementi con lo stesso valore della chiave di ordinamento, un'operazione `Query` restituisce tutti gli elementi che hanno lo stesso valore della chiave di partizione. Nella risposta, gli articoli corrispondenti non vengono restituiti in un ordine particolare.

È possibile eseguire una query su un indice secondario locale utilizzando letture a consistenza finale o fortemente coerenti. Per specificare il tipo di coerenza desiderato, utilizza il parametro `ConsistentRead` dell'operazione `Query`. Una lettura fortemente consistente da un indice secondario locale restituisce sempre gli ultimi valori aggiornati. Se la query deve recuperare attributi aggiuntivi dalla tabella di base, tali attributi saranno coerenti rispetto all'indice.

Example

Valuta i seguenti dati restituiti da una `Query` che richiede dati dai thread di discussione in un particolare forum.

```
{
    "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"}
    }
}
```

In questa query:

- DynamoDB `LastPostIndex` accede, utilizzando `ForumName` la chiave di partizione per individuare gli elementi dell'indice per "». EC2 Tutti gli elementi dell'indice con questa chiave sono memorizzati l'uno accanto all'altro per il recupero rapido.
- All'interno di questo forum, DynamoDB utilizza l'indice per cercare le chiavi che corrispondono alla condizione `LastPostDateTime` specificata.
- Poiché l'attributo `Replies` viene proiettato nell'indice, DynamoDB è in grado di recuperare questo attributo senza utilizzare alcuna velocità effettiva assegnata aggiuntiva.

- L'attributo `Tags` non viene proiettato nell'indice, quindi DynamoDB deve accedere alla tabella `Thread` e recuperare questo attributo.
- I risultati vengono restituiti, ordinati per `LastPostDateTime`. Le voci dell'indice vengono ordinate in base al valore della chiave di partizione e quindi in base al valore della chiave di ordinamento e `Query` li restituisce nell'ordine in cui sono memorizzati. È possibile utilizzare il parametro `ScanIndexForward` per restituire i risultati in ordine decrescente.

Poiché l'attributo `Tags` non viene proiettato nell'indice secondario locale, DynamoDB deve consumare unità di capacità di lettura aggiuntive per recuperare questo attributo dalla tabella di base. Se è necessario eseguire spesso questa query, è necessario proiettare `Tags` in `LastPostIndex` per evitare il recupero dalla tabella di base. Tuttavia, se è necessario accedere a `Tags` solo occasionalmente, il costo di archiviazione aggiuntivo per la proiezione `Tags` nell'indice potrebbe non valere la pena.

Scansione di un indice secondario locale

È possibile utilizzare l'operazione `Scan` per recuperare tutti i dati da un indice secondario globale. Devi fornire il nome della tabella di base e il nome dell'indice nella richiesta. Con un'operazione `Scan`, DynamoDB legge tutti i dati nell'indice e li restituisce all'applicazione. Inoltre puoi richiedere che vengano restituiti solo alcuni dati e che quelli rimanenti vengano eliminati. A questo scopo, utilizza il parametro `FilterExpression` dell'API `Scan`. Per ulteriori informazioni, consulta Espressioni di filtro per la scansione.

Scritture di elementi e indici secondari locali

DynamoDB conserva automaticamente tutti gli indici secondari locali sincronizzati con le rispettive tabelle di base. Le applicazioni non scrivono mai direttamente in un indice. Tuttavia, è importante comprendere le implicazioni di come DynamoDB mantiene questi indici.

Quando si crea un indice secondario locale, viene specificato un attributo da utilizzare come chiave di ordinamento per l'indice. È inoltre possibile specificare un tipo di dati per tale attributo. Questo significa che ogni volta che si scrive un elemento nella tabella di base, se l'elemento definisce un attributo della chiave di indice, il relativo tipo deve corrispondere al tipo di dati dello schema della chiave di indice. Nel caso di `LastPostIndex`, la chiave di ordinamento `LastPostDateTime` nell'indice è definita come un tipo di dati `String`. Se si prova ad aggiungere un elemento alla tabella `Thread` e si specifica un tipo di dati diverso per `LastPostDateTime` (come `Number`), DynamoDB restituisce un `ValidationException` perché il tipo di dati non corrisponde.

Non è richiesta una one-to-one relazione tra gli elementi di una tabella di base e gli elementi di un indice secondario locale. In effetti, questo comportamento può essere vantaggioso per molte applicazioni.

Una tabella con molti indici secondari globali comporta costi maggiori per l'attività di scrittura rispetto alle tabelle con meno indici. Per ulteriori informazioni, consulta Considerazioni sulla velocità di trasmissione effettiva assegnata per indici secondari locali.

> **Important**
>
> Per le tabelle con gli indici secondari locali, esiste un limite di 10 GB per valore di chiave di partizione. Una tabella con indici secondari locali può memorizzare qualsiasi numero di elementi, a condizione che la dimensione totale per qualsiasi valore di una chiave di partizione non superi 10 GB. Per ulteriori informazioni, consulta Limite delle dimensioni delle raccolte di elementi.

Considerazioni sulla velocità di trasmissione effettiva assegnata per indici secondari locali

Quando si crea una tabella in DynamoDB, vengono assegnate le unità di capacità di lettura e scrittura per il carico di lavoro previsto della tabella. Tale carico di lavoro include attività di lettura e scrittura sugli indici secondari locali della tabella.

Per visualizzare le tariffe correnti per la capacità di throughput assegnata, consulta Prezzi di Amazon DynamoDB.

Unità di capacità in lettura

Quando si esegue una query su un indice secondario locale, il numero di unità di capacità di lettura utilizzate dipende dal modo in cui si accede ai dati.

Come per le query su una tabella, una query su un indice può utilizzare letture a consistenza finale o fortemente consistenti a seconda del valore di `ConsistentRead`. Una lettura fortemente consistente utilizza una unità di capacità di lettura mentre una lettura a consistenza finale ne consuma solo la metà. Pertanto, scegliendo letture a consistenza finale, è possibile ridurre i costi delle unità di capacità di lettura.

Per le query su un indice che richiedono solo chiavi di indice e attributi proiettati, DynamoDB calcola l'attività di lettura assegnata come avviene per le query sulle tabelle. L'unica differenza è che il

calcolo si basa sulle dimensioni delle voci dell'indice, piuttosto che sulla dimensione dell'item nella tabella di base. Il numero di unità di capacità in lettura è la somma di tutte le dimensioni degli attributi proiettati di tutti gli elementi restituiti; il risultato viene quindi arrotondato al successivo limite di 4 KB. Per ulteriori informazioni sul modo in cui DynamoDB calcola l'uso della velocità effettiva assegnata, consulta Modalità con capacità allocata di DynamoDB.

Per le query sugli indici che leggono gli attributi non proiettati sull'indice secondario locale, oltre a leggere gli attributi proiettati dall'indice DynamoDB deve recuperare tali attributi dalla tabella di base. Questi recuperi si verificano quando si includono attributi non proiettati nei parametri `Select` o `ProjectionExpression` dell'operazione `Query`. Il recupero causa una latenza aggiuntiva nelle risposte alle query e comporta anche un costo della velocità effettiva assegnata più elevato: oltre alle letture dall'indice secondario locale descritto in precedenza, vengono addebitate le unità di capacità di lettura per ogni elemento della tabella di base recuperato. Questo addebito è per la lettura di ogni elemento intero dalla tabella, non solo degli attributi richiesti.

La dimensione massima dei risultati restituiti da un'operazione `Query` è 1 MB. Sono incluse le dimensioni di tutti i nomi e i valori degli attributi in tutti gli elementi restituiti. Tuttavia, se una query su un indice secondario locale fa sì che DynamoDB recuperi gli attributi dell'elemento dalla tabella di base, la dimensione massima dei dati nei risultati potrebbe essere inferiore. In questo caso, la dimensione del risultato è la somma di:

- La dimensione degli elementi corrispondenti nell'indice, arrotondata per eccesso ai 4 KB successivi.
- La dimensione di ogni elemento corrispondente nella tabella di base, con ogni elemento arrotondato singolarmente ai 4 KB successivi.

Utilizzando questa formula, la dimensione massima dei risultati restituiti da un'operazione Query è comunque 1 MB.

Ad esempio, si consideri una tabella in cui la dimensione di ogni elemento è 300 byte. Su quella tabella è presente un indice secondario locale, ma solo 200 byte di ogni elemento vengono proiettati sull'indice. Si supponga adesso che venga eseguita una `Query` su questo indice, che la query richieda recuperi di tabella per ogni elemento e che la query restituisca 4 elementi. DynamoDB riassume quanto segue:

- Dimensione degli elementi corrispondenti nell'indice: 200 byte × 4 elementi = 800 byte; questo valore viene quindi arrotondato a 4 KB.

- Dimensione di ogni elemento corrispondente nella tabella di base: (300 byte, arrotondato a 4 KB) × 4 elementi = 16 KB.

La dimensione totale dei dati nel risultato è pertanto 20 KB.

Unità di capacità in scrittura

Quando un elemento viene aggiunto, aggiornato o eliminato in una tabella, l'aggiornamento degli indici secondari locali utilizza le unità di capacità di scrittura assegnate per la tabella. Il costo totale della velocità effettiva assegnata per una scrittura è la somma delle unità di capacità di scrittura utilizzate scrivendo sulla tabella e di quelle utilizzate dall'aggiornamento degli indici secondari locali.

Il costo della scrittura di un elemento in un indice secondario locale dipende da diversi fattori:

- Se scrivi un nuovo item nella tabella che definisce un attributo indicizzato o aggiorni un item esistente per definire un attributo indicizzato in precedenza non definito, è necessario eseguire un'operazione di scrittura per inserire l'item nell'indice.
- Se un aggiornamento alla tabella modifica il valore di un attributo chiave indicizzato (da A a B), sono necessarie due scritture, una per eliminare l'elemento precedente dall'indice e un'altra per inserire il nuovo elemento nell'indice.
- Se un item era presente nell'indice, ma una scrittura nella tabella ha causato la cancellazione dell'attributo indicizzato, è necessaria una scrittura per eliminare la proiezione dell'item precedente dall'indice.
- Se un item non è presente nell'indice prima o dopo l'aggiornamento dell'item, non vi è alcun costo di scrittura aggiuntivo per l'indice.

Tutti questi fattori presuppongono che la dimensione di ciascun elemento nell'indice sia minore o uguale alla dimensione dell'elemento di 1 KB per il calcolo delle unità di capacità di scrittura. Le voci di indice più grandi richiedono unità di capacità in scrittura aggiuntive. È possibile ridurre al minimo i costi di scrittura considerando gli attributi che le query devono restituire e proiettando solo tali attributi nell'indice.

Considerazioni sull'archiviazione per indici secondari locali

Quando un'applicazione scrive un elemento in una tabella, DynamoDB copia automaticamente il sottoinsieme di attributi corretto in qualsiasi indice secondario locale in cui devono essere presenti gli attributi. All' AWS account vengono addebitati i costi per la memorizzazione dell'elemento nella

tabella di base e anche per l'archiviazione degli attributi in qualsiasi indice secondario locale di tale tabella.

La quantità di spazio utilizzata da un item dell'indice è la somma di quanto segue:

- La dimensione in byte della chiave primaria della tabella di base (chiave di partizione e chiave di ordinamento)
- La dimensione in byte dell'attributo della chiave di indicizzazione
- La dimensione in byte degli attributi proiettati (se presenti)
- 100 byte di sovraccarico per elemento dell'indice

Per stimare i requisiti di archiviazione per un indice secondario locale, è possibile stimare la dimensione media di un elemento nell'indice e quindi moltiplicarla per il numero di elementi nella tabella di base.

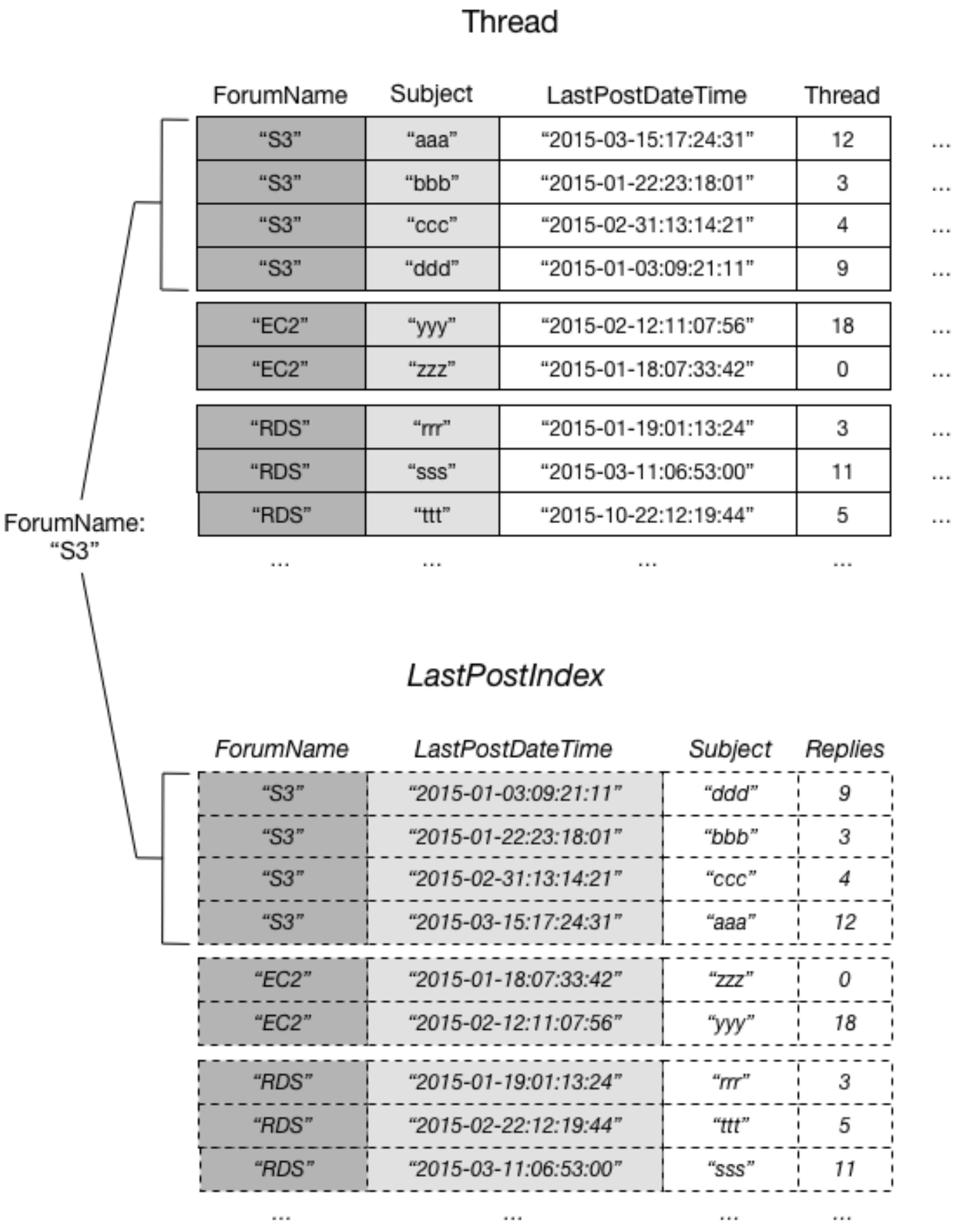
Se una tabella contiene un elemento in cui non è definito un attributo specifico, ma l'attributo è definito come una chiave di ordinamento dell'indice, DynamoDB non scrive alcun dato per tale elemento nell'indice.

Raccolte di elementi negli indici secondari locali

> **Note**
>
> In questa sezione vengono trattate solo le tabelle con indici secondari locali.

In DynamoDB, una raccolta di elementi è un gruppo di elementi in che hanno lo stesso valore della chiave di partizione in una tabella e in tutti gli indici secondari locali. Negli esempi utilizzati in tutta questa sezione, la chiave di partizione per la tabella `Thread` è `ForumName` e la chiave di partizione per `LastPostIndex` è anch'essa `ForumName`. Tutti gli elementi della tabella e dell'indice con lo stesso `ForumName` fanno parte della stessa raccolta di elementi. Ad esempio, nella tabella `Thread` e nell'indice secondario locale `LastPostIndex` esiste una raccolta di elementi per il forum `EC2` e una raccolta di elementi diversa per il forum `RDS`.

Il seguente diagramma mostra la raccolta di elementi per il forum `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

In questo diagramma, la raccolta di elementi è costituita da tutti gli elementi in `Thread` e `LastPostIndex` dove il valore della chiave di partizione `ForumName` è "S3". Se nella tabella sono presenti altri indici secondari locali, tutti gli elementi in tali indici con `ForumName` uguale a "S3" fanno parte della raccolta di elementi.

È possibile utilizzare una delle seguenti operazioni in DynamoDB per restituire informazioni sulle raccolte di elementi:

- `BatchWriteItem`
- `DeleteItem`
- `PutItem`
- `UpdateItem`
- `TransactWriteItems`

Ognuna di queste operazioni supporta il parametro `ReturnItemCollectionMetrics`. Quando si .imposta questo parametro su `SIZE`, è possibile visualizzare informazioni sulle dimensioni di ogni raccolta di elementi nell'indice.

Example

Di seguito è riportato un esempio dall'output di una operazione `UpdateItem` sulla tabella `Thread`, con `ReturnItemCollectionMetrics` impostato su `SIZE`. L'elemento che è stato aggiornato aveva `ForumName` il valore "EC2«, quindi l'output include informazioni su quella raccolta di elementi.

```
{
    ItemCollectionMetrics: {
        ItemCollectionKey: {
            ForumName: "EC2"
        },
        SizeEstimateRangeGB: [0.0, 1.0]
    }
}
```

L'oggetto `SizeEstimateRangeGB` indica che la dimensione di questa raccolta di elementi è compresa tra 0 e 1 GB. DynamoDB aggiorna periodicamente questa stima delle dimensioni, quindi i numeri potrebbero essere diversi la volta successiva che l'elemento viene modificato.

Limite delle dimensioni delle raccolte di elementi

La dimensione massima di qualsiasi raccolta di elementi per una tabella con uno o più indici secondari locali è di 10 GB. Ciò non si applica alle raccolte di elementi nelle tabelle senza indici secondari locali e non si applica alle raccolte di elementi negli indici secondari globali. Sono interessate solo le tabelle con uno o più indici secondari locali.

Se una raccolta di elementi supera il limite di 10 GB, DynamoDB restituisce una `ItemCollectionSizeLimitExceededException` e non sarà possibile aggiungere altri elementi alla raccolta o aumentare le dimensioni degli elementi inclusi nella raccolta. Le operazioni di lettura e scrittura che riducono le dimensioni della raccolta di elementi sono ancora consentite. Gli elementi possono comunque essere aggiunti ad altre raccolte di elementi.

Per ridurre le dimensioni di una raccolta di elementi, è possibile procedere in uno dei seguenti modi:

- Eliminare eventuali elementi non necessari con il valore della chiave di partizione in questione. Quando si eliminano questi elementi dalla tabella di base, DynamoDB rimuove anche tutte le voci di indice che hanno lo stesso valore della chiave di partizione.
- Aggiorna gli elementi rimuovendo gli attributi o riducendo le dimensioni degli attributi. Se questi attributi sono proiettati su qualsiasi indice secondario locale, DynamoDB riduce anche la dimensione delle voci di indice corrispondenti.
- Crea una nuova tabella con la stessa chiave di partizione e la stessa chiave di ordinamento, quindi sposta gli elementi dalla tabella precedente alla nuova tabella. Questo potrebbe essere un buon approccio se una tabella dispone di dati cronologici a cui si accede raramente. È possibile anche considerare l'archiviazione di questi dati della cronologia in Amazon Simple Storage Service (Amazon S3).

Quando la dimensione totale della raccolta di elementi scende al di sotto di 10 GB, è possibile aggiungere nuovamente elementi con lo stesso valore della chiave di partizione.

Come best practice, consigliamo di strumentalizzare l'applicazione in modo da monitorare le dimensioni delle raccolte di elementi. Un modo per farlo è impostare il parametro `ReturnItemCollectionMetrics` su `SIZE`ogni volta che si utilizza `BatchWriteItem`, `DeleteItem`, `PutItem` o `UpdateItem`. L'applicazione dovrebbe esaminare l'oggetto `ReturnItemCollectionMetrics` nell'output e registrare un messaggio di errore ogni volta che una raccolta di elementi supera un limite definito dall'utente (ad esempio, 8 GB). L'impostazione di un limite inferiore a 10 GB fornirebbe un sistema di avviso anticipato in modo da sapere che una raccolta di elementi si sta avvicinando al limite in tempo per intervenire.

Raccolte di elementi e partizioni

In una tabella con uno o più indici secondari locali, ogni raccolta di elementi viene archiviata in una partizione. La dimensione totale di tale raccolta di elementi è limitata alla capacità di quella partizione: 10 GB. Per un'applicazione in cui il modello di dati include raccolte di elementi con dimensioni illimitate o in cui si potrebbe ragionevolmente aspettarsi che alcune raccolte di elementi crescano oltre i 10 GB in futuro, è consigliabile prendere in considerazione l'utilizzo di un indice secondario globale.

Progettare le applicazioni in modo che i dati della tabella siano distribuiti uniformemente tra valori distinti della chiave di partizione. Per le tabelle con indici secondari locali, le applicazioni non devono creare "hot spot" di attività di lettura e scrittura all'interno di una singola raccolta di elementi su una singola partizione.

Utilizzo di indici secondari locali: Java

È possibile utilizzare l'API documento AWS SDK per Java per creare una tabella Amazon DynamoDB con uno o più indici secondari locali, descrivere gli indici sulla tabella ed eseguire query utilizzando gli indici.

Di seguito sono riportate le fasi comuni per le operazioni delle tabelle che utilizzano l'API di documento AWS SDK per Java.

1. Creare un'istanza della classe `DynamoDB`.
2. Fornisci i parametri obbligatori e facoltativi per l'operazione creando gli oggetti di richiesta corrispondenti.
3. Chiama il metodo appropriato fornito dal client creato nella fase precedente.

Argomenti

- Creazione di una tabella con un indice secondario locale
- Descrizione di una tabella con un indice secondario locale
- Esecuzione di query su un indice secondario locale
- Esempio: indici secondari locali che utilizzano l'API del documento Java

Creazione di una tabella con un indice secondario locale

Gli indici secondari locali devono essere creati al momento della creazione di una tabella. A tale scopo, utilizza il metodo `createTable` e fornire le specifiche per uno o più indici secondari locali.

Il seguente esempio di codice Java crea una tabella per contenere le informazioni sui brani in una raccolta musicale. La chiave di partizione è `Artist` e la chiave di ordinamento è `SongTitle`. Un indice secondario, `AlbumTitleIndex`, facilita le query in base al titolo dell'album.

Di seguito sono riportate le operazioni necessarie per creare una tabella con un indice secondario locale, utilizzando l'API documento di DynamoDB.

1. Creare un'istanza della classe `DynamoDB`.
2. Crea un'istanza della classe `CreateTableRequest` per fornire le informazioni della richiesta.

   È necessario fornire il nome della tabella, la sua chiave primaria e i valori del throughput assegnato. Per l'indice secondario locale, è necessario fornire il nome dell'indice, il nome e il tipo di dati della chiave di ordinamento dell'indice, lo schema della chiave per l'indice e la proiezione degli attributi.
3. Chiama il metodo `createTable` fornendo l'oggetto richiesta come parametro.

Il seguente esempio di codice Java mostra le fasi precedenti. Il codice crea una tabella (`Music`) con un indice secondario sull'attributo `AlbumTitle`. La chiave di partizione e la chiave di ordinamento della tabella, più la chiave di ordinamento dell'indice, sono gli unici attributi proiettati sull'indice.

```
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());
```

È necessario attendere fino a quando DynamoDB crea la tabella e imposta lo stato su `ACTIVE`. Dopodiché, puoi iniziare a inserire item di dati nella tabella.

Descrizione di una tabella con un indice secondario locale

Per ottenere informazioni sugli indici secondari locali in una tabella, utilizza il metodo `describeTable`. Puoi accedere al nome, allo schema della chiave e agli attributi proiettati di ciascun indice.

Di seguito sono riportate le fasi per accedere alle informazioni sull'indice secondario locale di una tabella utilizzando l'API documento AWS SDK per Java.

1. Creare un'istanza della classe `DynamoDB`.
2. Creare un'istanza della classe `Table`. Devi specificare il nome della tabella.
3. Chiama il metodo `describeTable` per l'oggetto `Table`.

Il seguente esempio di codice Java mostra le fasi precedenti.

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());
    }
}
```

Esecuzione di query su un indice secondario locale

È possibile utilizzare l'operazione `Query` su un indice secondario locale nello stesso modo in cui si esegue una `Query` su una tabella. È necessario specificare il nome dell'indice, i criteri della query per la chiave di ordinamento dell'indice e gli attributi da restituire. In questo esempio, l'indice è `AlbumTitleIndex` e la chiave di ordinamento dell'indice è `AlbumTitle`.

Gli unici attributi restituiti sono quelli proiettati nell'indice. È possibile modificare questa query per selezionare anche attributi non chiave, ma ciò richiederebbe un'attività di recupero della tabella relativamente costosa. Per ulteriori informazioni sul recupero delle tabelle, consulta Proiezioni di attributi.

Di seguito sono riportate le fasi per eseguire una query su un indice secondario locale utilizzando l'API documento di AWS SDK per Java.

1. Creare un'istanza della classe `DynamoDB`.
2. Creare un'istanza della classe `Table`. Devi specificare il nome della tabella.
3. Creare un'istanza della classe `Index`. È necessario specificare il nome dell'indice.
4. Chiamare il metodo `query` della classe `Index`.

Il seguente esempio di codice Java mostra le fasi precedenti.

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());
}
```

Esempio: indici secondari locali che utilizzano l'API del documento Java

Il seguente esempio di codice Java mostra come utilizzare gli indici secondari globali in Amazon DynamoDB. Nell'esempio viene creata una tabella denominata `CustomerOrders` con una chiave di partizione `CustomerId` e una chiave di ordinamento `OrderId`. In questa tabella esistono due indici secondari locali:

- `OrderCreationDateIndex`: la chiave di ordinamento è `OrderCreationDate` e i seguenti attributi sono proiettati sull'indice.
  - `ProductCategory`
  - `ProductName`
  - `OrderStatus`
  - `ShipmentTrackingId`
- `IsOpenIndex`: la chiave di ordinamento è `IsOpen` e tutti gli attributi della tabella vengono proiettati nell'indice.

Una volta creata la tabella `CustomerOrders`, il programma carica la tabella con i dati che rappresentano gli ordini dei clienti. Quindi esegue una query sui dati utilizzando gli indici secondari locali. Infine, il programma elimina la tabella `CustomerOrders`.

Per le istruzioni dettagliate per eseguire il test del seguente esempio, consulta Esempi di codice 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;
        }
```

```
}
```

Utilizzo di indici secondari locali: .NET

Argomenti

- Creazione di una tabella con un indice secondario locale
- Descrizione di una tabella con un indice secondario locale
- Esecuzione di query su un indice secondario locale
- Esempio: indici secondari locali che utilizzano l'API di basso livello AWS SDK per .NET

È possibile utilizzare l'API di basso livello AWS SDK per .NET per creare una tabella Amazon DynamoDB con uno o più indici secondari globali, descrivere gli indici sulla tabella ed eseguire query utilizzando gli indici. Queste operazioni vengono mappate alle corrispondenti azioni dell'API DynamoDB di basso livello. Per ulteriori informazioni, consulta Esempi di codice .NET.

Di seguito sono riportate le operazioni comuni per le operazioni delle tabelle che utilizzano l'API di basso livello .NET.

1. Creare un'istanza della classe `AmazonDynamoDBClient`.
2. Fornisci i parametri obbligatori e facoltativi per l'operazione creando gli oggetti di richiesta corrispondenti.

   Ad esempio, creare un oggetto `CreateTableRequest` per creare una tabella e un oggetto `QueryRequest` per eseguire una query su una tabella o un indice.
3. Eseguire il metodo appropriato fornito dal client creato nella fase precedente.

Creazione di una tabella con un indice secondario locale

Gli indici secondari locali possono essere creati al momento della creazione di una tabella. A tale scopo, utilizza `CreateTable` e fornire le specifiche per uno o più indici secondari locali. Il seguente esempio di codice C# crea una tabella per contenere le informazioni sui brani in una raccolta musicale. La chiave di partizione è `Artist` e la chiave di ordinamento è `SongTitle`. Un indice secondario, `AlbumTitleIndex`, facilita le query in base al titolo dell'album.

Di seguito sono riportate le operazioni per creare una tabella con un indice secondario locale, utilizzando l'API di basso livello di DynamoDB.

1. Creare un'istanza della classe `AmazonDynamoDBClient`.
2. Crea un'istanza della classe `CreateTableRequest` per fornire le informazioni della richiesta.

   È necessario fornire il nome della tabella, la sua chiave primaria e i valori del throughput assegnato. Per l'indice secondario locale, è necessario fornire il nome dell'indice, il nome e il tipo di dati della chiave di ordinamento dell'indice, lo schema della chiave per l'indice e la proiezione degli attributi.
3. Eseguire il metodo `CreateTable` fornendo l'oggetto della richiesta come parametro.

Il seguente esempio di codice C# mostra le fasi precedenti. Il codice crea una tabella (`Music`) con un indice secondario sull'attributo `AlbumTitle`. La chiave di partizione e la chiave di ordinamento della tabella, più la chiave di ordinamento dell'indice, sono gli unici attributi proiettati sull'indice.

```
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);
```

È necessario attendere fino a quando DynamoDB crea la tabella e imposta lo stato su `ACTIVE`. Dopodiché, puoi iniziare a inserire item di dati nella tabella.

Descrizione di una tabella con un indice secondario locale

Per ottenere informazioni sugli indici secondari locali in una tabella, utilizza l'API `DescribeTable`. Puoi accedere al nome, allo schema della chiave e agli attributi proiettati di ciascun indice.

Di seguito sono riportate le operazioni per accedere alle informazioni sull'indice secondario locale su una tabella utilizzando l'API di basso livello .NET.

1. Creare un'istanza della classe `AmazonDynamoDBClient`.
2. Crea un'istanza della classe `DescribeTableRequest` per fornire le informazioni della richiesta. Devi specificare il nome della tabella.
3. Eseguire il metodo `describeTable` fornendo l'oggetto della richiesta come parametro.
4.

Il seguente esempio di codice C# mostra le fasi precedenti.

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);
        }

    }
}
```

Esecuzione di query su un indice secondario locale

È possibile utilizzare l'operazione `Query` su un indice secondario locale nello stesso modo in cui si esegue una `Query` su una tabella. È necessario specificare il nome dell'indice, i criteri della query per la chiave di ordinamento dell'indice e gli attributi da restituire. In questo esempio, l'indice è `AlbumTitleIndex` e la chiave di ordinamento dell'indice è `AlbumTitle`.

Gli unici attributi restituiti sono quelli proiettati nell'indice. È possibile modificare questa query per selezionare anche attributi non chiave, ma ciò richiederebbe un'attività di recupero della tabella relativamente costosa. Per ulteriori informazioni sul recupero delle tabelle, consulta Proiezioni di attributi

Di seguito sono riportate le operazioni per eseguire una query su un indice secondario locale utilizzando l'API di basso livello .NET.

1. Creare un'istanza della classe `AmazonDynamoDBClient`.
2. Crea un'istanza della classe `QueryRequest` per fornire le informazioni della richiesta.
3. Eseguire il metodo `query` fornendo l'oggetto della richiesta come parametro.

Il seguente esempio di codice C# mostra le fasi precedenti.

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();
}
```

Esempio: indici secondari locali che utilizzano l'API di basso livello AWS SDK per .NET

Il seguente esempio di codice C# mostra come utilizzare gli indici secondari locali in Amazon DynamoDB. Nell'esempio viene creata una tabella denominata `CustomerOrders` con una chiave di partizione `CustomerId` e una chiave di ordinamento `OrderId`. In questa tabella esistono due indici secondari locali:

- `OrderCreationDateIndex`: la chiave di ordinamento è `OrderCreationDate` e i seguenti attributi sono proiettati sull'indice.
  - `ProductCategory`
  - `ProductName`
  - `OrderStatus`
  - `ShipmentTrackingId`

- `IsOpenIndex`: la chiave di ordinamento è `IsOpen` e tutti gli attributi della tabella vengono proiettati nell'indice.

Una volta creata la tabella `CustomerOrders`, il programma carica la tabella con i dati che rappresentano gli ordini dei clienti. Quindi esegue una query sui dati utilizzando gli indici secondari locali. Infine, il programma elimina la tabella `CustomerOrders`.

Per le istruzioni dettagliate per eseguire il test del seguente esempio, consulta Esempi di codice .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;
                }
            }
        }
    }
}
```

Utilizzo di indici secondari locali nella AWS CLI di DynamoDB

Puoi utilizzare la AWS CLI per creare una tabella Amazon DynamoDB con uno o più indici secondari locali, descrivere gli indici nella tabella ed eseguire query utilizzando gli indici.

Argomenti

- Creazione di una tabella con un indice secondario locale
- Descrizione di una tabella con un indice secondario locale
- Esecuzione di query su un indice secondario locale

Creazione di una tabella con un indice secondario locale

Gli indici secondari locali devono essere creati al momento della creazione di una tabella. A tale scopo, utilizza il parametro `create-table` e fornisci le specifiche per uno o più indici secondari locali. Nel seguente esempio viene creata una tabella (`Music`) per contenere le informazioni sui brani in una raccolta musicale. La chiave di partizione è `Artist` e la chiave di ordinamento è `SongTitle`. Un indice secondario, `AlbumTitleIndex` sull'attributo `AlbumTitle`, facilita le query in base al titolo dell'album.

```
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\"]}}]"
```

È necessario attendere fino a quando DynamoDB crea la tabella e imposta lo stato su `ACTIVE`. Dopodiché, puoi iniziare a inserire item di dati nella tabella. È possibile utilizzare describe-table per determinare lo stato di avanzamento della creazione della tabella.

Descrizione di una tabella con un indice secondario locale

Per ottenere informazioni sugli indici secondari locali in una tabella, utilizza il parametro `describe-table`. Puoi accedere al nome, allo schema della chiave e agli attributi proiettati di ciascun indice.

```
aws dynamodb describe-table --table-name Music
```

Esecuzione di query su un indice secondario locale

Puoi utilizzare l'operazione `query` su un indice secondario locale nello stesso modo in cui esegui una `query` su una tabella. È necessario specificare il nome dell'indice, i criteri della query per la chiave di ordinamento dell'indice e gli attributi da restituire. In questo esempio, l'indice è `AlbumTitleIndex` e la chiave di ordinamento dell'indice è `AlbumTitle`.

Gli unici attributi restituiti sono quelli proiettati nell'indice. È possibile modificare questa query per selezionare anche attributi non chiave, ma ciò richiederebbe un'attività di recupero della tabella relativamente costosa. Per ulteriori informazioni sul recupero delle tabelle, consulta Proiezioni di attributi.

```
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"} }'
```

Gestione di flussi di lavoro complessi con transazioni DynamoDB

Le transazioni Amazon DynamoDB semplificano l'esperienza degli sviluppatori di apportare modifiche coordinate all-or-nothing a più elementi sia all'interno che tra tabelle. La transazioni forniscono

atomicità, coerenza, isolamento e durabilità (ACID) in DynamoDB, consentendo di conservare più facilmente dati corretti nelle applicazioni.

Puoi utilizzare la lettura e la APIs scrittura transazionale di DynamoDB per gestire flussi di lavoro aziendali complessi che richiedono l'aggiunta, l'aggiornamento o l'eliminazione di più elementi in un'unica operazione. all-or-nothing Ad esempio, uno sviluppatore di videogiochi può garantire che i profili dei giocatori siano aggiornati correttamente quando si scambiano le voci in un videogioco o effettuano acquisti all'interno del gioco.

Con l'API di scrittura di transazione, è possibile raggruppare diverse operazioni `Put`, `Update`, `Delete` e `ConditionCheck`. Puoi quindi inviare le operazioni come un'operazione `TransactWriteItems` singola con esito positivo o negativo a livello di unità. Lo stesso vale per diverse azioni `Get`, che possono essere raggruppate e inviate come un'operazione `TransactGetItems` singola.

Non è previsto alcun costo aggiuntivo per abilitare le transazioni per le tabelle DynamoDB. Si paga solo per le letture o le scritture che fanno parte della transazione. DynamoDB esegue due letture o scritture sottostanti per ciascun elemento nella transazione: uno per preparare la transazione uno per eseguire il commit della transazione. Queste due read/write operazioni sottostanti sono visibili nelle CloudWatch metriche di Amazon.

Per iniziare a utilizzare le transazioni DynamoDB, scarica l'SDK AWS più recente o (). AWS Command Line Interface AWS CLI Quindi segui la procedura Esempio di transazioni di DynamoDB.

Le seguenti sezioni forniscono una panoramica dettagliata della transazione APIs e di come utilizzarla in DynamoDB.

Argomenti

- Transazioni Amazon DynamoDB: come funzionano
- Utilizzo di IAM con transazioni DynamoDB
- Esempio di transazioni di DynamoDB

Transazioni Amazon DynamoDB: come funzionano

Con le transazioni Amazon DynamoDB, puoi raggruppare più azioni e inviarle come singola all-or-nothing `TransactWriteItems` operazione. `TransactGetItems` Le seguenti sezioni descrivono le operazioni delle API, la gestione della capacità, le best practice e altri dettagli sulle operazioni transazionali in DynamoDB.

Argomenti

- TransactWriteItems API
- TransactGetItems API
- Livelli di isolamento per le transazioni DynamoDB
- Gestione dei conflitti nelle transazioni in DynamoDB
- Utilizzo delle transazioni in APIs DynamoDB Accelerator (DAX)
- Gestione della capacità per le transazioni
- Best practice per le transazioni
- Utilizzo di tabelle transazionali con tabelle globali APIs
- DynamoDB Transactions e la libreria client delle transazioni AWSLabs

TransactWriteItems API

`TransactWriteItems`è un'operazione di scrittura sincrona e idempotente che raggruppa fino a 100 azioni di scrittura in un'unica operazione. all-or-nothing Queste azioni possono indirizzare fino a 100 elementi distinti in una o più tabelle DynamoDB all'interno dello AWS stesso account e nella stessa regione. La dimensione aggregata degli elementi nella transazione non può superare i 4 MB. Le operazioni vengono completate in modo atomico, in maniera tale che abbiano tutte esito positivo oppure abbiano tutte esito negativo.

> **Note**
>
> - Un'operazione `TransactWriteItems` differisce da un'operazione `BatchWriteItem` nel fatto che tutte le operazioni che contiene devono essere completate con successo, altrimenti non viene apportata alcuna modifica. Con un'operazione `BatchWriteItem`, è possibile che solo alcune azioni nel batch abbiano esito positivo, contrariamente alle altre.
> - Le transazioni non possono essere eseguite utilizzando gli indici.

Non è possibile fare riferimento allo stesso item con diverse operazioni all'interno della stessa transazione. Ad esempio, non è possibile eseguire un'operazione `ConditionCheck` e `Update` per lo stesso item nella stessa transazione.

È possibile aggiungere uno dei seguenti tipi di operazioni a una transazione:

- `Put`: avvia un'operazione `PutItem` per creare un nuovo elemento o sostituire un vecchio elemento con uno nuovo, in base a condizioni o senza specificare alcuna condizione.
- `Update`: avvia un'operazione `UpdateItem` per modificare gli attributi di un elemento esistente o aggiungere un nuovo elemento alla tabella se non è già presente. Utilizza questa operazione per aggiungere, eliminare o aggiornare attributi su un item esistente in base a condizioni o senza una condizione.
- `Delete`: avvia un'operazione `DeleteItem` per eliminare un singolo elemento in una tabella identificata dalla chiave primaria.
- `ConditionCheck`: verifica che un elemento esista o controlla la condizione di attributi specifici dell'elemento.

Quando una transazione viene completata in DynamoDB, le sue modifiche iniziano a propagarsi agli indici secondari globali (), ai flussi e ai backupGSIs. Questa propagazione avviene gradualmente: i record dei flussi della stessa transazione potrebbero apparire in momenti diversi e potrebbero risultare interleaved con i record di altre transazioni. I consumer dei flussi non dovrebbero presumere l'atomicità delle transazioni o garanzie d'ordine.

Per garantire un'istantanea atomica degli elementi modificati in una transazione, utilizza l'operazione per leggere tutti gli elementi pertinenti insieme. TransactGetItems Questa operazione fornisce una visione coerente dei dati, garantendo la visualizzazione di tutte le modifiche rispetto a una transazione completata o di nessuna.

Poiché la propagazione non è immediata, se una tabella viene ripristinata da backup (RestoreTableFromBackup) o esportata in un punto temporale (ExportTableToPointInTime) durante la propagazione, può contenere solo alcune delle modifiche apportate durante una transazione recente.

Idempotenza

È possibile includere un token client quando si effettua una chiamata `TransactWriteItems` per garantire che la richiesta sia idempotente. Rendere le transazioni idempotenti impedisce errori nelle applicazioni se la stessa operazione viene inviata più volte a causa di un timeout della connessione o altri problemi di connettività.

Se la chiamata `TransactWriteItems` originale è andata a buon fine, allora le successive chiamate `TransactWriteItems` con lo stesso token client hanno esito positivo senza apportare alcuna modifica. Se viene impostato il parametro `ReturnConsumedCapacity`, la chiamata

`TransactWriteItems` iniziale restituisce il numero di unità di capacità in scrittura consumate mentre venivano apportate le modifiche. Le successive chiamate `TransactWriteItems` con lo stesso token client restituiscono il numero di unità di capacità in lettura consumate durante la lettura dell'item.

Considerazioni importanti sull'idempotenza

- Un token client è valido per 10 minuti dopo il termine della richiesta che lo utilizza. Dopo 10 minuti, qualsiasi richiesta che utilizza lo stesso token client viene trattata come una nuova richiesta. Lo stesso token client non deve essere riutilizzato per la stessa richiesta dopo 10 minuti.
- Se si ripete la richiesta con lo stesso token client all'interno della finestra di idempotenza di 10 minuti, ma vengono modificati altri parametri, DynamoDB restituisce un'eccezione `IdempotentParameterMismatch`.

Gestione degli errori per scrittura

Le transazioni di scrittura non hanno esito positivo nelle seguenti circostanze:

- Quando una condizione in una delle espressioni di condizione non viene soddisfatta.
- Quando un errore di convalida della transazione si verifica in quanto un'operazione all'interno della stessa operazione `TransactWriteItems` mira allo stesso item.
- Quando una richiesta `TransactWriteItems` è in conflitto con un'operazione `TransactWriteItems` in corso su uno o più item nella richiesta `TransactWriteItems`. In questo caso, la richiesta ha esito negativo con una `TransactionCanceledException`.
- Quando la capacità assegnata è insufficiente per il completamento della transazione.
- Quando la dimensione di un item diventa eccessiva (superiore a 400 KB), un indice secondario locale (LSI) diventa eccessivo o si verifica un errore di convalida simile a causa delle modifiche apportate dalla transazione.
- Quando è presente un errore utente, come un formato di dati invalido.

Per ulteriori informazioni sulla gestione dei conflitti con le operazioni `TransactWriteItems`, consulta Gestione dei conflitti nelle transazioni in DynamoDB.

TransactGetItems API

`TransactGetItems` è un'operazione di lettura sincrona che raggruppa fino a 100 operazioni `Get`. Queste azioni possono indirizzare fino a 100 elementi distinti in una o più tabelle DynamoDB

all'interno dello stesso account e della AWS stessa regione. La dimensione aggregata degli elementi nella transazione non può superare i 4 MB.

Le operazioni `Get` vengono eseguite in modo atomico, in maniera tale che abbiano tutte esito positivo oppure abbiano tutte esito negativo.

- `Get`: avvia un'operazione `GetItem` per recuperare un set di attributi dell'elemento con la chiave primaria specificata. Se non viene trovato un item corrispondente, `Get` non restituisce dati.

Gestione degli errori per lettura

Le transazioni di lettura non hanno esito positivo nelle seguenti circostanze:

- Quando una richiesta `TransactGetItems` è in conflitto con un'operazione `TransactWriteItems` in corso su uno o più item nella richiesta `TransactGetItems`. In questo caso, la richiesta ha esito negativo con una `TransactionCanceledException`.
- Quando la capacità assegnata è insufficiente per il completamento della transazione.
- Quando è presente un errore utente, come un formato di dati invalido.

Per ulteriori informazioni sulla gestione dei conflitti con le operazioni `TransactGetItems`, consulta Gestione dei conflitti nelle transazioni in DynamoDB.

Livelli di isolamento per le transazioni DynamoDB

I livelli di isolamento delle operazioni transazionali (`TransactWriteItems` o `TransactGetItems`) e di altre operazioni sono:

SERIALIZZABILI

L'isolamento serializzabile garantisce che i risultati di diverse operazioni simultanee siano le stesse, come se nessuna operazione avesse inizio prima che l'ultima fosse finita.

L'isolamento serializzabile è presente tra i seguenti tipi di operazione:

- Tra una qualsiasi operazione transazionale e una qualsiasi operazione di scrittura standard (`PutItem`, `UpdateItem` o `DeleteItem`).
- Tra una qualsiasi operazione transazionale e una qualsiasi operazione di lettura standard (`GetItem`).

- Tra un'operazione `TransactWriteItems` e un'operazione `TransactGetItems`.

Nonostante l'isolamento serializzabile sia presente tra le operazioni transazionali e ogni singola scrittura standard in un'operazione `BatchWriteItem`, l'isolamento serializzabile non è presente tra la transazione e l'operazione `BatchWriteItem` come un'unità.

Allo stesso modo, il livello di isolamento tra un'operazione transazionale e un `GetItems` individuale in un'operazione `BatchGetItem` è serializzabile. Tuttavia, il livello di isolamento tra la transazione e l'operazione `BatchGetItem`come unità è read-committed.

Una singola richiesta `GetItem` è serializzabile rispetto a una richiesta `TransactWriteItems` in uno dei due modi seguenti, prima o dopo la richiesta `TransactWriteItems`. Più richieste `GetItem`, rispetto alle chiavi in una richiesta `TransactWriteItems`simultanea, possono essere eseguite in qualsiasi ordine, e quindi i risultati sono letti–commessi.

Ad esempio, se le richieste `GetItem` per l'elemento A e l'elemento B vengono eseguite contemporaneamente a una richiesta `TransactWriteItems` che modifica sia l'elemento A che l'elemento B, esistono quattro possibilità:

- Entrambe le richieste `GetItem` vengono eseguite prima della richiesta `TransactWriteItems`.
- Entrambe le richieste `GetItem` vengono eseguite dopo la richiesta `TransactWriteItems`.
- La richiesta `GetItem` per l'elemento A viene eseguita prima della richiesta `TransactWriteItems`. Per l'elemento B, `GetItem` viene eseguito dopo `TransactWriteItems`.
- La richiesta `GetItem` per l'elemento B viene eseguita prima della richiesta `TransactWriteItems`. Per l'elemento A, `GetItem` viene eseguito dopo `TransactWriteItems`.

Sarebbe opportuno utilizzare `TransactGetItems` se si preferisce un livello di isolamento serializzabile per più richieste `GetItem`.

Se viene effettuata una lettura non transazionale su più elementi che facevano parte della stessa richiesta di scrittura della transazione in corso, è possibile che si sia in grado di leggere il nuovo stato di alcuni elementi e il vecchio stato degli altri elementi. Sarà possibile leggere il nuovo stato di tutti gli elementi che facevano parte della richiesta di scrittura della transazione solo quando si riceverà una risposta positiva per la scrittura transazionale, a indicare che la transazione è stata completata.

Una volta completata con successo la transazione e ricevuta una risposta, le successive operazioni di lettura a coerenza finale possono comunque restituire il vecchio stato per un breve periodo a

causa di un modello di consistenza finale di DynamoDB. Per garantire la lettura della maggior parte up-to-date dei dati subito dopo una transazione, è necessario utilizzare letture fortemente coerenti `ConsistentRead` impostando su true.

READ-COMMITTED

L'isolamento Read-committed garantisce che le operazioni di lettura restituiscano sempre valori sottoposti a commit per un elemento: la lettura non presenterà mai una vista dell'elemento che rappresenta uno stato di una scrittura transazionale terminata con un errore. L'isolamento read-committed non impedisce le modifiche dell'item immediatamente dopo l'operazione di lettura.

Il livello di isolamento è read-committed tra una qualsiasi operazione transazionale e una qualsiasi operazione di lettura che coinvolga diverse letture standard (`BatchGetItem`, `Query` o `Scan`). Se una scrittura transazionale aggiorna un elemento durante un'operazione `BatchGetItem`, `Query` o `Scan`, la fase successiva dell'operazione di lettura restituisce il nuovo valore sottoposto a commit, con `ConsistentRead`) o eventualmente un valore sottoposto a commit precedente (letture a consistenza finale).

Riepilogo dell'operazione

Per riassumere, la seguente tabella mostra i livelli di isolamento tra un'operazione transazionale (`TransactWriteItems` o `TransactGetItems`) e altre operazioni.

Operazioni	Livello di isolamento
`DeleteItem`	Serializzabile
`PutItem`	Serializzabile
`UpdateItem`	Serializzabile
`GetItem`	Serializzabile
`BatchGetItem`	Read-committed*
`BatchWriteItem`	NON serializzabile*
`Query`	Read-committed
`Scan`	Read-committed

Operazioni	Livello di isolamento
Altra operazione transazionale	Serializzabile

I livelli contrassegnati da un asterisco (*) si applicano all'operazione come un'unità. Tuttavia, le singole operazioni all'interno di tali operazioni hanno un livello di isolamento serializzabile.

Gestione dei conflitti nelle transazioni in DynamoDB

Un conflitto transazionale si può verificare durante richieste simultanee a livello di voci su una singola voce all'interno di una transazione. I conflitti tra transazioni possono verificarsi nei seguenti scenari:

- Una richiesta `PutItem`,`UpdateItem` e `DeleteItem` per una voce è in conflitto con una richiesta `TransactWriteItems` continua che include la stessa voce.
- Una voce all'interno di una richiesta `TransactWriteItems` è parte di un'altra richiesta `TransactWriteItems` continua.
- Una voce all'interno di una richiesta `TransactGetItems` è parte di una richiesta continua `TransactWriteItems`, `BatchWriteItem`, `PutItem`, `UpdateItem` o `DeleteItem`.

> **Note**
>
> - Quando una richiesta `PutItem`, `UpdateItem` o `DeleteItem` viene rifiutata, la richiesta dà esito negativo con un `TransactionConflictException`.
> - Se una richiesta a livello di voce all'interno di `TransactWriteItems` o `TransactGetItems` viene rifiutata, la richiesta dà esito negativo con `TransactionCanceledException`. Se la richiesta fallisce, AWS SDKs non riprovare.
>
> Se si utilizza il AWS SDK per Java, l'eccezione contiene l'elenco di CancellationReasons, ordinato in base all'elenco di elementi nel parametro di `TransactItems` richiesta. Per altre lingue, una rappresentazione della stringa dell'elenco è inclusa nel messaggio di errore dell'eccezione.
> - Quando un'operazione `TransactWriteItems` o `TransactGetItems` in corso è in conflitto con una richiesta `GetItem` simultanea, entrambe le operazione possono avere esito positivo.

La TransactionConflict CloudWatch metrica viene incrementata per ogni richiesta a livello di elemento non riuscita.

Utilizzo delle transazioni in APIs DynamoDB Accelerator (DAX)

`TransactWriteItems` e `TransactGetItems` sono entrambi supportati in DynamoDB Accelerator (DAX) con gli stessi livelli di isolamento di DynamoDB.

`TransactWriteItems` scrive tramite DAX. DAX passa una chiamata `TransactWriteItems` a DynamoDB e restituisce la risposta. Per popolare la cache dopo la scrittura, DAX chiama `TransactGetItems` in background per ogni elemento nell'operazione `TransactWriteItems`che consuma unità di capacità di lettura aggiuntive. Per ulteriori informazioni, consulta Gestione della capacità per le transazioni. Questa funzionalità consente di mantenere la logica dell'applicazione semplice e di utilizzare DAX per operazioni sia transazionali che non transazionali.

Le chiamate `TransactGetItems` vengono passate tramite DAX senza che gli elementi vengano memorizzati nella cache in locale. Si tratta dello stesso comportamento utilizzato per una lettura fortemente coerente in DAX. APIs

Gestione della capacità per le transazioni

Non è previsto alcun costo aggiuntivo per abilitare le transazioni per le tabelle DynamoDB. Si paga solo per le letture o le scritture che fanno parte della transazione. DynamoDB esegue due letture o scritture sottostanti per ciascun elemento nella transazione: uno per preparare la transazione uno per eseguire il commit della transazione. Le due read/write operazioni sottostanti sono visibili nelle CloudWatch metriche di Amazon.

Pianifica le letture e le scritture aggiuntive richieste da Transactional APIs per fornire capacità alle tabelle. Ad esempio, si supponga che l'applicazione esegua una transazione al secondo e che ciascuna transazione scriva nella tabella tre elementi da 500 byte. Ogni articolo richiede due unità di capacità di scrittura (WCUs): una per preparare la transazione e l'altra per confermare la transazione. Pertanto, è necessario assegnarne sei WCUs alla tabella.

Se nell'esempio precedente si utilizzava DynamoDB Accelerator (DAX), si utilizzerebbero anche due unità di capacità di lettura RCUs () per ogni elemento della chiamata. `TransactWriteItems` Quindi è necessario aggiungere altre RCUs sei unità alla tabella.

Analogamente, se l'applicazione esegue una transazione di lettura al secondo e ogni transazione legge tre elementi da 500 byte della tabella, è necessario fornire sei unità di capacità di lettura

(RCUs) alla tabella. La lettura di ogni elemento richiede due elementi RCUs: uno per preparare la transazione e uno per confermare la transazione.

Inoltre, il comportamento dell'SDK predefinito prevede di ritentare le transazioni in caso di un'eccezione `TransactionInProgressException`. Pianifica le unità di capacità di lettura aggiuntive (RCUs) utilizzate da questi nuovi tentativi. Lo stesso vale qualora si ritentino le transazioni nel proprio codice utilizzandone uno `ClientRequestToken`.

Best practice per le transazioni

Durante l'utilizzo delle transazioni DynamoDB, considerare le seguenti best practice suggerite.

- Abilita il dimensionamento automatico o assicurati di aver effettuato il provisioning di una capacità di throughput sufficiente da eseguire le due operazioni di lettura o scrittura per ogni item della transazione.
- Se non utilizzi un SDK AWS fornito, includi un `ClientRequestToken` attributo quando effettui una `TransactWriteItems` chiamata per assicurarti che la richiesta sia idempotente.
- Non raggruppare le operazioni in una transazione qualora non sia necessario. Ad esempio, se una singola transazione con 10 operazioni può essere suddivisa in diverse transazioni senza compromettere la correttezza dell'applicazione, si raccomanda di frazionare la transazione. Le transazioni più semplici migliorano il throughput e hanno maggiori probabilità di successo.
- Diverse transazioni che aggiornano gli stessi item simultaneamente possono causare conflitti che annullano le transazioni. Per minimizzare tali conflitti, si consiglia di utilizzare le seguenti best practice DynamoDB per la modellazione dei dati.
- Se un set di attributi viene spesso aggiornato tra più item come parte di una transazione singola, considera di raggruppare gli attributi in un singolo item per ridurre l'ambito della transazione.
- Evita l'uso di transazioni per l'inserimento di dati in blocco. Per le scritture in blocco, è consigliabile utilizzare `BatchWriteItem`.

Utilizzo di tabelle transazionali con tabelle globali APIs

Le operazioni transazionali forniscono garanzie di atomicità, coerenza, isolamento e durabilità (ACID) solo all'interno della AWS regione in cui è stata richiamata l'API di scrittura. Le transazioni non sono supportate tra le Regioni nelle tabelle globali. Ad esempio, supponi di avere una tabella globale con repliche nelle Regioni Stati Uniti orientali (Ohio) e Stati Uniti occidentali (Oregon) e di eseguire un'operazione `TransactWriteItems` nella Regione Stati Uniti orientali (Virginia settentrionale). In

questo caso, è possibile osservare transazioni parzialmente completate nella Regione degli Stati Uniti occidentali (Oregon) mentre le modifiche vengono replicate. Le modifiche vengono replicate in altre Regioni solo dopo essere state confermate nella Regione di origine.

DynamoDB Transactions e la libreria client delle transazioni AWSLabs

Le transazioni DynamoDB forniscono un sostituto più economico, robusto e performante della libreria client delle transazioni. AWSLabs Ti suggeriamo di aggiornare le tue applicazioni per utilizzare la transazione nativa lato server. APIs

Utilizzo di IAM con transazioni DynamoDB

Puoi usare AWS Identity and Access Management (IAM) per limitare le azioni che le operazioni transazionali possono eseguire in Amazon DynamoDB. Per ulteriori informazioni sull'uso di policy IAM in DynamoDB, consulta Policy basate su identità per DynamoDB.

Le autorizzazioni per le operazioni `Put`, `Update`, `Delete` e `Get` sono regolate dalle autorizzazioni utilizzate per le operazioni `PutItem`, `UpdateItem`, `DeleteItem` e `GetItem` sottostanti. Per l'operazione `ConditionCheck`, è possibile utilizzare l'autorizzazione `dynamodb:ConditionCheckItem` nelle policy IAM.

Di seguito sono riportati degli esempi di policy IAM che è possibile utilizzare per configurare le transazioni DynamoDB.

Esempio 1: consentire le operazioni transazionali

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"
            ]
        }
    ]
}
```

Esempio 2: consentire solo le operazioni transazionali

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"
                    ]
                }
            }
        }
    ]
}
```

Esempio 3: consentire le letture e le scritture non transazionali e bloccare le letture e le scritture transazionali

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"
             ]
         }
    ]
```

```
}
```

Esempio 4: impedire che le informazioni vengano restituite in caso di errore ConditionCheck

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"
                }
            }
        }
    ]
}
```

Esempio di transazioni di DynamoDB

Come esempio di una situazione in cui le transazioni Amazon DynamoDB possono essere utili, si consideri questa applicazione Java di esempio per un marketplace online.

L'applicazione ha tre tabelle DynamoDB nel backend:

- `Customers`: questa tabella archivia i dettagli sui clienti del marketplace. La sua chiave primaria è un identificatore univoco `CustomerId`.

- `ProductCatalog`: questa tabella archivia dettagli quali prezzo e disponibilità dei prodotti in vendita nel marketplace. La sua chiave primaria è un identificatore univoco `ProductId`.
- `Orders`: questa tabella archivia i dettagli sugli ordini del marketplace. La sua chiave primaria è un identificatore univoco `OrderId`.

Come effettuare un ordine

I seguenti frammenti di codice illustrano come utilizzare le transazioni DynamoDB per coordinare i vari passaggi necessari per creare ed elaborare un ordine. L'utilizzo di una singola all-or-nothing operazione garantisce che, se una parte della transazione fallisce, non vengano eseguite azioni nella transazione e non vengano apportate modifiche.

In questo esempio, si configura un ordine da un cliente il cui `customerId` è `09e8e9c8-ec48`. Si esegue quindi come una singola transazione utilizzando il seguente flusso di lavoro semplice di elaborazione degli ordini:

1. Determinare che l'ID cliente sia valido.
2. Assicurati che il prodotto sia `IN_STOCK` e aggiornare lo stato del prodotto in `SOLD`.
3. Assicurati che l'ordine non esista già e creare l'ordine.

Convalida del cliente

Per prima cosa, definire un'azione per verificare che un cliente con `customerId` uguale a `09e8e9c8-ec48` esista nella tabella dei clienti.

```
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 + ")");
```

Aggiornamento dello stato dei prodotti

Quindi, definire un'operazione per aggiornare lo stato del prodotto in `SOLD` se la condizione in cui lo stato del prodotto è attualmente impostato su `IN_STOCK` è `true`. La configurazione del parametro `ReturnValuesOnConditionCheckFailure` restituisce l'elemento se l'attributo dello stato del prodotto dell'elemento non era uguale 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);
```

Creazione dell'ordine

Infine, creare l'ordine se un ordine con quell'`OrderId` non esiste già.

```
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 + ")");
```

Esecuzione della transazione

L'esempio seguente illustra come eseguire le azioni definite in precedenza come singola all-or-nothing operazione.

```
   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());
   }
```

Lettura dei dettagli dell'ordine

L'esempio seguente mostra come leggere l'ordine completato in modo transazionale attraverso le tabelle `Orders` e `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());
}
```

Change Data Capture con Amazon DynamoDB

Molte applicazioni traggono vantaggio dalla possibilità di acquisire le modifiche apportate a elementi archiviati in una tabella DynamoDB, nel momento in cui si verificano tali modifiche. Di seguito sono riportati alcuni esempi di casi d'uso:

- Un'applicazione per dispositivi mobili popolare modifica i dati in una tabella DynamoDB alla velocità di migliaia di aggiornamenti al secondo. Un'altra applicazione acquisisce e archivia i dati relativi a questi aggiornamenti, fornendo metriche di near-real-time utilizzo per l'app mobile.
- Un'applicazione finanziaria modifica i dati del mercato azionario in una tabella DynamoDB. Diverse applicazioni in esecuzione in parallelo tracciano queste variazioni in tempo reale, calcolano e value-at-risk ribilanciano automaticamente i portafogli in base ai movimenti dei prezzi delle azioni.

- I sensori nei veicoli di trasporto e nelle attrezzature industriali inviano dati a una tabella DynamoDB. Diverse applicazioni monitorano le prestazioni e inviano avvisi di messaggistica quando viene rilevato un problema, prevedono eventuali difetti applicando algoritmi di machine learning e comprimono e archiviano i dati in Amazon Simple Storage Service (Amazon S3).
- Un'applicazione invia automaticamente notifiche ai dispositivi mobili di tutti gli amici inclusi in un gruppo non appena un amico carica una nuova immagine.
- Un nuovo cliente aggiunge dati a una tabella DynamoDB. Questo evento richiama un'altra applicazione che invia un'e-mail di benvenuto al nuovo cliente.

DynamoDB supporta lo streaming dei record di acquisizione dei dati delle modifiche a livello di elemento in tempo quasi reale. È possibile creare applicazioni che utilizzano questi flussi e agiscono in base al contenuto.

> **Note**
>
> L'aggiunta di tag ai flussi DynamoDB e l'utilizzo del controllo degli accessi basato su attributi (ABAC) con i flussi DynamoDB non sono supportati.

Il seguente video ti fornirà un'introduzione sul concetto di acquisizione dei dati di modifica.

Modalità di capacità delle tabelle

Argomenti

- Opzioni di streaming per Change Data Capture
- Utilizzo di Kinesis Data Streams per acquisire le modifiche apportate a Dynamo DB.
- Acquisizione dei dati di modifica per DynamoDB Streams

Opzioni di streaming per Change Data Capture

DynamoDB offre due modelli di streaming per l'acquisizione dei dati delle modifiche: Kinesis Data Streams per DynamoDB e DynamoDB Streams.

Per scegliere la soluzione più adatta per l'applicazione, la tabella seguente riassume le caratteristiche di ciascun modello di streaming.

Properties	Kinesis Data Streams per DynamoDB	DynamoDB Streams
Conservazione dei dati	Fino a 1 anno.	24 ore.
Supporto per Kinesis Client Library (KCL)	Supporta KCL, versioni 1.X, 2.X e 3.X.	Supporta KCL versioni 1.X e 2.X.
Numero di consumatori	Fino a 5 consumatori simultanei per partizione, o fino a 20 consumatori simultanei per partizione con fan-out potenziato.	Fino a 2 consumatori simultanei per partizione.
Quote di velocità effettiva	Illimitate.	Soggetto alle quote di velocità effettiva stabilite dalla tabella DynamoDB e dalla regione. AWS
Modello di distribuzione record	Pull model over HTTP GetRecordse con fan-out migliorato, Kinesis Data Streams trasferisce i record su HTTP/2 utilizzando. SubscribeToShard	Recupera il modello tramite HTTP utilizzando. GetRecords
Ordinamento dei record	L'attributo timestamp su ogni record di flusso può essere utilizzato per identificare l'ordine effettivo in cui si sono verificate le modifiche nella tabella DynamoDB.	Per ogni elemento modificato in una tabella DynamoDB, i record di flusso vengono visualizzati nella stessa sequenza delle modifiche effettive apportate all'elemento.
Duplicazione di record	Nel flusso potrebbero apparire occasionalmente record duplicati.	Nel flusso non vengono visualizzati record duplicati.

Properties	Kinesis Data Streams per DynamoDB	DynamoDB Streams
Opzioni di elaborazione del flusso	Elabora i record di flusso utilizzando AWS Lambda, il Servizio gestito da Amazon per Apache Flink, Kinesis data firehose o lo streaming ETL di AWS Glue.	Elabora i record di flusso utilizzando AWS Lambda o l'adattatore DynamoDB Streams Kinesis.
Livello di durabilità	Zone di disponibilità per fornire un failover automatico senza interruzioni.	Zone di disponibilità per fornire un failover automatico senza interruzioni.

È possibile abilitare entrambi i modelli di streaming sulla stessa tabella DynamoDB.

Il video seguente analizza ulteriormente le differenze tra le due opzioni.

DynamoDB Streams e Kinesis Data Streams a confronto

Utilizzo di Kinesis Data Streams per acquisire le modifiche apportate a Dynamo DB.

Utilizzare Amazon Kinesis Data Streams per acquisire le modifiche apportate a Amazon DynamoDB.

Kinesis Data Streams acquisisce le modifiche a livello di elemento in qualsiasi tabella DynamoDB e le replica in un flusso dei dati Kinesis. Le applicazioni possono accedere a questo flusso e visualizzare le modifiche a livello di elemento quasi in tempo reale. Puoi acquisire e archiviare in modo continuo terabyte di dati ogni ora. Puoi sfruttare tempi di conservazione dei dati più lunghi e, grazie alla funzionalità di fan-out migliorata, puoi raggiungere contemporaneamente due o più applicazioni downstream. Altri vantaggi includono controlli aggiuntivi e trasparenza della sicurezza.

Il flusso di dati Kinesis consente inoltre di accedere ad Amazon Data Firehose e al Servizio gestito da Amazon per Apache Flink. Questi servizi consentono di creare applicazioni per utilizzare pannelli di controllo in tempo reale, generare avvisi, implementare prezzi e inserzioni dinamici ed eseguire analisi dei dati e algoritmi di machine learning sofisticati.

> **Note**
>
> L'utilizzo di Kinesis Data Streams per DynamoDB è soggetto ai prezzi di Kinesis Data Streams per il flusso di dati e ai prezzi di DynamoDB per la tabella di origine.

Funzionamento di Kinesis Data Streams con DynamoDB

Quando un flusso dei dati Kinesis è abilitato per una tabella DynamoDB, quest'ultima invia un record di dati che acquisisce eventuali modifiche ai dati della tabella. Questo record di dati include:

- L'ora specifica in cui qualsiasi elemento è stato creato, aggiornato o eliminato di recente
- La chiave primaria di quell'elemento
- Uno snapshot del record prima della modifica
- Uno snapshot del record dopo la modifica

Questi record di dati vengono acquisiti e pubblicati quasi in tempo reale. Dopo essere stati scritti sul flusso dei dati Kinesis, possono essere letti come qualsiasi altro record. È possibile utilizzare la Kinesis Client Library, utilizzare AWS Lambda, chiamare l'API Kinesis Data Streams e utilizzare altri servizi connessi. Per ulteriori informazioni, consulta Lettura di dati da Amazon Kinesis Data Streams nella Guida per gli sviluppatori di Amazon Kinesis Data Streams.

Queste modifiche dei dati vengono acquisite anche in modo asincrono. Kinesis non ha alcun impatto sulle prestazioni di una tabella da cui esegue lo streaming. I record del flusso archiviati nel flusso dei dati Kinesis sono crittografati anche a riposo. Per ulteriori informazioni, consulta Protezione dei dati in Amazon Kinesis Data Streams.

I record del flusso di dati Kinesis potrebbero essere visualizzati in un ordine diverso rispetto alle modifiche apportate all'elemento. Le notifiche dello stesso elemento potrebbero apparire anche più di una volta nel flusso. È possibile controllare l'attributo `ApproximateCreationDateTime` per identificare l'ordine in cui sono state apportate le modifiche all'elemento e i record duplicati.

Abilitando un flusso di dati Kinesis come destinazione di streaming di una tabella DynamoDB, è possibile configurare la precisione dei valori `ApproximateCreationDateTime` in millisecondi o microsecondi. Per impostazione predefinita, `ApproximateCreationDateTime` indica l'ora della modifica in millisecondi. Inoltre, è possibile modificare questo valore su una destinazione di streaming attiva. Dopo tale aggiornamento, i record di flussi scritti su Kinesis avranno i valori `ApproximateCreationDateTime` della precisione desiderata.

I valori binari scritti in DynamoDB devono essere codificati in formato con codifica base64. Tuttavia, quando i record di dati vengono scritti su un flusso di dati Kinesis, questi valori binari codificati vengono codificati una seconda volta con la codifica base64. Quando leggono questi record da un flusso di dati Kinesis, per recuperare i valori binari non elaborati, le applicazioni devono decodificare questi valori due volte.

DynamoDB addebita l'utilizzo di Kinesis Data Streams nelle unità di acquisizione dei dati di modifica. 1 KB di modifica per singolo elemento conta come un'unità di acquisizione dei dati di modifica. I KB di modifica in ogni elemento sono calcolati in base alla più grande delle immagini "prima" e "dopo" dell'elemento scritto nello stream, utilizzando la stessa logica di consumo di unità di capacità per operazioni di scrittura. In modo analogo al funzionamento della modalità on demand di DynamoDB, non è necessario effettuare il provisioning del throughput della capacità per le unità di acquisizione dei dati di modifica.

Attivazione di un flusso di dati Kinesis per la tabella DynamoDB

Puoi abilitare o disabilitare lo streaming su Kinesis dalla tabella DynamoDB esistente utilizzando l' Console di gestione AWS, l' AWS SDK o il (). AWS Command Line Interface AWS CLI

- Puoi trasmettere dati da DynamoDB a Kinesis Data Streams solo nello stesso account e nella stessa regione AWS della tabella. AWS
- Puoi eseguire lo streaming dei dati solo da una tabella DynamoDB a un flusso dei dati Kinesis.

Applicazione di modifiche a una destinazione di un flusso di dati Kinesis su una tabella DynamoDB

Per impostazione predefinita, tutti i record del flusso di dati Kinesis includono un attributo `ApproximateCreationDateTime`. Questo attributo rappresenta un timestamp in millisecondi dell'ora approssimativa in cui ogni record è stato creato. È possibile modificare la precisione di questi valori utilizzando https://console.aws.amazon.com/kinesis, l'SDK o il AWS CLI

Nozioni di base su Kinesis Data Streams per Amazon DynamoDB

Questa sezione descrive come utilizzare Kinesis Data Streams per le tabelle Amazon DynamoDB con la console Amazon DynamoDB, il () e l'API AWS Command Line Interface .AWS CLI

Creazione di un flusso di dati Amazon Kinesis attivo

Tutti questi esempi utilizzano la tabella DynamoDB `Music` che è stata creata come parte del tutorial Nozioni di base su DynamoDB.

Per ulteriori informazioni su come creare consumatori e connettere il flusso di dati Kinesis ad altri servizi AWS , consulta Leggere i dati dal flusso di dati Amazon Kinesis, nella Guida per gli sviluppatori del flusso di dati Amazon Kinesis.

> **Note**
>
> Quando utilizzi per la prima volta le partizioni KDS, consigliamo di impostarle in modo che aumentino o diminuiscano in base ai modelli di utilizzo. Dopo aver accumulato ulteriori dati sui modelli di utilizzo, è possibile adattare le partizioni nel flusso di conseguenza.

Console

1. Accedi Console di gestione AWS e apri la console Kinesis all'indirizzo. https://console.aws.amazon.com/kinesis/
2. Scegli Crea flusso di dati e segui le istruzioni per creare un flusso denominato `samplestream`.
3. Apri la console DynamoDB all'indirizzo. https://console.aws.amazon.com/dynamodb/
4. Nel riquadro di navigazione sul lato sinistro della console scegli Tables (Tabelle).
5. Seleziona la tabella Music.
6. Scegli la scheda Exports and streams (Esportazioni e flussi).
7. (Facoltativo) Nella sezione Dettagli del flusso di dati Amazon Kinesis, è possibile modificare la precisione del timestamp del record da microsecondi (impostazione predefinita) a millisecondi.
8. Scegli samplestream dall'elenco a discesa.
9. Seleziona il pulsante Attiva.

AWS CLI

1. Crea un flusso di dati Kinesis denominato `samplestream` utilizzando il comando create-stream.

   ```
   aws kinesis create-stream --stream-name samplestream --shard-count 3
   ```

   Prima di configurare il numero di partizioni per il flusso di dati Kinesis, consulta Considerazioni sulla gestione delle partizioni per Kinesis Data Streams.

2. Verificare che il flusso Kinesis sia attivo e pronto per l'uso utilizzando il comando describe-stream.

```
aws kinesis describe-stream --stream-name samplestream
```

3. Abilitare lo streaming di Kinesis sulla tabella DynamoDB utilizzando il comando DynamoDB `enable-kinesis-streaming-destination`. Sostituisci il valore `stream-arn` con quello restituito da `describe-stream` nella fase precedente. Facoltativamente, abilita lo streaming con una precisione più granulare (microsecondi) dei valori di timestamp restituiti su ogni record.

   Abilita lo streaming con la precisione del timestamp in microsecondi:

```
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
```

   Oppure abilita lo streaming con la precisione del timestamp predefinita (millisecondi):

```
aws dynamodb enable-kinesis-streaming-destination \
  --table-name Music \
  --stream-arn arn:aws:kinesis:us-west-2:12345678901:stream/samplestream
```

4. Abilitare lo streaming di Kinesis sulla tabella DynamoDB utilizzando il comando `describe-kinesis-streaming-destination` DynamoDB.

```
aws dynamodb describe-kinesis-streaming-destination --table-name Music
```

5. Scrivere i dati nella tabella DynamoDB utilizzando il comando `put-item`, come descritto nella Guida per gli sviluppatori di 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. Utilizza il comando get-record della CLI di Kinesis per recuperare il contenuto del flusso Kinesis. Quindi utilizzare il seguente frammento di codice per deserializzare il contenuto del flusso.

```
/**
 * 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. Seguire le istruzioni contenute nella guida per gli sviluppatori di Kinesis Data Streams per creare un flusso di dati Kinesis denominato `samplestream` tramite Java.

   Prima di configurare il numero di partizioni per il flusso di dati Kinesis, consulta Considerazioni sulla gestione delle partizioni per Kinesis Data Streams.

2. Utilizzo il seguente frammento di codice per abilitare lo streamingKinesis sulla tabella DynamoDB. Facoltativamente, abilita lo streaming con una precisione più granulare (microsecondi) dei valori di timestamp restituiti su ogni record.

   Abilita lo streaming con la precisione del timestamp in microsecondi:

```
EnableKinesisStreamingConfiguration enableKdsConfig =
 EnableKinesisStreamingConfiguration.builder()

 .approximateCreationDateTimePrecision(ApproximateCreationDateTimePrecision.MICROSECOND)
  .build();

EnableKinesisStreamingDestinationRequest enableKdsRequest =
 EnableKinesisStreamingDestinationRequest.builder()
  .tableName(tableName)
  .streamArn(kdsArn)
  .enableKinesisStreamingConfiguration(enableKdsConfig)
  .build();

EnableKinesisStreamingDestinationResponse enableKdsResponse =
 ddbClient.enableKinesisStreamingDestination(enableKdsRequest);
```

   Oppure abilita lo streaming con la precisione del timestamp predefinita (millisecondi):

```
EnableKinesisStreamingDestinationRequest enableKdsRequest =
 EnableKinesisStreamingDestinationRequest.builder()
  .tableName(tableName)
  .streamArn(kdsArn)
  .build();

EnableKinesisStreamingDestinationResponse enableKdsResponse =
 ddbClient.enableKinesisStreamingDestination(enableKdsRequest);
```

3. Segui le istruzioni contenute nella guida per gli sviluppatori di Kinesis Data Streams per leggere dal flusso di dati creato.
4. Utilizza il seguente frammento di codice per deserializzare il contenuto del flusso.

```
/**
 * 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());
}
```

Applicazione di modifiche a un flusso di dati Amazon Kinesis attivo

Questa sezione descrive come apportare modifiche a una configurazione attiva di Kinesis Data Streams for DynamoDB utilizzando la console e l'API. AWS CLI

Console di gestione AWS

1. Apri la console DynamoDB all'indirizzo https://console.aws.amazon.com/dynamodb/
2. Vai alla tabella.
3. Seleziona la scheda Esportazioni e flussi.

AWS CLI

1. Chiama `describe-kinesis-streaming-destination` per confermare che il flusso sia `ACTIVE`.
2. Chiama `UpdateKinesisStreamingDestination`, come in questo esempio:

```
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. Chiama `describe-kinesis-streaming-destination` per confermare che il flusso sia `UPDATING`.
4. Chiama `describe-kinesis-streaming-destination` periodicamente fino a quando lo stato dello streaming non è nuovamente `ACTIVE`. In genere sono necessari fino a 5 minuti perché gli aggiornamenti alla precisione del timestamp entrino in vigore. Una volta che lo stato è aggiornato, significa che l'aggiornamento è completo e il nuovo valore di precisione verrà applicato ai record futuri.
5. Scrivi nella tabella usando `putItem`.
6. Utilizza il comando `get-records` di Kinesis per recuperare il contenuto del flusso.
7. Verifica che `ApproximateCreationDateTime` delle scritture presenti la precisione desiderata.

API Java

1. Fornisci un frammento di codice che costruisce una richiesta `UpdateKinesisStreamingDestination` e una risposta `UpdateKinesisStreamingDestination`.
2. Fornisci un frammento di codice che costruisce una richiesta `DescribeKinesisStreamingDestination` e una `DescribeKinesisStreamingDestination response`.
3. Chiama `describe-kinesis-streaming-destination` periodicamente fino a quando lo stato dello streaming non è nuovamente `ACTIVE`, a indicare che l'aggiornamento è completo e il nuovo valore di precisione verrà applicato ai record futuri.
4. Effettua le scritture sulla tabella.
5. Effettua la lettura dal flusso e deserializza il contenuto del flusso.
6. Verifica che `ApproximateCreationDateTime` delle scritture presenti la precisione desiderata.

Utilizzo di shard e metriche con i flussi DynamoDB e il flusso di dati Kinesis

Considerazioni sulla gestione delle partizioni per Kinesis Data Streams

Un flusso dei dati Kinesis calcola il proprio throughput in partizioni. Nel flusso di dati Amazon Kinesis è possibile scegliere tra la modalità con capacità on demand e la modalità con provisioning per i flussi di dati.

Si consiglia di utilizzare la modalità on demand per il flusso di dati Kinesis se il carico di lavoro di scrittura di DynamoDB è altamente variabile e imprevedibile. Con la modalità on demand, non è richiesta pianificazione della capacità in quanto il flusso di dati Kinesis gestisce automaticamente gli shard per fornire il throughput necessario.

Per carichi di lavoro prevedibili è possibile utilizzare la modalità con provisioning per il flusso di dati Kinesis. Con la modalità con provisioning, è necessario specificare il numero di shard per il flusso di dati per ricevere i record di Change Data Capture da DynamoDB. Per determinare il numero di shard di cui il flusso di dati Kinesis avrà bisogno per supportare la tabella DynamoDB sono necessari i seguenti valori di input:

- La dimensione media del record della tabella DynamoDB in byte (`average_record_size_in_bytes`).
- Il numero massimo di operazioni di scrittura che la tabella DynamoDB eseguirà al secondo. Questo include le operazioni di creazione, eliminazione e aggiornamento eseguite dalle applicazioni,

nonché le operazioni generate automaticamente come le eliminazioni generate dal Time to Live (TTL) (`write_throughput`).

- La percentuale di operazioni di aggiornamento e sovrascrittura eseguite sulla tabella, rispetto alle operazioni di creazione o eliminazione (`percentage_of_updates`). Tieni a mente che le operazioni di aggiornamento e sovrascrittura replicano nel flusso sia le vecchie che le nuove immagini dell'elemento modificato al flusso. Questo genera il doppio della dimensione dell'elemento DynamoDB.

È possibile calcolare il numero di shard (`number_of_shards`) di cui il flusso di dati Kinesis ha bisogno utilizzando i valori di input nella formula seguente:

```
number_of_shards = ceiling( max( ((write_throughput * (4+percentage_of_updates) *
 average_record_size_in_bytes) / 1024 / 1024), (write_throughput/1000)), 1)
```

Ad esempio, si potrebbe avere un throughput massimo di 1040 operazioni di scrittura al secondo (`write_throughput`) con una dimensione media dei record di 800 byte (`average_record_size_in_bytes`). Se il 25% di queste operazioni di scrittura sono operazioni di aggiornamento (`percentage_of_updates`) allora saranno necessari due shard (`number_of_shards`) per supportare il throughput di streaming DynamoDB:

```
ceiling( max( ((1040 * (4+25/100) * 800)/ 1024 / 1024), (1040/1000)), 1).
```

Considera quanto segue prima di utilizzare la formula per calcolare il numero di shard necessari con la modalità con provisioning per il flusso di dati Kinesis:

- Questa formula aiuta a stimare il numero di shard necessari per ospitare i record di dati di modifica di DynamoDB. Non rappresenta il numero totale di shard necessari nel flusso di dati Kinesis, ad esempio gli shard necessari per supportare consumer del flusso di dati Kinesis aggiuntivi.
- Nella modalità assegnata è possibile che si verifichino comunque eccezioni di velocità di trasmissione effettiva di lettura e scrittura se non si configura il flusso di dati per gestire i picchi di velocità di trasmissione effettiva. In questo caso, è necessario dimensionare manualmente il flusso di dati in modo da adattarlo al traffico di dati.
- Questa formula prende in considerazione il bloat aggiuntivo generato da DynamoDB prima di trasmettere i record di dati dei log delle modifiche ai flussi di dati Kinesis.

Per ulteriori informazioni sulle modalità di capacità sul flusso di dati Kinesis, consulta Scelta della modalità di capacità del flusso di dati. Per ulteriori informazioni sulla differenza di prezzo tra le diverse modalità di capacità, consulta Prezzi del flusso di dati Amazon Kinesis.

Monitoraggio di Change Data Capture con Kinesis Data Streams

DynamoDB fornisce diversi parametri CloudWatch Amazon per aiutarti a monitorare la replica dell'acquisizione dei dati di modifica su Kinesis. Per un elenco completo delle metriche, consulta. CloudWatch Parametri e dimensioni di DynamoDB

Per determinare se il flusso dispone di capacità sufficiente, si consiglia di monitorare i seguenti elementi sia durante l'abilitazione del flusso che in produzione:

- `ThrottledPutRecordCount`: il numero di record che sono stati limitati dal flusso dei dati Kinesis a causa della capacità insufficiente del flusso di dati Kinesis. Potrebbe verificarsi una limitazione della larghezza di banda della rete durante i picchi di utilizzo eccezionali, ma il `ThrottledPutRecordCount` dovrebbe rimanere il più basso possibile. DynamoDB prova a inviare i record con limitazione sul flusso di dati Kinesis, ma ciò potrebbe comportare una maggiore latenza di replica.

  Se si verifica una limitazione eccessiva e regolare, potrebbe essere necessario aumentare il numero di partizioni del flusso Kinesis proporzionalmente alla velocità di scrittura osservata della tabella. Per ulteriori informazioni su come determinare le dimensioni di un flusso di dati Kinesis, consulta Determinazione delle dimensioni iniziali di un flusso di dati Kinesis.
- `AgeOfOldestUnreplicatedRecord`: il tempo trascorso dalla modifica a livello di elemento meno recente da replicare nel flusso di dati Kinesis è apparso nella tabella DynamoDB. In condizioni di funzionamento normale, `AgeOfOldestUnreplicatedRecord` dovrebbe essere nell'ordine dei millisecondi. Questo numero aumenta in base ai tentativi di replica non riusciti quando questi sono causati da scelte di configurazione controllate dal cliente.

  Se la metrica `AgeOfOldestUnreplicatedRecord` supera le 168 ore, la replica delle modifiche a livello di elemento dalla tabella DynamoDB al flusso di dati Kinesis verrà automaticamente disabilitata.

  Gli esempi di configurazioni controllate dal cliente che portano a tentativi di replica non riusciti sono una capacità del flusso dei dati Kinesis con provisioning insufficiente, che comporta una limitazione eccessiva o un aggiornamento manuale delle policy di accesso del flusso dei dati Kinesis, che impedisce a DynamoDB di aggiungere dati al flusso dei dati. Per mantenere questo parametro il

più basso possibile, potrebbe essere necessario garantire il corretto provisioning della capacità del flusso di dati Kinesis e assicurarsi che le autorizzazioni di DynamoDB siano invariate.

- `FailedToReplicateRecordCount`: il numero di record che DynamoDB non è riuscito a replicare nel flusso dei dati Kinesis. Alcuni elementi di dimensioni superiori a 34 KB potrebbero espandersi per modificare i record di dati superiori al limite di dimensioni di 1 MB degli elementi di Kinesis Data Streams. Questo aumento delle dimensioni si verifica quando gli elementi più grandi di 34 KB includono un numero elevato di valori booleani o vuoti degli attributi. I valori booleani e vuoti degli attributi vengono archiviati come 1 byte in DynamoDB, ma si espandono fino a 5 byte quando vengono serializzati utilizzando JSON standard per la replica di Kinesis Data Streams. DynamoDB non può replicare tali record di modifica nel flusso dei dati Kinesis. DynamoDB ignora questi record di dati di modifica e continua automaticamente a replicare i record successivi.

Puoi creare CloudWatch allarmi Amazon che inviano un messaggio Amazon Simple Notification Service (Amazon SNS) per la notifica quando una delle metriche precedenti supera una soglia specifica.

Utilizzo di policy IAM per il flusso di dati Amazon Kinesis e Amazon DynamoDB

La prima volta che abiliti Amazon Kinesis Data Streams per Amazon DynamoDB, DynamoDB crea automaticamente un ruolo collegato al servizio (IAM) per te. AWS Identity and Access Management Questo ruolo, `AWSServiceRoleForDynamoDBKinesisDataStreamsReplication`, consente a DynamoDB di gestire la replica delle modifiche a livello di elemento ai Kinesis Data Streams per conto dell'utente. Non eliminare questo ruolo collegato al servizio.

Per ulteriori informazioni sui ruoli collegati al servizio, consulta Utilizzo dei ruoli collegati al servizio nella Guida per l'utente IAM.

> **Note**
>
> DynamoDB non supporta le condizioni basate sui tag per le policy IAM.

Per abilitare il flusso di dati Amazon Kinesis per Amazon DynamoDB, è necessario disporre delle seguenti autorizzazioni sulla tabella:

- `dynamodb:EnableKinesisStreamingDestination`
- `kinesis:ListStreams`

- `kinesis:PutRecords`
- `kinesis:DescribeStream`

Per descrivere il flusso di dati Amazon Kinesis per Amazon DynamoDB per una determinata tabella DynamoDB, è necessario disporre delle seguenti autorizzazioni sulla tabella.

- `dynamodb:DescribeKinesisStreamingDestination`
- `kinesis:DescribeStreamSummary`
- `kinesis:DescribeStream`

Per disabilitare il flusso di dati Amazon Kinesis per Amazon DynamoDB, è necessario disporre delle seguenti autorizzazioni sulla tabella.

- `dynamodb:DisableKinesisStreamingDestination`

Per aggiornare un flusso di dati Amazon Kinesis per Amazon DynamoDB, è necessario disporre delle seguenti autorizzazioni sulla tabella.

- `dynamodb:UpdateKinesisStreamingDestination`

Gli esempi seguenti mostrano come utilizzare le policy IAM per concedere autorizzazioni per Amazon Kinesis Data Streams per Amazon DynamoDB.

Esempio: abilitazione di Amazon Kinesis Data Streams per Amazon DynamoDB

La seguente policy IAM concede le autorizzazioni per abilitare un flusso di dati Amazon Kinesis per Amazon DynamoDB per la tabella `Music`. Non concede le autorizzazioni per disabilitare, aggiornare o descrivere un flusso di dati Kinesis per DynamoDB per la tabella `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"
        }
    ]
}
```

Esempio: abilitazione di un flusso di dati Amazon Kinesis per Amazon DynamoDB

La seguente policy IAM concede le autorizzazioni per aggiornare un flusso di dati Amazon Kinesis per Amazon DynamoDB per la tabella `Music`. Non concede le autorizzazioni per abilitare, disabilitare o descrivere un flusso di dati Amazon Kinesis per Amazon DynamoDB per la tabella `Music`.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:UpdateKinesisStreamingDestination"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:111122223333:table/Music"
        }
    ]
}
```

Esempio: disabilitazione di Amazon Kinesis Data Streams per Amazon DynamoDB

La seguente policy IAM concede le autorizzazioni per aggiornare un flusso di dati Amazon Kinesis per Amazon DynamoDB per la tabella `Music`. Non concede le autorizzazioni per abilitare, aggiornare o descrivere un flusso di dati Amazon Kinesis per Amazon DynamoDB per la tabella `Music`.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:DisableKinesisStreamingDestination"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:111122223333:table/Music"
        }
    ]
}
```

Esempio: applicazione selettiva delle autorizzazioni per il flusso di dati Amazon Kinesis per Amazon DynamoDB in base alla risorsa

La seguente policy IAM concede le autorizzazioni per abilitare e descrivere il flusso di dati Amazon Kinesis per Amazon DynamoDB per la tabella `Music` e non concede le autorizzazioni per disabilitare il flusso di dati Amazon Kinesis per Amazon DynamoDB per la tabella `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"
        }
    ]
}
```

Utilizzo di ruoli collegati ai servizi per Kinesis Data Streams per DynamoDB

Amazon Kinesis Data Streams per Amazon DynamoDB utilizza ruoli collegati ai servizi (IAM). AWS Identity and Access Management Un ruolo collegato al servizio è un tipo univoco di ruolo IAM collegato direttamente a Kinesis Data Streams per DynamoDB. I ruoli collegati ai servizi sono predefiniti da Kinesis Data Streams for DynamoDB e includono tutte le autorizzazioni richieste dal servizio per chiamare altri servizi per tuo conto. AWS

Un ruolo collegato al servizio semplifica la configurazione di Kinesis Data Streams per DynamoDB perché non è necessario aggiungere manualmente le autorizzazioni necessarie. Kinesis Data Streams per DynamoDB definisce le autorizzazioni dei relativi ruoli associati ai servizi e, salvo diversamente definito, solo Kinesis Data Streams potrà assumere i propri ruoli. Le autorizzazioni definite includono la policy di attendibilità e la policy delle autorizzazioni che non può essere collegata a nessun'altra entità IAM.

Per informazioni sugli altri servizi che supportano i ruoli collegati ai servizi, consulta Servizi AWS che funzionano con IAM e cerca i servizi che riportano Sì nella colonna Ruolo associato ai servizi. Scegli Sì in corrispondenza di un link per visualizzare la documentazione relativa al ruolo collegato ai servizi per tale servizio.

Autorizzazioni dei ruoli collegati ai servizi per Kinesis Data Streams per DynamoDB

Kinesis Data Streams for DynamoDB utilizza il ruolo collegato al servizio denominato. AWSServiceRoleForDynamoDBKinesisDataStreamsReplication Lo scopo di questo ruolo collegato al servizio è di consentire ad Amazon DynamoDB di gestire la replica delle modifiche a livello di elemento al flusso di dati Kinesis per conto dell'utente.

Ai fini dell'assunzione del ruolo, il ruolo collegato ai servizi `AWSServiceRoleForDynamoDBKinesisDataStreamsReplication` considera attendibili i seguenti servizi:

- `kinesisreplication.dynamodb.amazonaws.com`

La policy delle autorizzazioni del ruolo consente a Kinesis Data Streams per DynamoDB per completare le seguenti operazioni sulle risorse specificate:

- Operazione: `Put records and describe` su `Kinesis stream`
- Azione: `Generate data keys` attiva per inserire dati `AWS KMS` negli stream Kinesis crittografati utilizzando chiavi generate dall'utente. AWS KMS

Per i contenuti esatti del documento informativo, consulta Dynamo. DBKinesis ReplicationServiceRolePolicy

Per consentire a un'entità IAM (come un utente, un gruppo o un ruolo) di creare, modificare o eliminare un ruolo collegato ai servizi devi configurare le relative autorizzazioni. Per ulteriori informazioni, consulta Autorizzazioni del ruolo collegato ai servizi nella Guida per l'utente di IAM.

Creazione di un ruolo collegato ai servizi per Kinesis Data Streams per DynamoDB

Non hai bisogno di creare manualmente un ruolo collegato ai servizi. Quando abiliti Kinesis Data Streams for DynamoDB nella, o nell' AWS CLI AWS API, Kinesis Data Streams for Console di gestione AWS DynamoDB crea automaticamente il ruolo collegato al servizio.

Se elimini questo ruolo collegato ai servizi, è possibile ricrearlo seguendo lo stesso processo utilizzato per ricreare il ruolo nell'account. Quando si abilita Kinesis Data Streams per DynamoDB, questo crea automaticamente il ruolo collegato ai servizi.

Modifica di un ruolo collegato ai servizi per Kinesis Data Streams per DynamoDB

Kinesis Data Streams per DynamoDB non consente di modificare il ruolo collegato ai servizi `AWSServiceRoleForDynamoDBKinesisDataStreamsReplication`. Dopo aver creato un ruolo collegato al servizio, non potrai modificarne il nome perché varie entità potrebbero farvi riferimento. È possibile tuttavia modificarne la descrizione utilizzando IAM. Per ulteriori informazioni, consulta la sezione Modifica di un ruolo collegato ai servizi nella Guida per l'utente di IAM.

Eliminazione di un ruolo collegato ai servizi per Kinesis Data Streams per DynamoDB

Puoi anche utilizzare la console IAM, the o l'API per eliminare manualmente il ruolo collegato al servizio AWS CLI . AWS Per farlo, sarà necessario prima eseguire manualmente la pulizia delle risorse associate al ruolo collegato ai servizi e poi eliminarlo manualmente.

> **Note**
>
> Se il servizio Kinesis Data Streams per DynamoDB utilizza tale ruolo quando si prova a eliminare le risorse, è possibile che l'eliminazione non abbia esito positivo. In questo caso, attendi alcuni minuti e quindi ripeti l'operazione.

Per eliminare manualmente il ruolo collegato ai servizi mediante IAM

Utilizza la console IAM AWS CLI, o l' AWS API per eliminare il ruolo collegato al `AWSServiceRoleForDynamoDBKinesisDataStreamsReplication` servizio. Per ulteriori informazioni, consulta Eliminazione del ruolo collegato ai servizi nella Guida per l'utente di IAM.

Acquisizione dei dati di modifica per DynamoDB Streams

DynamoDB Streams acquisisce una sequenza cronologica di modifiche a livello di elemento in una tabella DynamoDB qualsiasi e le archivia in un log per un massimo di 24 ore. Le applicazioni possono accedere a questo log e visualizzare gli elementi di dati nel rispettivo stato prima e dopo la modifica, praticamente in tempo reale.

La crittografia a riposo crittografa i dati in DynamoDB Streams. Per ulteriori informazioni, consulta Crittografia a riposo per DynamoDB.

Un flusso DynamoDB è un flusso ordinato di informazioni sulle modifiche apportate agli elementi in una tabella DynamoDB. Quando si abilita un flusso in una tabella, DynamoDB acquisisce informazioni su ogni modifica apportata agli elementi di dati nella tabella.

Ogni volta che un'applicazione crea, aggiorna o elimina elementi nella tabella, DynamoDB Streams scrive un record di flusso con gli attributi della chiave primaria degli elementi che sono stati modificati. Un record di flusso contiene informazioni su una modifica di dati apportata a un singolo elemento in una tabella DynamoDB. Puoi configurare il flusso in modo che i record di flusso acquisiscano informazioni aggiuntive, come le immagini "prima" e "dopo" degli elementi modificati.

DynamoDB Streams garantisce quanto segue:

- Ogni record di flusso viene visualizzato esattamente una volta nel flusso.
- Per ogni elemento modificato in una tabella DynamoDB, i record di flusso vengono visualizzati nella stessa sequenza delle modifiche effettive apportate all'elemento.

DynamoDB Streams scrive i record di flusso praticamente in tempo reale per permettere di creare applicazioni che utilizzano questi flussi e agiscono in base al contenuto.

Argomenti

- Endpoint per DynamoDB Streams
- Abilitazione di un flusso
- Lettura ed elaborazione di un flusso
- DynamoDB Streams e Time to Live
- Utilizzo dell'adattatore DynamoDB Streams Kinesis per elaborare i record di flusso
- API di basso livello DynamoDB Streams: esempio Java
- Streams e trigger DynamoDB AWS Lambda
- Flussi DynamoDB e Apache Flink

Endpoint per DynamoDB Streams

AWS mantiene endpoint separati per DynamoDB e DynamoDB Streams. Per usare tabelle e indici di database, l'applicazione deve accedere a un endpoint DynamoDB. Per leggere ed elaborare i record di DynamoDB Streams, l'applicazione deve accedere a un endpoint DynamoDB Streams nella stessa regione.

DynamoDB Streams offre due set di endpoint. Questi sono:

- IPv4-only endpoint: endpoint con la convenzione di denominazione. `streams.dynamodb.<region>.amazonaws.com`
- Endpoint dual-stack: nuovi endpoint compatibili con entrambi e che seguono la convenzione di denominazione. IPv4 IPv6 `streams-dynamodb.<region>.api.aws`

> **Note**
>
> Per l'elenco completo delle regioni e degli endpoint di DynamoDB e DynamoDB Streams, consulta Regioni ed endpoint in Riferimenti generali di AWS.

AWS SDKs Forniscono client separati per DynamoDB e DynamoDB Streams. A seconda dei requisiti, l'applicazione può accedere a un endpoint DynamoDB, a un endpoint DynamoDB Streams o a entrambi contemporaneamente. Per connettersi a entrambi gli endpoint, l'applicazione deve creare un'istanza di due client, uno per DynamoDB e uno per DynamoDB Streams.

Abilitazione di un flusso

È possibile abilitare uno stream su una nuova tabella quando lo si crea utilizzando o uno dei AWS CLI . AWS SDKs È inoltre possibile abilitare o disabilitare un flusso in una tabella esistente oppure modificare le impostazioni di un flusso. DynamoDB Streams opera in modo asincrono, pertanto non vi è alcun impatto sulle prestazioni di una tabella se si abilita un flusso.

Il modo più semplice per gestire DynamoDB Streams consiste nell'usare la Console di gestione AWS.

1. Accedi Console di gestione AWS e apri la console DynamoDB all'indirizzo. https://console.aws.amazon.com/dynamodb/
2. Nel pannello di controllo della console DynamoDB, scegli Tabelle e seleziona una tabella esistente.
3. Scegli la scheda Exports and streams (Esportazioni e flussi).
4. Nella sezione Dettagli del flusso DynamoDB, seleziona Attiva.
5. Nella pagina Attiva il flusso DynamoDB, seleziona le informazioni che verranno scritte nel flusso a ogni modifica dei dati nella tabella:

   - Key attributes only (Solo attributi chiave): solo gli attributi chiave dell'elemento modificato.
   - Nuova immagine: l'intero elemento, così come visualizzato dopo che è stato modificato.
   - Vecchia immagine: l'intero elemento, così come visualizzato prima della modifica.
   - Immagini nuove e vecchie: le immagini dell'elemento nuove e vecchie.

   Dopo aver configurato le impostazioni, seleziona Attiva il flusso.
6. (Facoltativo) Per disabilitare un flusso esistente, seleziona Disattivain Dettagli del flusso DynamoDB.

Per abilitare o modificare un flusso, puoi anche usare le operazioni API `CreateTable` o `UpdateTable`. Il parametro `StreamSpecification` determina la configurazione del flusso:

- `StreamEnabled`: specifica se un flusso è abilitato (`true`) o disabilitato (`false`) per la tabella.
- `StreamViewType`: specifica le informazioni che verranno scritte nel flusso a ogni modifica dei dati nella tabella:
  - `KEYS_ONLY`: solo gli attributi chiave dell'elemento modificato.
  - `NEW_IMAGE`: l'intero elemento, così come visualizzato dopo che è stato modificato.
  - `OLD_IMAGE`: l'intero elemento, così come visualizzato prima della modifica.
  - `NEW_AND_OLD_IMAGES`: le immagini dell'elemento nuove e vecchie.

Puoi abilitare o disabilitare un flusso in qualsiasi momento. Tuttavia, se provi ad abilitare un flusso in una tabella che include già un flusso riceverai un'eccezione `ValidationException`. Inoltre, se si tenta di disabilitare un flusso in una tabella che non include alcun flusso, verrà mostrata un'eccezione `ValidationException`.

Quando `StreamEnabled` viene impostato su `true`, DynamoDB crea un nuovo flusso con un descrittore di flusso univoco assegnato. Se disabiliti e quindi riabiliti un flusso nella tabella, viene creato un nuovo flusso con un descrittore di flusso diverso.

Ogni flusso è identificato in modo univoco da un Amazon Resource Name (ARN). Di seguito è riportato un ARN di esempio per un flusso in una tabella DynamoDB denominata `TestTable`.

```
arn:aws:dynamodb:us-west-2:111122223333:table/TestTable/stream/2015-05-11T21:21:33.291
```

Per determinare l'ultimo descrittore di flusso per una tabella, inviare una richiesta `DescribeTable` DynamoDB e cercare l'elemento `LatestStreamArn` nella risposta.

> **Note**
>
> Non è possibile modificare un `StreamViewType` una volta configurato un flusso. Se è necessario apportare modifiche a un flusso dopo che è stato configurato, è necessario disabilitare il flusso corrente e crearne uno nuovo.

Lettura ed elaborazione di un flusso

Per leggere ed elaborare un flusso, l'applicazione deve connettersi a un endpoint DynamoDB Streams ed emettere le richieste API.

Un flusso è costituito da record di flusso. Ogni record di flusso rappresenta una singola modifica dei dati nella tabella DynamoDB cui appartiene il flusso. A ogni record di flusso è assegnato un numero in sequenza, corrispondente all'ordine in cui il record è stato pubblicato nel flusso.

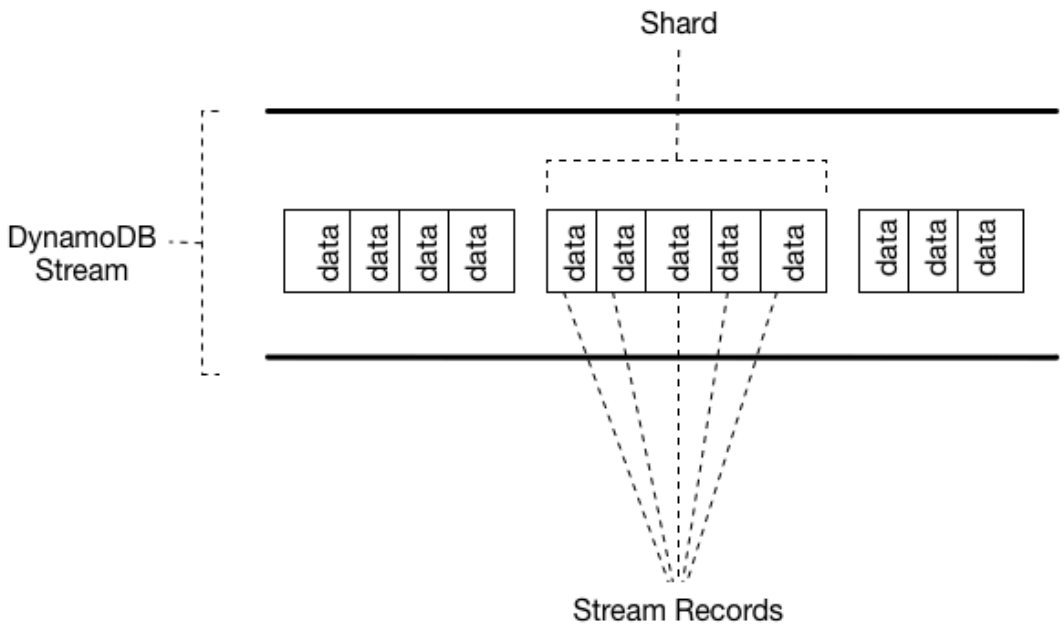
I record di flusso sono organizzati in gruppi o shard. Ogni shard funge da container per più record di flusso e contiene le informazioni necessarie per l'accesso e l'iterazione attraverso i record. I record di flusso all'interno di uno shard vengono automaticamente rimossi dopo 24 ore.

Gli shard sono effimeri, ovvero vengono creati ed eliminati automaticamente in base alle necessità. Qualsiasi shard può anche essere suddiviso in più nuovi shard e anche questa operazione viene eseguita automaticamente. È anche possibile che per uno shard padre esista un solo shard figlio. Uno shard può essere suddiviso in risposta a livelli elevati di attività di scrittura nella tabella padre corrispondente, in modo che le applicazioni possano elaborare record da più shard in parallelo.

Se disabiliti un flusso, tutti gli shard aperti verranno chiusi. I dati nel flusso continueranno a essere leggibili per 24 ore.

Poiché gli shard hanno una derivazione (padre e figlio), un'applicazione deve sempre elaborare uno shard padre prima dello shard figlio. In questo modo, anche l'elaborazione dei record di flusso avviene sicuramente in base all'ordine corretto. Se si utilizza DynamoDB Streams Kinesis Adapter, questa operazione viene gestita automaticamente. L'applicazione elabora le partizioni e i record di flusso nell'ordine corretto e gestisce automaticamente partizioni nuove o scadute, nonché le partizioni che vengono suddivise durante l'esecuzione dell'applicazione. Per ulteriori informazioni, consultareUtilizzo dell'adattatore DynamoDB Streams Kinesis per elaborare i record di flusso.

Il diagramma seguente mostra la relazione tra un flusso, gli shard nel flusso e i record di flusso negli shard.



> **Note**
>
> Se si esegue un'operazione `PutItem` o `UpdateItem` che non modifica alcun dato in un elemento, DynamoDB Streams non scrive un record di flusso per tale operazione.

Per accedere a un flusso ed elaborare i record di flusso al suo interno, devi eseguire queste operazioni:

- Determinare l'ARN univoco del flusso a cui vuoi accedere.
- Determinare quali shard nel flusso contengono i record di flusso che ti interessano.
- Accedi alle partizioni e recupera i record di flusso desiderati.

> **Note**
>
> Non più di due processi devono leggere dalla stessa partizione di flussi contemporaneamente. La presenza di più di due lettori per shard può causare il throttling.

L'API DynamoDB Streams fornisce le operazioni seguenti, che possono essere usate dai programmi applicativi:

- `ListStreams`: restituisce un elenco dei descrittori di flusso per il conto corrente e l'endpoint. Puoi facoltativamente richiedere solo i descrittori di flusso per un nome di tabella specifico.
- `DescribeStream`: restituisce informazioni su un flusso, inclusi lo stato attuale del flusso, il suo nome della risorsa Amazon (ARN), la composizione dei suoi shard e la corrispondente tabella su DynamoDB. Facoltativamente, è possibile utilizzare il campo `ShardFilter` per recuperare lo shard secondario esistente associato allo shard principale.
- `GetShardIterator`: restituisce un iteratore di partizioni, che descrive una posizione all'interno di una partizione. Puoi richiedere che l'iterazione fornisca accesso al punto meno recente, al punto più recente o a un punto particolare nel flusso.
- `GetRecords`: restituisce i record di flusso in una determinata partizione. Devi specificare l'iterazione shard restituita da una richiesta `GetShardIterator`.

Per la descrizione completa di queste operazioni API, incluse le richieste e le risposte di esempio, consulta la Documentazione di riferimento delle API di Amazon DynamoDB Streams.

Individuazione di shard

Come individuare nuovi shard in un flusso DynamoDB con due potenti metodi. Gli utenti dei flussi Amazon DynamoDB hanno a disposizione due modi efficaci per tracciare e identificare nuovi shard:

Polling della topologia dell'intero flusso

Utilizzo dell'API `DescribeStream` per eseguire regolarmente il polling del flusso. Questa operazione restituisce tutti gli shard del flusso, inclusi tutti i nuovi shard creati. Confrontando i risultati nel tempo è possibile rilevare i nuovi shard aggiunti.

Individuazione di shard secondari

Usa l'API `DescribeStream`con il parametro `ShardFilter` per trovare un sottoinsieme di shard. Specificando uno shard principale nella richiesta, i flussi DynamoDB restituiranno i relativi shard secondari immediati. Questo approccio è utile quando è necessario solo tracciare la derivazione degli shard senza scansionare l'intero flusso.

Le applicazioni che utilizzano dati dai flussi DynamoDB possono passare in modo efficiente dalla lettura di uno shard chiuso al relativo shard secondario utilizzando questo parametro `ShardFilter`, evitando chiamate ripetute all'API `DescribeStream` per recuperare e attraversare la mappa degli shard per tutti gli shard chiusi e aperti. Questo aiuta a individuare

rapidamente gli shard secondari dopo la chiusura di uno shard principale, rendendo le applicazioni di elaborazione dei flussi più reattive e convenienti.

Entrambi i metodi consentono di rimanere aggiornati sulla struttura in evoluzione dei propri flussi DynamoDB, assicurandosi di non perdere mai aggiornamenti cruciali dei dati o modifiche agli shard.

Limite di conservazione dei dati per DynamoDB Streams

Tutti i dati in DynamoDB Streams hanno una durata di 24 ore. Puoi recuperare e analizzare le ultime 24 ore di attività per qualsiasi tabella specifica. Tuttavia, i dati più vecchi di 24 ore sono soggetti a taglio (rimozione) in qualsiasi momento.

Se disabiliti un flusso in una tabella, i dati nel flusso restano leggibili per 24 ore. Dopo questo intervallo di tempo, i dati scadono e i record di flusso vengono automaticamente eliminati. Non esiste un meccanismo per eliminare manualmente un flusso esistente. Devi attendere fino allo scadere del limite di conservazione (24 ore) perché tutti i record di flusso vengano eliminati.

DynamoDB Streams e Time to Live

È possibile eseguire il backup o elaborare gli elementi che sono stati eliminati da Time to Live (TTL, Time to Live) abilitando Amazon DynamoDB Streams sulla tabella ed elaborando i record di flusso degli elementi scaduti. Per ulteriori informazioni, consulta Lettura ed elaborazione di un flusso.

Il record Streams contiene un campo di identità utente `Records[<index>].userIdentity`.

Gli elementi eliminati dal processo Time to Live (TTL, Time to Live) dopo la scadenza hanno i seguenti campi:

- `Records[<index>].userIdentity.type`

  `"Service"`
- `Records[<index>].userIdentity.principalId`

  `"dynamodb.amazonaws.com"`

> **Note**
>
> Quando si utilizza il TTL in una tabella globale, il campo `userIdentity` verrà impostato nella Regione in cui è stato eseguito il TTL. Questo campo non verrà impostato in altre Regioni quando l'eliminazione viene replicata.

Il JSON seguente mostra la porzione rilevante di un singolo record Streams.

```
"Records": [
    {
        ...

        "userIdentity": {
            "type": "Service",
            "principalId": "dynamodb.amazonaws.com"
        }

        ...

    }
]
```

Utilizzo di DynamoDB Streams e Lambda per archiviare gli elementi eliminati TTL

La combinazione di DynamoDB Time to Live (TTL), DynamoDB Streams e AWS Lambda può contribuire a semplificare l'archiviazione dei dati, ridurre i costi di storage DynamoDB e ridurre la complessità del codice. L'utilizzo di Lambda come consumatore di flussi offre molti vantaggi, in particolare la riduzione dei costi rispetto ad altri consumatori come Kinesis Client Library (KCL). Non ti viene addebitato alcun costo per le chiamate API `GetRecords` sul flusso DynamoDB quando si utilizza Lambda per consumare eventi e Lambda può fornire il filtraggio degli eventi identificando i modelli JSON in un evento stream. Attraverso il filtraggio dei contenuti del modello di eventi, è possibile definire fino a cinque filtri diversi per controllare quali eventi vengono inviati a Lambda per l'elaborazione. Ciò aiuta a ridurre le invocazioni delle funzioni Lambda, semplifica il codice e riduce i costi complessivi.

Sebbene DynamoDB Streams contenga tutte le modifiche ai dati, ad esempio `Create`, `Modify` e le azioni `Remove`, questo può causare invocazioni indesiderate della funzione Lambda di archivio. Ad esempio, supponiamo di avere una tabella con 2 milioni di modifiche ai dati all'ora che fluiscono nello

stream, ma meno del 5% di queste sono eliminazioni di elementi che scadranno durante il processo TTL e devono essere archiviate. Con Filtri di origine evento Lambda, la funzione Lambda richiamerà solo 100.000 volte all'ora. Il risultato con il filtraggio degli eventi è che ti vengono addebitati solo le invocazioni necessarie anziché i 2 milioni di invocazioni che avresti avuto senza filtraggio degli eventi.

Il filtraggio degli eventi viene applicato alla mappatura di origine evento Lambda, una risorsa che legge da un evento scelto, ovvero il flusso DynamoDB, e invoca una funzione Lambda. Nel diagramma seguente, puoi vedere come un elemento eliminato Time to Live viene consumato da una funzione Lambda utilizzando flussi e filtri eventi.

Modello di filtro evento DynamoDB Time to Live

L'aggiunta del seguente JSON ai criteri di filtro della mappatura dell'origine eventi consente l'invocazione della funzione Lambda solo per gli elementi eliminati dal TTL:

```
{
    "Filters": [
        {
            "Pattern": { "userIdentity": { "type": ["Service"], "principalId":
 ["dynamodb.amazonaws.com"] } }
        }
    ]
}
```

AWS Lambda Crea una mappatura delle sorgenti degli eventi

Utilizza i seguenti frammenti di codice per creare una mappatura dell'origine eventi filtrata che è possibile connettere al flusso DynamoDB di una tabella. Ciascun blocco di codice include il modello di filtro eventi.

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"]
   }
}
```

Utilizzo dell'adattatore DynamoDB Streams Kinesis per elaborare i record di flusso

L'utilizzo di Amazon Kinesis Adapter è il modo consigliato per utilizzare flussi da Amazon DynamoDB. L'API dei flussi DynamoDB è progettata in maniera simile a quella del flusso di dati Kinesis. In entrambi i servizi, i flussi di dati sono composti da partizioni, che sono container per i record di flusso. Entrambi i servizi APIs contengono`ListStreams`, `DescribeStream``GetShards`, e `GetShardIterator` operazioni. Sebbene queste operazioni DynamoDB Streams siano simili alle loro controparti in Kinesis Data Streams, non sono identiche al 100%.

Come utente di DynamoDB Streams, è possibile utilizzare i modelli di progettazione presenti all'interno di KCL per elaborare partizioni e record di flusso DynamoDB Streams. Per fare ciò, si utilizza l'adattatore Kinesis DynamoDB Streams. L'adattatore Kinesis implementa l'interfaccia Kinesis Data Streams in modo che si possa utilizzare KCL per l'uso e l'elaborazione di record da DynamoDB Streams. Per istruzioni su come configurare e installare il DynamoDB Streams Kinesis Adapter, consulta il repository. GitHub

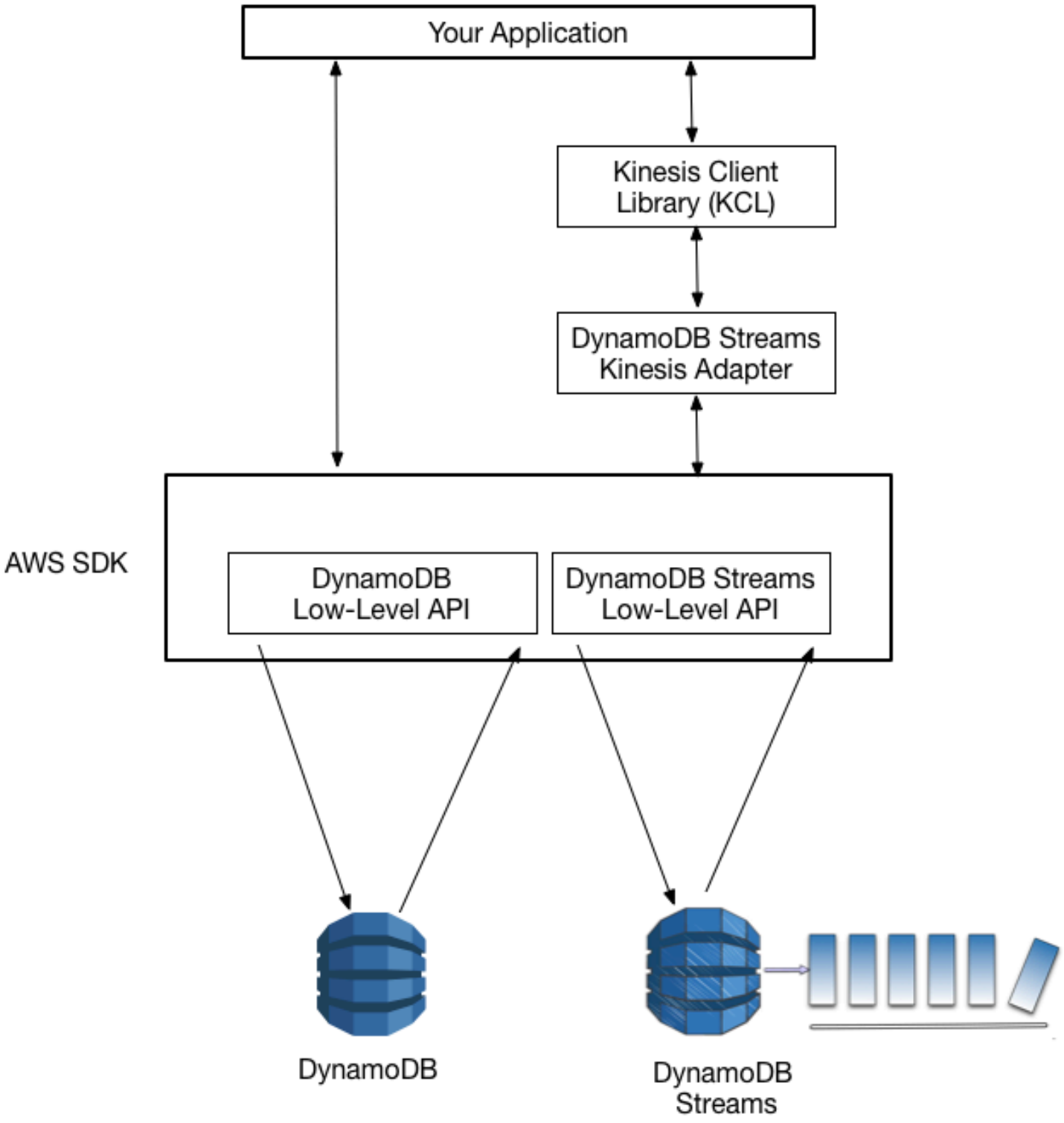
È possibile scrivere applicazioni per Kinesis Data Streams utilizzando la libreria client Kinesis (Kinesis Client Library, KCL). KCL semplifica la codifica fornendo astrazioni utili sull'API Kinesis Data Streams di basso livello. Per ulteriori informazioni su KCL, consulta Sviluppo di consumatori utilizzando la libreria client Kinesis nella Guida per gli sviluppatori di Amazon Kinesis Data Streams.

DynamoDB consiglia di utilizzare la versione 3.x di KCL con SDK AWS for Java v2.x. L'attuale versione 1.x di DynamoDB Streams Kinesis Adapter con AWS SDK per Java SDK per la versione

AWS 1.x continuerà a essere pienamente supportata per tutto il suo ciclo di vita, come previsto durante il periodo di transizione in linea con la politica di manutenzione degli strumenti.AWS SDKs

> **Note**
>
> Le versioni 1.x e 2.x di Amazon KCL sono obsolete. end-of-supportKCL 1.x sarà disponibile il 30 gennaio 2026. Si consiglia vivamente di migrare le applicazioni KCL che utilizzano la versione 1.x all'ultima versione KCL prima del 30 gennaio 2026. Per trovare la versione più recente di KCL, consulta la pagina Amazon Kinesis Client Library su. GitHub Per informazioni sulle versioni più recenti di Kinesis Client Library, consulta Use Kinesis Client Library. Per ulteriori informazioni sulla migrazione da KCL 1.x a 3.x, consulta Migrating from KCL 1.x to KCL 3.x.

Nel seguente diagramma viene illustrato come queste librerie interagiscono tra loro.



Con l'adattatore Kinesis DynamoDB Streams abilitato, è possibile iniziare a sviluppare rispetto a KCL, con le chiamate API dirette senza soluzione di continuità all'endpoint DynamoDB Streams.

Quando si avvia l'applicazione , quest'ultima richiama la libreria KCL per creare un'istanza di un worker. È necessario fornire al lavoratore le informazioni di configurazione per l'applicazione, come il descrittore di flusso e AWS le credenziali, e il nome di una classe di processore di record fornita. Durante l'esecuzione del codice nel processore di record, il worker completa le seguenti attività:

- Si collega al flusso
- Enumera le partizioni all'interno del flusso
- Controlla ed enumera gli shard secondari di uno shard principale chiuso all'interno del flusso
- Coordina le associazioni di shard con altri processi di lavoro (se presenti)
- Crea istanze di un elaboratore di record per ogni shard che gestisce
- Estrae i record di dati dal flusso
- Inserisce i record nell'elaboratore di record corrispondente
- Controlla i record elaborati
- Bilancia le associazioni tra shard e processi di lavoro quando il conteggio delle istanze del lavoro cambia
- Bilancia le associazioni tra partizioni e worker quando le partizioni sono suddivise

> **Note**
>
> Per una descrizione dei concetti su KCL elencati qui, consulta Sviluppo di consumatori utilizzando la libreria client Kinesis nella Guida per gli sviluppatori di Amazon Kinesis Data Streams.
> Per ulteriori informazioni sull'utilizzo degli stream, vedere AWS Lambda Streams e trigger DynamoDB AWS Lambda

Migrazione di KLC da 1.x a 3.x

Panoramica

Questa guida fornisce istruzioni per la migrazione dell'applicazione consumer da KCL 1.x a KCL 3.x. A causa delle differenze di architettura tra KCL 1.x e KCL 3.x, la migrazione richiede l'aggiornamento di diversi componenti per garantire la compatibilità.

KCL 1.x utilizza classi e interfacce diverse rispetto a KCL 3.x. È necessario prima migrare l'elaboratore di record, il generatore dell'elaboratore di record e le classi di lavoratori al formato

compatibile con KCL 3.x e seguire la procedura di migrazione per la migrazione da KCL 1.x a KCL 3.x.

Fasi della migrazione

Argomenti

- Fase 1: migrare l'elaboratore di record
- Fase 2: migrare il generatore dell'elaboratore di record
- Fase 3: eseguire la migrazione del lavoratore
- Fase 4: panoramica e consigli sulla configurazione di KCL 3.x
- Fase 5: migrare da KCL 2.x a KCL 3.x

Fase 1: migrare l'elaboratore di record

L'esempio seguente mostra un elaboratore di record implementato per l'Adattatore Kinesis per i flussi DynamoDB con 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();
        }
    }
}
```

Per migrare la classe RecordProcessor

1. Modifica le interfacce da `com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessor` e `com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IShutdownNotificat`

verso `software.amazon.dynamodb.streamsadapter.processor.DynamoDBStreamsShardRecordP` come segue:

```
// 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. Aggiorna le istruzioni di importazione per i metodi `initialize` e `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. Sostituisci il metodo `shutdownRequested` con i seguenti nuovi metodi: `leaseLost`, `shardEnded` e `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();
        }
    }
```

Segue la versione aggiornata della classe dell'elaboratore di record:

```
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**
>
> L'adattatore Kinesis di DynamoDB Streams ora utilizza il modello Record. SDKv2 In SDKv2, `AttributeValue` gli oggetti complessi (,BS,, NS ML,SS) non restituiscono mai null. Utilizza i metodi `hasBs()`, `hasNs()`, `hasM()`, `hasL()`, `hasSs()` per verificare se questi valori esistono.

Fase 2: migrare il generatore dell'elaboratore di record

La fabbrica dell'elaboratore di record è responsabile per la creazione di elaboratori di record quando un lease è acquisito. Di seguito è illustrato un esempio di un generatore di KCL 1.x:

```
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);
    }
}
```

Per migrare **`RecordProcessorFactory`**

- Modifica l'interfaccia implementata da `com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessorFa` a `software.amazon.kinesis.processor.ShardRecordProcessorFactory`, come segue:

```
// 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() {
```

Di seguito è riportato un esempio di generatore di elaboratore di record in 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();
    }
}
```

Fase 3: eseguire la migrazione del lavoratore

Nella versione 3.0 della KCL, una nuova classe, denominata Pianificatore, sostituisce la classe Lavoratore. Di seguito è illustrato un esempio di un lavoratore di KCL 1.x:

```
final KinesisClientLibConfiguration config = new KinesisClientLibConfiguration(...)
final IRecordProcessorFactory recordProcessorFactory = new RecordProcessorFactory();
final Worker worker = StreamsWorkerFactory.createDynamoDbStreamsWorker(
        recordProcessorFactory,
        workerConfig,
        adapterClient,
        amazonDynamoDB,
        amazonCloudWatchClient);
```

Per migrare il lavoratore

1. Modifica la dichiarazione `import` per la classe `Worker` nelle dichiarazioni di importazione delle classi `Scheduler` e `ConfigsBuilder`.

   ```
   // import com.amazonaws.services.kinesis.clientlibrary.lib.worker.Worker;
   import software.amazon.kinesis.coordinator.Scheduler;
   import software.amazon.kinesis.common.ConfigsBuilder;
   ```

2. Importa `StreamTracker` e modifica l'importazione di `StreamsWorkerFactory` in `StreamsSchedulerFactory`.

   ```
   import software.amazon.kinesis.processor.StreamTracker;
   // import software.amazon.dynamodb.streamsadapter.StreamsWorkerFactory;
   import software.amazon.dynamodb.streamsadapter.StreamsSchedulerFactory;
   ```

3. Seleziona la posizione da cui avviare l'applicazione. Può essere `TRIM_HORIZON` o `LATEST`.

   ```
   import software.amazon.kinesis.common.InitialPositionInStream;
   import software.amazon.kinesis.common.InitialPositionInStreamExtended;
   ```

4. Creazione di un'istanza `StreamTracker`.

   ```
   StreamTracker streamTracker = StreamsSchedulerFactory.createSingleStreamTracker(
           streamArn,

    InitialPositionInStreamExtended.newInitialPosition(InitialPositionInStream.TRIM_HORIZON)
   ```

```
);
```

5. Crea l'oggetto `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. Crea l'oggetto `ConfigsBuilder`.

```
import software.amazon.kinesis.common.ConfigsBuilder;

...
ConfigsBuilder configsBuilder = new ConfigsBuilder(
                streamTracker,
                applicationName,
                adapterClient,
                dynamoDbAsyncClient,
                cloudWatchAsyncClient,
                UUID.randomUUID().toString(),
                new StreamsRecordProcessorFactory());
```

7. Crea `Scheduler` con `ConfigsBuilder` come mostrato nell'esempio seguente:

```
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

L'impostazione `CLIENT_VERSION_CONFIG_COMPATIBLE_WITH_2X` mantiene la compatibilità tra l'Adattatore Kinesis per i flussi DynamoDB per KCL v3 e KCL v1, non tra KCL v2 e v3.

Fase 4: panoramica e consigli sulla configurazione di KCL 3.x

Per una descrizione dettagliata delle configurazioni introdotte dopo KCL 1.x e rilevanti in KCL 3.x, consulta KCL configurations and KCL migration client configuration.

> **Important**
>
> Invece di creare direttamente oggetti di `checkpointConfig`, `coordinatorConfig`, `leaseManagementConfig`, `metricsConfig`, `processorConfig` e `retrievalConfig`, si consiglia di utilizzare `ConfigsBuilder` per impostare le configurazioni in KCL 3.x e versioni successive per evitare problemi di inizializzazione del Pianificatore. `ConfigsBuilder` offre un modo più flessibile e gestibile per configurare l'applicazione KCL.

Configurazioni con aggiornamento del valore predefinito in KCL 3.x

`billingMode`

Nella versione 1.x di KCL, il valore predefinito per `billingMode` è impostato su `PROVISIONED`. Tuttavia, con la versione 3.x di KCL, l'impostazione predefinita `billingMode` è `PAY_PER_REQUEST` (modalità on demand). Si consiglia di utilizzare la modalità con capacità on demand per la tabella di lease per regolare automaticamente la capacità in base all'utilizzo. Per indicazioni sull'utilizzo della capacità allocata per le tabelle di lease, consulta Best practices for the lease table with provisioned capacity mode.

`idleTimeBetweenReadsInMillis`

Nella versione 1.x di KCL, il valore predefinito per `idleTimeBetweenReadsInMillis` è impostato su 1.000 (o 1 secondo). La versione 3.x di KCL imposta il valore predefinito per `idleTimeBetweenReadsInMillis` su 1.500 (o 1,5 secondi), ma l'Adattatore Kinesis per i flussi Amazon DynamoDB sostituisce il valore predefinito su 1.000 (o 1 secondo).

Nuove configurazioni in KCL 3.x

`leaseAssignmentIntervalMillis`

Questa configurazione definisce l'intervallo di tempo prima che gli shard di nuovo riscontro inizino l'elaborazione e viene calcolata come 1,5 × `leaseAssignmentIntervalMillis`. Se questa impostazione non è configurata in modo esplicito, l'intervallo di tempo predefinito è 1,5 × `failoverTimeMillis`. L'elaborazione di nuovi shard prevede la scansione della tabella di

lease e l'interrogazione di un indice secondario globale (GSI) sulla tabella di lease. La riduzione di `leaseAssignmentIntervalMillis` aumenta la frequenza delle operazioni Scan e Query, con conseguente aumento dei costi di DynamoDB. Si consiglia di impostare questo valore su 2000 (o 2 secondi) per ridurre al minimo il ritardo nell'elaborazione di nuovi shard.

`shardConsumerDispatchPollIntervalMillis`

Questa configurazione definisce l'intervallo tra polling successivi da parte del consumer dello shard per attivare le transizioni di stato. Nella versione 1.x di KCL, questo comportamento era controllato dal parametro `idleTimeInMillis`, che non era esposto come impostazione configurabile. Con la versione 3.x di KCL, si consiglia di impostare questa configurazione in modo che corrisponda al valore utilizzato per `idleTimeInMillis` nella configurazione della versione 1.x di KCL.

Fase 5: migrare da KCL 2.x a KCL 3.x

Per garantire una transizione e una compatibilità fluide con l'ultima versione di Kinesis Client Library (KCL), segui le fasi 5 – 8 nelle istruzioni della guida alla migrazione per l'aggiornamento da KCL 2.x a KCL 3.x.

Per la risoluzione dei problemi più comuni di KCL 3.x, consulta Troubleshooting KCL consumer applications.

Rollback a una versione di KCL precedente

Questo argomento illustra come eseguire il rollback di un'applicazione consumer alla versione di KCL precedente. Il processo di rollback consiste in due fasi:

1. Esegui lo Strumento di migrazione di KCL.
2. Reimplementa il codice della versione precedente di KCL.

Fase 1: eseguire lo Strumento di migrazione di KCL

Quando è necessario tornare alla versione precedente di KCL, occorre eseguire lo Strumento di migrazione KCL. Lo strumento svolge due attività importanti:

- Rimuove una tabella di metadati denominata tabella delle metriche dei lavoratori e l'indice secondario globale nella tabella di lease in DynamoDB. Questi artefatti sono creati da KCL 3.x ma non sono necessari al ritorno alla versione precedente.

- Consente a tutti i lavoratori di funzionare in una modalità compatibile con KCL 1.x e di iniziare a utilizzare l'algoritmo di bilanciamento del carico utilizzato nelle versioni precedenti di KCL. In caso di problemi con il nuovo algoritmo di bilanciamento del carico in KCL 3.x, questo ridurrà immediatamente il problema.

> **Important**
>
> La tabella dello stato del coordinatore in DynamoDB deve esistere e non deve essere eliminata durante il processo di migrazione, rollback e rollforward.

> **Note**
>
> È importante che tutti i lavoratori dell'applicazione consumer utilizzino lo stesso algoritmo di bilanciamento del carico in un determinato momento. Lo Strumento di migrazione di KCL assicura che tutti i lavoratori dell'applicazione consumer KCL 3.x passino alla modalità compatibile con KCL 1.x in modo che tutti i lavoratori eseguano lo stesso algoritmo di bilanciamento del carico durante il rollback dell'applicazione alla versione precedente di KCL.

Puoi scaricare lo strumento di migrazione KCL nella directory degli script del repository KCL. GitHub
Esegui lo script da un lavoratore o da un host con le autorizzazioni appropriate per scrivere nella tabella dello stato del coordinatore, nella tabella delle metriche dei lavoratori e nella tabella di lease. Assicurati che per le applicazioni consumer KCL siano configurate le autorizzazioni IAM appropriate. Esegui lo script solo una volta per applicazione KCL utilizzando il comando specificato:

```
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`

Sostituisci con il tuo. *region* Regione AWS

`--application_name`

Questo parametro è obbligatorio in caso di utilizzo di nomi predefiniti per le tabelle dei metadati di DynamoDB (tabella di lease, tabella dello stato del coordinatore e tabella delle metriche dei lavoratori). Se sono stati specificati nomi personalizzati per queste tabelle, è possibile omettere questo parametro. Sostituisci *`applicationName`* con il nome effettivo dell'applicazione KCL. Lo strumento utilizza questo nome per ricavare i nomi delle tabelle predefiniti se non vengono forniti nomi personalizzati.

`--lease_table_name`

Questo parametro è necessario se si imposta un nome personalizzato per la tabella di lease nella configurazione KCL. Se si utilizza il nome della tabella predefinito, è possibile omettere questo parametro. Sostituisci *`leaseTableName`* con il nome della tabella personalizzata che hai specificato per la tabella di leasing.

`--coordinator_state_table_name`

Questo parametro è necessario se si imposta un nome personalizzato per la tabella dello stato del coordinatore nella configurazione KCL. Se si utilizza il nome della tabella predefinito, è possibile omettere questo parametro. *`coordinatorStateTableName`*Sostituiscilo con il nome della tabella personalizzata che hai specificato per la tabella degli stati del coordinatore.

`--worker_metrics_table_name`

Questo parametro è necessario se si imposta un nome personalizzato per la tabella delle metriche dei lavoratori nella configurazione KCL. Se si utilizza il nome della tabella predefinito, è possibile omettere questo parametro. *`workerMetricsTableName`*Sostituiscilo con il nome della tabella personalizzata che hai specificato per la tabella delle metriche dei lavoratori.

Fase 2: reimplementare il codice con la versione precedente di KCL

> **Important**
>
> Qualsiasi menzione della versione 2.x nell'output generato dallo Strumento di migrazione di KCL deve essere interpretata come riferita alla versione 1.x di KCL. L'esecuzione dello script non esegue un rollback completo, ma passa solo l'algoritmo di bilanciamento del carico a quello utilizzato nella versione 1.x di KCL.

Dopo aver eseguito lo strumento di migrazione di KCL per un rollback, verrà mostrato uno di questi messaggi:

Messaggio 1

> “Rollback completato. L’applicazione eseguiva funzionalità della versione 2x compatibili. Torna ai file binari dell’applicazione precedente implementando il codice con la versione precedente di KCL.”
>
> Azione richiesta: significa che i lavoratori erano in esecuzione nella modalità compatibile con KCL 1.x. Reimplementa il codice con la versione precedente di KCL sui lavoratori.

Messaggio 2

> “Rollback completato. L’applicazione KCL eseguiva funzionalità della versione 3 e verrà ripristinata a funzionalità compatibili con la versione 2x. Se non è possibile visualizzare alcuna mitigazione dopo un breve periodo di tempo, esegui il rollback ai file binari dell’applicazione precedente implementando il codice con la versione KCL precedente.”
>
> Azione richiesta: significa che i lavoratori erano in esecuzione in modalità KCL 3.x e lo Strumento di migrazione di KCL ha portato tutti i lavoratori alla modalità compatibile con KCL 1.x. Reimplementa il codice con la versione precedente di KCL sui lavoratori.

Messaggio 3

> “Il rollback dell’applicazione è già stato effettuato. Tutte KCLv3 le risorse che potevano essere eliminate sono state ripulite per evitare addebiti fino a quando non è stato possibile avviare l'applicazione con la migrazione».
>
> Azione richiesta: significa che i lavoratori erano già stati sottoposti a rollback per essere eseguiti nella modalità compatibile con KCL 1.x. Reimplementa il codice con la versione precedente di KCL sui lavoratori.

Rollforward a KCL 3.x dopo un rollback

Questo argomento illustra come eseguire il rollforward di un’applicazione consumer a KCL 3.x dopo un rollback. Quando è necessario eseguire il rollforward, occorre completare una procedura in due fasi:

1. Esegui lo Strumento di migrazione di KCL.
2. Implementa il codice con KCL 3.x.

Fase 1: eseguire lo Strumento di migrazione di KCL

Esegui lo Strumento di migrazione di KCL con il seguente comando per eseguire il rollforward a KCL 3.x:

```
python3 ./KclMigrationTool.py --region region --mode rollforward [--
application_name applicationName] [--
coordinator_state_table_name coordinatorStateTableName]
```

Parameters

`--region`

*region*Sostituiscilo con il tuo Regione AWS.

`--application_name`

Questo parametro è obbligatorio se utilizzi i nomi predefiniti per la tabella dello stato del coordinatore. Se sono stati specificati nomi personalizzati per la tabella dello stato del coordinatore, è possibile omettere questo parametro. Sostituisci *applicationName* con il nome effettivo dell'applicazione KCL. Lo strumento utilizza questo nome per ricavare i nomi delle tabelle predefiniti se non vengono forniti nomi personalizzati.

`--coordinator_state_table_name`

Questo parametro è necessario se si imposta un nome personalizzato per la tabella dello stato del coordinatore nella configurazione KCL. Se si utilizza il nome della tabella predefinito, è possibile omettere questo parametro. Sostituisci *coordinatorStateTableName* con il nome della tabella personalizzata che hai specificato per la tabella degli stati del coordinatore.

Dopo aver eseguito lo strumento di migrazione in modalità rollforward, KCL crea le seguenti risorse DynamoDB necessarie per KCL 3.x:

- Un indice secondario globale nella tabella di lease
- Una tabella delle metriche dei lavoratori

Fase 2: implementare il codice con KCL 3.x

Dopo aver eseguito lo Strumento di migrazione di KCL per un rollforward, implementa il codice con KCL 3.x sui lavoratori. Per completare la migrazione, consulta Fase 8: completare la migrazione.

Spiegazione passo per passo: Adattatore Kinesis DynamoDB Streams

In questa sezione viene riportata una spiegazione passo per passo di un'applicazione Java che utilizza Amazon Kinesis Client Library e l'adattatore Amazon DynamoDB Streams Kinesis. L'applicazione mostra un esempio di replica dei dati, in cui l'attività di scrittura da una tabella viene applicata a una seconda tabella e i contenuti di entrambe le tabelle rimangono sincronizzati. Per il codice sorgente, consulta Programma completo: Adattatore Kinesis di DynamoDB Streams.

Il programma effettua le seguenti operazioni:

1. Crea due tabelle DynamoDB denominate `KCL-Demo-src` e `KCL-Demo-dst`. Su ognuna di queste tabelle è abilitato un flusso.
2. Genera l'attività di aggiornamento nella tabella di origine aggiungendo, aggiornando ed eliminando gli elementi. Questo fa sì che i dati vengano scritti nel flusso della tabella.
3. Legge i record dal flusso, li ricostruisce come richieste DynamoDB e applica le richieste alla tabella di destinazione.
4. Esegue la scansione delle tabelle di origine e di destinazione per garantire che i contenuti siano identici.
5. Esegue la pulizia eliminando le tabelle.

Queste fasi sono descritte nelle sezioni seguenti e l'applicazione completa viene mostrata alla fine della procedura guidata.

Argomenti

- Fase 1: creazione di tabelle DynamoDB
- Fase 2: generazione dell'attività di aggiornamento nella tabella di origine
- Fase 3: elaborazione del flusso
- Fase 4: verifica che entrambe le tabelle abbiano contenuti identici
- Fase 5: rimozione
- Programma completo: Adattatore Kinesis di DynamoDB Streams

Fase 1: creazione di tabelle DynamoDB

Il primo passo consiste nel creare due tabelle DynamoDB, una di origine e una di destinazione. `StreamViewType` sul flusso della tabella di origine è `NEW_IMAGE`. Questo significa che ogni volta

che un item viene modificato in questa tabella, l'immagine "successiva" dell'item viene scritta nel flusso. In questo modo, il flusso tiene traccia di tutte le attività di scrittura della tabella.

Il seguente esempio mostra il codice utilizzato per creare entrambe le tabelle.

```
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)
```

Fase 2: generazione dell'attività di aggiornamento nella tabella di origine

La fase successiva consiste nel generare le attività di scrittura sulla tabella di origine. Mentre questa attività è in corso, il flusso della tabella di origine viene aggiornato pressoché in tempo reale.

L'applicazione definisce una classe helper con metodi che chiamano le operazioni API `PutItem`, `UpdateItem` e `DeleteItem` per scrivere i dati. Il seguente esempio di codice mostra come vengono utilizzati questi metodi.

```
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");
```

Fase 3: elaborazione del flusso

Ora il programma inizia l'elaborazione del flusso. L'adattatore Kinesis di DynamoDB Streams agisce come un livello trasparente tra KCL e l'endpoint DynamoDB Streams in modo che il codice possa utilizzare appieno KCL piuttosto che effettuare chiamate a DynamoDB Streams di basso livello. Il programma esegue le attività di seguito elencate:

- Definisce una classe di elaboratore di record, `StreamsRecordProcessor`, con metodi conformi alla definizione dell'interfaccia KCL: `initialize`, `processRecords` e `shutdown`. Il metodo `processRecords` contiene la logica necessaria per la lettura dal flusso della tabella di origine e la scrittura nella tabella di destinazione.
- Definisce una factory di classe per la classe di elaboratore di record (`StreamsRecordProcessorFactory`). Ciò è richiesto per i programmi Java che utilizzano KCL.
- Crea un'istanza di un nuovo `Worker` KCL che è associato alla factory di classe.
- Arresta il `Worker` quando l'elaborazione del record è completata.

Per ulteriori informazioni sulla definizione dell'interfaccia KCL, consulta Sviluppo di consumatori utilizzando la Kinesis Client Library nella Guida per gli sviluppatori di Amazon Kinesis Data Streams.

Il seguente esempio di codice mostra il loop principale in `StreamsRecordProcessor`. L'istruzione `case` determina quale operazione eseguire, sulla base dell'item `OperationType` presente nel record del flusso.

```
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();
        }
    }
}
```

Fase 4: verifica che entrambe le tabelle abbiano contenuti identici

A questo punto, i contenuti delle tabelle di origine e destinazione sono sincronizzati. L'applicazione emette le richieste `Scan` su entrambe le tabelle per verificare che i loro contenuti siano effettivamente identici.

La classe `DemoHelper` contiene un metodo `ScanTable` che chiama l'API `Scan` di basso livello. L'esempio seguente mostra come viene utilizzato.

```
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!");
}
```

Fase 5: rimozione

La demo è completata, quindi l'applicazione elimina le tabelle di origine e di destinazione. Vedere l'esempio di codice seguente. Anche dopo l'eliminazione delle tabelle, i flussi rimangono disponibili per altre 24 ore, dopo di che vengono automaticamente eliminati.

```
dynamoDBClient.deleteTable(new DeleteTableRequest().withTableName(srcTable));
dynamoDBClient.deleteTable(new DeleteTableRequest().withTableName(destTable));
```

Programma completo: Adattatore Kinesis di DynamoDB Streams

Di seguito è riportato il programma Java completo che esegue le attività descritte in Spiegazione passo per passo: Adattatore Kinesis DynamoDB Streams. Quando lo esegui, dovresti vedere un output simile al seguente.

```
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**
>
> Per eseguire questo programma, assicurati che l'applicazione client abbia accesso a DynamoDB e CloudWatch Amazon utilizzando le policy. Per ulteriori informazioni, consulta Policy basate su identità per DynamoDB.

I codice sorgente è composto da quattro file `.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 di basso livello DynamoDB Streams: esempio Java

> **Note**
>
> Il codice in questa pagina non è completo e non prevede la gestione di tutti gli scenari di utilizzo di Amazon DynamoDB Streams. Il modo consigliato di utilizzare record di flusso da DynamoDB è tramite l'adattatore Amazon Kinesis utilizzando la Kinesis Client Library (KCL), come descritto in Utilizzo dell'adattatore DynamoDB Streams Kinesis per elaborare i record di flusso.

Questa sezione contiene un programma Java che mostra il funzionamento di DynamoDB Streams. Il programma effettua le seguenti operazioni:

1. Crea una tabella DynamoDB con un flusso abilitato.
2. Descrive le impostazioni del flusso per questa tabella.
3. Modifica i dati nella tabella.
4. Descrive gli shard nel flusso.
5. Legge i record del flusso dagli shard.
6. Recupera gli shard secondari e continua a leggere i record.
7. Elimina.

Quando esegui il programma, vedrai un output simile al seguente:

```
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 Esempio

```
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;
    }



}
```

Streams e trigger DynamoDB AWS Lambda

Amazon DynamoDB è integrato AWS Lambda in modo da poter creare trigger, parti di codice che rispondono automaticamente agli eventi in DynamoDB Streams. Con i trigger è possibile creare applicazioni che rispondono alle modifiche di dati nelle tabelle DynamoDB.

Argomenti

- Tutorial #1: Utilizzo dei filtri per elaborare tutti gli eventi con Amazon DynamoDB e utilizzo di AWS LambdaAWS CLI
- Tutorial n. 2: utilizzo di filtri per elaborare alcuni eventi con DynamoDB e Lambda
- Best practice per l'utilizzo dei flussi DynamoDB con Lambda

Se abiliti DynamoDB Streams su una tabella, puoi associare lo stream Amazon Resource Name (ARN) a una funzione che scrivi. AWS Lambda Tutte le operazioni di mutazione su quella tabella DynamoDB possono quindi essere acquisite come elemento nel flusso. Ad esempio, è possibile impostare un trigger in modo che quando un elemento in una tabella viene modificato, nel flusso della tabella venga immediatamente visualizzato un nuovo record.

> **Note**
>
> In caso di sottoscrizione di più di due funzioni Lambda in un flusso DynamoDB, potrebbe verificarsi una limitazione (della larghezza di banda della rete) della lettura.

Il servizio AWS Lambda esegue il polling del flusso alla ricerca di nuovi record quattro volte al secondo. Quando sono disponibili nuovi record di flusso, la funzione Lambda viene richiamata in modo sincrono. Puoi sottoscrivere fino a due funzioni Lambda allo stesso flusso DynamoDB. In caso di sottoscrizione di più di due funzioni Lambda nello stesso flusso DynamoDB, potrebbe verificarsi una limitazione (della larghezza di banda della rete) della lettura.

La funzione Lambda può inviare una notifica, avviare un flusso di lavoro o eseguire numerose altre operazioni specificate. Ad esempio, è possibile scrivere una funzione Lambda semplicemente per copiare ogni record di flusso in un'archiviazione persistente, come il Gateway di file di Amazon S3 (Amazon S3), per creare un percorso di verifica permanente dell'attività di scrittura della tabella. Oppure, supponi di avere un'applicazione di gioco per dispositivi mobili che scrive in una tabella `GameScores`. Quando l'attributo `TopScore` della tabella `GameScores` viene aggiornato, viene scritto un record di flusso corrispondente nel flusso della tabella. Questo evento potrebbe quindi attivare una funzione Lambda che pubblica un messaggio di congratulazioni su un social network. È anche possibile scrivere questa funzione per ignorare tutti i record di flusso che non sono aggiornamenti di `GameScores` o che non modificano l'attributo `TopScore`.

Se la funzione restituisce un errore, Lambda ritenta il batch fino a quando l'elaborazione non riesce o i dati scadono. È inoltre possibile configurare Lambda in modo da riprovare con un batch di dimensioni inferiori, limitare il numero di tentativi, eliminare i record una volta che diventano troppo vecchi e altre opzioni.

Come best practice in materia di prestazioni, la funzione Lambda deve essere di breve durata. Per evitare di introdurre ritardi di elaborazione non necessari, inoltre, non dovrebbe eseguire una logica complessa. In particolare, per un flusso a velocità elevata, è meglio attivare flussi di lavoro Step Function di post-elaborazione asincrona rispetto a funzioni Lambda sincrone a lunga durata.

Non è possibile utilizzare lo stesso trigger Lambda su account diversi AWS . Sia la tabella DynamoDB che le funzioni Lambda devono appartenere allo stesso account. AWS

Per ulteriori informazioni in merito AWS Lambda, consulta la Guida per gli sviluppatori AWS Lambda .

Tutorial #1: Utilizzo dei filtri per elaborare tutti gli eventi con Amazon DynamoDB e utilizzo di AWS LambdaAWS CLI

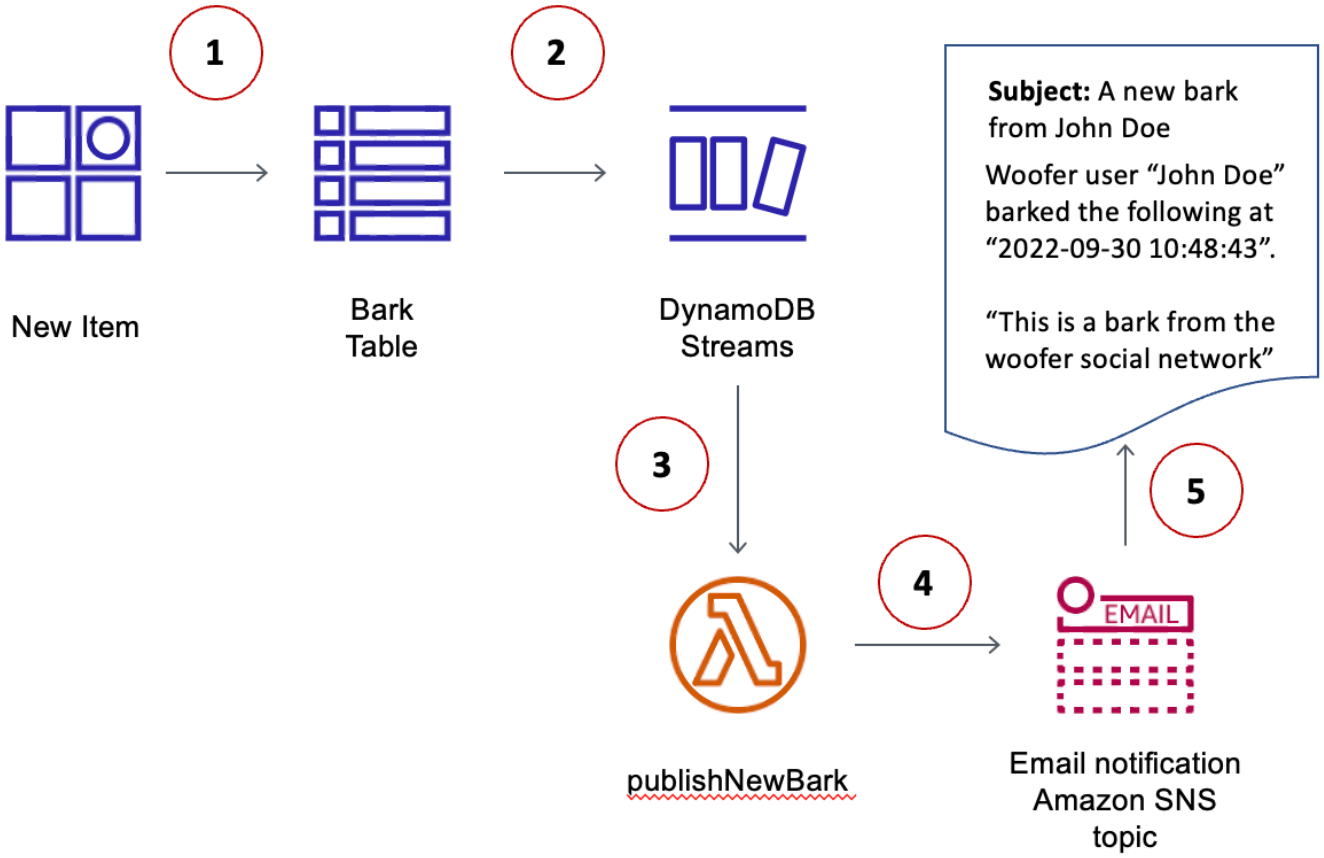
In questo tutorial, creerai un AWS Lambda trigger per elaborare un flusso da una tabella DynamoDB.

Argomenti

- Fase 1: creazione di una tabella DynamoDB con un flusso abilitato

- Fase 2: creazione di un ruolo di esecuzione Lambda
- Fase 3: creazione di un argomento Amazon SNS
- Fase 4: creazione e test di una funzione Lambda
- Fase 5: creazione e test di un trigger

Lo scenario di questo tutorial è Woofer, un semplice social network. Gli utenti di Woofer comunicano tramite i bark, brevi messaggi di testo che vengono inviati ad altri utenti di Woofer. Il seguente diagramma illustra i componenti e il flusso di lavoro di questa applicazione.



1. Un utente scrive un elemento in una tabella DynamoDB (`BarkTable`). Ogni item della tabella rappresenta un bark.
2. Viene scritto un nuovo record di flusso per riflettere l'aggiunta di un nuovo item a `BarkTable`.
3. Il nuovo record di flusso attiva una AWS Lambda funzione (). `publishNewBark`
4. Se il record di flusso indica che è stato aggiunto un nuovo elemento a `BarkTable`, la funzione Lambda legge i dati dal record di flusso e pubblica un messaggio in un argomento di Amazon Simple Notification Service (Amazon SNS).

5. Questo messaggio è ricevuto dai sottoscrittori dell'argomento Amazon SNS. (In questo tutorial, l'unico sottoscrittore è un indirizzo e-mail).

Prima di iniziare

Questo tutorial utilizza il. AWS Command Line Interface AWS CLI Se non è stato ancora fatto, seguire le istruzioni contenute nella Guida per l'utente di AWS Command Line Interface per installare e configurare la AWS CLI.

Fase 1: creazione di una tabella DynamoDB con un flusso abilitato

In questa fase, viene creata crea una tabella DynamoDB (`BarkTable`) per memorizzare tutti i bark degli utenti di Woofer. La chiave primaria è costituita da `Username` (chiave di partizione) e `Timestamp` (chiave di ordinamento). Entrambi questi attributi sono di tipo stringa.

In `BarkTable` è abilitato un flusso. Più avanti in questo tutorial, crei un trigger associando una AWS Lambda funzione allo stream.

1. Immetti il seguente comando per creare la tabella.

```
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. Nell'output, cerca `LatestStreamArn`.

```
...
"LatestStreamArn": "arn:aws:dynamodb:region:accountID:table/BarkTable/
stream/timestamp
...
```

   Prendi nota dei valori *region* e *accountID*, in quanto sono necessari nelle altre fasi di questo tutorial.

Fase 2: creazione di un ruolo di esecuzione Lambda

In questo passaggio, crei un ruolo AWS Identity and Access Management (IAM) (`WooferLambdaRole`) e gli assegni le autorizzazioni. Questo ruolo viene utilizzato dalla funzione Lambda creata in Fase 4: creazione e test di una funzione Lambda.

Puoi creare anche una policy per il ruolo. La policy conterrà tutte le autorizzazioni necessarie alla funzione Lambda nella fase di runtime.

1. Crea un file denominato `trust-relationship.json` con i seguenti contenuti.

   JSON

   ```
   {
      "Version":"2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Principal": {
            "Service": "lambda.amazonaws.com"
          },
          "Action": "sts:AssumeRole"
        }
      ]
    }
   ```

2. Immetti il seguente comando per creare `WooferLambdaRole`.

   ```
   aws iam create-role --role-name WooferLambdaRole \
       --path "/service-role/" \
       --assume-role-policy-document file://trust-relationship.json
   ```

3. Crea un file denominato `role-policy.json` con i seguenti contenuti. (Sostituisci *`region`* e inserisci *`accountID`* la tua AWS regione e l'ID dell'account.)

   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 policy include quattro dichiarazioni che consentono a `WooferLambdaRole` di eseguire le seguenti operazioni:

- Eseguire una funzione Lambda (`publishNewBark`). Questa funzione viene creata in una fase successiva di questo tutorial.
- Accedi ad Amazon CloudWatch Logs. La funzione Lambda scrive la diagnostica nei CloudWatch registri in fase di esecuzione.
- Leggere i dati dal flusso di DynamoDB per `BarkTable`.
- Pubblicare i messaggi su Amazon SNS.

4. Immetti il seguente comando per collegare la policy a `WooferLambdaRole`.

```
aws iam put-role-policy --role-name WooferLambdaRole \
    --policy-name WooferLambdaRolePolicy \
    --policy-document file://role-policy.json
```

Fase 3: creazione di un argomento Amazon SNS

In questa fase, viene creato un argomento Amazon SNS (`wooferTopic`) e viene registrato un indirizzo e-mail su di esso. La funzione Lambda utilizza questo argomento per pubblicare nuovi bark degli utenti di Woofer.

1. Immettere il seguente comando per creare un nuovo argomento Amazon SNS.

```
aws sns create-topic --name wooferTopic
```

2. Immetti il comando seguente per sottoscrivere un indirizzo e-mail a `wooferTopic`. Sostituisci *`region`* e *`accountID`* con la regione e l'ID account AWS e sostituisci *`example@example.com`* con un indirizzo e-mail valido.

```
aws sns subscribe \
    --topic-arn arn:aws:sns:region:accountID:wooferTopic \
    --protocol email \
    --notification-endpoint example@example.com
```

3. Amazon SNS invia un messaggio di conferma al tuo indirizzo e-mail. Scegli il link Confirm subscription (Conferma sottoscrizione) per completare la procedura di sottoscrizione.

Fase 4: creazione e test di una funzione Lambda

In questo passaggio, crei una AWS Lambda funzione (`publishNewBark`) da cui elaborare i record di flusso. `BarkTable`

La funzione `publishNewBark` elabora solo gli eventi di flusso che corrispondono a nuovi item in `BarkTable`. La funzione legge i dati di questo evento, quindi richiama Amazon SNS perché li pubblichi.

1. Crea un file denominato `publishNewBark.js` con i seguenti contenuti. Sostituisci *`region`* e *`accountID`* con la tua AWS regione e l'ID dell'account.

```
'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. Crea un file zip che contenga `publishNewBark.js`. Per fare ciò, se disponi di una utility a riga di comando zip, puoi immettere il comando seguente.

```
zip publishNewBark.zip publishNewBark.js
```

3. Quando si crea la funzione Lambda, si specifica l'Amazon Resource Name (ARN) per `WooferLambdaRole` che hai creato in Fase 2: creazione di un ruolo di esecuzione Lambda. Immetti il comando seguente per recuperare questo ARN.

```
aws iam get-role --role-name WooferLambdaRole
```

Nell'output, cerca l'ARN di `WooferLambdaRole`.

```
...
"Arn": "arn:aws:iam::region:role/service-role/WooferLambdaRole"
...
```

Immetti il seguente comando per creare la funzione Lambda. Sostituire *`roleARN`* con l'ARN per. `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. Ora esegui il test di `publishNewBark` per verificare che funziona. Per fare ciò, fornire un input simile a un record reale da DynamoDB Streams.

   Crea un file denominato `payload.json` con i seguenti contenuti. Sostituisci *`region`* e *`accountID`* con la Regione AWS e l'ID account.

```
{
    "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"
        }
    ]
}
```

Immetti il comando seguente per eseguire il test della funzione `publishNewBark`.

```
aws lambda invoke --function-name publishNewBark --payload file://payload.json --
cli-binary-format raw-in-base64-out output.txt
```

Se il test viene superato, viene visualizzato il seguente output.

```
{
    "StatusCode": 200,
    "ExecutedVersion": "$LATEST"
}
```

Inoltre, il file `output.txt` conterrà il testo seguente.

```
"Successfully processed 1 records."
```

Entro pochi minuti riceverai anche un nuovo messaggio e-mail.

> **Note**
>
> AWS Lambda scrive informazioni diagnostiche su Amazon CloudWatch Logs. Se si verificano errori nella funzione Lambda, è possibile utilizzare queste informazioni diagnostiche per la risoluzione dei problemi:
>
> 1. Apri la CloudWatch console all'indirizzo https://console.aws.amazon.com/cloudwatch/.
> 2. Nel riquadro di navigazione scegli Logs (Log).
> 3. Scegli il seguente gruppo di log: `/aws/lambda/publishNewBark`
> 4. Scegli il flusso di log più recente per visualizzare l'output e gli errori della funzione.

Fase 5: creazione e test di un trigger

In Fase 4: creazione e test di una funzione Lambda, è stato eseguito il test della funzione Lambda per verificarne la corretta esecuzione. In questa fase, è possibile creare un trigger associando la funzione Lambda (`publishNewBark`) a un'origine eventi (il flusso `BarkTable`).

1. Quando crei il trigger, è necessario specificare l'ARN del flusso `BarkTable`. Immetti il comando seguente per recuperare questo ARN.

   ```
   aws dynamodb describe-table --table-name BarkTable
   ```

   Nell'output, cerca `LatestStreamArn`.

   ```
   ...
    "LatestStreamArn": "arn:aws:dynamodb:region:accountID:table/BarkTable/
   stream/timestamp
   ...
   ```

2. Immetti il comando seguente per creare il trigger. Sostituisci *`streamARN`* con l'ARN del flusso effettivo.

   ```
   aws lambda create-event-source-mapping \
       --region region \
       --function-name publishNewBark \
   ```

```
    --event-source streamARN  \
    --batch-size 1 \
    --starting-position TRIM_HORIZON
```

3. Esegui il test del trigger. Immetti il comando seguente per aggiungere 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"}
```

   Dovresti ricevere un nuovo messaggio e-mail entro pochi minuti.

4. Aprire la console DynamoDB e aggiungere altri elementi a `BarkTable`. È necessario specificare i valori degli attributi `Username` e `Timestamp`. (Sebbene non sia obbligatorio, si dovrebbe inoltre specificare un valore per `Message`) Dovresti ricevere un nuovo messaggio e-mail per ogni item aggiunto a `BarkTable`.

   La funzione Lambda elabora solo i nuovi elementi aggiunti a `BarkTable`. Se aggiorni o elimini un item della tabella, la funzione non esegue alcuna azione.

> **Note**
>
> AWS Lambda scrive informazioni diagnostiche su Amazon CloudWatch Logs. Se si verificano errori nella funzione Lambda, è possibile utilizzare queste informazioni diagnostiche per la risoluzione dei problemi:
>
> 1. Apri la CloudWatch console all'indirizzo https://console.aws.amazon.com/cloudwatch/.
> 2. Nel riquadro di navigazione scegli Logs (Log).
> 3. Scegli il seguente gruppo di log: `/aws/lambda/publishNewBark`
> 4. Scegli il flusso di log più recente per visualizzare l'output e gli errori della funzione.

Tutorial n. 2: utilizzo di filtri per elaborare alcuni eventi con DynamoDB e Lambda

In questo tutorial, creerai un AWS Lambda trigger per elaborare solo alcuni eventi in un flusso da una tabella DynamoDB.

Argomenti

- Mettere tutto insieme - CloudFormation
- Mettere tutto insieme - CDK

Con il filtro eventi Lambda è possibile utilizzare espressioni di filtro per controllare quali eventi Lambda invia alla funzione per l'elaborazione. Puoi configurare fino a 5 diversi filtri per i flussi DynamoDB. Se si utilizzano finestre di batch, Lambda applica i criteri di filtro a ogni nuovo evento per stabilire se aggiungerlo al batch corrente.

I filtri vengono applicati tramite strutture chiamate `FilterCriteria`. I 3 attributi principali di `FilterCriteria` sono `metadata properties`, `data properties` e `filter patterns`.

Ecco una struttura di esempio di un evento di flussi DynamoDB:


```
{
  "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"
}
```

`metadata properties` sono i campi dell'oggetto evento. Nel caso di flussi DynamoDB, `metadata properties` sono campi come dynamodb o `eventName`.

`data properties` sono i campi del corpo dell'evento. Per filtrare su `data properties`, bisogna assicurarsi di contenerli in `FilterCriteria` all'interno della chiave appropriata. Per le origini eventi Dynamo DB, la chiave dati è `NewImage` o `OldImage`.

Infine, le regole di filtro definiranno l'espressione del filtro che si desidera applicare a una proprietà specifica. Ecco alcuni esempi:

Operatore di confronto	Esempio	Sintassi delle regole (parziale)
Null	Il tipo di prodotto è null	`{ "product_type": { "S": null } }`
Empty	Il nome del prodotto è vuoto	`{ "product_name": { "S": [ ""] } }`
Equals	Lo stato equivale a Florida	`{ "state": { "S": ["FL"] } }`
And	Lo stato del prodotto equivale a Florida e la categoria di prodotto è Chocolate	`{ "state": { "S": ["FL"] } , "category": { "S": [ "CHOCOLATE"] } }`
Or	Lo stato del prodotto è Florida o California	`{ "state": { "S": ["FL","CA"] } }`
Not	Lo stato del prodotto non è Florida	`{"state": {"S": [{"anything-but": ["FL"]}]}}`
Exists	Esiste il prodotto artigianale	`{"homemade": {"S": [{"exists": true}]}}`

Operatore di confronto	Esempio	Sintassi delle regole (parziale)
Does not exist	Il prodotto "homemade" non esiste	`{"homemade": {"S": [{"exists": false}]}}`
Begins with	PK inizia con COMPANY	`{"PK": {"S": [{"prefix": "COMPANY"}]}}`

Per una funzione Lambda è possibile specificare fino a 5 modelli di filtro eventi. Si noti che ognuno di questi 5 eventi verrà valutato come un OR logico. Quindi, se configuri due filtri denominati `Filter_One` e `Filter_Two`, la funzione Lambda eseguirà `Filter_One` OR `Filter_Two`.

> **Note**
>
> Nella pagina di filtraggio degli eventi Lambda ci sono alcune opzioni per filtrare e confrontare valori numerici, tuttavia nel caso degli eventi di filtro DynamoDB ciò non si applica perché i numeri in DynamoDB vengono memorizzati come stringhe. Ad esempio `"quantity": { "N": "50" }`, sappiamo che è un numero a causa della proprietà `"N"`.

Mettere tutto insieme - CloudFormation

Per mostrare in pratica la funzionalità di filtraggio degli eventi, ecco un modello di esempio. CloudFormation Questo modello genererà una tabella DynamoDB semplice con una chiave di partizione PK e una chiave di ordinamento SK con flussi Amazon DynamoDB abilitati. Creerà una funzione Lambda e un semplice ruolo di esecuzione Lambda che consentirà di scrivere registri su Amazon Cloudwatch e leggere gli eventi dai flussi Amazon DynamoDB. Aggiungerà anche la mappatura dell'origine eventi tra flussi DynamoDB e la funzione Lambda, in modo che la funzione possa essere eseguita ogni volta che c'è un evento nei flussi Amazon DynamoDB.

```
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
```

Dopo aver distribuito questo modello di CloudFormation, puoi inserire il seguente elemento Amazon DynamoDB:

```
{
 "PK": "COMPANY#1000",
 "SK": "PRODUCT#CHOCOLATE#DARK",
 "company_id": "1000",
 "type": "",
 "state": "FL",
 "stores": 5,
 "price": 15,
 "quantity": 50,
 "fabric": "Florida Chocolates"
}
```

Grazie alla semplice funzione lambda inclusa in linea in questo modello di formazione cloud, vedrai gli eventi nei gruppi di CloudWatch log di Amazon per la funzione lambda come segue:

```
{
  "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"
}
```

Esempi di filtri

- Solo prodotti che corrispondono a un determinato stato

Questo esempio modifica il CloudFormation modello per includere un filtro per abbinare tutti i prodotti provenienti dalla Florida, con l'abbreviazione «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
```

Dopo aver ridistribuito lo stack, puoi aggiungere il seguente elemento DynamoDB alla tabella. Si noti che non verrà visualizzato nei registri delle funzioni Lambda, poiché il prodotto in questo esempio proviene dalla 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 gli elementi che iniziano con alcuni valori in PK e SK

Questo esempio modifica il CloudFormation modello per includere la seguente condizione:

```
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
```

Si noti che la condizione AND richiede che la condizione sia all'interno del modello, dove le chiavi PK e SK sono nella stessa espressione separate da una virgola.

O inizia con alcuni valori su PK e SK o proviene da un determinato stato.

Questo esempio modifica il CloudFormation modello per includere le seguenti condizioni:

```
 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
```

Si noti che la condizione OR viene aggiunta introducendo nuovi modelli nella sezione del filtro.

Mettere tutto insieme - CDK

Il seguente modello di formazione del progetto CDK di esempio illustra la funzionalità di filtro degli eventi. Prima di lavorare con questo progetto CDK è necessario installare i prerequisiti, inclusa l'esecuzione degli script di preparazione.

Creazione di un progetto CDK

Per prima cosa crea un nuovo AWS CDK progetto, invocandolo `cdk init` in una directory vuota.

```
mkdir ddb_filters
cd ddb_filters
cdk init app --language python
```

Il comando `cdk init` utilizza il nome della cartella del progetto per denominare vari elementi del progetto, tra cui classi, sottocartelle e file. Tutti i trattini nel nome della cartella vengono convertiti in caratteri di sottolineatura. Altrimenti il nome dovrebbe seguire il formato di un identificatore Python. Ad esempio, non dovrebbe iniziare con un numero o contenere spazi.

Per lavorare con il nuovo progetto, attivare il suo ambiente virtuale. Ciò consente di installare le dipendenze del progetto localmente nella cartella del progetto, anziché globalmente.

```
source .venv/bin/activate
python -m pip install -r requirements.txt
```

> **Note**
>
> Potresti riconoscerlo come il Mac/Linux comando per attivare un ambiente virtuale. I modelli Python includono un file batch, `source.bat`, che consente di utilizzare lo stesso comando su Windows. Funziona anche il comando Windows tradizionale `.venv\Scripts\activate.bat`. Se hai inizializzato il tuo AWS CDK progetto utilizzando AWS CDK Toolkit v1.70.0 o precedente, il tuo ambiente virtuale si trova invece nella directory. `.env .venv`

Infrastruttura di base

Apri il file `./ddb_filters/ddb_filters_stack.py` con l'editor di testo preferito. Questo file è stato generato automaticamente al momento della creazione del AWS CDK progetto.

Quindi, aggiungi le funzioni `_create_ddb_table` e `_set_ddb_trigger_function`. Queste funzioni creeranno una tabella DynamoDB con chiave di partizione PK e una chiave di ordinamento SK in modalità di assegnazione in modalità on-demand, con flussi Amazon DynamoDB abilitato per impostazione predefinita per mostrare immagini nuove e vecchie.

La funzione Lambda verrà archiviata nella cartella `lambda` nel file `app.py`. Questo file verrà creato in seguito. Comprenderà una variabile di ambiente `APP_TABLE_NAME`, che sarà il nome della tabella Amazon DynamoDB creata da questo stack. Nella stessa funzione concederemo alla funzione Lambda le autorizzazioni di lettura del flusso. Infine, verrà effettuata la sottoscrizione a flussi DynamoDB come origine degli eventi per la funzione Lambda.

Alla fine del file nel metodo `__init__`, richiamerai i rispettivi costrutti per inizializzarli nello stack. Per progetti più grandi che richiedono componenti e servizi aggiuntivi, potrebbe essere meglio definire questi costrutti al di fuori dello stack di 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)
```

Ora creeremo una funzione lambda molto semplice che stamperà i log in Amazon. CloudWatch Per farlo, crea una nuova cartella denominata `lambda`.

```
mkdir lambda
touch app.py
```

Usando l'editor di testo preferito, aggiungi il seguente contenuto al file `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)
```

Assicurati di essere nella cartella `/ddb_filters/`, digita il seguente comando per creare l'applicazione di esempio:

```
cdk deploy
```

A un certo punto ti verrà chiesto di confermare se desideri implementare la soluzione. Accetta le modifiche digitando Y.

```
##########################################################################################
# + # ${LambdaHandler/ServiceRole} # arn:${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 volta implementate le modifiche, apri la AWS console e aggiungi un elemento alla tabella.

```
{
 "PK": "COMPANY#1000",
 "SK": "PRODUCT#CHOCOLATE#DARK",
 "company_id": "1000",
 "type": "",
```

```
 "state": "FL",
 "stores": 5,
 "price": 15,
 "quantity": 50,
 "fabric": "Florida Chocolates"
}
```

I CloudWatch log dovrebbero ora contenere tutte le informazioni di questa voce.

Esempi di filtri

- Solo prodotti che corrispondono a un determinato stato

Apri il file `ddb_filters/ddb_filters/ddb_filters_stack.py` e modificalo per includere il filtro che corrisponde a tutti i prodotti equivalenti a "FL". Questo può essere modificato appena sotto `event_subscription` nella riga 45.

```
event_subscription.add_property_override(
    property_path="FilterCriteria",
    value={
        "Filters": [
            {
                "Pattern": json.dumps(
                    {"dynamodb": {"NewImage": {"state": {"S": ["FL"]}}}}
                )
            },
        ]
    },
)
```

- Solo gli elementi che iniziano con alcuni valori in PK e SK

Modifica lo script Python per includere la seguente condizione:

```
event_subscription.add_property_override(
    property_path="FilterCriteria",
    value={
        "Filters": [
            {
                "Pattern": json.dumps(
                    {
```

```
                        {
                            "dynamodb": {
                                "Keys": {
                                    "PK": {"S": [{"prefix": "COMPANY"}]},
                                    "SK": {"S": [{"prefix": "PRODUCT"}]},
                                }
                            }
                        }
                    }
                )
            },
        ]
    },
```

- O inizia con alcuni valori su PK e SK o proviene da un determinato stato.

Modifica lo script Python per includere le seguenti condizioni:

```
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"]}}}}
                )
            },
        ]
```

```
    },
)
```

Si noti che la condizione OR viene aggiunta aggiungendo altri elementi all'array Filters.

Pulizia

Individua lo stack di filtri nella base della tua directory di lavoro ed esegui `cdk destroy`. Ti verrà chiesto di confermare l'eliminazione della risorsa:

```
cdk destroy
Are you sure you want to delete: DdbFiltersStack (y/n)? y
```

Best practice per l'utilizzo dei flussi DynamoDB con Lambda

Una AWS Lambda funzione viene eseguita all'interno di un contenitore, un ambiente di esecuzione isolato da altre funzioni. Quando esegui una funzione per la prima volta, AWS Lambda crea un nuovo contenitore e inizia a eseguire il codice della funzione.

Una funzione Lambda dispone di un gestore che viene eseguito a ogni richiamo. Il gestore contiene la logica di business principale della funzione. Ad esempio, la funzione Lambda illustrata in Fase 4: creazione e test di una funzione Lambda dispone di un gestore in grado di elaborare record in un flusso DynamoDB.

Puoi anche fornire un codice di inizializzazione che venga eseguito una sola volta, dopo la creazione del contenitore, ma prima che il gestore venga AWS Lambda eseguito per la prima volta. La funzione Lambda mostrata in Fase 4: creazione e test di una funzione Lambda ha un codice di inizializzazione che importa l'SDK per JavaScript Node.js e crea un client per Amazon SNS. Questi oggetti dovrebbero essere definiti una sola volta, esternamente al gestore.

Dopo l'esecuzione della funzione, AWS Lambda potrebbe scegliere di riutilizzare il contenitore per le successive chiamate della funzione. In questo caso, il gestore della funzione potrebbe essere in grado di utilizzare nuovamente le risorse definite nel codice di inizializzazione. (Non puoi controllare per quanto tempo AWS Lambda conserva il container, né se questo verrà nuovamente utilizzato o meno).

Per l' AWS Lambda utilizzo dei trigger DynamoDB, consigliamo quanto segue:

- AWS i client di servizio devono essere istanziati nel codice di inizializzazione, non nel gestore. Ciò consente di AWS Lambda riutilizzare le connessioni esistenti, per tutta la durata del contenitore.

- In generale, non è necessario gestire in modo esplicito le connessioni o implementare il pool di connessioni perché lo AWS Lambda gestisce per te.

Un consumer Lambda per un flusso DynamoDB non garantisce una consegna esatta di una sola volta e può portare a duplicati occasionali. Assicurati che il codice della funzione Lambda sia idempotente per evitare che si verifichino problemi imprevisti dovuti all'elaborazione duplicata.

Per ulteriori informazioni, consulta Best practice per l'utilizzo delle AWS Lambda funzioni nella AWS Lambda Developer Guide.

Flussi DynamoDB e Apache Flink

È possibile utilizzare i record dei flussi Amazon DynamoDB con Apache Flink. Con il Servizio gestito da Amazon per Apache Flink è possibile trasformare e analizzare i dati in streaming in tempo reale con Apache Flink. Apache Flink è un framework di elaborazione di flussi open source per l'elaborazione di dati in tempo reale. Il connettore dei flussi Amazon DynamoDB per Apache Flink semplifica la creazione e la gestione dei carichi di lavoro Apache Flink e consente di integrare le applicazioni con altri Servizi AWS.

Amazon Managed Service for Apache Flink ti aiuta a creare rapidamente applicazioni di elaborazione dei end-to-end flussi per l'analisi dei log, l'analisi dei clickstream, l'Internet of Things (IoT), la tecnologia pubblicitaria, i giochi e altro ancora. I quattro casi d'uso più comuni sono lo streaming extract-transform-load (ETL), le applicazioni basate sugli eventi, l'analisi reattiva in tempo reale e l'interrogazione interattiva dei flussi di dati. Per ulteriori informazioni sulla scrittura in Apache Flink dai flussi Amazon DynamoDB, consulta Amazon DynamoDB Streams Connector.

Utilizzo del ShardFilter parametro con il connettore DynamoDB Streams per Apache Flink

I flussi Amazon DynamoDB supportano il parametro `ShardFilter` nell'API `DescribeStream` per trovare un sottoinsieme di shard. Specificando uno shard principale nella richiesta, i flussi DynamoDB restituiranno i relativi shard secondari immediati. È possibile usare il parametro `ShardFilter` per individuare rapidamente gli shard secondari dopo la chiusura di uno shard principale, rendendo le applicazioni di elaborazione dei flussi più reattive e convenienti.

Il servizio gestito da Amazon per Apache Flink supporta il parametro `ShardFilter` durante la lettura dai flussi DynamoDB. Per utilizzare questa funzionalità, è necessario richiedere l'accesso a una versione speciale del connettore Flink che include il supporto `ShardFilter`. Per iniziare, apri una richiesta di supporto nel AWS Support Center e richiedi l'accesso a questi file binari. Per ulteriori

informazioni sull'utilizzo dei flussi DynamoDB con Apache Flink, consulta Lettura ed elaborazione di un flusso nella Guida per gli sviluppatori di DynamoDB.

Accelerazione in memoria con DynamoDB Accelerator (DAX)

Amazon DynamoDB è progettato per dimensionamento e prestazioni. Nella maggior parte dei casi, i tempi di risposta di DynamoDB possono essere misurati nell'ordine di millisecondi a una cifra. Tuttavia, alcuni casi d'uso richiedono tempi di risposta nell'ordine di microsecondi. Per questi casi d'uso, DynamoDB Accelerator (DAX) offre tempi di risposta rapidi per l'accesso a dati a consistenza finale.

DAX è un servizio di memorizzazione nella cache compatibile con DynamoDB che permette di trarre vantaggio da rapide prestazioni in memoria per le applicazioni più complesse. DAX affronta tre scenari principali:

1. Come cache in memoria, DAX riduce in misura esponenziale i tempi di risposta dei carichi di lavoro di lettura a consistenza finale, da millisecondi a una cifra a microsecondi.
2. DAX riduce la complessità operativa e dell'applicazione fornendo un servizio gestito compatibile con DynamoDB a livello di API e di conseguenza richiede solo modifiche funzionali minime per l'uso con un'applicazione esistente.
3. Per carichi di lavoro gravosi in lettura o con lunghi intervalli di inattività, DAX offre velocità effettiva maggiore e possibili risparmi sui costi operativi riducendo la necessità di una assegnazione eccessiva di unità di capacità di lettura. Questa caratteristica è particolarmente vantaggiosa per le applicazioni che richiedono letture ripetute per singole chiavi.

DAX supporta la crittografia lato server. Con la crittografia a riposo, i dati persistenti da DAX su disco verranno crittografati. DAX; scrive i dati sul disco durante la propagazione delle modifiche dal nodo primario alle repliche di lettura. Per ulteriori informazioni, consulta Crittografia DAX dei dati inattivi.

DAX inoltre supporta la crittografia in transito, garantendo che tutte le richieste e le risposte tra l'applicazione e il cluster siano crittografate tramite TLS (Transport Level Security) e che le connessioni al cluster possano essere autenticate mediante la verifica di un certificato cluster x509. Per ulteriori informazioni, consulta Crittografia DAX in transito.

Argomenti

- Casi d'uso per DAX
- Note per l'utilizzo di DAX

- DAX: come funziona
- Componenti del cluster DAX
- Creazione di un cluster DAX
- Modelli di consistenza DAX e DynamoDB
- Sviluppo con il client DynamoDB Accelerator (DAX)
- Gestione dei cluster DAX
- Monitoraggio di DynamoDB Accelerator
- Istanze espandibili T3/T2 DAX
- Controllo degli accessi DAX
- Crittografia DAX dei dati inattivi
- Crittografia DAX in transito
- Utilizzo di ruoli IAM collegati ai servizi per DAX
- Accesso a DAX su più account AWS
- Guida alle dimensioni del cluster DAX

Casi d'uso per DAX

DAX permette di accedere a dati a consistenza finale da tabelle DynamoDB con latenza nell'ordine di microsecondi. Un cluster DAX multi-AZ può gestire milioni di richieste al secondo.

DAX è l'ideale per i seguenti tipi di applicazioni:

- Applicazioni che richiedono il tempo di risposta minore possibile per le letture. Alcuni esempi includono applicazioni di aste in tempo reale, videogiochi, social e trading. DAX offre prestazioni di lettura in memoria rapide per questi casi d'uso.
- Applicazioni che leggono un numero ridotto di item più spesso di altri. Ad esempio, considera un sistema di e-commerce che organizza la vendita per un solo giorno di un prodotto molto richiesto. Durante la vendita, la richiesta del prodotto (e dei relativi dati in DynamoDB) aumenta nettamente rispetto a tutti gli altri prodotti. Per ridurre gli impatti di una chiave hot e una distribuzione non uniforme del traffico, è possibile eseguire l'offload dell'attività di lettura in una cache DAX fino al termine della vendita della durata di un giorno.
- Applicazioni con attività di lettura intensiva, ma anche sensibili ai costi. Con DynamoDB, è possibile assegnare il numero di letture al secondo richieste dall'applicazione. Se l'attività di lettura aumenta, puoi aumentare il throughput di lettura assegnato delle tabelle (a un costo aggiuntivo). Oppure, è

possibile eseguire l'offload dell'attività dall'applicazione a un cluster DAX e ridurre il numero di unità di capacità di lettura che altrimenti si dovrebbe acquistare.

- Applicazioni che richiedono letture ripetute su set di dati di grandi dimensioni. Un'applicazione di questo tipo può sottrarre risorse del database ad altre applicazioni. Ad esempio, un'analisi a esecuzione prolungata dei dati meteo regionali può temporaneamente utilizzare tutta la capacità di lettura di una tabella DynamoDB. e di conseguenza avere un impatto negativo sulle applicazioni che devono poter accedere agli stessi dati. Con DAX, l'analisi dei dati meteo può invece essere eseguita sui dati memorizzati nella cache.

DAX non è adatto ai seguenti tipi di applicazioni:

- Applicazioni che richiedono letture consistenti assolute o che non possono tollerare letture consistenti finali.
- Applicazioni che non richiedono tempi di risposta nell'ordine di microsecondi per le letture o che non devono eseguire l'offload di attività di lettura ripetute dalle tabelle sottostanti.
- Applicazioni che richiedono un uso intensivo di scrittura. Un volume elevato di scritture porta a un aumento delle repliche tra i nodi DAX di un cluster. Ciò causa un maggiore consumo di risorse e il rischio di problemi di disponibilità.
- Applicazioni senza molte letture ripetute. DAX offre prestazioni ottimali quando le percentuali di riscontri nella cache superano il 90%. Percentuali di riscontri nella cache inferiori aumentano i mancati riscontri nella cache, il che consuma più risorse nel cluster DAX.

Note per l'utilizzo di DAX

- Per un elenco delle AWS regioni in cui è disponibile DAX, consulta i prezzi di Amazon DynamoDB.
- DAX supporta applicazioni scritte in Go, Java, Node.js, Python e.NET, AWS utilizzando client forniti per tali linguaggi di programmazione.
- DAX è disponibile solo per la piattaforma EC2 -VPC.
- La policy del ruolo di servizio del cluster DAX deve consentire l'operazione `dynamodb:DescribeTable` per conservare i metadati relativi alla tabella DynamoDB.
- I cluster DAX mantengono i metadati sui nomi degli attributi degli elementi da archiviare. e tali metadati sono mantenuti a tempo indeterminato (anche dopo che l'item è scaduto o è stato eliminato dalla cache). Le applicazioni che utilizzano un numero illimitato di nomi di attributi possono col tempo causare il riempimento totale della memoria nel cluster DAX. Questo limite si

applica solo ai nomi di attributi di primo livello, non ai nomi di attributi nidificati. Esempi di nomi di attributi di primo livello problematici includono timestamp e session. UUIDs IDs

Questo limite si applica solo ai nomi di attributi, non ai loro valori. Item simile al seguente non sono un problema.

```
{
    "Id": 123,
    "Title": "Bicycle 123",
    "CreationDate": "2017-10-24T01:02:03+00:00"
}
```

Ma item simili al seguente sono un problema se sono in numero sufficiente e ciascuno ha un timestamp diverso.

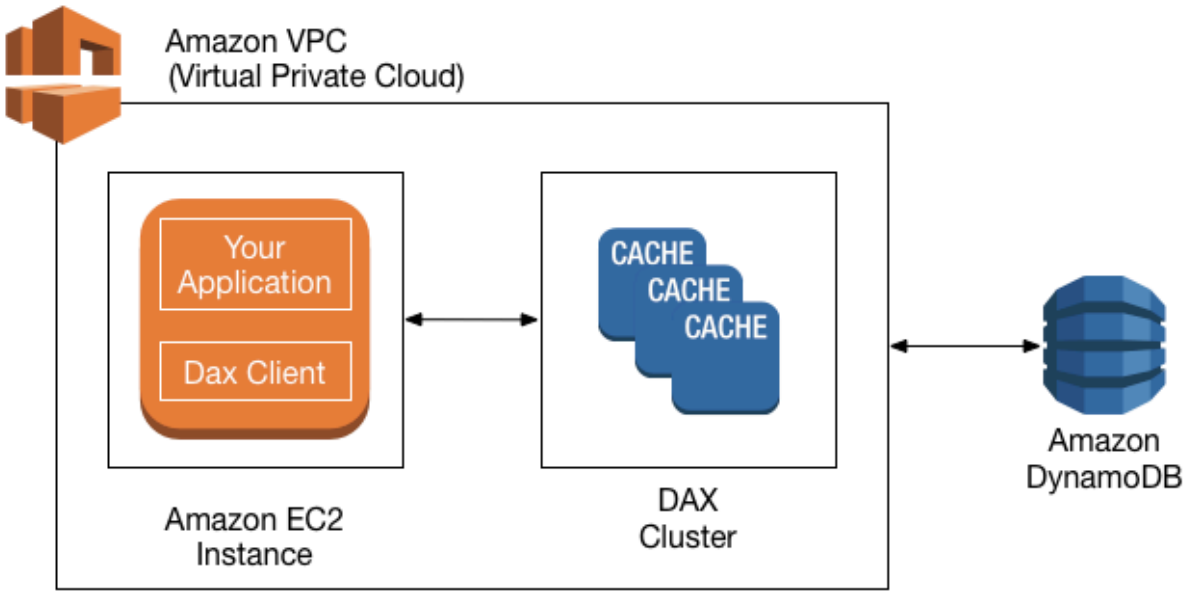
```
{
    "Id": 123,
    "Title": "Bicycle 123",
    "2017-10-24T01:02:03+00:00": "created"
}
```

DAX: come funziona

Amazon DynamoDB Accelerator (DAX) è progettato per funzionare in un ambiente Amazon Virtual Private Cloud (Amazon VPC). Il servizio Amazon VPC definisce una rete virtuale che ricorda molto un data center tradizionale. Con un VPC, puoi controllarne l'intervallo di indirizzi IP, le sottoreti, le tabelle di routing e i gateway di rete, nonché le impostazioni di sicurezza. È possibile avviare un cluster DAX nella rete virtuale e controllare l'accesso al cluster usando i gruppi di sicurezza di Amazon VPC.

> **Note**
>
> Se l'account AWS è stato creato dopo il 4 dicembre 2013, si dispone già di un VPC predefinito in ogni regione AWS. Il VPC è pronto per l'uso: può essere utilizzato immediatamente senza dover eseguire ulteriori operazioni di configurazione.
> Per ulteriori informazioni, consulta VPC predefinito e sottoreti predefinite nella Guida per l'utente di Amazon VPC.

Il seguente diagramma mostra una panoramica di alto livello di DAX.



Per creare un cluster DAX, viene utilizzata la Console di gestione AWS. Se non diversamente specificato, il cluster DAX viene eseguito all'interno del VPC predefinito. Per eseguire l'applicazione, avviare un'istanza Amazon EC2 nell'Amazon VPC. Quindi distribuire l'applicazione (con il client DAX) nell'istanza EC2.

In fase di runtime, il client DAX indirizza tutte le richieste API DynamoDB dell'applicazione al cluster DAX. Se può, DAX elabora una di queste richieste API direttamente. In caso contrario, passerà la richiesta a DynamoDB.

Infine, il cluster DAX restituisce i risultati all'applicazione.

Argomenti

- Come sono elaborate le richieste da DAX
- Cache degli elementi
- Cache delle query

Come sono elaborate le richieste da DAX

Un cluster DAX è costituito da uno o più nodi. Ogni nodo esegue la propria istanza del software memorizzazione nella cache di DAX. Uno dei nodi funge da nodo principale per il cluster. I nodi aggiuntivi (se presenti) servono da repliche di lettura. Per ulteriori informazioni, consulta Nodi.

La tua applicazione può accedere a DAX specificando l'endpoint per il cluster DAX. Il software client DAX funziona con l'endpoint del cluster per eseguire il routing e il bilanciamento del carico intelligente.

Operazioni di lettura

DAX può rispondere alle seguenti chiamate API:

- `GetItem`
- `BatchGetItem`
- `Query`
- `Scan`

Se la richiesta specifica letture a consistenza finale (comportamento predefinito), prova a leggere l'elemento da DAX:

- Se in DAX è disponibile l'elemento, (hit della cache), DAX restituisce l'elemento all'applicazione senza accedere a DynamoDB.
- Se in DAX l'elemento non è disponibile (mancato riscontro nella cache), DAX passa la richiesta a DynamoDB. Quando riceve la risposta da DynamoDB, DAX restituisce i risultati all'applicazione e li scrive nella cache del nodo primario.

> **Note**
>
> Se ci sono delle repliche di lettura nel cluster, DAX mantiene automaticamente le repliche sincronizzate con il nodo primario. Per ulteriori informazioni, consulta Cluster.

Se la richiesta specifica letture fortemente consistenti, DAX passa la richiesta a DynamoDB. I risultati di DynamoDB non vengono memorizzati nella cache in DAX. ma vengono restituiti all'applicazione.

Operazioni di scrittura

Le seguenti operazioni API DAX sono considerate "write-through":

- `BatchWriteItem`
- `UpdateItem`
- `DeleteItem`
- `PutItem`

Con queste operazioni, i dati vengono prima scritti nella tabella DynamoDB e quindi nel cluster DAX. L'operazione ha esito positivo solo se i dati vengono scritti sia sulla tabella che in DAX.

Altre operazioni

DAX non riconosce le operazioni DynamoDB per la gestione delle tabelle (ad esempio `CreateTable`, `UpdateTable` e così via). Se l'applicazione deve eseguire queste operazioni, deve accedere direttamente a DynamoDB anziché usare DAX.

Per informazioni dettagliate sulla coerenza di DAX e DynamoDB, consulta Modelli di consistenza DAX e DynamoDB.

Per informazioni sul funzionamento delle transazioni in DAX, consulta Utilizzo delle transazioni in APIs DynamoDB Accelerator (DAX).

Limitazione del tasso di richiesta

Se il numero di richieste inviate a DAX supera la capacità di un nodo, DAX limita il tasso con cui accetta le richieste aggiuntive restituendo una ThrottlingException. DAX valuta continuamente l'utilizzo della CPU per determinare il volume di richieste che può elaborare mantenendo allo stesso tempo uno stato del cluster integro.

È possibile monitorare il parametro ThrottledRequestCount che DAX pubblica in Amazon CloudWatch. Se queste eccezioni vengono visualizzate regolarmente, è consigliabile prendere in considerazione la possibilità di aumentare la dimensione del cluster.

Cache degli elementi

DAX gestisce una cache di elementi per archiviare i risultati delle operazioni `GetItem` e `BatchGetItem`. Gli elementi nella cache rappresentano i dati a consistenza finale da DynamoDB e sono archiviati in base ai rispettivi valori delle chiavi primarie.

Quando un'applicazione invia una richiesta `GetItem` o `BatchGetItem`, DAX prova a leggere gli elementi direttamente dalla cache di elementi utilizzando i valori delle chiavi specificati. Se gli elementi vengono trovati (hit della cache), DAX li restituisce immediatamente all'applicazione. Se gli elementi non vengono trovati (cache miss), DAX invia la richiesta a DynamoDB. DynamoDB elabora le richieste utilizzando letture a consistenza finale e restituisce gli elementi a DAX. DAX li memorizza nella cache degli elementi e quindi li restituisce all'applicazione.

Per impostazione predefinita, la cache degli elementi ha un'impostazione di durata (TTL, Time to Live), che è di 5 minuti. DAX assegna un timestamp a ogni elemento che scrive nella cache degli elementi. Un item scade se è rimasto nella cache più a lungo dell'impostazione TTL. Se si emetti una richiesta `GetItem` per un elemento scaduto, la richiesta viene considerata un mancato riscontro nella cache e DAX invia la richiesta `GetItem` a DynamoDB.

> **Note**
>
> È possibile specificare l'impostazione TTL per la cache di elementi quando si crea un nuovo cluster DAX. Per ulteriori informazioni, consulta Gestione dei cluster DAX .

DAX conserva anche un elenco degli elementi utilizzati meno di recente (LRU, last recently used) per la cache di elementi. L'elenco LRU monitora quando un item è stato scritto per la prima volta nella cache e quando è stato letto l'ultima volta dalla cache. Se la cache di elementi si riempie, DAX elimina gli elementi più vecchi (anche se non sono ancora scaduti) per fare spazio a quelli nuovi. L'algoritmo LRU è sempre abilitato per la cache degli elementi e non è configurabile dall'utente.

Se si specifica zero come impostazione TTL della cache degli elementi, gli elementi nella cache degli elementi verranno aggiornati solo a causa di un'espulsione LRU o di una operazione "write-through".

Per informazioni dettagliate sulla coerenza della cache di elementi in DAX, consulta Comportamento della cache di elementi DAX.

Cache delle query

DAX gestisce anche una cache di query per archiviare i risultati delle operazioni `Query` e `Scan`. Gli elementi in questa cache rappresentano set di risultati da query e scansioni delle tabelle DynamoDB. Questi set di risultati sono memorizzati in base ai rispettivi valori dei parametri.

Quando un'applicazione invia una richiesta `Query` o `Scan`, DAX prova a leggere un set di risultati corrispondente dalla cache di query utilizzando i valori dei parametri specificati. Se il set di risultati

viene trovato (hit della cache), DAX lo restituisce immediatamente all'applicazione. Se il set di risultati non viene trovato (cache miss), DAX invia la richiesta a DynamoDB. DynamoDB elabora le richieste utilizzando letture a consistenza finale e restituisce il set di risultati a DAX. DAX lo memorizza nella cache delle query e lo restituisce all'applicazione.

> **Note**
>
> È possibile specificare l'impostazione TTL per la cache di query quando si crea un nuovo cluster DAX. Per ulteriori informazioni, consulta Gestione dei cluster DAX .

DAX mantiene un elenco LRU anche per la cache di query. L'elenco monitora quando un set di risultati è stato scritto per la prima volta nella cache e quando il risultato è stato letto l'ultima volta dalla cache. Se la cache di query si riempie, DAX elimina i set di risultati più vecchi (anche se non sono ancora scaduti) per fare spazio a quelli nuovi. L'algoritmo LRU è sempre abilitato per la cache delle query e non è configurabile dall'utente.

Se si specifica zero come impostazione TTL della cache delle query, la risposta alla query non verrà memorizzata nella cache.

Per informazioni dettagliate sulla coerenza della cache di query in DAX, consulta Comportamento della cache di query DAX.

Componenti del cluster DAX

Un cluster Amazon DynamoDB Accelerator (DAX) è costituito dai componenti dell'infrastruttura AWS. In questa sezione sono descritti questi componenti e come funzionano insieme.

Argomenti

- Nodi
- Cluster
- Regioni zone di disponibilità
- Gruppi di parametri
- Gruppi di sicurezza
- ARN del cluster
- Endpoint del cluster

- Endpoint dei nodi
- Gruppi di sottoreti
- Eventi
- Finestra di manutenzione

Nodi

Un nodo è il più piccolo elemento di base di un cluster DAX. Ogni nodo esegue un'istanza del software DAX e mantiene una singola replica dei dati memorizzati nella cache.

È possibile dimensionare il cluster DAX in due modi diversi:

- Aggiungendo più nodi al cluster. Questo aumenta il throughput di lettura complessivo del cluster.
- Usando un tipo di nodo più grande. I tipi di nodi più grandi forniscono più capacità e possono aumentare il throughput. (Devi creare un nuovo cluster con il nuovo tipo di nodo.)

Ogni nodo all'interno di un cluster è dello stesso tipo di nodo ed esegue lo stesso software di memorizzazione nella cache DAX. Per un elenco dei tipi di nodi disponibili, consulta Prezzi di Amazon DynamoDB.

Cluster

Un cluster è un raggruppamento logico di uno o più nodi gestiti da DAX come un'unità. Uno dei nodi nel cluster è designato come il nodo primario e gli altri nodi, se presenti, sono le repliche di lettura.

Il nodo primario è responsabile per quanto segue:

- Soddisfare le richieste delle applicazioni per i dati memorizzati nella cache.
- Gestione delle operazioni di scrittura in DynamoDB.
- Espellere i dati dalla cache in base alla policy di espulsione del cluster.

Quando vengono apportate modifiche ai dati memorizzati nella cache del nodo primario, DAX propaga le modifiche a tutti i nodi di replica di lettura mediante i log di replica. Dopo aver ricevuto la conferma da tutte le repliche di lettura, DynamoDB elimina i log di replica dal nodo primario.

Un cluster DAX può supportare fino a 11 nodi per cluster (il nodo primario, più un massimo di 10 repliche di lettura).

Le repliche di lettura sono responsabili per quanto segue:

- Soddisfare le richieste delle applicazioni per i dati memorizzati nella cache.
- Espellere i dati dalla cache in base alla policy di espulsione del cluster.

Tuttavia, a differenza del nodo primario, le repliche di lettura non scrivono su DynamoDB.

Le repliche di lettura hanno due scopi aggiuntivi:

- Scalabilità. Se si ha un numero elevato di client applicativi che devono accedere a DAX contemporaneamente, è possibile aggiungere più repliche per il dimensionamento della lettura. DAX distribuisce il carico in modo uniforme su tutti i nodi del cluster. Un altro modo per aumentare il throughput è utilizzare tipi di nodi di cache più grandi.
- Elevata disponibilità. Nel caso di un errore del nodo primario, DAX esegue automaticamente il failover su un nodo di replica di lettura e lo designa come nuovo nodo primario. Se un nodo di replica riporta un errore, altri nodi nel cluster DAX saranno ancora in grado di assolvere le richieste finché il nodo con l'errore non viene ripristinato. Per la massima tolleranza ai guasti, devi distribuire le repliche di lettura in zone di disponibilità separate. Questa configurazione garantisce che il cluster DAX possa continuare a funzionare anche se un'intera zona di disponibilità diventa non disponibile.

> **Important**
>
> Per l'uso in produzione, consigliamo vivamente di utilizzare DAX con almeno tre nodi, posizionati in diverse zone di disponibilità. Per fare in modo che un cluster DAX sia tollerante ai guasti sono richiesti tre nodi.
> Un cluster DAX può essere distribuito con uno o due nodi per lo sviluppo o il test di carichi di lavoro. I cluster con un nodo non sono tolleranti ai guasti e non si consiglia di utilizzare meno di tre nodi in produzione. Se un cluster con un nodo rileva errori software o hardware, il cluster può diventare non disponibile o perdere i dati archiviati nella cache.

> **Important**
>
> Un cluster DAX supporta un massimo di 500 tabelle DynamoDB. Superate le 500 tabelle DynamoDB, il cluster potrebbe subire un peggioramento della disponibilità e delle prestazioni.

Regioni zone di disponibilità

Un cluster DAX in una regione AWS può interagire solo con le tabelle DynamoDB che si trovano nella stessa regione. Per questo motivo, assicurati di avviare il cluster DAX nella regione corretta. Se sono presenti tabelle DynamoDB in altre regioni, sarà necessario avviare i cluster DAX anche in quelle regioni.

Ogni regione è pensata per essere completamente isolata dalle altre regioni . All'interno di ciascuna regione ci sono più zone di disponibilità. Avviando i nodi in diverse zone di disponibilità, puoi ottenere la massima tolleranza ai guasti possibile.

> **Important**
>
> Non posizionare tutti i nodi del cluster in un'unica zona di disponibilità. In questa configurazione, il cluster DAX diventa non disponibile in caso di un fallimento della zona di disponibilità.
> Per l'uso in produzione, consigliamo vivamente di utilizzare DAX con almeno tre nodi, posizionati in diverse zone di disponibilità. Per fare in modo che un cluster DAX sia tollerante ai guasti sono richiesti tre nodi.
> Un cluster DAX può essere distribuito con uno o due nodi per lo sviluppo o il test di carichi di lavoro. I cluster con uno o due nodi non sono tolleranti ai guasti e non consigliamo di utilizzare meno di tre nodi in produzione. Se un cluster con uno o due nodi rileva errori software o hardware, il cluster può diventare non disponibile o perdere i dati memorizzati nella cache.

Gruppi di parametri

I gruppi di parametri vengono utilizzati per gestire le impostazioni di runtime per i cluster DAX. DAX ha diversi parametri che è possibile utilizzare per ottimizzare le prestazioni (come la definizione di una policy TTL per i dati memorizzati nella cache). Un gruppo di parametri è un set denominato di parametri applicabili a un cluster. Puoi assicurarti che tutti i nodi di quel cluster siano configurati esattamente nello stesso modo.

Gruppi di sicurezza

Un cluster DAX viene eseguito in un ambiente Amazon Virtual Private Cloud (Amazon VPC). Questo ambiente è una rete virtuale dedicata all'account AWS e isolata dagli altri VPC. Un gruppo

di sicurezza funge da firewall virtuale per il tuo VPC, permettendoti di controllare il traffico di rete in entrata e in uscita.

Quando avvii un cluster nel VPC, aggiungi una regola di ingresso al tuo gruppo di sicurezza per consentire il traffico di rete in entrata. La regola di ingresso specifica il protocollo (TCP) e il numero di porta (8111) per il cluster. Dopo aver aggiunto questa regola al gruppo di sicurezza, le applicazioni in esecuzione all'interno del VPC possono accedere al cluster DAX.

ARN del cluster

A ogni cluster DAX viene assegnato un Amazon Resource Name (ARN). Di seguito è riportato il formato ARN.

```
arn:aws:dax:region:accountID:cache/clusterName
```

Utilizza l'ARN del cluster in una policy IAM per definire le autorizzazioni per le operazioni API DAX. Per ulteriori informazioni, consulta Controllo degli accessi DAX.

Endpoint del cluster

Ogni cluster DAX fornisce un endpoint del cluster per l'uso da parte dell'applicazione. Accedendo al cluster tramite questo endpoint, l'applicazione non ha bisogno dei nomi host e dei numeri di porta dei singoli nodi nel cluster. L'applicazione viene a "conoscenza" automaticamente di tutti i nodi del cluster, anche se aggiungi o rimuovi le repliche di lettura.

Di seguito è riportato un esempio di endpoint cluster nella regione us-est-1 che non è configurato per l'utilizzo della crittografia in transito.

```
dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
```

Di seguito è riportato un esempio di endpoint cluster nella stessa regione che è configurata per l'utilizzo della crittografia in transito.

```
daxs://my-encrypted-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
```

Endpoint dei nodi

Ciascuno dei singoli nodi in un cluster DAX ha il proprio nome host e il numero di porta. Di seguito è riportato un esempio di un endpoint del nodo.

```
myDAXcluster-a.2cmrwl.clustercfg.dax.use1.cache.amazonaws.com:8111
```

L'applicazione può accedere a un nodo direttamente utilizzando il suo endpoint. Tuttavia, consigliamo di trattare il cluster DAX come una singola unità e accedervi utilizzando l'endpoint del cluster. L'endpoint del cluster evita all'applicazione di dover gestire un elenco di nodi e di doverlo mantenere aggiornato quando aggiungi o rimuovi nodi dal cluster.

Gruppi di sottoreti

L'accesso ai nodi del cluster DAX è limitato alle applicazioni in esecuzione su istanze Amazon EC2 all'interno di un ambiente Amazon VPC. È possibile utilizzare i gruppi di sottoreti per concedere l'accesso al cluster dalle istanze Amazon EC2 in esecuzione su sottoreti specifiche. Un gruppo di sottoreti è una raccolta di sottoreti (generalmente private) che è possibile designare per i cluster in esecuzione in un ambiente Amazon VPC.

Quando si crea un cluster DAX, è necessario specificare un gruppo di sottoreti. DAX utilizza quel gruppo di sottoreti per selezionare una sottorete e gli indirizzi IP all'interno di quella sottorete da associare ai nodi.

Eventi

DAX registra gli eventi significativi all'interno dei cluster, ad esempio l'impossibilità di aggiungere un nodo, l'aggiunta di un nodo o le modifiche ai gruppi di sicurezza. Tramite il monitoraggio degli eventi chiave, puoi conoscere lo stato corrente dei cluster e, in base all'evento, essere in grado di intraprendere operazioni correttive. È possibile accedere a questi eventi usando la Console di gestione AWS o l'operazione `DescribeEvents` nell'API di gestione DAX.

È possibile anche richiedere che le notifiche vengano inviate a un argomento Amazon Simple Notification Service (Amazon SNS) specifico. In questo modo si saprà immediatamente quando un evento si verifica nel cluster DAX.

Finestra di manutenzione

Ogni cluster ha una finestra di manutenzione settimanale durante la quale applicare le modifiche al sistema. Man mano che le modifiche vengono applicate in sequenza, un nodo esistente viene sostituito e un nuovo nodo con le modifiche applicate viene aggiunto al cluster. Durante questo periodo, l'applicazione potrebbe riscontrare errori o rallentamenti temporanei. Pertanto, si consiglia di pianificare la finestra di manutenzione durante il periodo di utilizzo minimo e di modificare questa pianificazione periodicamente in base alle esigenze. Puoi specificare un intervallo di tempo di 24 ore al massimo durante il quale si verificano le attività di manutenzione richieste.

Se non si specifica una finestra di manutenzione preferita durante la creazione o la modifica di un cluster di cache, DAX assegna una finestra di manutenzione di 60 minuti in un giorno della settimana casuale. La finestra di manutenzione di 60 minuti è selezionata casualmente da un blocco di tempo di 8 ore per ogni Regione AWS. La seguente tabella elenca i blocchi temporali per ciascuna regione da cui sono assegnate le finestre di manutenzione predefinite.

Codice regione	Nome della Regione	Finestra di manutenzione
ap-northeast-1	Regione Asia Pacifico (Tokyo)	13:00 - 21:00 UTC
ap-southeast-1	Regione Asia Pacifico (Singapore)	14:00 - 22:00 UTC
ap-southeast-2	Asia Pacifico (Sydney)	12:00 - 20:00 UTC
ap-south-1	Regione Asia Pacifico (Mumbai)	17:30 - 01:30 UTC
cn-northwest-1	Regione Cina (Ningxia)	23:00 - 07:00 UTC
cn-north-1	Regione Cina (Pechino)	14:00 - 22:00 UTC
eu-central-1	Regione Europa (Francoforte)	23:00 - 07:00 UTC
eu-north-1	Regione Europa (Stoccolma)	01:00 - 09:00 UTC
eu-south-2	Regione Europa (Spagna)	21:00 - 05:00 UTC
eu-west-1	Europa (Irlanda)	22:00 - 06:00 UTC
eu-west-2	Regione Europa (Londra)	23:00 - 07:00 UTC
eu-west-3	Regione Europa (Parigi)	23:00 - 07:00 UTC
sa-east-1	Regione Sud America (San Paolo)	01:00 - 09:00 UTC
us-east-1	Stati Uniti orientali (Virginia settentrionale)	03:00 - 11:00 UTC
us-east-2	Stati Uniti orientali (Ohio)	23:00 - 07:00 UTC

Codice regione	Nome della Regione	Finestra di manutenzione
us-west-1	Regione Stati Uniti occidentali (California settentrionale)	06:00 - 14:00 UTC
us-west-2	Stati Uniti occidentali (Oregon)	06:00 - 14:00 UTC

Creazione di un cluster DAX

In questa sezione viene illustrato come configurare e utilizzare Amazon DynamoDB Accelerator (DAX) nell'ambiente Amazon Virtual Private Cloud (Amazon VPC) predefinito. È possibile creare il primo cluster DAX utilizzando () o. AWS Command Line Interface AWS CLI Console di gestione AWS

Dopo aver creato il cluster DAX, puoi accedervi da un' EC2 istanza Amazon in esecuzione nello stesso VPC. Puoi quindi utilizzare il cluster DAX con un programma applicativo. Per ulteriori informazioni, consulta Sviluppo con il client DynamoDB Accelerator (DAX).

Argomenti

- Creazione di un ruolo di servizio IAM per DAX per l'accesso a DynamoDB
- DAX e IPv6
- Creazione di un cluster DAX utilizzando AWS CLI
- Creazione di un cluster DAX mediante la Console di gestione AWS

Creazione di un ruolo di servizio IAM per DAX per l'accesso a DynamoDB

Affinché il cluster DAX possa accedere alle tabelle DynamoDB per conto tuo, è necessario creare un ruolo del servizio. Un ruolo di servizio è un ruolo AWS Identity and Access Management (IAM) che autorizza un AWS servizio ad agire per tuo conto. Il ruolo del servizio consente a DAX di accedere alle tabelle DynamoDB come se l'accesso venisse effettuato da te. Il ruolo del servizio deve essere creato prima del cluster DAX.

Se si utilizza la console, il flusso di lavoro di creazione di un cluster verifica la presenza di un ruolo del servizio DAX preesistente. Se non viene trovato, la console crea automaticamente un nuovo ruolo. Per ulteriori informazioni, consulta the section called “Fase 2: creazione di un cluster DAX”.

Se si utilizza il AWS CLI, è necessario specificare un ruolo di servizio DAX creato in precedenza. Oppure, devi crearne uno nuovo prima di procedere. Per ulteriori informazioni, consulta Fase 1: Creare un ruolo di servizio IAM per DAX per accedere a DynamoDB utilizzando AWS CLI.

Autorizzazioni necessarie per creare un ruolo di servizio

La policy `AdministratorAccess` gestita da AWS fornisce tutte le autorizzazioni necessarie per creare un cluster DAX e un ruolo del servizio. Se l'utente dispone di `AdministratorAccess` collegato, non sono richieste altre operazioni.

In caso contrario, sarà necessario aggiungere le autorizzazioni seguenti alla policy IAM, in modo che l'utente possa creare il ruolo di servizio:

- `iam:CreateRole`
- `iam:CreatePolicy`
- `iam:AttachRolePolicy`
- `iam:PassRole`

Collega queste autorizzazioni all'utente che sta cercando di eseguire l'operazione.

> **Note**
>
> Le `iam:PassRole` autorizzazioni `iam:CreateRoleiam:CreatePolicy`,`iam:AttachRolePolicy`, e non sono incluse nelle politiche AWS gestite per DynamoDB. Ciò è intenzionale, perché queste autorizzazioni forniscono la possibilità di un'escalation di privilegi. Ovvero, un utente potrebbe servirsi di queste autorizzazioni per creare una nuova policy di amministratore e collegare tale policy a un ruolo esistente. Per questo motivo, in qualità di amministratore del cluster DAX, è necessario aggiungere esplicitamente queste autorizzazioni alla policy.

Risoluzione dei problemi

Se nella tua policy utente mancano le autorizzazioni `iam:CreateRole`, `iam:CreatePolicy` e `iam:AttachPolicy`, riceverai dei messaggi di errore. La tabella seguente elenca questi messaggi e descrive come correggere i problemi.

Se vedi questo messaggio di errore...	Esegui questa operazione:
`User: arn:aws:iam:: accountID :user/userName is not authorize d to perform: iam:CreateRole on resource: arn:aws:iam:: accountID :role/service-role/ roleName`	Aggiungi `iam:CreateRole` alla tua policy utente.
`User: arn:aws:iam:: accountID :user/userName is not authorize d to perform: iam:CreatePolicy on resource: policy policyName`	Aggiungi `iam:CreatePolicy` alla tua policy utente.
`User: arn:aws:iam:: accountID :user/userName is not authorized to perform: iam:AttachRolePolicy on resource: role daxServiceRole`	Aggiungi `iam:AttachRolePolicy` alla tua policy utente.

Per ulteriori informazioni sulle policy IAM richieste per l'amministrazione dei cluster DAX, consulta Controllo degli accessi DAX.

DAX e IPv6

DynamoDB DAX ora IPv6 supporta l'indirizzamento, che consente di creare cluster che operano IPv4 in modalità di rete -only IPv6, -only o dual-stack. Questo aiuta a migliorare le funzionalità di rete per soddisfare i requisiti dell'infrastruttura in evoluzione.

Tipi di rete:

È possibile creare cluster con i seguenti tipi di rete:

- IPv4-solo
- IPv6-solo
- Dual-stack (supporta entrambi e) IPv4 IPv6

Caratteristiche principali:

Con IPv6 il supporto, puoi fare quanto segue:

- Opzioni di configurazione della rete:
  - IPv4-only e dual-stack cluster attivi. `dual_stack subnets`
  - IPv6-only cluster su sottoreti -only. IPv6
- Gestione dei gruppi di sottoreti:
  - Crea gruppi di sottoreti con supporto -only, IPv4 IPv6 -only o dual-stack
  - Modifica i gruppi di sottoreti esistenti con sottoreti VPC aggiuntive
  - Aggiungi sottoreti IPv6 -only ai gruppi di sottoreti -configuration IPv6
  - Aggiungi IPv4 o dual-stack sottoreti a gruppi configurati in dual-stack IPv4
- Configurazione del client:
  - Quando si effettuano chiamate sul piano dati, è possibile impostare il protocollo IP preferito per i cluster dual_stack utilizzando:
    - `ip_discovery`parametro in Python SDK
    - `ipDiscovery`parametro in altro SDKs
  - Predefinito: IPv4 quando la preferenza del protocollo non è specificata

Prima dell'implementazione IPv6 nei cluster DAX, è necessario considerare quanto segue:

- Il tipo di rete non può essere modificato dopo la creazione del cluster
- Per i cluster dual-stack, il `ip_discovery/ipDiscovery` parametro nella configurazione del client determina il protocollo IP da utilizzare (o) IPv4 IPv6
- Applicazioni diverse possono connettersi allo stesso cluster dual-stack utilizzando protocolli IP diversi in base alla loro configurazione

Example Esempio di configurazione del client

```
DynamoDbAsyncClient client = ClusterDaxAsyncClient.builder()
        .overrideConfiguration(Configuration.builder()
            .url(endpoint)             // DAX cluster endpoint
            .ipDiscovery(ipDiscovery)       // IP discovery type (IPv4 or IPv6)
            .build())
        .build();
```

Important

Quando si utilizzano policy IAM basate su risorse per limitare gli indirizzi IP per le tabelle DynamoDB in ambienti IPv6 -only con DAX, è necessario creare un'eccezione per il ruolo IAM del cluster DAX se si blocca lo spazio degli indirizzi (). IPv4 `0.0.0.0/0` Aggiungi una `ArnNotEquals` condizione alla tua policy che consenta specificamente l'accesso al ruolo IAM del cluster DAX mantenendo le restrizioni basate su IP per altri percorsi di accesso. Senza questa eccezione, DAX non può accedere alla tabella DynamoDB.
Ad esempio:

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"
        }
      }
    }
  ]
}
```

Creazione di un cluster DAX utilizzando AWS CLI

In questa sezione viene spiegato come creare un cluster Amazon DynamoDB Accelerator (DAX) utilizzando la AWS Command Line Interface (AWS CLI). Se non l'hai ancora fatto, installa e configura AWS CLI. Per le istruzioni, consulta i seguenti argomenti nella Guida per l'utente di AWS Command Line Interface :

- Installazione di AWS CLI
- Configurazione della AWS CLI

> **Important**
>
> Per gestire i cluster DAX utilizzando AWS CLI, installa o esegui l'aggiornamento alla versione 1.11.110 o successiva.

Tutti gli AWS CLI esempi utilizzano la `us-west-2` regione e l'account fittizio. IDs

Argomenti

- Fase 1: Creare un ruolo di servizio IAM per DAX per accedere a DynamoDB utilizzando AWS CLI
- Fase 2: creazione di un gruppo di sottoreti
- Fase 3: Creare un cluster DAX utilizzando AWS CLI
- Passaggio 4: configura le regole in entrata dei gruppi di sicurezza utilizzando AWS CLI

Fase 1: Creare un ruolo di servizio IAM per DAX per accedere a DynamoDB utilizzando AWS CLI

Prima di poter creare un cluster Amazon DynamoDB Accelerator (DAX), è necessario creare un ruolo del servizio per lo stesso. Un ruolo di servizio è un ruolo AWS Identity and Access Management (IAM) che autorizza un AWS servizio ad agire per tuo conto. Il ruolo del servizio consente a DAX di accedere alle tabelle DynamoDB come se l'accesso venisse effettuato da te.

In questa fase, viene creata una policy IAM e quindi questa policy viene collegata a un ruolo IAM. Puoi quindi assegnare il ruolo a un cluster DAX in modo che possa eseguire le operazioni DynamoDB per conto tuo.

Per creare un ruolo del servizio IAM per DAX

1. Crea un file denominato `service-trust-relationship.json` con i seguenti contenuti.

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
       {
            "Effect": "Allow",
            "Principal": {
                "Service": "dax.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

2. Crea il ruolo del servizio.

```
aws iam create-role \
    --role-name DAXServiceRoleForDynamoDBAccess \
    --assume-role-policy-document file://service-trust-relationship.json
```

3. Crea un file denominato `service-role-policy.json` con i seguenti contenuti.

   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:*"
            ]
        }
    ]
}
```

*accountID*Sostituiscilo con l'ID AWS del tuo account. Per trovare l'ID del tuo AWS account, scegli il tuo ID di accesso nell'angolo in alto a destra della console. L'ID AWS del tuo account viene visualizzato nel menu a discesa.

Nell'Amazon Resource Name (ARN) dell'esempio, *accountID* deve essere un numero di 12 cifre. Non usare trattini o altra punteggiatura.

4. Crea una policy IAM per il ruolo del servizio.

```
aws iam create-policy \
    --policy-name DAXServicePolicyForDynamoDBAccess \
    --policy-document file://service-role-policy.json
```

Nell'output, prendi nota dell'ARN per la policy creata, come l'esempio seguente.

```
arn:aws:iam::123456789012:policy/DAXServicePolicyForDynamoDBAccess
```

5. Collega la policy al ruolo del servizio. Sostituire *arn* il codice seguente con il ruolo ARN effettivo del passaggio precedente.

```
aws iam attach-role-policy \
    --role-name DAXServiceRoleForDynamoDBAccess \
    --policy-arn arn
```

Quindi, specifica un gruppo di sottoreti per il VPC predefinito. Un gruppo di sottoreti è una raccolta di una o più sottoreti all'interno del VPC. Per informazioni, consulta Fase 2: creazione di un gruppo di sottoreti.

Fase 2: creazione di un gruppo di sottoreti

Segui questa procedura per creare un gruppo di sottoreti per il tuo cluster Amazon DynamoDB Accelerator (DAX) utilizzando (). AWS Command Line Interface AWS CLI

> **Note**
>
> Se hai già creato un gruppo di sottoreti per il VPC predefinito, questa fase può essere ignorata.

DAX è progettato per funzionare all'interno di un ambiente Amazon Virtual Private Cloud (Amazon VPC). Se hai creato il tuo account AWS dopo il 4 dicembre 2013, disponi già di un VPC predefinito in ogni regione AWS . Per ulteriori informazioni, consulta VPC predefinito e sottoreti predefinite nella Guida per l'utente di Amazon VPC.

Come creare un gruppo di sottoreti

1. Per determinare l'identificatore per il tuo VPC predefinito, immetti il seguente comando.

   ```
   aws ec2 describe-vpcs
   ```

   Nell'output, prendi nota dell'identificatore per il VPC predefinito, come nell'esempio seguente.

   ```
   vpc-12345678
   ```

2. Determina la sottorete IDs associata al VPC predefinito. *vpcID*Sostituiscilo con il tuo ID VPC effettivo, ad esempio. `vpc-12345678`

   ```
   aws ec2 describe-subnets \
       --filters "Name=vpc-id,Values=vpcID" \
       --query "Subnets[*].SubnetId"
   ```

   Nell'output prendere nota degli identificatori delle sottoreti, ad esempio `subnet-11111111`.

3. Crea il gruppo di sottoreti. Assicurati di specificare almeno un ID sottorete nel parametro `--subnet-ids`.

   ```
   aws dax create-subnet-group \
       --subnet-group-name my-subnet-group \
   ```

```
    --subnet-ids subnet-11111111 subnet-22222222 subnet-33333333 subnet-44444444
```

Per creare il cluster, consulta Fase 3: Creare un cluster DAX utilizzando AWS CLI.

Fase 3: Creare un cluster DAX utilizzando AWS CLI

Segui questa procedura per utilizzare AWS Command Line Interface (AWS CLI) per creare un cluster Amazon DynamoDB Accelerator (DAX) nel tuo Amazon VPC predefinito.

Come creare un cluster DAX

1. Ottieni l'Amazon Resource Name (ARN) per il ruolo del servizio.

   ```
   aws iam get-role \
       --role-name DAXServiceRoleForDynamoDBAccess \
       --query "Role.Arn" --output text
   ```

   Nell'output, prendi nota dell'ARN del ruolo del servizio, come nell'esempio seguente.

   ```
   arn:aws:iam::123456789012:role/DAXServiceRoleForDynamoDBAccess
   ```

2. Crea il cluster DAX. Sostituisci *roleARN* con l'ARN ottenuto nella fase precedente.

   ```
   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
   ```

   Tutti i nodi nel cluster sono di tipo `dax.r4.large` (`--node-type`). Sono presenti sono tre nodi (`--replication-factor`), un nodo primario e due repliche.

   > **Note**
   >
   > Poiché `sudo` e `grep` sono parole chiave riservate, non è possibile creare un cluster DAX con queste parole nel nome. Ad esempio, `sudo` e `sudocluster` sono nomi cluster non validi.

Per visualizzare lo stato del cluster, immetti il seguente comando.

```
aws dax describe-clusters
```

Lo stato viene visualizzato nell'output, ad esempio `"Status": "creating"`.

> **Note**
>
> La creazione del cluster richiede diversi minuti. Quando il cluster è pronto, il suo stato diventa `available`. Nel frattempo, passa a Passaggio 4: configura le regole in entrata dei gruppi di sicurezza utilizzando AWS CLI e segui le istruzioni indicate.

Passaggio 4: configura le regole in entrata dei gruppi di sicurezza utilizzando AWS CLI

I nodi nel cluster Amazon DynamoDB Accelerator (DAX) utilizzano il gruppo di sicurezza predefinito per Amazon VPC. Per il gruppo di sicurezza predefinito, è necessario autorizzare il traffico in entrata sulla porta TCP 8111 per i cluster non crittografati o la porta 9111 per i cluster crittografati. Ciò consente alle EC2 istanze Amazon nel tuo Amazon VPC di accedere al tuo cluster DAX.

> **Note**
>
> Se il cluster DAX è stato avviato con un altro gruppo di sicurezza (diverso da `default`), è necessario eseguire questa procedura per quel gruppo.

Per configurare le regole in entrata del gruppo di sicurezza

1. Per determinare l'identificatore del gruppo di sicurezza predefinito, immetti il seguente comando: Sostituisci *`vpcID`* con il tuo ID VPC effettivo (da Fase 2: creazione di un gruppo di sottoreti).

   ```
   aws ec2 describe-security-groups \
       --filters Name=vpc-id,Values=vpcID Name=group-name,Values=default \
       --query "SecurityGroups[*].{GroupName:GroupName,GroupId:GroupId}"
   ```

   Nell'output prendere nota dell'identificatore del gruppo di sicurezza, ad esempio `sg-01234567`.
2. Quindi immetti i seguenti dati. Sostituisci *`sgID`* con l'identificatore del gruppo di sicurezza effettivo. Usare la porta 8111 per cluster non crittografati e 9111 per cluster crittografati.

```
aws ec2 authorize-security-group-ingress \
    --group-id sgID --protocol tcp --port 8111
```

Creazione di un cluster DAX mediante la Console di gestione AWS

In questa sezione viene spiegato come creare un cluster Amazon DynamoDB Accelerator (DAX) utilizzando la Console di gestione AWS.

Argomenti

- Passaggio 1: creare un gruppo di sottoreti utilizzando il Console di gestione AWS
- Fase 2: Creare un cluster DAX utilizzando il Console di gestione AWS
- Passaggio 3: Configurare le regole in entrata dei gruppi di sicurezza utilizzando il Console di gestione AWS

Passaggio 1: creare un gruppo di sottoreti utilizzando il Console di gestione AWS

Seguire questa procedura per creare un gruppo di sottoreti per il cluster Amazon DynamoDB Accelerator (DAX) utilizzando Console di gestione AWS.

> **Note**
>
> Se hai già creato un gruppo di sottoreti per il VPC predefinito, questa fase può essere ignorata.

DAX è progettato per funzionare all'interno di un ambiente Amazon Virtual Private Cloud (Amazon VPC). Se hai creato il tuo account AWS dopo il 4 dicembre 2013, disponi già di un VPC predefinito in ogni regione AWS . Per ulteriori informazioni, consulta VPC predefinito e sottoreti predefinite nella Guida per l'utente di Amazon VPC.

Come parte del processo di creazione di un cluster DAX, è necessario specificare un gruppo di sottoreti. Un gruppo di sottoreti è una raccolta di una o più sottoreti all'interno del VPC. Quando si crea il cluster DAX, i nodi vengono distribuiti alle sottoreti all'interno del gruppo di sottoreti.

> **Note**
>
> Il VPC con questo cluster DAX può contenere altre risorse e persino endpoint VPC per gli altri servizi tranne l'endpoint VPC per ElastiCache e può causare errori nelle operazioni del cluster DAX.

Come creare un gruppo di sottoreti

1. Apri la console DynamoDB all'indirizzo. https://console.aws.amazon.com/dynamodb/
2. Nel pannello di navigazione, in DAX scegli Subnet groups (Gruppi di sottoreti).
3. Scegli Crea gruppo di sottoreti.
4. Nella finestra Create subnet group (Crea gruppo di sottoreti) eseguire le operazioni seguenti:

    a. Nome: immettere un nome breve per il gruppo di sottoreti.
    b. Descrizione: immettere una descrizione del gruppo di sottoreti.
    c. ID VPC: scegli l'identificatore per il tuo ambiente Amazon VPC.
    d. Sottoreti: scegli una o più sottoreti dall'elenco.

    > **Note**
    >
    > Le sottoreti sono distribuite tra più zone di disponibilità. Se intendi creare un cluster DAX multinodo (un nodo primario e una o più repliche di lettura), ti consigliamo di scegliere più sottoreti. IDs Quindi DAX può distribuire i nodi del cluster in più zone di disponibilità. Se una zona di disponibilità non è più disponibile, DAX esegue automaticamente il failover verso una zona di disponibilità attiva. Il cluster DAX continuerà a funzionare senza interruzioni.

    Quando le impostazioni sono quelle desiderate, scegli Crea gruppo di sottoreti.

Per creare il cluster, consulta Fase 2: Creare un cluster DAX utilizzando il Console di gestione AWS.

Fase 2: Creare un cluster DAX utilizzando il Console di gestione AWS

Seguire questa procedura per creare un cluster Amazon DynamoDB Accelerator (DAX) nell'Amazon VPC predefinito.

Come creare un cluster DAX

1. Apri la console DynamoDB all'indirizzo. https://console.aws.amazon.com/dynamodb/
2. Nel riquadro di navigazione, in DAX, scegli Clusters (Cluster).
3. Scegli Crea cluster.
4. Nella finestra Create cluster (Crea cluster) procedere nel seguente modo:

   a. Nome cluster: immettere un nome breve per il cluster DAX.

   > **Note**
   >
   > Poiché `sudo` e `grep` sono parole chiave riservate, non è possibile creare un cluster DAX con queste parole nel nome. Ad esempio, `sudo` e `sudocluster` sono nomi cluster non validi.

   b. Descrizione cluster: immettere una descrizione per il cluster.

   c. Node types (Tipi di nodi): scegli il tipo di nodo per tutti i nodi del cluster.

   d. Dimensione del cluster: scegli il numero di nodi nel cluster. Un cluster è costituito da un nodo primario e fino a nove repliche di lettura.

   > **Note**
   >
   > Se intendi creare un cluster a nodo singolo, seleziona 1. Il cluster sarà costituito da un nodo primario.
   > Se desideri creare un cluster a più nodi, scegli un numero tra 3 (un nodo primario e due repliche di lettura) e 10 (un nodo primario e nove repliche di lettura).
   >
   > > **Important**
   > >
   > > Per l'uso in produzione, consigliamo vivamente di utilizzare DAX con almeno tre nodi, in cui ogni nodo è collocato in una zone di disponibilità diversa. Per fare in modo che un cluster DAX sia tollerante ai guasti sono richiesti tre nodi.
   > > Un cluster DAX può essere distribuito con uno o due nodi per lo sviluppo o il test di carichi di lavoro. I cluster con uno o due nodi non sono tolleranti ai guasti e non consigliamo di utilizzare meno di tre nodi in produzione. Se un

cluster con uno o due nodi rileva errori software o hardware, il cluster può diventare non disponibile o perdere i dati memorizzati nella cache.

e. Scegli Next (Successivo).

f. Subnet group (Gruppo di sottoreti): scegli Choose existing (Scegli esistente) e scegli il gruppo di sottoreti che hai creato in Passaggio 1: creare un gruppo di sottoreti utilizzando il Console di gestione AWS.

g. Access control (Controllo degli accessi): scegli il gruppo di sicurezza predefinito.

h. Availability Zones AZ (Zone di disponibilità): scegli Automatic (Automatiche).

i. Scegli Next (Successivo).

j. Ruolo del servizio IAM per l'accesso a DynamoDB: scegli Crea nuovo e specifica le informazioni seguenti:

- Nome ruolo IAM: immettere un nome per un ruolo IAM, ad esempio, `DAXServiceRole`. La console crea un nuovo ruolo IAM, che verrà assunto dal cluster DAX al runtime.
- Seleziona la casella accanto a Create policy (Crea policy).
- Policy del ruolo IAM: scegli Lettura/scrittura. Questo consente al cluster DAX di eseguire operazioni di lettura e scrittura in DynamoDB.
- Nuovo nome della policy IAM: questo campo verrà compilato quando si immette il nome del ruolo IAM. Puoi anche inserire un nome per una policy IAM, ad esempio, `DAXServicePolicy`. La console crea una nuova policy IAM e collega la policy al ruolo IAM.
- Accesso alle tabelle DynamoDB: scegli Tutte le tabelle.

k. Crittografia: scegli Abilita la crittografia a riposo e Abilita la crittografia in transito. Per ulteriori informazioni, consulta Crittografia DAX dei dati inattivi e Crittografia DAX in transito.

È inoltre richiesto un ruolo di servizio separato per consentire a DAX di accedere ad Amazon EC2 . DAX crea automaticamente questo ruolo del servizio per l'utente. Per ulteriori informazioni, consulta Utilizzo di ruoli collegati ai servizi per DAX.

5. Dopo aver selezionato le impostazioni desiderate, scegliere Avanti.

6. Gruppo di parametri: scegli Scegli esistente.

7. Finestra di manutenzione: scegli Nessuna preferenza se non hai preferenze su quando vengono applicati gli aggiornamenti software oppure scegli Specifica la finestra temporale e fornisci le opzioni Giorno feriale, Ora e Inizia entro (ore) per pianificare la finestra di manutenzione.
8. Tag: scegli Aggiungi nuovo tag per inserirne una key/value coppia a scopo di etichettatura.
9. Scegli Next (Successivo).

Nella schermata Rivedi e crea, puoi rivedere tutte le impostazioni. Se sei pronto per creare il cluster, scegli Crea cluster.

Nella schermata Cluster, il cluster DAX verrà elencato con lo stato Creazione in corso.

> **Note**
>
> La creazione del cluster richiede diversi minuti. Quando il cluster è pronto, il suo stato cambia in Available (Disponibile).
> Nel frattempo, passa a Passaggio 3: Configurare le regole in entrata dei gruppi di sicurezza utilizzando il Console di gestione AWS e segui le istruzioni indicate.

Passaggio 3: Configurare le regole in entrata dei gruppi di sicurezza utilizzando il Console di gestione AWS

Il cluster Amazon DynamoDB Accelerator (DAX) comunica tramite la porta TCP 8111 (per cluster non crittografati) o 9111 (per cluster crittografati), pertanto sarà necessario autorizzare il traffico in entrata su quella porta. Ciò consente alle EC2 istanze Amazon nel tuo Amazon VPC di accedere al tuo cluster DAX.

> **Note**
>
> Se il cluster DAX è stato avviato con un altro gruppo di sicurezza (diverso da `default`), è necessario eseguire questa procedura per quel gruppo.

Per configurare le regole in entrata del gruppo di sicurezza

1. Apri la EC2 console Amazon all'indirizzo https://console.aws.amazon.com/ec2/.
2. Fai clic su Gruppi di sicurezza nel pannello di navigazione.

3. Scegli il gruppo di sicurezza di default (predefinito). Nel menu Actions (Operazioni) scegli Edit inbound rules (Modifica regole in entrata).
4. Scegli Add Rule (Aggiungi regola) quindi immettere le informazioni seguenti:
   - Intervallo porte: immettere 8111 (se il cluster non è crittografato) o 9111 (se il cluster è crittografato).
   - Fonte: lascia come Personalizzato e scegli il campo di ricerca a destra. Verrà visualizzato un menu a discesa. Puoi selezionare l'identificatore per il gruppo di sicurezza predefinito.
5. Scegli Salva regole per salvare le modifiche.
6. Per aggiornare il nome nella console, vai alla proprietà Nome e scegli l'opzione Modifica.

Modelli di consistenza DAX e DynamoDB

Amazon DynamoDB Accelerator (DAX) è un servizio di memorizzazione nella cache write-through progettato per semplificare il processo di aggiunta di una cache alle tabelle DynamoDB. Poiché DAX opera separatamente da DynamoDB, è importante comprendere i modelli di consistenza sia di DAX che di DynamoDB in modo da assicurarsi che le applicazioni si comportino come previsto.

In molti casi d'uso, il modo in cui l'applicazione utilizza DAX influenza la coerenza dei dati all'interno del cluster DAX e quella dei dati tra DAX e DynamoDB.

Argomenti

- Consistenza tra i nodi del cluster DAX
- Comportamento della cache di elementi DAX
- Comportamento della cache di query DAX
- Letture fortemente consistenti e transazionali
- Caching negativo
- Strategie di scrittura

Consistenza tra i nodi del cluster DAX

Per ottenere un'alta disponibilità dell'applicazione, è preferibile effettuare il provisioning del cluster DAX con almeno tre nodi e di posizionare tali nodi in più zone di disponibilità all'interno di una regione.

Durante l'esecuzione, il cluster DAX replica i dati tra tutti i nodi del cluster (supponendo che sia stato effettuato il provisioning di più di un nodo). Si consideri un'applicazione che esegue un'operazione `UpdateItem` riuscita tramite DAX. Tale operazione determina la modifica della cache degli elementi del nodo primario con il nuovo valore, il quale viene quindi replicato in tutti gli altri nodi nel cluster. Questa replica è di tipo consistente finale e richiede in genere meno di un secondo per il completamento.

In questo scenario, è possibile che due client leggano la stessa chiave dallo stesso cluster DAX ma ricevano valori diversi, a seconda del nodo a cui ciascun client accede. I nodi sono tutti consistenti dopo che l'aggiornamento è stato completamente replicato in tutti i nodi del cluster. Questo comportamento è simile alla natura di consistenza finale di DynamoDB.

Se si sta creando un'applicazione che utilizza DAX, questa deve essere progettata in modo da poter tollerare dati a consistenza finale.

Comportamento della cache di elementi DAX

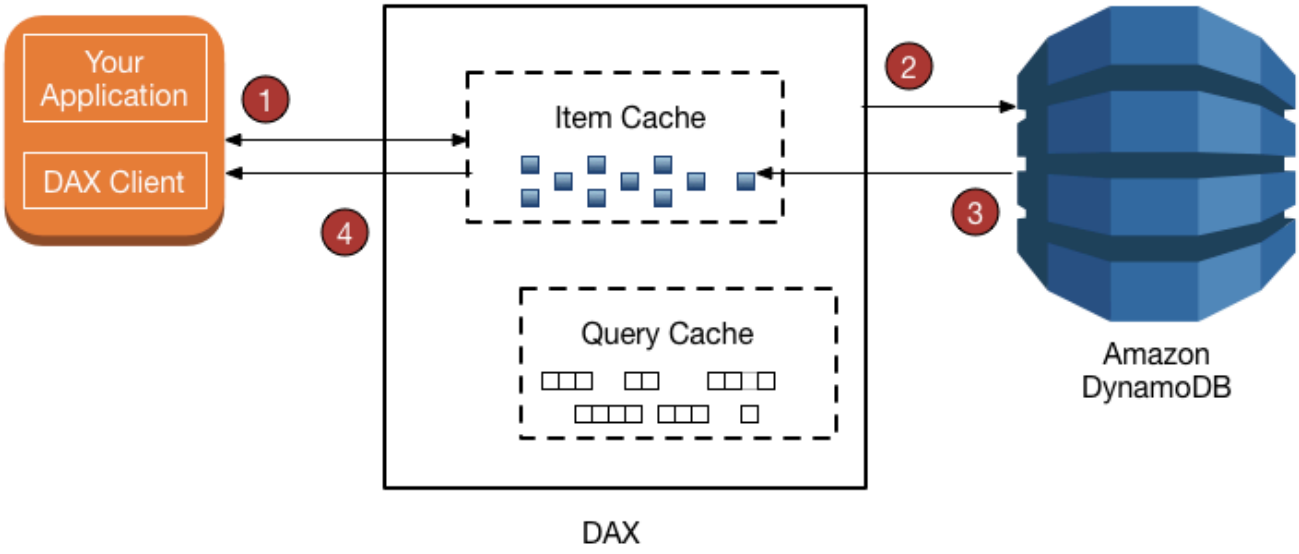
Ogni cluster DAX dispone di due cache distinte: una cache di elementi e una cache di query. Per ulteriori informazioni, consulta DAX: come funziona.

In questa sezione vengono descritte le implicazioni della coerenza relative alla lettura e alla scrittura nella cache di elementi DAX.

Consistenza delle letture

Con DynamoDB, l'operazione `GetItem` esegue una lettura a consistenza finale per impostazione predefinita. Si supponga di utilizzare `UpdateItem` con il client DynamoDB. Se tenti di leggere lo stesso item subito dopo, è possibile che i dati visualizzati siano quelli relativi a prima dell'aggiornamento. Ciò è dovuto al ritardo di propagazione all'interno di tutte le posizioni di archiviazione di DynamoDB. La consistenza si ottiene generalmente in pochi secondi. è probabile che l'item aggiornato venga visualizzato dopo un nuovo tentativo di lettura.

Quando si utilizza `GetItem` con il client DAX, l'operazione (in questo caso, una lettura a consistenza finale) procede come mostrato di seguito.



1. Il client DAX emette una richiesta `GetItem`. DAX cerca di leggere l'elemento richiesto dalla cache di elementi. Se l'elemento si trova nella cache (hit della cache), DAX lo restituisce all'applicazione.
2. Se l'elemento non è disponibile (mancato riscontro nella cache), DAX esegue un'operazione `GetItem` a consistenza finale in DynamoDB.
3. DynamoDB restituisce l'elemento richiesto e DAX lo archivia nella cache di elementi.
4. DAX restituisce l'elemento all'applicazione.
5. (Non visualizzato) Se il cluster DAX contiene più di un nodo, l'elemento viene replicato in tutti gli altri nodi del cluster.

Il nuovo elemento rimane nella cache di elementi DAX, soggetto all'impostazione durata (TTL, Time to Live) e all'algoritmo utilizzato meno di recente (LRU) della cache. Per ulteriori informazioni, consulta DAX: come funziona.

Tuttavia, durante questo periodo, DAX non legge nuovamente l'elemento da DynamoDB. Se un altro utente aggiorna l'elemento tramite un client DynamoDB, ignorando del tutto DAX, una richiesta `GetItem` che utilizza il client DAX genererà risultati diversi rispetto alla stessa richiesta `GetItem` che utilizza il client DynamoDB. In questo scenario, DAX e DynamoDB contengono valori incoerenti per la stessa chiave finché il TTL dell'elemento DAX non scade.

Se un'applicazione modifica i dati all'interno di una tabella DynamoDB sottostante, ignorando DAX, è necessario anticipare e tollerare le incoerenze che potrebbero verificarsi nei dati.

> **Note**
>
> Oltre a `GetItem`, il client DAX supporta anche le richieste `BatchGetItem`. `BatchGetItem` è essenzialmente un wrapper intorno a una o più richieste `GetItem`, pertanto DAX tratta ognuna di queste come una singola operazione `GetItem`.

Consistenza delle scritture

DAX è una cache write-through che semplifica il processo di mantenimento della coerenza della cache di elementi DAX con le tabelle DynamoDB sottostanti.

Il client DAX supporta le stesse operazioni API di scrittura di DynamoDB (`PutItem`, `UpdateItem` `DeleteItem`, `BatchWriteItem` e `TransactWriteItems`). Quando si utilizzano tali operazioni con il client DAX, gli elementi vengono modificati sia in DAX che in DynamoDB. DAX aggiorna gli elementi nella cache degli elementi, a prescindere dal valore TTL degli elementi.

Ad esempio, si assuma di aver emesso una richiesta `GetItem` dal client DAX per leggere un elemento dalla tabella `ProductCatalog`. La chiave di partizione è `Id`; non è presente una chiave di ordinamento. Recupera l'item il cui `Id` è `101`. Il valore `QuantityOnHand` per quell'elemento è `42`. DAX archivia l'elemento nella sua cache di elementi con un TTL specifico. Supponiamo che il TTL di questo esempio sia 10 minuti. Dopo tre minuti, un'altra applicazione utilizza il client DAX per aggiornare lo stesso elemento in modo che il suo valore `QuantityOnHand` ora sia `41`. Ipotizzando che l'item non venga nuovamente aggiornato, tutte le letture successive dello stesso item nei 10 minuti successivi restituiscono il valore memorizzato nella cache di `QuantityOnHand` (`41`).

Come DAX elabora le scritture

DAX è destinato all'uso in applicazioni che richiedono letture ad alte prestazioni. Come cache write-through, DAX passa le scritture a DynamoDB in modo sincrono, quindi replica automaticamente e in maniera asincrona gli aggiornamenti risultanti nella cache degli elementi in tutti i nodi del cluster. Non è necessario gestire la logica dell'invalidamento della cache perché verrà gestita automaticamente da DAX.

DAX supporta le operazioni di scrittura seguenti: `PutItem`, `UpdateItem`, `DeleteItem`, `BatchWriteItem` e `TransactWriteItems`.

Quando si invia una richiesta `PutItem`, `UpdateItem`, `DeleteItem` o `BatchWriteItem` a DAX, si verifica quanto segue:

- DAX invia la richiesta a DynamoDB.
- DynamoDB risponde a DAX, confermando l'esito positivo della scrittura.
- DAX scrive l'elemento nella sua cache di elementi.
- DAX riporta al richiedente che l'operazione è stata eseguita correttamente.

Quando si invia una richiesta `TransactWriteItems` a DAX, si verifica quanto segue:

- DAX invia la richiesta a DynamoDB.
- DynamoDB risponde a DAX, confermando che la transazione è stata completata.
- DAX riporta al richiedente che l'operazione è stata eseguita correttamente.
- In background, DAX effettua una richiesta `TransactGetItems` per ogni elemento nella richiesta `TransactWriteItems` per archiviare l'elemento nella cache di elementi. `TransactGetItems` viene usato per garantire un isolamento serializzabile.

Se una scrittura su DynamoDB non va a buon fine per un qualsiasi motivo, inclusa la limitazione, l'elemento non viene memorizzato nella cache in DAX. e il richiedente riceve un'eccezione relativa all'errore. In questo modo, i dati non vengono scritti nella cache DAX a meno che non siano stati scritti prima correttamente in DynamoDB.

> **Note**
>
> Ogni scrittura in DAX altera lo stato della cache di elementi. Tuttavia, le scritture nella cache degli elementi non influiscono sulla cache delle query. La cache di elementi e la cache di query DAX hanno scopi diversi e funzionano in maniera indipendente l'una dall'altra.

Comportamento della cache di query DAX

DAX memorizza nella cache i risultati delle richieste `Query` e `Scan` nella propria cache di query. Tuttavia, questi risultati non influiscono sulla cache degli elementi. Quando l'applicazione emette una richiesta `Query` o `Scan` con DAX, il set di risultati viene salvato nella cache di query, non nella cache di elementi. Non puoi preparare la cache degli elementi eseguendo un'operazione `Scan` perché la cache degli elementi e la cache delle query sono entità separate.

Consistenza di query-aggiornamento-query

Gli aggiornamenti della cache di elementi o della tabella DynamoDB sottostante non invalidano né modificano i risultati memorizzati nella cache di query.

A titolo illustrativo, considera il seguente scenario in cui un'applicazione utilizza la tabella `DocumentRevisions`, la cui chiave di partizione è `DocId` e la cui chiave di ordinamento è `RevisionNumber`.

1. Un client emette un `Query` per `DocId` `101`, per tutti gli elementi con `RevisionNumber` maggiore o uguale a `5`. DAX archivia il set di risultati nella cache di query e lo restituisce all'utente.
2. Il client emette una richiesta `PutItem` per `DocId` `101` con un valore `RevisionNumber` di `20`.
3. Il client emette la stessa richiesta `Query` descritta nella fase 1 (`DocId` `101` e `RevisionNumber` **>=** `5`).

In questo scenario, il set di risultati memorizzato nella cache per la richiesta `Query` emessa nella fase 3 è identico al set di risultati che è stato memorizzato nella cache nella fase 1. Il motivo è che DAX non invalida i set di risultati `Query` o `Scan` in base agli aggiornamenti dei singoli elementi. L'operazione `PutItem` della fase 2 viene riportata nella cache di query DAX solo quando il TTL per la `Query` scade.

L'applicazione deve considerare il valore TTL della cache delle query e per quanto tempo l'applicazione può tollerare risultati incoerenti tra la cache delle query e la cache degli elementi.

Letture fortemente consistenti e transazionali

Per eseguire una richiesta `GetItem`, `BatchGetItem`, `Query` o `Scan` fortemente consistente, impostare il parametro `ConsistentRead` su true. DAX passa le richieste di lettura fortemente consistente a DynamoDB. Quando riceve una risposta da DynamoDB, DAX restituisce i risultati al client senza memorizzarli nella cache. DAX non è in grado di servire direttamente le letture fortemente consistenti in quanto non è strettamente accoppiato con DynamoDB. Per questo motivo, le eventuali letture successive di DAX devono essere letture a consistenza finale. Inoltre, le eventuali letture fortemente consistenti successive devono essere trasferite a DynamoDB.

DAX gestisce le richieste `TransactGetItems` nello stesso modo in cui gestisce le letture fortemente consistenti. DAX passa tutte le richieste `TransactGetItems` a DynamoDB. Quando riceve una risposta da DynamoDB, DAX restituisce i risultati al client ma senza memorizzarli nella cache.

Caching negativo

DAX supporta voci negative sia nella cache di elementi sia nella cache di query. Una voce negativa nella cache si verifica quando DAX non è in grado di individuare gli elementi richiesti in una tabella DynamoDB sottostante. Anziché generare un errore, DAX memorizza nella cache un risultato vuoto e lo restituisce all'utente.

Ad esempio, si assuma che un'applicazione invii una richiesta `GetItem` a un cluster DAX e che nella cache di elementi DAX non esista un elemento corrispondente. Ciò fa sì che DAX legga l'elemento corrispondente dalla tabella DynamoDB sottostante. Se l'elemento non esiste in DynamoDB, DAX archivia un elemento vuoto nella sua cache di elementi e lo restituisce all'applicazione. Ora supponiamo che l'applicazione invii un'altra richiesta `GetItem` per lo stesso articolo. DAX individua l'elemento vuoto nella cache di elementi e lo restituisce immediatamente all'applicazione. Non consulta DynamoDB.

Una voce negativa nella cache rimane nella cache di elementi DAX fino a che il TTL dell'elemento scade, viene richiamato l'algoritmo LRU o l'elemento viene modificato tramite `PutItem`, `UpdateItem` o `DeleteItem`.

La cache di query DAX gestisce i risultati di cache negativi in modo simile. Se un'applicazione esegue un'operazione `Query` o `Scan` e la cache di query DAX non contiene un risultato memorizzato nella cache, DAX invia la richiesta a DynamoDB. Se non sono presenti elementi corrispondenti nel set di risultati, DAX archivia un set di risultati vuoto nella cache di query e lo restituisce all'applicazione. Le richieste `Query` o `Scan` successive generano lo stesso set di risultati (vuoto) finché il TTL per tali risultati non scade.

Strategie di scrittura

Il comportamento write-through di DAX è appropriato per molti modelli di applicazione. Tuttavia, il modello write-through potrebbe non essere appropriato per tutti i modelli di applicazione.

Nel caso di applicazioni sensibili alla latenza, la scrittura tramite DAX genera un hop di rete aggiuntivo. Di conseguenza, la scrittura in DAX è leggermente più lenta rispetto alla scrittura diretta in DynamoDB. Se l'applicazione è sensibile alla latenza di scrittura, è possibile ridurre la latenza scrivendo direttamente in DynamoDB. Per ulteriori informazioni, consulta Write-Around.

Nel caso di applicazioni a scrittura intensiva (come quelle che eseguono il caricamento in blocco dei dati), è consigliabile non scrivere tutti i dati tramite DAX perché solo una piccola percentuale di tali dati viene letta dall'applicazione. Quando scrivi grandi quantità di dati tramite DAX, è necessario

richiamare il relativo algoritmo LRU per fare spazio ai nuovi elementi da leggere nella cache. Ciò riduce l'efficacia di DAX come cache di lettura.

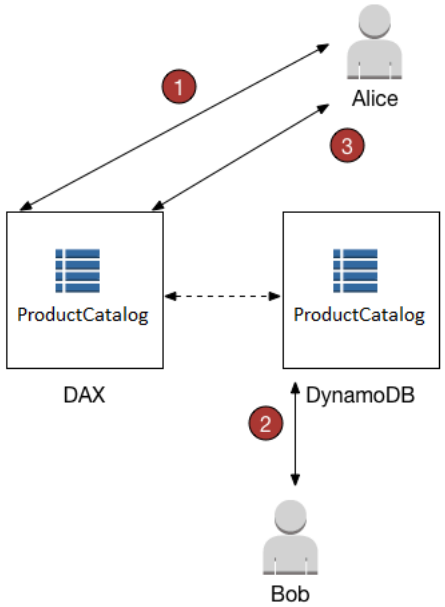
Quando si scrive un elemento in DAX, lo stato della cache di elementi viene modificato per ricevere il nuovo elemento. Ad esempio, per far spazio al nuovo elemento, DAX potrebbe dover eliminare i dati meno recenti dalla cache di elementi. Il nuovo item rimane nella cache degli elementi, soggetto all'algoritmo LRU della cache e all'impostazione TTL della cache. Finché l'elemento persiste nella cache di elementi, DAX non esegue una nuova lettura dell'elemento da DynamoDB.

Write-Through

La cache di elementi DAX implementa una policy write-through. Per ulteriori informazioni, consulta Come DAX elabora le scritture.

Quando si scrive un elemento, DAX si assicura che l'elemento memorizzato nella cache sia sincronizzato con l'elemento presente in DynamoDB. Questo è utile per le applicazioni che devono leggere nuovamente un item subito dopo averlo scritto. Tuttavia, se altre applicazioni scrivono direttamente in una tabella DynamoDB, l'elemento nella cache di elementi DAX non sarà più sincronizzato con DynamoDB.

A titolo illustrativo, considera due utenti (Alice e Bob) che utilizzano la tabella `ProductCatalog`. Alice accede alla tabella tramite DAX, Bob invece ignora DAX e accede alla tabella direttamente in DynamoDB.



1. Alice aggiorna un elemento nella tabella `ProductCatalog`. DAX inoltra la richiesta a DynamoDB e l'aggiornamento va a buon fine. DAX scrive quindi l'elemento nella sua cache di elementi e restituisce una risposta di operazione completata ad Alice. Da quel punto in poi, fino a quando l'elemento viene definitivamente rimosso dalla cache, tutti gli utenti che leggono l'elemento da DAX vedono l'elemento con l'aggiornamento di Alice.
2. Poco tempo dopo, Bob aggiorna lo stesso item `ProductCatalog` scritto da Alice. Tuttavia, Bob aggiorna l'elemento direttamente in DynamoDB. DAX non aggiorna automaticamente la sua cache di elementi in risposta agli aggiornamenti tramite DynamoDB. Di conseguenza, gli utenti di DAX non vedranno l'aggiornamento di Bob.
3. Alice legge nuovamente l'elemento da DAX. Poiché l'elemento si trova nella cache di elementi, DAX lo restituisce ad Alice senza accedere alla tabella DynamoDB.

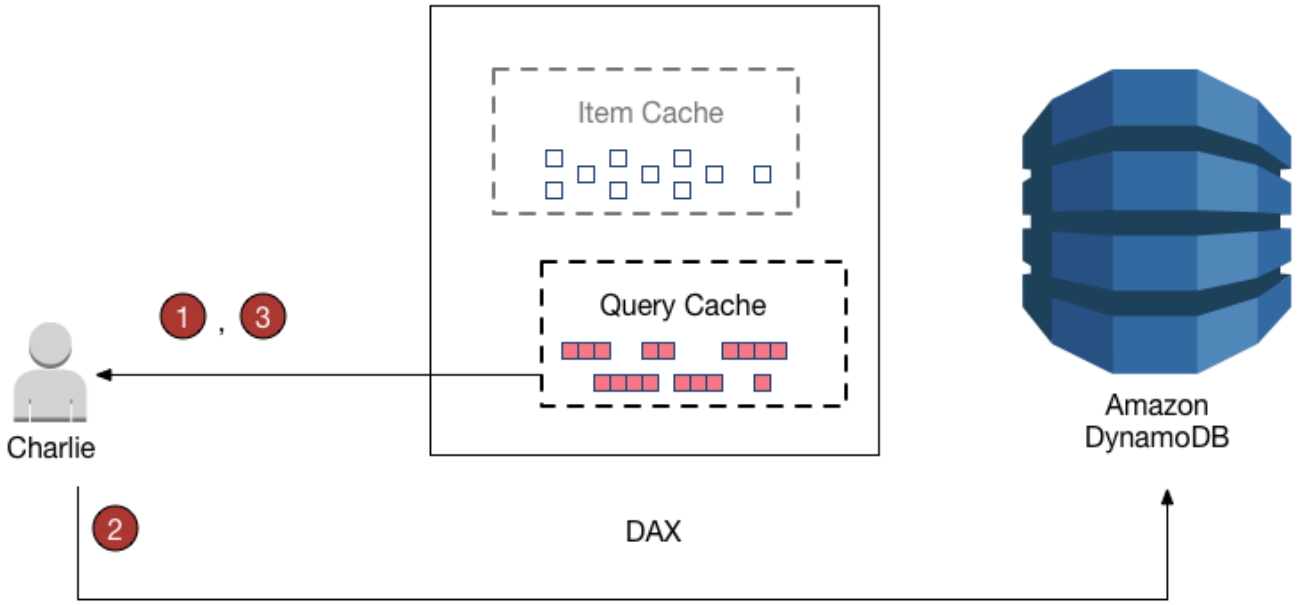
In questo scenario, Alice e Bob vedono rappresentazioni diverse dello stesso item `ProductCatalog`. Questa situazione persiste fino a quando DAX non rimuove l'elemento dalla cache di elementi o finché un altro utente aggiorna nuovamente lo stesso elemento tramite DAX.

Write-Around

Se l'applicazione deve scrivere grandi quantità di dati (come ad esempio, nel caso di un caricamento in blocco dei dati), potrebbe essere preferibile ignorare DAX e scrivere i dati direttamente in DynamoDB. Una strategia write-around (scrittura diretta) riduce la latenza di scrittura. Tuttavia, la cache di elementi non rimane sincronizzata con i dati in DynamoDB.

Se si decide di utilizzare una strategia write-around, tenere presente che DAX popola la sua cache di elementi ogni volta che le applicazioni utilizzano il client DAX per leggere i dati. In alcuni casi, ciò potrebbe rappresentare un vantaggio in quanto assicura che vengano memorizzati nella cache solo i dati letti più frequentemente (anziché i dati scritti più frequentemente).

Ad esempio, si consideri un utente (Charlie) che desidera utilizzare una tabella diversa, la tabella `GameScores`, tramite DAX. La chiave di partizione per `GameScores` è `UserId`, quindi tutti i punteggi di Charlie hanno lo stesso `UserId`.



1. Charlie intende recuperare tutti i suoi punteggi, pertanto invia una `Query` a DAX. Ipotizzando che questa query non sia stata emessa in precedenza, DAX la inoltra a DynamoDB per l'elaborazione, Memorizza i risultati nella cache di query DAX e infine restituisce i risultati a Charlie. Il set di risultati rimane disponibile nella cache delle query finché non viene rimosso.
2. Supponiamo ora che Charlie giochi a Meteor Blasters e ottenga un punteggio elevato. Charlie invia una richiesta `UpdateItem` a DynamoDB, modificando un elemento nella tabella `GameScores`.
3. Infine, Charlie decide di eseguire nuovamente la richiesta `Query` precedente per recuperare tutti i suoi dati da `GameScores`. Charlie non visualizza nei risultati il suo punteggio massimo di Meteor Blasters in quanto i risultati della query provengono dalla cache delle query, non dalla cache degli elementi. Poiché le due cache sono indipendenti l'una dall'altra, le modifiche apportate in una cache non influiscono sull'altra cache.

DAX non aggiorna i set di risultati nella cache di query con i dati più recenti di DynamoDB. Ogni set di risultati nella cache delle query è attuale al momento dell'esecuzione dell'operazione `Query` o `Scan`. Pertanto, i risultati della `Query` di Charlie non riflettono l'operazione `PutItem`. Questa situazione persiste fino a quando DAX rimuove il set di risultati dalla cache di query.

Sviluppo con il client DynamoDB Accelerator (DAX)

Per utilizzare DAX da un'applicazione è necessario utilizzare il client DAX relativo al linguaggio di programmazione in uso. Il client DAX è progettato in modo da non interrompere le applicazioni Amazon DynamoDB esistenti. Sono necessarie solo alcune semplici modifiche al codice.

> **Note**
>
> I client DAX per diversi linguaggi di programmazione sono disponibili sul seguente sito:
>
> - http://dax-sdk.s3 - website-us-west -2.amazonaws.com

Questa sezione mostra come avviare un' EC2 istanza Amazon nel tuo Amazon VPC predefinito, connettersi all'istanza ed eseguire un'applicazione di esempio. Fornisce inoltre informazioni su come modificare l'applicazione esistente in modo da poter utilizzare il cluster DAX.

Argomenti

- Tutorial: Esecuzione di un'applicazione di esempio utilizzando DynamoDB Accelerator (DAX)
- Modifica di un'applicazione esistente per l'utilizzo con DAX

Tutorial: Esecuzione di un'applicazione di esempio utilizzando DynamoDB Accelerator (DAX)

Questo tutorial dimostra come avviare un' EC2 istanza Amazon nel tuo cloud privato virtuale (VPC) predefinito, connettersi all'istanza ed eseguire un'applicazione di esempio che utilizza Amazon DynamoDB Accelerator (DAX).

> **Note**
>
> Per completare questo tutorial, è necessario avere un cluster DAX in esecuzione nel VPC predefinito. Se non è stato creato un cluster DAX, consulta Creazione di un cluster DAX per le relative istruzioni.

Argomenti

- Fase 1: Avvio di un' EC2istanza Amazon
- Fase 2: creare un utente e una policy
- Fase 3: configurare un' EC2istanza Amazon
- Fase 4: esecuzione di una applicazione di esempio

Fase 1: Avvio di un' EC2istanza Amazon

Quando il cluster Amazon DynamoDB Accelerator (DAX) è disponibile, puoi avviare un'istanza Amazon nel tuo EC2 Amazon Virtual Private Cloud (Amazon VPC) predefinito. È quindi possibile installare ed eseguire il software client DAX su tale istanza.

Per avviare un'istanza EC2

1. Accedi a Console di gestione AWS e apri la EC2 console Amazon all'indirizzo https://console.aws.amazon.com/ec2/.
2. Scegli Launch Instance (Avvia istanza) ed effettua le seguenti operazioni:

   Fase 1: scegliere un'Amazon Machine Image (AMI)

   1. Nell'elenco di AMIs, trova l'AMI Amazon Linux e scegli Seleziona.

   Fase 2: scelta di un tipo di istanza

   1. Nell'elenco dei tipi di istanza, scegli t2.micro.
   2. Scegliere Next: Configure Instance Details (Successivo: Configura i dettagli dell'istanza).

   Fase 3: configurare i dettagli dell'istanza

   1. In Rete seleziona il VPC predefinito.
   2. Scegli Passaggio successivo: aggiunta dello storage.

   Fase 4: aggiungere storage

   1. Ignora questa fase scegliendo Next: Add tags (Successivo: Aggiungi i tag).

   Fase 5: aggiungi i tag

1. Ignora questa fase scegliendo Next: Configure Security Group (Successivo: Configura il gruppo di sicurezza).

   Fase 6: configura il gruppo di sicurezza

   1. Scegli Seleziona un gruppo di sicurezza esistente.
   2. Nell'elenco dei gruppi di sicurezza, scegli default (predefinito). Questo è il gruppo di sicurezza di default per il tuo ambiente VPC.
   3. Scegli Next: Review and Launch (Successivo: Verifica e avvia).

   Fase 7: verifica il lancio dell'istanza

   1. Scegli Avvia.
3. Nella finestra Select an existing key pair or create a new key pair (Seleziona una coppia di chiavi esistente oppure crea una nuova coppia di chiavi), effettua una delle seguenti operazioni:

   - Se non disponi di una coppia di EC2 chiavi Amazon, scegli Crea una nuova coppia di chiavi e segui le istruzioni. Viene chiesto di scaricare un file della chiave privata (file `.pem`). Questo file ti servirà in un secondo momento, quando accederai alla tua EC2 istanza Amazon.
   - Se disponi già di una coppia di EC2 chiavi Amazon, vai a Seleziona una coppia di chiavi e scegli la tua coppia di chiavi dall'elenco. Devi già avere il file della chiave privata (`.pem`file) disponibile per accedere alla tua EC2 istanza Amazon.
4. Dopo aver configurato la coppia di chiavi, scegli Launch Instances (Avvia istanze).
5. Nel pannello di navigazione della console, scegli EC2 Dashboard, quindi scegli l'istanza che hai avviato. Nel riquadro inferiore, nella scheda Description (Descrizione), trova il Public DNS (DNS pubblico) per la tua istanza, ad esempio `ec2-11-22-33-44.us-west-2.compute.amazonaws.com`. Prendi nota di questo nome DNS pubblico, perché sarà necessario per Fase 3: configurare un' EC2istanza Amazon.

> **Note**
>
> Bastano alcuni minuti prima che la tua EC2 istanza Amazon diventi disponibile. Nel frattempo, passa a Fase 2: creare un utente e una policy e segui le istruzioni indicate.

Fase 2: creare un utente e una policy

In questa fase, crei un utente con una policy che concede l'accesso al tuo cluster Amazon DynamoDB Accelerator (DAX) e all'utilizzo di DynamoDB. AWS Identity and Access Management Quindi è possibile eseguire le applicazioni che interagiscono con il cluster DAX.

Registrati per un Account AWS

Se non ne hai uno Account AWS, completa i seguenti passaggi per crearne uno.

Per iscriverti a un Account AWS

1. Apri la https://portal.aws.amazon.com/billing/registrazione.
2. Segui le istruzioni online.

   Nel corso della procedura di registrazione riceverai una telefonata o un messaggio di testo e ti verrà chiesto di inserire un codice di verifica attraverso la tastiera del telefono.

   Quando ti iscrivi a un Account AWS, Utente root dell'account AWSviene creato un. L'utente root dispone dell'accesso a tutte le risorse e tutti i Servizi AWS nell'account. Come best practice di sicurezza, assegna l'accesso amministrativo a un utente e utilizza solo l'utente root per eseguire le attività che richiedono l'accesso di un utente root.

AWS ti invia un'email di conferma dopo il completamento della procedura di registrazione. In qualsiasi momento, puoi visualizzare l'attività corrente del tuo account e gestirlo accedendo a https://aws.amazon.com/e scegliendo Il mio account.

Crea un utente con accesso amministrativo

Dopo esserti registrato Account AWS, proteggi Utente root dell'account AWS AWS IAM Identity Center, abilita e crea un utente amministrativo in modo da non utilizzare l'utente root per le attività quotidiane.

Proteggi i tuoi Utente root dell'account AWS

1. Accedi Console di gestione AWScome proprietario dell'account scegliendo Utente root e inserendo il tuo indirizzo Account AWS email. Nella pagina successiva, inserisci la password.

   Per informazioni sull'accesso utilizzando un utente root, consulta la pagina Signing in as the root user della Guida per l'utente di Accedi ad AWS .

2. Abilita l'autenticazione a più fattori (MFA) per l'utente root.

   Per istruzioni, consulta Abilitare un dispositivo MFA virtuale per l'utente Account AWS root (console) nella Guida per l'utente IAM.

Crea un utente con accesso amministrativo

1. Abilita Centro identità IAM.

   Per istruzioni, consulta Abilitazione di AWS IAM Identity Center nella Guida per l'utente di AWS IAM Identity Center .

2. In IAM Identity Center, assegna l'accesso amministrativo a un utente.

   Per un tutorial sull'utilizzo di IAM Identity Center directory come fonte di identità, consulta Configurare l'accesso utente con l'impostazione predefinita IAM Identity Center directory nella Guida per l'AWS IAM Identity Center utente.

Accesso come utente amministratore

- Per accedere con l'utente IAM Identity Center, utilizza l'URL di accesso che è stato inviato al tuo indirizzo e-mail quando hai creato l'utente IAM Identity Center.

  Per informazioni sull'accesso utilizzando un utente IAM Identity Center, consulta AWS Accedere al portale di accesso nella Guida per l'Accedi ad AWS utente.

Assegna l'accesso a ulteriori utenti

1. In IAM Identity Center, crea un set di autorizzazioni conforme alla best practice dell'applicazione di autorizzazioni con il privilegio minimo.

   Segui le istruzioni riportate nella pagina Creazione di un set di autorizzazioni nella Guida per l'utente di AWS IAM Identity Center .

2. Assegna al gruppo prima gli utenti e poi l'accesso con autenticazione unica (Single Sign-On).

   Per istruzioni, consulta Aggiungere gruppi nella Guida per l'utente di AWS IAM Identity Center .

Per fornire l'accesso, aggiungi autorizzazioni agli utenti, gruppi o ruoli:

- Utenti e gruppi in AWS IAM Identity Center:

  Crea un set di autorizzazioni. Segui le istruzioni riportate nella pagina Create a permission set (Creazione di un set di autorizzazioni) nella Guida per l'utente di AWS IAM Identity Center .

- Utenti gestiti in IAM tramite un provider di identità:

  Crea un ruolo per la federazione delle identità. Segui le istruzioni riportate nella pagina Create a role for a third-party identity provider (federation) della Guida per l'utente IAM.

- Utenti IAM:
  - Crea un ruolo che l'utente possa assumere. Segui le istruzioni riportate nella pagina Create a role for an IAM user della Guida per l'utente IAM.
  - (Non consigliato) Collega una policy direttamente a un utente o aggiungi un utente a un gruppo di utenti. Segui le istruzioni riportate nella pagina Aggiunta di autorizzazioni a un utente (console) nella Guida per l'utente IAM.

Come utilizzare l'editor di policy JSON per creare una policy

1. Accedi Console di gestione AWS e apri la console IAM all'indirizzo https://console.aws.amazon.com/iam/.
2. Nel riquadro di navigazione a sinistra, seleziona Policies (Policy).

   Se è la prima volta che selezioni Policy, verrà visualizzata la pagina Benvenuto nelle policy gestite. Seleziona Inizia.
3. Nella parte superiore della pagina, scegli Crea policy.
4. Nella sezione Editor di policy, scegli l'opzione JSON.
5. Immettere o incollare un documento di policy JSON. Per dettagli sul linguaggio della policy IAM, consulta la Documentazione di riferimento delle policy JSON IAM.
6. Risolvi eventuali avvisi di sicurezza, errori o avvisi generali generati durante la convalida delle policy, quindi scegli Next (Successivo).

   > **Note**
   >
   > È possibile alternare le opzioni dell'editor Visivo e JSON in qualsiasi momento. Se tuttavia si apportano modifiche o si seleziona Successivo nell'editor Visivo, IAM potrebbe

ristrutturare la policy in modo da ottimizzarla per l'editor visivo. Per ulteriori informazioni, consulta Modifica della struttura delle policy nella Guida per l'utente di IAM.

7. (Facoltativo) Quando crei o modifichi una policy in Console di gestione AWS, puoi generare un modello di policy JSON o YAML da utilizzare nei modelli. CloudFormation

   Per fare ciò, nell'editor delle politiche scegli Azioni, quindi scegli Genera modello. CloudFormation Per saperne di più CloudFormation, consulta il riferimento al tipo di AWS Identity and Access Management risorsa nella Guida AWS CloudFormation per l'utente.
8. Una volta terminata l'aggiunta delle autorizzazioni alla policy, scegli Successivo.
9. Nella pagina Rivedi e crea, immettere un valore in Nome policy e Descrizione (facoltativo) per la policy in fase di creazione. Rivedi Autorizzazioni definite in questa policy per visualizzare le autorizzazioni concesse dalla policy.
10. (Facoltativo) Aggiungere metadati alla policy collegando i tag come coppie chiave-valore. Per ulteriori informazioni sull'utilizzo dei tag in IAM, consulta Tags for AWS Identity and Access Management resources in the IAM User Guide.
11. Seleziona Crea policy per salvare la nuova policy.

Documento policy: copiare e incollare il documento seguente per creare la policy JSON.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Action": [
                "dax:*"
            ],
            "Effect": "Allow",
            "Resource": [
                "*"
            ]
        },
        {
            "Action": [
                "dynamodb:*"
            ],
```

```
            "Effect": "Allow",
            "Resource": [
                "*"
            ]
        }
    ]
}
```

Fase 3: configurare un' EC2istanza Amazon

Quando l' EC2 istanza Amazon è disponibile, puoi accedere all'istanza e prepararla per l'uso.

> **Note**
>
> I passaggi seguenti presuppongono che ti stia connettendo alla tua EC2 istanza Amazon da un computer che esegue Linux. Per altri modi di connessione, consulta Connect to Your Linux Instance nella Amazon EC2 User Guide.

Per configurare l' EC2 istanza

1. Apri la EC2 console Amazon all'indirizzo https://console.aws.amazon.com/ec2/.
2. Usa il `ssh` comando per accedere alla tua EC2 istanza Amazon, come mostrato nell'esempio seguente.

   ```
   ssh -i my-keypair.pem ec2-user@public-dns-name
   ```

   È necessario specificare il file di chiave privata (file `.pem`) e il nome DNS pubblico dell'istanza. Consultare Fase 1: Avvio di un' EC2istanza Amazon.

   L'ID accesso è `ec2-user`. Non è richiesta una password.
3. Dopo aver effettuato l'accesso all' EC2 istanza, configura le AWS credenziali come illustrato di seguito. Inserisci AWS l'ID della chiave di accesso e la chiave segreta (daFase 2: creare un utente e una policy) e imposta il nome predefinito della regione sulla regione corrente. Nell'esempio che segue, il nome della regione predefinito è `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]:
```

Dopo aver avviato e configurato l' EC2 istanza Amazon, puoi testare la funzionalità di DAX utilizzando una delle applicazioni di esempio disponibili. Per ulteriori informazioni, consulta Fase 4: esecuzione di una applicazione di esempio.

Fase 4: esecuzione di una applicazione di esempio

Per aiutarti a testare la funzionalità di Amazon DynamoDB Accelerator (DAX), puoi eseguire una delle applicazioni di esempio disponibili sulla tua istanza Amazon. EC2

Argomenti

- Node.js e DAX
- SDK per Go di DAX
- Java e DAX
- .NET e DAX
- Python e DAX

Node.js e DAX

Configurazione di un client predefinito per Node.js

Durante la configurazione del client DAX JavaScript SDK, è possibile personalizzare vari parametri per ottimizzare le prestazioni, la gestione della connessione e la resilienza agli errori. La tabella seguente descrive le impostazioni di configurazione predefinite che controllano il modo in cui il client interagisce con il cluster DAX, inclusi i valori di timeout, i meccanismi di ripetizione dei tentativi, la gestione delle credenziali e le opzioni di monitoraggio dello stato. Per ulteriori informazioni, consulta Dynamo Operations. DBClient

Impostazioni predefinite del client SDK DAX per JS

Parametro	Tipo	Description
`region` facoltativo	`string`	Regione AWS Da usare per il client DAX (esempio: 'us-east-1'). Questo è un parametro

Parametro	Tipo	Description
		obbligatorio se non viene fornito tramite la variabile di ambiente.
`endpoint` obbligatorio	`string`	L'endpoint del Cluster a cui si connette l'SDK. Esempi: Non crittografato: .region.amazonaws.com dax-cluster-name Crittografato: daxs://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
`requestTimeout` valore predefinito 6000 ms	`number`	Definisce il tempo massimo di attesa da parte del client per ricevere una risposta da DAX.
`writeRetries` valore predefinito 1	`number`	Il numero di ripetizioni di tentativi per le richieste di scrittura che hanno esito negativo.
`readRetries` valore predefinito 1	`number`	Il numero di ripetizioni di tentativi per le richieste di lettura che hanno esito negativo.

Parametro	Tipo	Description
`maxRetries` valore predefinito 1	`number`	Il numero massimo di ripetizioni di tentativi in caso di richieste che hanno esito negativo. Se readRetries/writeRetries sono impostati, la configurazione impostata in readRetries e writeRetries ha la priorità su maxRetries.
`connectTimeout` valore predefinito 10000 ms	`number`	Il timeout (in millisecondi) per stabilire una connessione a uno qualsiasi dei nodi del cluster.
`maxConcurrentConnections` valore predefinito 100	`number`	Limita il numero totale di connessioni simultanee che un'istanza client può creare per nodo in un cluster DAX.

Parametro	Tipo	Description
`maxRetryDelay` valore predefinito 7000 ms	`number`	Quando il server DAX indica che è necessario il ripristino impostando il flag `waitForRecoveryBeforeRetrying` su true, il client si interrompe prima della ripetizione del tentativo. Durante questi periodi di recupero, il parametro `maxRetryDelay` determina il tempo massimo di attesa tra le ripetizioni di tentativi. Questa configurazione specifica per il recupero si applica solo quando il server DAX è in modalità di recupero. In tutti gli altri scenari, il comportamento delle ripetizioni di tentativi segue uno dei due modelli seguenti: un ritardo esponenziale basato sul numero di ripetizioni di tentativi (regolato da parametri `writeRetries`, `readRetries` o `maxRetries`) o una ripetizione di tentativo immediata a seconda del tipo di eccezione.

Parametro	Tipo	Description
`credentials` facoltativo	`AwsCredentialIdentity` | `AwsCredentialIdentityProvider`	Le credenziali da utilizzare per l'autenticazione delle richieste. AWS Questo può essere fornito come un o un AwsCredentialIdentity . AwsCredentialIdentityProvider Se non fornite, l'SDK AWS utilizzerà automaticamente la catena di provider di credenziali predefinita. Esempio: `{ accessKeyId: 'AKIA... ' , secretAccessKey: ' ... ', sessionToken:'... '} `* @default Utilizza la catena di provider di credenziali AWS predefinita.
`healthCheckInterval` valore predefinito 5000 ms	`number`	L'intervallo (in millisecondi) tra i controlli di integrità del cluster. Un intervallo più basso verrà controllato più frequentemente.
`healthCheckTimeout` valore predefinito 1000 ms	`number`	Il timeout (in millisecondi) per il completamento del controllo dell'integrità.

Parametro	Tipo	Description
`skipHostnameVerification` valore predefinito false	`boolean`	Salta la verifica del nome host delle connessioni TLS. Non ha alcun impatto sui cluster non crittografati. L'impostazione predefinita prevede la verifica del nome host; impostandola su True, la verifica verrà saltata. Assicurati di comprendere le implicazioni della disattivazione, ovvero l'impossibilità di autenticare il cluster a cui ti stai connettendo.
`unhealthyConsecutiveErrorCount` valore predefinito 5	`number`	Imposta il numero di errori consecutivi necessari per segnalare un nodo non integro entro l'intervallo di controllo dell'integrità.
`clusterUpdateInterval` valore predefinito 4000 ms	`number`	Restituisce l'intervallo tra polling di membri del cluster per le modifiche ai membri del cluster.
`clusterUpdateThreshold` valore predefinito 125	`number`	Restituisce la soglia al di sotto della quale sul cluster non verranno eseguite query per le modifiche ai membri.
`credentailProvider` opzionale | valore predefinito null	`AwsCredentialIdentityProvider`	Provider definito dall'utente per AWS le credenziali utilizzate per autenticare le richieste a DAX.

Configurazione della paginazione per DaxDocument

Nome	Tipo	Dettaglio
`client`	DaxDocument	Istanza di DaxDocument tipo.
`pageSize`	number	Determina il numero di elementi per pagina.
`startingToken` Facoltativo	any	LastEvaluatedKey dalla risposta precedente può essere utilizzato per richieste successive.

Per l'utilizzo dell'impaginazione, consulta the section called "TryDax.js".

Migrazione a SDK DAX per Node.js V3

Questa guida alla migrazione aiuterà a effettuare la transizione delle applicazioni Node.js esistenti con DAX. Il nuovo SDK richiede Node.js 18 o versione successiva e introduce diverse importanti modifiche alla struttura del codice di DynamoDB Accelerator. Questa guida illustrerà le differenze principali, tra cui modifiche alla sintassi, nuovi metodi di importazione e modelli di programmazione asincrona aggiornati.

Utilizzo di DAX per V2 Node.js

```
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);
    }
});
```

Utilizzo di DAX per V3 Node.js

Per l'utilizzo di DAX per Node.js V3, Node versione 18 o successiva è la versione preferita. Per passare a Node 18, utilizza quanto segue:

```
// 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)
}
```

L'SDK DAX per Node.js v3.x è compatibile con SDK AWS per Node.js v3.x. L'SDK DAX per Node.js v3.x supporta l'uso di client aggregati. Si noti che DAX non supporta la creazione di client essenziali (bare-bones). Per ulteriori informazioni sulle funzionalità non supportate, consulta the section called "Funzionalità non in parità con AWS SDK V3".

Segui questi passaggi per eseguire l'applicazione di esempio Node.js sulla tua EC2 istanza Amazon.

Come eseguire l'applicazione di esempio Node.js per DAX

1. Configura Node.js sulla tua EC2 istanza Amazon, come segue:

   a. Installa il gestore delle versioni del nodo (nvm).

   ```
   curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.40.1/install.sh | bash
   ```

   b. Usa nvm per installare Node.js.

   ```
   nvm install 18
   ```

   c. Usa nvm per utilizzare Node 18

   ```
   nvm use 18
   ```

   d. Verifica che Node.js sia installato e funzioni correttamente.

   ```
   node -e "console.log('Running Node.js ' + process.version)"
   ```

   Si dovrebbe visualizzare il seguente messaggio.

```
Running Node.js v18.x.x
```

2. Installa il client DaxDocument Node.js utilizzando il gestore di pacchetti del nodo (npm).

```
npm install @amazon-dax-sdk/lib-dax
```

TryDaxcodice di esempio

Per valutare i vantaggi prestazionali di DynamoDB Accelerator (DAX), segui questa procedura per eseguire un test di esempio che confronta i tempi di lettura tra DynamoDB standard e un cluster DAX.

1. Dopo aver configurato il workspace e installato `lib-dax` come dipendenza, copia the section called “TryDax.js” nel progetto.
2. Esegui il programma sul cluster DAX. Per determinare l'endpoint per il cluster DAX, scegli una delle seguenti opzioni:

   - Utilizzo della console DynamoDB: scegli il cluster DAX. L'endpoint del cluster viene visualizzato nella console, come nell'esempio seguente.

     ```
     dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
     ```

   - Utilizzando il comando AWS CLI— Immettere il seguente comando.

     ```
     aws dax describe-clusters --query "Clusters[*].ClusterDiscoveryEndpoint"
     ```

     L'endpoint del cluster viene visualizzato nell'output, come nell'esempio seguente.

     ```
     {
         "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. Ora esegui il programma, specificando l’endpoint del cluster come parametro della riga di comando.

```
node TryDax.js dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
```

Verrà visualizzato un output simile al seguente:

```
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.
```

Prendi nota delle informazioni sui tempi. Il numero di microsecondi necessari per i test `GetItem`, `Query`, `Scan`.

4. In questo caso, i programmi sono stati eseguiti sul cluster DAX. Ora eseguirai di nuovo il programma, questa volta su DynamoDB.
5. A questo punto, esegui di nuovo il programma, ma questa volta senza l'URL dell'endpoint del cluster come parametro della riga di comando.

```
node TryDax.js
```

Consultare l'output e prendere nota delle informazioni sui tempi. I tempi trascorsi per `GetItem`, `Query` e `Scan` devono essere significativamente più bassi con DAX rispetto a DynamoDB.

Funzionalità non in parità con AWS SDK V3

- Client essenziali: Dax per Node.js V3 non supporta client essenziali.

```
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);
```

- Stack middleware: Dax per Node.js V3 non supporta l'utilizzo di funzioni 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 per Go di DAX

Segui questa procedura per eseguire l'applicazione di esempio Amazon DynamoDB Accelerator (DAX) SDK for Go sulla tua istanza Amazon. EC2

Come eseguire l'esempio SDK per Go per DAX

1. Configura l'SDK for Go sulla tua istanza EC2 Amazon:

   a. Installa il linguaggio di programmazione Go (`Golang`).

   ```
   sudo yum install -y golang
   ```

   b. Verifica che Golang sia installata e funzioni correttamente.

   ```
   go version
   ```

   Dovrebbe apparire un messaggio come questo.

   ```
   go version go1.23.4 linux/amd64
   ```

2. Installa l'applicazione Golang di esempio.

   ```
   go get github.com/aws-samples/sample-aws-dax-go-v2
   ```

3. Esegui i seguenti programmi Golang. Il primo programma crea una tabella DynamoDB denominata `TryDaxGoTable`. Il secondo programma scrive i dati nella tabella.

```
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. Esegui i seguenti programmi 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
```

   Prendi nota delle informazioni sui tempi: il numero di millisecondi richiesto per i test `GetItem`, `Query` e `Scan`.

5. Nella fase precedente, i programmi sono stati eseguiti sull'endpoint DynamoDB. A questo punto, eseguire nuovamente i programmi, ma questa volta le operazioni `GetItem`, `Query` e `Scan` saranno elaborate dal cluster DAX.

   Per determinare l'endpoint per il cluster DAX, scegli una delle seguenti opzioni:

   - Utilizzo della console DynamoDB: scegli il cluster DAX. L'endpoint del cluster viene visualizzato nella console, come nell'esempio seguente.

     ```
     dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
     ```

   - Utilizzando il AWS CLI — Inserisci il seguente comando.

     ```
     aws dax describe-clusters --query "Clusters[*].ClusterDiscoveryEndpoint"
     ```

     L'endpoint del cluster viene visualizzato nell'output, come nell'esempio seguente.

```
{
    "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"
}
```

Ora esegui di nuovo i programmi, ma questa volta specifica l'endpoint del cluster come parametro della riga di comando.

```
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
```

Guarda il resto dell'output e prendi nota delle informazioni sui tempi. I tempi trascorsi per `GetItem`, `Query` e `Scan` devono essere significativamente più bassi con DAX che con DynamoDB.

6. Esegui il seguente programma Golang per eliminare `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
```

Funzionalità non in parità con la versione 2 AWS SDK per Go

Middleware Stack: DAX Go V2 non supporta l'uso di Middleware Stacks tramite. APIoptions Per ulteriori informazioni, consulta Personalizzazione delle richieste client v2 con Middleware. AWS SDK per Go

Esempio:

```
// 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
}
```

Output:

```
failed to execute command: custom middleware through APIOptions is not supported in DAX
 client
exit status 1
```

Configurazione di un client predefinito per Go

In questa guida verranno illustrate le opzioni di configurazione che consentono di ottimizzare le prestazioni, la gestione delle connessioni e il comportamento di registrazione di log del client DAX. Comprendendo le impostazioni predefinite e come personalizzarle è possibile ottimizzare l'interazione dell'applicazione Go con DAX.

In questa sezione

- Impostazioni predefinite del client SDK DAX per Go
- Creazione di client

Impostazioni predefinite del client SDK DAX per Go

Parametro	Tipo	Description
`Region` obbligatorio	`string`	Regione AWS Da usare per il client DAX (esempio: 'us-east-1'). Questo è un parametro obbligatorio se non viene fornito tramite l'ambiente.
`HostPorts` obbligatorio	`[] string`	Elenco degli endpoint del cluster DAX a cui si connette l'SDK. Ad esempio: Non crittografato: dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com

Parametro	Tipo	Description
		Crittografato: daxs://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
`MaxPendingConnectionsPerHost` valore predefinito 10	`number`	Numero di tentativi di connessione concomitanti. (È possibile che le connessioni siano in fase di creazione simultanea.)
`ClusterUpdateThreshold` valore predefinito 125 * time.Millisecond	`time.Duration`	Il tempo minimo che deve trascorrere tra gli aggiornamenti del cluster.
`ClusterUpdateInterval` valore predefinito 4 * time.Second	`time.Duration`	L'intervallo con cui il client aggiornerà automaticamente le informazioni del cluster DAX.
`IdleConnectionsReapDelay` valore predefinito 30 * time.Second	`time.Duration`	L'intervallo in base al quale il client chiuderà le connessioni inattive nel client DAX.
`ClientHealthCheckInterval` valore predefinito 5 * time.Second	`time.Duration`	L'intervallo con cui il client eseguirà i controlli di integrità sugli endpoint del cluster DAX.

Parametro	Tipo	Description
`Credentials` default	`aws.CredentialsProvider`	Le AWS credenziali utilizzate dal client DAX per autenticare le richieste al servizio DAX. Consulta Credentials and Credential Providers.
`DialContext` default	`func`	Una funzione personalizzata utilizzata dal client DAX per stabilire connessioni al cluster DAX.
`SkipHostnameVerification` valore predefinito false	bool	Salta la verifica del nome host delle connessioni TLS. Questa impostazione ha effetto solo sui cluster crittografati. Se impostata su True, disabilita la verifica del nome host. La disabilitazione della verifica significa che non è possibile autenticare l'identità del cluster a cui si sta effettuando la connessione, il che comporta rischi per la sicurezza. Per impostazione predefinita, la verifica del nome host è abilitata.
`RouteManagerEnabled` valore predefinito false	`bool`	Questo flag viene utilizzato per rimuovere gli instradamenti soggetti a errori di rete.

Parametro	Tipo	Description
`RequestTimeout` valore predefinito 60 * time.Second	`time.Duration`	Definisce il tempo massimo di attesa da parte del client per ricevere una risposta da DAX. Priorità: timeout contestuale (se impostato) > `RequestTimmeout` (se impostato) > Valore predefinito 60 s `RequestTimeout` .
`WriteRetries` valore predefinito 2	`number`	Il numero di ripetizioni di tentativi per le richieste di scrittura che hanno esito negativo.
`ReadRetries` valore predefinito 2	`number`	Il numero di ripetizioni di tentativi per le richieste di lettura che hanno esito negativo.
`RetryDelay` valore predefinito 0	`time.Duration`	Il ritardo per gli errori non sottoposti a limitazione (della larghezza di banda della rete) (in secondi) per le ripetizioni di tentativi quando una richiesta ha esito negativo.
`Logger` facoltativo	`logging.Logger`	Logger è un'interfaccia per la registrazione di log delle voci in determinate classificazioni.

Parametro	Tipo	Description
`LogLevel` utilità predefinite. LogOff	`number`	Questo loglevel è definito solo per DAX. Può essere importato usando github. com/aws/aws-dax-go-v2/tree/main/dax/utils. `const ( LogOff LogLevelType = 0 LogDebug LogLevelType = 1 LogDebugWithReques tRetries LogLevelType = 2 )`

> **Note**
>
> Per `time.Duration`, l'unità predefinita è il nanosecondo. Se non si specifica alcuna unità per un parametro, verranno utilizzati i nano secondi: `daxCfg.ClusterUpdateInterval = 10` significa 10 nano secondi. (`daxCfg.ClusterUpdateInterval = 10 * time.Millisecond` significa 10 millisecondi).

Creazione di client

Come creare un client DAX:

- Crea la configurazione DAX, quindi crea il client DAX utilizzando la configurazione DAX. In questo modo è possibile sovrascrivere una configurazione DAX, se necessario.

```
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)
```

Migrazione a SDK DAX per Go V2

Questa guida alla migrazione aiuterà a effettuare la transizione delle applicazioni DAX per Go esistenti.

Utilizzo dell'SDK DAX per Go 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)
}
```

Utilizzo dell'SDK DAX per Go 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)
}
```

Per ulteriori dettagli ed esempi di API, consulta AWS samples.

Java e DAX

SDK per Java 2.x per DAX è compatibile con AWS SDK per Java 2.x. È basato su Java 8+ e include il supporto per I/O non bloccanti. Per informazioni sull'utilizzo di DAX con SDK for AWS Java 1.x, consulta. Uso di DAX con AWS SDK per Java 1.x

Utilizzo del client come dipendenza Maven

Segui queste fasi per utilizzare il client per l'SDK DAX per Java nella tua applicazione come una dipendenza:

1. Scarica e installa Apache Maven. Per ulteriori informazioni, consulta le pagine relative al download di Apache Maven e all'installazione di Apache Maven.
2. Aggiungi la dipendenza Maven client al file POM (Project Object Model) dell'applicazione. In questo esempio, sostituiscilo *x.x.x* con il numero di versione effettivo del client.

```
<!--Dependency:-->
<dependencies>
    <dependency>
        <groupId>software.amazon.dax</groupId>
        <artifactId>amazon-dax-client</artifactId>
        <version>x.x.x</version>
    </dependency>
</dependencies>
```

TryDax codice di esempio

Dopo aver configurato il workspace e aggiunto l'SDK DAX come dipendenza, copiare TryDax.java nel progetto.

Eseguire il codice utilizzando questo comando.

```
java -cp classpath TryDax
```

Verrà visualizzato un output simile al seguente.

```
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.
```

Prendi nota delle informazioni sui tempi: il numero di millisecondi richiesto per i test `GetItem` e `Query`. In questo caso, il programma è stato eseguito sull'endpoint DynamoDB. Ora eseguirai di nuovo il programma, questa volta rispetto al cluster DAX.

Per determinare l'endpoint per il cluster DAX, scegli una delle seguenti opzioni:

- Nella console DynamoDB, seleziona il cluster DAX. L'endpoint del cluster viene visualizzato nella console, come nell'esempio seguente.

  ```
  dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
  ```

- Usando AWS CLI, inserisci il seguente comando:

  ```
  aws dax describe-clusters --query "Clusters[*].ClusterDiscoveryEndpoint"
  ```

  L'indirizzo, la porta e l'URL dell'endpoint del cluster vengono visualizzati nell'output, come nell'esempio seguente.

  ```
  {
      "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"
  }
  ```

A questo punto, eseguire di nuovo il programma, ma questa volta specificare l'endpoint del cluster come parametro della riga di comando.

```
java -cp classpath TryDax dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
```

Consultare l'output e prendere nota delle informazioni sui tempi. I tempi trascorsi per `GetItem` e `Query` devono essere significativamente più bassi con DAX che con DynamoDB.

Parametri SDK

Con DAX SDK for Java 2.x, puoi raccogliere metriche sui client di servizio nella tua applicazione e analizzare l'output in Amazon. CloudWatch Per ulteriori informazioni, consulta Abilitazione dei parametri SDK.

> **Note**
>
> L'SDK DAX per Java raccoglie solo le metriche `ApiCallSuccessful` e `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 e DAX

Segui questi passaggi per eseguire l'esempio.NET sulla tua EC2 istanza Amazon.

> **Note**
>
> Questo tutorial utilizza l'SDK.NET 9. Mostra come eseguire un programma nell'Amazon VPC predefinito per accedere al cluster Amazon DynamoDB Accelerator (DAX). Funziona con l'AWS SDK v4 per.NET. Per dettagli sulle modifiche alla V4 e informazioni sulla migrazione, consulta Migrazione alla versione 4 dell'SDK for AWS .NET. Se preferisci, puoi usare il AWS Toolkit for Visual Studio per scrivere un'applicazione.NET e distribuirla nel tuo VPC.
> Per ulteriori informazioni, consulta Creazione e implementazione di applicazioni Elastic Beanstalk in .NET mediante AWS Toolkit for Visual Studio nella Guida per gli sviluppatori di AWS Elastic Beanstalk .

Come eseguire l'applicazione di esempio .NET per DAX

1. Vai alla pagina Microsoft Downloads e scarica l'ultima versione di.NET 9 SDK per Linux. Il file scaricato è `dotnet-sdk-N.N.N-linux-x64.tar.gz`.
2. Estrai i file SDK.

   ```
   mkdir dotnet
   tar zxvf dotnet-sdk-N.N.N-linux-x64.tar.gz -C dotnet
   ```

   Sostituisci *N.N.N* con il numero effettivo della versione dell'SDK .NET (ad esempio: `9.0.305`).
3. Verifica l'installazione.

   ```
   alias dotnet=$HOME/dotnet/dotnet
   dotnet --version
   ```

   Ciò dovrebbe stampare il numero di versione dell'SDK .NET.

   > **Note**
   >
   > Invece del numero di versione, potresti ricevere l'errore seguente:

error: libunwind.so.8: cannot open shared object file: No such file or directory
Per risolvere l'errore, installa il pacchetto `libunwind`.

```
sudo yum install -y libunwind
```

Dopo aver eseguito questa operazione, dovresti essere in grado di eseguire il comando `dotnet --version` senza errori.

4. Crea un nuovo progetto .NET.

```
dotnet new console -o myApp
```

Questa operazione richiede alcuni minuti per eseguire una one-time-only configurazione. Al termine, esegui il progetto di esempio.

```
dotnet run --project myApp
```

Dovresti ricevere il seguente messaggio: `Hello World!`

5. Il file `myApp/myApp.csproj` contiene metadati sul tuo progetto. Per utilizzare il client DAX nell'applicazione, modificare il file nel modo seguente.

```
<Project Sdk="Microsoft.NET.Sdk">
    <PropertyGroup>
        <OutputType>Exe</OutputType>
        <TargetFramework>net9.0</TargetFramework>
    </PropertyGroup>
    <ItemGroup>
        <PackageReference Include="AWSSDK.DAX.Client" Version="*" />
    </ItemGroup>
</Project>
```

6. Scarica il codice sorgente del programma di esempio (file `.zip`).

```
wget http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/samples/
TryDax.zip
```

Al termine del download, estrai i file di origine.

```
unzip TryDax.zip
```

7. Ora esegui i programmi di esempio di dotNet, uno alla volta. Per ogni programma, copia i suoi contenuti in `myApp/Program.cs` ed esegui il progetto `MyApp`.

   Esegui i seguenti programmi .NET. Il primo programma crea una tabella DynamoDB denominata `TryDaxTable`. Il secondo programma scrive i dati nella tabella.

   ```
   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. Quindi, avviare alcuni programmi per eseguire le operazioni `GetItem`, `Query` e `Scan` sul cluster DAX. Per determinare l'endpoint per il cluster DAX, scegli una delle seguenti opzioni:

   - Utilizzo della console DynamoDB: scegli il cluster DAX. L'endpoint del cluster viene visualizzato nella console, come nell'esempio seguente.

     ```
     dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
     ```

   - Utilizzando il AWS CLI — Immettere il seguente comando.

     ```
     aws dax describe-clusters --query "Clusters[*].ClusterDiscoveryEndpoint"
     ```

     L'endpoint del cluster viene visualizzato nell'output, come nell'esempio seguente.

     ```
     {
         "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"
     }
     ```

   Ora esegui i seguenti programmi, specificando il tuo endpoint del cluster come parametro della riga di comando. Sostituisci l'endpoint di esempio con l'endpoint del cluster DAX.

   ```
   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
```

Prendi nota delle informazioni sui tempi: il numero di millisecondi richiesto per i test `GetItem`, `Query` e `Scan`.

9. Esegui il seguente programma .NET per eliminare `TryDaxTable`:

```
cp TryDax/dotNet/06-DeleteTable.cs myApp/Program.cs
dotnet run --project myApp
```

Per ulteriori informazioni sui programmi, consulta le seguenti sezioni:

- 0-1 CreateTable cs
- 02-Write-Data.cs
- 03- -Test.cs GetItem
- 04-Query-Test.cs
- 05-Scan-Test.cs
- 0-6 pezzi DeleteTable

0-1 CreateTable cs

Il programma `01-CreateTable.cs` crea una tabella (`TryDaxTable`). I programmi .NET rimanenti in questa sezione dipendono da questa tabella.

```
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

Il programma `02-Write-Data.cs` scrive i dati di test in `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- -Test.cs GetItem

Il programma `03-GetItem-Test.cs` esegue le operazioni `GetItem` in `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

Il programma `04-Query-Test.cs` esegue le operazioni `Query` in `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

Il programma `05-Scan-Test.cs` esegue le operazioni `Scan` in `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();
        }
    }
}
```

0-6 pezzi DeleteTable

Il programma `06-DeleteTable.cs` elimina `TryDaxTable`. Esegui questo programma dopo aver completato i test.

```
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 e DAX

Segui questa procedura per eseguire l'applicazione di esempio Python sulla tua istanza Amazon EC2.

Come eseguire l'applicazione di esempio Python per DAX

1. Installare il client Python per DAX mediante l'utilità `pip`.

   ```
   pip install amazon-dax-client
   ```

2. Scarica il codice sorgente del programma di esempio (file `.zip`).

   ```
   wget http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/samples/
   TryDax.zip
   ```

   Al termine del download, estrai i file di origine.

   ```
   unzip TryDax.zip
   ```

3. Esegui i seguenti programmi Python. Il primo programma crea una tabella Amazon DynamoDB denominata `TryDaxTable`. Il secondo programma scrive i dati nella tabella.

   ```
   python 01-create-table.py
   python 02-write-data.py
   ```

4. Esegui i seguenti programmi Python.

   ```
   python 03-getitem-test.py
   python 04-query-test.py
   python 05-scan-test.py
   ```

Prendi nota delle informazioni sui tempi: il numero di millisecondi richiesto per i test `GetItem`, `Query` e `Scan`.

5. Nella fase precedente, i programmi sono stati eseguiti sull'endpoint DynamoDB. A questo punto, eseguire nuovamente i programmi, ma questa volta le operazioni `GetItem`, `Query` e `Scan` saranno elaborate dal cluster DAX.

   Per determinare l'endpoint per il cluster DAX, scegli una delle seguenti opzioni:

   - Utilizzo della console DynamoDB: scegli il cluster DAX. L'endpoint del cluster viene visualizzato nella console, come nell'esempio seguente.

     ```
     dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
     ```

   - Utilizzando il comando AWS CLI— Inserisci il seguente comando.

     ```
     aws dax describe-clusters --query "Clusters[*].ClusterDiscoveryEndpoint"
     ```

     L'endpoint del cluster viene visualizzato nell'output, come in questo esempio.

     ```
     {
         "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"
     }
     ```

   Esegui nuovamente i programmi, ma questa volta specifica l'endpoint del cluster come parametro della riga di comando.

   ```
   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
   ```

   Guarda il resto dell'output e prendi nota delle informazioni sui tempi. I tempi trascorsi per `GetItem`, `Query` e `Scan` devono essere significativamente più bassi con DAX che con DynamoDB.

6. Esegui il seguente programma Python per eliminare `TryDaxTable`.

```
python 06-delete-table.py
```

Per ulteriori informazioni sui programmi, consulta le seguenti sezioni:

- 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

Il programma `01-create-table.py` crea una tabella (`TryDaxTable`). I programmi Python rimanenti in questa sezione dipendono da questa tabella.

```
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

Il programma `02-write-data.py` scrive i dati di test in `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

Il programma `03-getitem-test.py` esegue le operazioni `GetItem` in `TryDaxTable`. Questo esempio è fornito per la Regione 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

Il programma `04-query-test.py` esegue le operazioni `Query` in `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

Il programma `05-scan-test.py` esegue le operazioni `Scan` in `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

Il programma `06-delete-table.py` elimina `TryDaxTable`. Eseguire questo programma dopo aver completato i test della funzionalità 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!")
```

Modifica di un'applicazione esistente per l'utilizzo con DAX

Se si dispone già di un'applicazione Java che utilizza Amazon DynamoDB, sarà necessario modificarla in modo che possa accedere al cluster DynamoDB Accelerator (DAX). Non è necessario riscrivere l'intera applicazione perché il client Java DAX è simile al client di basso livello DynamoDB incluso nell'SDK for Java 2.x. AWS Consultare Utilizzo degli elementi in DynamoDBper maggiori dettagli.

> **Note**
>
> Questo esempio utilizza AWS SDK for Java 2.x. Per la versione legacy dell'SDK per Java 1.x, vedereModifica dell'applicazione SDK per Java 1.x esistente per l'utilizzo di DAX.

Per modificare il programma, sostituire il client DynamoDB con un client 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();
```

Puoi anche utilizzare la libreria di alto livello che fa parte dell' AWS SDK for Java 2.x, sostituendo il client DynamoDB con un client 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();
```

Per ulteriori informazioni, consulta Mappatura di elementi nelle tabelle DynamoDB.

Gestione dei cluster DAX

In questa sezione vengono descritte alcune attività di gestione comuni per i cluster Amazon DynamoDB Accelerator (DAX).

Argomenti

- Autorizzazioni IAM per la gestione di un cluster DAX
- Dimensionamento di un cluster DAX
- Personalizzazione delle impostazioni del cluster DAX
- Configurazione delle impostazioni TTL
- Supporto per l'assegnazione di tag per DAX
- AWS CloudTrail integrazione
- Eliminazione di un cluster DAX

Autorizzazioni IAM per la gestione di un cluster DAX

Quando amministri un cluster DAX utilizzando Console di gestione AWS o il AWS Command Line Interface (AWS CLI), ti consigliamo vivamente di restringere l'ambito delle azioni che gli utenti possono eseguire. Così facendo, puoi mitigare i rischi seguendo il principio del minimo privilegio.

La discussione che segue si concentra sul controllo degli accessi per la gestione DAX. APIs Per ulteriori informazioni, consulta Amazon DynamoDB Accelerator nella Documentazione di riferimento delle API di Amazon DynamoDB.

> **Note**
>
> Per informazioni più dettagliate sulla gestione delle autorizzazioni AWS Identity and Access Management (IAM), consulta quanto segue:
>
> - IAM e creazione di cluster DAX: Creazione di un cluster DAX.
> - IAM e le operazioni del piano dati DAX: Controllo degli accessi DAX.

Per la gestione DAX APIs, non è possibile assegnare le azioni API a una risorsa specifica. L'elemento `Resource` deve essere impostato su `"*"`. Ciò differisce dalle operazioni API del piano dati DAX come `GetItem`, `Query` e `Scan`. Le operazioni del piano dati sono esposte tramite il client DAX e possono essere orientate verso risorse specifiche.

A titolo illustrativo, si consideri il documento della policy IAM riportato di seguito.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Action": [
                "dax:*"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:dax:us-west-2:123456789012:cache/DAXCluster01"
            ]
        }
    ]
}
```

Si supponga che l'intento di questa policy sia quello di consentire le chiamate API di gestione DAX per il cluster `DAXCluster01` e solo per quel cluster.

Supponiamo ora che un utente emetta il seguente AWS CLI comando.

```
aws dax describe-clusters
```

Questo comando non riesce con un'eccezione Not Authorized, perché la chiamata API `DescribeClusters` sottostante non può essere orientata a un cluster specifico. Anche se la policy è sintatticamente valida, il comando non riesce perché l'elemento `Resource` deve essere impostato su `"*"`. Tuttavia, se l'utente esegue un programma che invia le chiamate del piano dati DAX (ad esempio `GetItem` o `Query`) su `DAXCluster01`, queste chiamate avranno esito positivo. Questo perché il piano dati DAX APIs può essere limitato a risorse specifiche (in questo caso,). `DAXCluster01`

Se desideri scrivere un'unica policy IAM completa che comprenda sia la gestione APIs DAX che il piano dati DAX APIs, ti suggeriamo di includere due dichiarazioni distinte nel documento sulla policy. Una di queste istruzioni dovrebbe riguardare il piano dati DAX APIs, mentre l'altra si riferisce alla gestione DAX. APIs

Di seguito viene illustrato un esempio di policy che mostra questo approccio. Nota come l'istruzione `DAXDataAPIs` è orientata alla risorsa `DAXCluster01`, ma la risorsa per `DAXManagementAPIs`

deve essere "*". Le operazioni mostrate in ogni istruzione sono a solo scopo illustrativo. Puoi personalizzarle in base alle esigenze dell'applicazione.

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": [
                "*"
            ]
        }
    ]
}
```

Dimensionamento di un cluster DAX

Esistono sono due opzioni per il dimensionamento di un cluster DAX. La prima opzione è dimensionamento orizzontale, dove aggiungi le repliche di lettura al cluster. La seconda opzione è dimensionamento verticale, dove selezioni diversi tipi di nodo. Per informazioni su come approcciare la scelta della dimensione del cluster e del tipo di nodo appropriati per l'applicazione, consulta Guida alle dimensioni del cluster DAX.

Dimensionamento orizzontale

Con il dimensionamento orizzontale puoi migliorare il throughput delle operazioni di lettura aggiungendo più repliche di lettura al cluster. Un singolo cluster DAX supporta fino a 10 repliche di lettura e puoi aggiungere o rimuovere le repliche mentre il cluster è in esecuzione.

Quando si aggiunge un nuovo nodo, è necessario sincronizzare i dati della cache da un nodo peer. Pertanto, il tempo di aggiunta varia in base alla dimensione della cache e al carico di lavoro dell'applicazione. Come best practice, si consiglia di pre-scalare il cluster per soddisfare i picchi di traffico previsti. Per informazioni sulle linee guida per il corretto dimensionamento e i consigli di monitoraggio, consulta Guida alle dimensioni del cluster DAX.

AWS CLI Gli esempi seguenti mostrano come aumentare o diminuire il numero di nodi. L'argomento `--new-replication-factor` specifica il numero totale di nodi nel cluster. Uno è il nodo primario e gli altri nodi sono repliche di lettura.

```
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**
>
> Lo stato del cluster cambia in `modifying` quando modifichi il fattore di replica. Lo stato cambia in `available` quando la modifica è completata.

Dimensionamento verticale

Se hai un ampio set di dati funzionanti, l'applicazione potrebbe trarre vantaggio dall'utilizzo dei tipi di nodi più grandi. I nodi più grandi possono consentire al cluster di memorizzare più dati in memoria, riducendo i mancati riscontri nella cache e migliorando le prestazioni complessive dell'applicazione. Tutti i nodi in un cluster DAX devono essere dello stesso tipo.

Se il cluster DAX ha un tasso elevato di operazioni di scrittura o mancati riscontri nella cache, l'applicazione potrebbe anche trarre vantaggio dall'utilizzo di tipi di nodi più grandi. Le operazioni di scrittura e i mancati riscontri nella cache utilizzano le risorse sul nodo primario del cluster. Pertanto, l'utilizzo di tipi di nodi più grandi potrebbe aumentare le prestazioni del nodo primario e quindi consentire un throughput maggiore per questi tipi di operazioni.

Non è possibile modificare i tipi di nodo su un cluster DAX in esecuzione. Invece, devi creare un nuovo cluster con il tipo di nodo desiderato. Per un elenco dei tipi di nodo supportati, consulta Nodi.

È possibile creare un nuovo cluster DAX utilizzando il Console di gestione AWS, CloudFormation AWS CLI, o l'AWS SDK. (Per AWS CLI, utilizzate il `--node-type` parametro per specificare il tipo di nodo.)

Personalizzazione delle impostazioni del cluster DAX

Quando si crea un cluster DAX, vengono utilizzate le seguenti impostazioni predefinite:

- Pulizia della cache automatica abilitata con durata (TTL, Time to Live) di 5 minuti
- Nessuna preferenza per le zone di disponibilità
- Nessuna preferenza per le finestre di manutenzione
- Notifiche disabilitate

Per i nuovi cluster, puoi personalizzare le impostazioni al momento della creazione. Per eseguire questa operazione in Console di gestione AWS, deseleziona Use default settings (Usa impostazioni predefinite) per modificare le impostazioni seguenti:

- Rete e sicurezza: consente di eseguire singoli nodi del cluster DAX in diverse zone di disponibilità all'interno della regione corrente AWS . Se scegli No Preference (Nessuna preferenza), i nodi vengono distribuiti automaticamente tra le zone di disponibilità.
- Gruppo di parametri: un set denominato di parametri che vengono applicati a ogni nodo nel cluster. Puoi utilizzare un gruppo di parametri per specificare il comportamento TTL della cache. Puoi modificare il valore di un determinato parametro all'interno di un gruppo di parametri (ad eccezione del gruppo di parametri predefinito `default.dax.1.0`) in qualsiasi momento.
- Finestra di manutenzione: un periodo di tempo settimanale durante il quale vengono applicati aggiornamenti software e patch ai nodi del cluster. Puoi scegliere il giorno di inizio, l'ora di inizio e la durata della finestra di manutenzione. Se scegli No Preference (Nessuna preferenza), la finestra di manutenzione viene selezionata a caso da un blocco di tempo di 8 ore per regione. Per ulteriori informazioni, consulta Finestra di manutenzione.

> **Note**
>
> Il gruppo di parametri e la finestra di manutenzione possono essere modificati in qualsiasi momento in un cluster in esecuzione.

Quando si verifica un evento di manutenzione, DAX può inviare una notifica tramite Amazon Simple Notification Service (Amazon SNS). Per configurare le notifiche, scegli un'opzione dal selezionatore Argomento per notifica SNS. Puoi creare un nuovo argomento Amazon SNS o usarne uno esistente.

Per informazioni sulla configurazione e la sottoscrizione di un argomento Amazon SNS, consulta Nozioni di base su Amazon SNS nella Guida per gli sviluppatori di Amazon Simple Notification Service.

Configurazione delle impostazioni TTL

DAX gestisce due cache per i dati che legge da DynamoDB:

- Cache di elementi: per gli elementi recuperati tramite `GetItem` o `BatchGetItem`.
- Cache di query: per i set di risultati recuperati tramite `Query` o `Scan`.

Per ulteriori informazioni, consultare Cache degli elementi e Cache delle query.

Il TTL predefinito per ciascuna di queste cache è di 5 minuti. Se si desidera utilizzare diverse impostazioni TTL, è possibile avviare un cluster DAX usando un gruppo di parametri personalizzato. Per eseguire questa operazione nella console, scegli DAX | Parameter groups (Gruppi di parametri) nel pannello di navigazione.

Puoi eseguire queste attività anche utilizzando l' AWS CLI. L'esempio seguente mostra come avviare un nuovo cluster DAX utilizzando un gruppo di parametri personalizzato. In questo esempio, il TTL della cache degli elementi è impostato su 10 minuti e il TTL della cache delle query è impostato su 3 minuti.

1. Crea un nuovo set di parametri.

```
aws dax create-parameter-group \
    --parameter-group-name custom-ttl
```

2. Imposta il TTL della cache degli elementi su 10 minuti (600000 millisecondi).

```
aws dax update-parameter-group \
    --parameter-group-name custom-ttl \
    --parameter-name-values "ParameterName=record-ttl-millis,ParameterValue=600000"
```

3. Imposta il TTL della cache delle query su 3 minuti (180000 millisecondi).

```
aws dax update-parameter-group \
    --parameter-group-name custom-ttl \
    --parameter-name-values "ParameterName=query-ttl-millis,ParameterValue=180000"
```

4. Verifica che il parametri siano impostati correttamente.

```
aws dax describe-parameters --parameter-group-name custom-ttl \
    --query "Parameters[*].[ParameterName,Description,ParameterValue]"
```

A questi punto è possibile avviare un nuovo cluster DAX con questo gruppo di parametri.

```
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**
>
> Non è possibile modificare un gruppo di parametri che viene utilizzato da un'istanza DAX in esecuzione.

Supporto per l'assegnazione di tag per DAX

Molti AWS servizi, incluso DynamoDB, supportano il tagging, la capacità di etichettare le risorse con nomi definiti dall'utente. È possibile assegnare tag ai cluster DAX, in modo da identificare rapidamente tutte le AWS risorse che hanno lo stesso tag, o per classificare le fatture in base ai tag assegnati. AWS

Per ulteriori informazioni, consulta Aggiunta di tag ed etichette alle risorse in DynamoDB.

Utilizzando il Console di gestione AWS

Come gestire i tag del cluster DAX

1. Apri la console DynamoDB all'indirizzo. https://console.aws.amazon.com/dynamodb/
2. Nel riquadro di navigazione, in DAX, scegli Clusters (Cluster).
3. Scegli il cluster che desideri utilizzare.
4. Selezionare la scheda Tag. In questa scheda puoi aggiungere, elencare, modificare o eliminare i tag.

   Dopo aver selezionato le impostazioni desiderate, scegli Applica modifiche.

Usando la AWS CLI

Quando utilizzi AWS CLI per gestire i tag del cluster DAX, devi prima determinare l'Amazon Resource Name (ARN) per il cluster. L'esempio seguente mostra come determinare l'ARN per un cluster denominato `MyDAXCluster`.

```
aws dax describe-clusters \
    --cluster-name MyDAXCluster \
```

```
    --query "Clusters[*].ClusterArn"
```

Nell'output, l'aspetto dell'ARN sarà simile al seguente: `arn:aws:dax:us-west-2:123456789012:cache/MyDAXCluster`

L'esempio seguente mostra come aggiungere i tag al cluster.

```
aws dax tag-resource  \
    --resource-name arn:aws:dax:us-west-2:123456789012:cache/MyDAXCluster \
    --tags="Key=ClusterUsage,Value=prod"
```

Elenca tutti i tag per un cluster.

```
aws dax list-tags \
    --resource-name arn:aws:dax:us-west-2:123456789012:cache/MyDAXCluster
```

Per rimuovere un tag, specifica la chiave.

```
aws dax untag-resource  \
    --resource-name arn:aws:dax:us-west-2:123456789012:cache/MyDAXCluster \
    --tag-keys ClusterUsage
```

AWS CloudTrail integrazione

DAX è integrato con AWS CloudTrail e consente di controllare le attività del cluster DAX. È possibile utilizzare CloudTrail i log per visualizzare tutte le modifiche apportate a livello di cluster. Puoi anche visualizzare le modifiche ai componenti del cluster come nodi, gruppi di sottoreti e gruppi di parametri. Per ulteriori informazioni, consulta Registrazione delle operazioni di DynamoDB con AWS CloudTrail.

Eliminazione di un cluster DAX

Se non si utilizza più un cluster DAX, sarà necessario eliminarlo per evitare di ricevere addebiti per le risorse inutilizzate.

È possibile eliminare un cluster DAX utilizzando la console o la AWS CLI. Di seguito è riportato un esempio di :

```
aws dax delete-cluster --cluster-name mydaxcluster
```

Monitoraggio di DynamoDB Accelerator

Il monitoraggio è una parte importante per mantenere l'affidabilità, la disponibilità e le prestazioni di Amazon DynamoDB Accelerator (DAX) e delle tue soluzioni. AWS È necessario raccogliere i dati di monitoraggio da tutte le parti della AWS soluzione in modo da poter eseguire più facilmente il debug di un errore multipunto, se si verifica.

Prima di iniziare il monitoraggio di DAX, è opportuno creare un piano di monitoraggio che includa le risposte alle seguenti domande:

- Quali sono gli obiettivi del monitoraggio?
- Di quali risorse si intende eseguire il monitoraggio?
- Con quale frequenza sarà eseguito il monitoraggio di queste risorse?
- Quali strumenti di monitoraggio verranno utilizzati?
- Chi eseguirà i processi di monitoraggio?
- Chi deve ricevere una notifica quando si verifica un problema?

Argomenti

- Strumenti di monitoraggio per DynamoDB Accelerator
- Monitoraggio con Amazon CloudWatch
- Registrazione delle operazioni di DAX con AWS CloudTrail

Strumenti di monitoraggio per DynamoDB Accelerator

AWS fornisce strumenti che è possibile utilizzare per monitorare Amazon DynamoDB Accelerator (DAX). Puoi configurare alcuni di questi strumenti in modo che eseguano automaticamente il monitoraggio, mentre altri richiedono l'intervento manuale. Si consiglia di automatizzare il più possibile i processi di monitoraggio.

Argomenti

- Strumenti di monitoraggio automatici
- Strumenti di monitoraggio manuali

Strumenti di monitoraggio automatici

Per tenere sotto controllo DAX e segnalare l'eventuale presenza di problemi, puoi usare i seguenti strumenti di monitoraggio automatici:

- Amazon CloudWatch Alarms: monitora una singola metrica in un periodo di tempo specificato ed esegui una o più azioni in base al valore della metrica rispetto a una determinata soglia in diversi periodi di tempo. L'azione è una notifica inviata a un argomento di Amazon Simple Notification Service (Amazon SNS) o a una politica di Amazon Auto EC2 Scaling. CloudWatch gli allarmi non richiamano azioni semplicemente perché si trovano in uno stato particolare; lo stato deve essere cambiato e mantenuto per un determinato numero di periodi. Per ulteriori informazioni, consulta Monitoraggio di metriche in DynamoDB con Amazon CloudWatch.
- Amazon CloudWatch Logs: monitora, archivia e accedi ai tuoi file di registro da AWS CloudTrail o altre fonti. Per ulteriori informazioni, consulta Monitoring Log Files nella Amazon CloudWatch User Guide.
- Amazon CloudWatch Events: abbina gli eventi e li indirizza a una o più funzioni o stream di destinazione per apportare modifiche, acquisire informazioni sullo stato e intraprendere azioni correttive. Per ulteriori informazioni, consulta What is Amazon CloudWatch Events nella Amazon CloudWatch User Guide.
- AWS CloudTrail Monitoraggio dei log: condividi i file di CloudTrail registro tra account, monitora i file di registro in tempo reale inviandoli a CloudWatch Logs, scrivi applicazioni di elaborazione dei log in Java e verifica che i file di registro non siano cambiati dopo la consegna da parte di. CloudTrail Per ulteriori informazioni, consulta Lavorare con i file di CloudTrail registro nella Guida per l'AWS CloudTrail utente.

Strumenti di monitoraggio manuali

Un'altra parte importante del monitoraggio di DAX consiste nel monitorare manualmente gli elementi non coperti CloudWatch dagli allarmi. Il DAX e altri Console di gestione AWS dashboard forniscono una at-a-glance panoramica dello stato dell'ambiente. CloudWatch Trusted Advisor AWS Ti consigliamo anche di controllare i file di log in DAX.

- Il pannello di controllo DAX mostra quanto segue:
  - Integrità del servizio
- La CloudWatch home page mostra quanto segue:
  - Stato e allarmi attuali

- Grafici degli allarmi e delle risorse
- Stato di integrità dei servizi

Inoltre, è possibile utilizzare CloudWatch per effettuare le seguenti operazioni:

- Crea pannelli di controllo personalizzati per monitorare i servizi rilevanti.
- Crea grafici dei dati dei parametri per la risoluzione di problemi e il rilevamento di tendenze.
- Cerca e sfoglia tutte le metriche delle tue AWS risorse.
- Crea e modifica gli allarmi per ricevere le notifiche dei problemi.

Monitoraggio con Amazon CloudWatch

Puoi monitorare DynamoDB Accelerator (DAX) utilizzando CloudWatch Amazon, che raccoglie ed elabora i dati grezzi da DAX in metriche leggibili quasi in tempo reale. Queste statistiche vengono registrate per un periodo di due settimane. Puoi accedere alle informazioni storiche per una prospettiva migliore sulle prestazioni del servizio o dell'applicazione Web. Per impostazione predefinita, i dati metrici DAX vengono inviati automaticamente a. CloudWatch Per ulteriori informazioni, consulta What Is Amazon CloudWatch? nella Amazon CloudWatch User Guide.

Argomenti

- Come posso utilizzare utilizzano i parametri DAX?
- Visualizzazione di parametri e dimensioni DAX
- Creazione di CloudWatch allarmi per monitorare DAX
- Monitoraggio della produzione

Come posso utilizzare utilizzano i parametri DAX?

I parametri riportati da DAX forniscono informazioni che possono essere analizzate in diversi modi. L'elenco seguente mostra alcuni usi comuni dei parametri. Questi suggerimenti rappresentano solo nozioni di base e non costituiscono un elenco completo.

In che modo?	Parametri rilevanti
Posso determinare se si sono verificati errori di sistema	Monitora `FaultRequestCount` per determinare se qualche richiesta ha causato un codice HTTP 500 (errore del server). Ciò

In che modo?	Parametri rilevanti
	può indicare un errore interno del servizio DAX o un HTTP 500 nella SystemErrors metrica della tabella sottostante.
Posso determinare se si sono verificati errori dell'utente	Monitora `ErrorRequestCount` per determinare se qualche richiesta ha causato un codice HTTP 400 (errore del client). Se vedi crescere il conteggio degli errori, potresti voler indagare e assicurarti di inviare richieste client corrette.
Posso determinare se si sono verificati mancati riscontri nella cache	Monitora `ItemCacheMisses` per determinare il numero di volte in cui un elemento non è stato trovato nella cache e `QueryCacheMisses` e `ScanCacheMisses` per determinare il numero di volte in cui il risultato di una query o una scansione non sono è trovato nella cache.
Monitora l'hit rate della cache	Usa CloudWatch Metric Math per definire una metrica del tasso di accesso alla cache utilizzando espressioni matematiche. Ad esempio, per la cache degli elementi, è possibile utilizzare l'espressione m1/SUM([m1, m2])*100, dove m1 è il parametro `ItemCacheHits` e m2 è il parametro `ItemCacheMisses` per il cluster. Per le cache di query e scansioni, puoi seguire lo stesso modello utilizzando il parametro della cache corrispon dente per query e scansioni.

Visualizzazione di parametri e dimensioni DAX

Quando interagisci con Amazon DynamoDB, invia parametri e dimensioni ad Amazon. CloudWatch Puoi utilizzare le procedure riportate di seguito per visualizzare i parametri per DynamoDB Accelerator (DAX).

Come visualizzare i parametri (console)

I parametri vengono raggruppati prima in base allo spazio dei nomi del servizio e successivamente in base alle diverse combinazioni di dimensioni all'interno di ogni spazio dei nomi.

1. Apri la console all' CloudWatch indirizzo. https://console.aws.amazon.com/cloudwatch/
2. Nel riquadro di navigazione, seleziona Parametri.

3. Seleziona lo spazio dei nomi DAX.

Come visualizzare i parametri (AWS CLI)

- Al prompt dei comandi utilizza il comando seguente.

```
aws cloudwatch list-metrics --namespace "AWS/DAX"
```

Parametri e dimensioni di DAX

Le seguenti sezioni contengono le metriche e le dimensioni a cui vengono inviate le informazioni da DAX. CloudWatch

Parametri DAX

In DAX sono disponibili i parametri seguenti. DAX invia le metriche a CloudWatch solo quando hanno un valore diverso da zero.

> **Note**
>
> CloudWatch aggrega le seguenti metriche DAX a intervalli di 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

Non tutte le statistiche, come `Average` o `Sum`, si applicano a tutti i parametri. Tuttavia, tutti questi valori sono disponibili tramite la console DAX o utilizzando la console o per tutte le CloudWatch metriche. AWS CLI AWS SDKs Nella tabella seguente ciascun parametro presenta un elenco di statistiche valide applicabile a quel parametro.

Metrica	Description
`CPUUtilization`	La percentuale di utilizzo CPU del nodo o del cluster.

Metrica	Description
	Unità: `Percent` Statistiche valide: • `Minimum` • `Maximum` • `Average`
`CacheMemoryUtilization`	Percentuale di memoria cache disponibile utilizzata dalla cache degli elementi e dalla cache delle query nel nodo o nel cluster. I dati memorizzati nella cache iniziano a essere rimossi prima che l'utilizzo della memoria raggiunga il 100% (vedere il parametro `EvictedSize`). Se `CacheMemoryUtilization` raggiunge il 100% su qualsiasi nodo, le richieste di scrittura saranno limitate e si dovrebbe prendere in considerazione il passaggio a un cluster con un tipo di nodo di dimensioni maggiori. Unità: `Percent` Statistiche valide: • `Minimum` • `Maximum` • `Average`
`NetworkBytesIn`	Il numero di byte ricevuti dal nodo o dal cluster su tutte le interfacce di rete. Unità: `Bytes` Statistiche valide: • `Minimum` • `Maximum` • `Average`

Metrica	Description
`NetworkBytesOut`	Il numero di byte inviati dal nodo o dal cluster su tutte le interfacce di rete. Questo parametro identifica il volume del traffico in uscita in termini di numero di byte su un singolo nodo o cluster. Unità: `Bytes` Statistiche valide: • `Minimum` • `Maximum` • `Average`
`BaselineNetworkBytesInUtilization`	La percentuale della larghezza di banda di rete di base utilizzata in un dato momento per il traffico in ingresso. A titolo di riferimento, il 50% significa che viene utilizzata metà della larghezza di banda di rete disponibile per il traffico in ingresso. Unità: `Percent` Statistiche valide: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`

Metrica	Description
`BaselineNetworkBytesOutUtilization`	La percentuale della larghezza di banda di rete di base utilizzata in un dato momento per il traffico in uscita. A titolo di riferimento, il 50% significa che viene utilizzata metà della larghezza di banda di rete disponibile per il traffico in uscita. Unità: `Percent` Statistiche valide: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`
`NetworkPacketsIn`	Il numero di pacchetti ricevuti dal nodo o dal cluster su tutte le interfacce di rete. Unità: `Count` Statistiche valide: • `Minimum` • `Maximum` • `Average`

Metrica	Description
`NetworkPacketsOut`	Il numero di pacchetti inviati dal nodo o dal cluster su tutte le interfacce di rete. Questo parametro identifica il volume del traffico in uscita in termini di numero di pacchetti su un singolo nodo o cluster. Unità: `Count` Statistiche valide: • `Minimum` • `Maximum` • `Average`
`GetItemRequestCount`	Il numero di richieste `GetItem` gestite dal nodo o dal cluster. Unità: `Count` Statistiche valide: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`

Metrica	Description
`BatchGetItemRequestCount`	Il numero di richieste `BatchGetItem` gestite dal nodo o dal cluster. Unità: `Count` Statistiche valide: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`
`BatchWriteItemRequestCount`	Il numero di richieste `BatchWriteItem` gestite dal nodo o dal cluster. Unità: `Count` Statistiche valide: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`

Metrica	Description
`DeleteItemRequestCount`	Il numero di richieste `DeleteItem` gestite dal nodo o dal cluster. Unità: `Count` Statistiche valide: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`
`PutItemRequestCount`	Il numero di richieste `PutItem` gestite dal nodo o dal cluster. Unità: `Count` Statistiche valide: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`

Metrica	Description
`UpdateItemRequestCount`	Il numero di richieste `UpdateItem` gestite dal nodo o dal cluster. Unità: `Count` Statistiche valide: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`
`TransactWriteItemsCount`	Il numero di richieste `TransactWriteItems` gestite dal nodo o dal cluster. Unità: `Count` Statistiche valide: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`

Metrica	Description
`TransactGetItemsCount`	Il numero di richieste `TransactGetItems` gestite dal nodo o dal cluster. Unità: `Count` Statistiche valide: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`
`ItemCacheHits`	Il numero di volte in cui un elemento è stato restituito dalla cache dal nodo o dal cluster. Unità: `Count` Statistiche valide: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`

Metrica	Description
`ItemCacheMisses`	Il numero di volte in cui un elemento non si trovava nella cache del nodo o del cluster ed è stato necessario recuperarlo da DynamoDB. Unità: `Count` Statistiche valide: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`
`QueryCacheHits`	Il numero di volte in cui un risultato della query è stato restituito dalla cache dal nodo o dal cluster. Unità: `Count` Statistiche valide: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`

Metrica	Description
`QueryCacheMisses`	Il numero di volte in cui un risultato della query non si trovava nella cache del nodo o del cluster ed è stato necessario recuperarlo da DynamoDB. Unità: `Count` Statistiche valide: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`
`ScanCacheHits`	Il numero di volte in cui un risultato della scansione è stato restituito dalla cache dal nodo o dal cluster. Unità: `Count` Statistiche valide: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`

Metrica	Description
`ScanCacheMisses`	Il numero di volte in cui un risultato della scansione non si trovava nella cache del nodo o del cluster ed è stato necessario recuperarlo da DynamoDB. Unità: `Count` Statistiche valide: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`
`TotalRequestCount`	Il numero di richieste gestite dal nodo o dal cluster. Unità: `Count` Statistiche valide: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`

Metrica	Description
`ErrorRequestCount`	Il numero totale di richieste che hanno provocato un errore utente segnalato dal nodo o dal cluster. Sono incluse le richieste che sono state limitate dal nodo o dal cluster. Unità: `Count` Statistiche valide: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`
`ThrottledRequestCount`	Il numero di richieste limitate dal nodo o dal cluster. Le richieste che sono state limitate da DynamoDB non sono incluse e possono essere monitorate utilizzandoi parametri di DynamoDB. Unità: `Count` Statistiche valide: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`

Metrica	Description
`FaultRequestCount`	Il numero totale di richieste che hanno provocato un errore interno segnalato dal nodo o dal cluster. Unità: `Count` Statistiche valide: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`
`FailedRequestCount`	Il numero totale di richieste che hanno provocato un errore segnalato dal nodo o dal cluster. Unità: `Count` Statistiche valide: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`

Metrica	Description
`QueryRequestCount`	Il numero di richieste della query gestite dal nodo o dal cluster. Unità: `Count` Statistiche valide: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`
`ScanRequestCount`	Il numero di richieste di scansione gestite dal nodo o dal cluster. Unità: `Count` Statistiche valide: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`

Metrica	Description
`ClientConnections`	Il numero di connessioni simultanee effettuate dai client al nodo o al cluster. Unità: `Count` Statistiche valide: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`
`EstimatedDbSize`	Un'approssimazione della quantità di dati memorizzati nella cache degli elementi e della cache delle query dal nodo o dal cluster. Unità: `Bytes` Statistiche valide: • `Minimum` • `Maximum` • `Average`

Metrica	Description
`EvictedSize`	La quantità di dati rimossi dal nodo o dal cluster per fare spazio ai dati appena richiesti. Se il tasso di mancati riscontri aumenta, così come questo parametro, probabilmente significa che il tuo set di lavoro è aumentato. Si consiglia di passare a un cluster con un tipo di nodo di dimensioni maggiori. Unità: `Bytes` Statistiche valide: • `Minimum` • `Maximum` • `Average` • `Sum`
`CPUCreditUsage`	Il numero di crediti CPU spesi dal nodo per l'utilizzo della CPU. Un credito CPU equivale a una vCPU in esecuzione al 100% per un minuto o a una combinazione equivalente di vCPUs, utilizzo e tempo (ad esempio, una vCPU in esecuzione al 50% di utilizzo per due minuti o due v CPUs al 25% di utilizzo per due minuti). I parametri di credito CPU sono disponibili solo con una frequenza di 5 minuti. Se specifichi un periodo superiore a 5 minuti, usa la statistica `Sum` al posto di `Average`. Unità: `Credits (vCPU-minutes)` Statistiche valide: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`

Metrica	Description
`CPUCreditBalance`	Il numero di crediti CPU ottenuti che un nodo ha accumulato da quando è stata lanciato o avviato. I crediti vengono accumulati nel saldo del credito dopo che sono stati ottenuti e rimossi dal saldo del credito una volta spesi. Il saldo del credito ha un limite massimo, determinato dalla dimensione del nodo DAX. Una volta che il limite viene raggiunto, tutti i nuovi crediti guadagnati vengono scartati. I crediti in `CPUCreditBalance` sono disponibili affinché il nodo li spenda per andare oltre l'utilizzo di base della CPU. Unità: `Credits (vCPU-minutes)` Statistiche valide: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`

Metrica	Description
`CPUSurplusCreditBalance`	Il numero di crediti extra spesi da un nodo DAX quando il rispettivo valore `CPUCreditBalance` è pari a zero. Il valore `CPUSurplusCreditBalance` viene saldato con i crediti CPU ottenuti. Se il numero dei crediti extra va oltre il numero massimo di crediti che un nodo può ottenere in un periodo di 24 ore, i crediti extra spesi eccedenti il limite comporteranno un addebito aggiuntivo. Unità: `Credits (vCPU-minutes)` Statistiche valide: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`

Metrica	Description
`CPUSurplusCreditsCharged`	Il numero di crediti extra spesi da un'istanza, che non sono saldati con i crediti CPU ottenuti e che pertanto incorrono in costi aggiuntivi. I crediti extra spesi vengono addebitati quando superano il numero massimo di crediti che un nodo può ottenere in un periodo di 24 ore. Quando il nodo viene terminato, i crediti extra spesi che hanno superato il limite vengono addebitati alla fine dell'ora. Unità: `Credits (vCPU-minutes)` Statistiche valide: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`

> **Note**
>
> I parametri `CPUCreditUsage`, `CPUCreditBalance`, `CPUSurplusCreditBalance` e `CPUSurplusCreditsCharged` sono disponibili solo per i nodi T3.

Dimensioni per i parametri DAX

I parametri per DAX sono qualificati dai valori per la combinazione di account, ID cluster o ID cluster e ID nodo. È possibile utilizzare la CloudWatch console per recuperare i dati DAX lungo una qualsiasi delle dimensioni nella tabella seguente.

Dimensione	Description
`Account`	Fornisce statistiche aggregate su tutti i nodi di un account.
`ClusterId`	Limita i dati a un cluster.
`ClusterId, NodeId`	Limita i dati a un nodo all'interno di un cluster.

Creazione di CloudWatch allarmi per monitorare DAX

Puoi creare un CloudWatch allarme Amazon che invia un messaggio Amazon Simple Notification Service (Amazon SNS) quando l'allarme cambia stato. Un allarme monitora un singolo parametro per un periodo di tempo specificato. L'allarme esegue una o più operazioni basate sul valore del parametro relativo a una soglia prestabilita per un certo numero di periodi. L'azione è una notifica che viene inviata a un argomento di Amazon SNS o a una politica di Amazon Auto EC2 Scaling. Gli allarmi richiamano azioni solo per modifiche di stato sostenute. CloudWatch gli allarmi non richiamano azioni semplicemente perché si trovano in uno stato particolare. Lo stato deve essere cambiato e restare costante per un numero specificato di periodi.

Come mi vengono notificati i mancati riscontri nella cache delle query?

1. Crea un argomento Amazon SNS, `arn:aws:sns:us-west-2:522194210714:QueryMissAlarm`.

   Per ulteriori informazioni, consulta Configurare Amazon Simple Notification Service nella Amazon CloudWatch User Guide.

2. Crea l'allarme.

```
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. Testa l'allarme.

```
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**
>
> È possibile aumentare o diminuire la soglia in modo che soddisfi le esigenze dell'applicazione. Puoi anche utilizzare CloudWatch Metric Math per definire una metrica del tasso di mancata risposta alla cache e impostare un allarme in base a tale metrica.

Come mi vengono notificate le richieste che causano errori interni in un cluster?

1. Crea un argomento Amazon SNS, `arn:aws:sns:us-west-2:123456789012:notify-on-system-errors`.

   Per ulteriori informazioni, consulta Configurare Amazon Simple Notification Service nella Amazon CloudWatch User Guide.

2. Crea l'allarme.

```
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. Testa l'allarme.

```
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
```

Monitoraggio della produzione

Devi stabilire una baseline per le prestazioni normali di DAX nell'ambiente, misurando le prestazioni in diversi momenti e con condizioni di carico differenti. Quando esegui il monitoraggio di DAX, dovresti considerare di archiviare i dati storici sul monitoraggio. Questi dati archiviati forniscono una baseline rispetto cui confrontare i dati sulle prestazioni correnti e identificare i normali modelli o le anomalie di prestazioni e ideare metodi per risolvere i problemi.

Per stabilire una baseline, devi monitorare almeno gli elementi seguenti durante il test di carico e in produzione.

- Utilizzo della CPU e richieste limitate, in modo da poter determinare se è necessario usare un tipo di nodo più grande nel cluster. L'utilizzo della CPU del cluster è disponibile tramite la `CPUUtilization` CloudWatch metrica. La statistica media di questa metrica fornisce una visualizzazione dell'utilizzo medio della CPU su tutti i nodi del cluster. Per le decisioni sulla scalabilità del cluster, si consiglia di utilizzare la statistica massima, che rappresenta l'utilizzo massimo su tutti i nodi.

> **Note**
>
> AWS ha migliorato la granularità della `CPUUtilization` metrica. Si potrebbero osservare le modifiche alla metrica a partire dal 17/05/2024 al 22/06/2024.

- La latenza delle operazioni (come misura sul lato client) deve rimanere coerente entro i requisiti di latenza dell'applicazione.
- I tassi di errore dovrebbero rimanere bassi, come si evince dalle metriche `ErrorRequestCount`Fault`RequestCount`, and. `FailedRequestCount` CloudWatch
- Utilizzo di byte di rete, in modo da poter determinare se utilizzare più nodi o un tipo di nodo più grande nel cluster. Per monitorare il consumo, puoi impostare gli avvisi `BaselineNetworkBytesInUtilization` e le `BaselineNetworkBytesOutUtilization` metriche disponibili in CloudWatch, che indicano il consumo percentuale della larghezza di banda di rete disponibile per il tipo di istanza, rispettivamente per il traffico in ingresso e in uscita.
- Utilizzo della memoria cache e dimensione eliminata, in modo da poter determinare se il tipo di nodo del cluster ha memoria sufficiente per contenere il set di lavoro e, nel caso passare a un tipo di nodo più grande.

> **Note**
>
> In caso di un numero elevato di errori e scritture della cache, l'utilizzo della memoria cache può aumentare fino al 100% e causare interruzioni della disponibilità.

- Connessioni client, in modo da poter monitorare eventuali picchi imprevedibili nelle connessioni al cluster.

Registrazione delle operazioni di DAX con AWS CloudTrail

Amazon DynamoDB Accelerator (DAX) è integrato AWS CloudTrail con, un servizio che fornisce un registro delle azioni intraprese da un utente, un ruolo o un servizio in DAX. AWS

Per ulteriori informazioni su DAX e CloudTrail, consulta la sezione DynamoDB Accelerator (DAX) in. Registrazione delle operazioni di DynamoDB con AWS CloudTrail

Istanze espandibili T3/T2 DAX

DAX consente di scegliere tra istanze a prestazioni fisse (come R4, R5 e R7) e istanze a prestazioni espandibili (come T2 e T3). Le istanze espandibili forniscono un livello di base di prestazioni della CPU con la possibilità di superare questo livello temporaneamente.

Le prestazioni di base e la capacità di espansione sono regolate dai crediti CPU. Le istanze espandibili accumulano crediti CPU in modo continuo, a una velocità determinata dalla dimensione dell'istanza, quando il carico di lavoro è al di sotto della soglia di base. Questi crediti possono quindi essere consumati quando il carico di lavoro aumenta. Un credito CPU fornisce le prestazioni di un core CPU completo per un minuto.

Molti carichi di lavoro non necessitano di livelli di CPU costantemente elevati, ma beneficiano in modo significativo dell'accesso completo a CPU molto veloci quando ne hanno bisogno. Le istanze espandibili sono progettate specificamente per questi casi d'uso. Se sono necessarie prestazioni CPU costantemente elevate per il database, consigliamo di utilizzare istanze a prestazioni fisse.

Famiglia di istanze T2 DAX

Le istanze T2 DAX sono istanze per scopi generici espandibili che forniscono un livello di base di prestazioni della CPU e la possibilità di superare questo livello temporaneamente. Le istanze T2 sono una buona scelta per i carichi di lavoro di test e sviluppo che richiedono prevedibilità dei prezzi. Le istanze DAX T2 sono configurate per la modalità standard, il che significa che se l'istanza sta esaurendo i crediti accumulati, l'utilizzo della CPU viene gradualmente ridotto al livello di base. Per ulteriori informazioni sulla modalità standard, consulta Modalità standard per istanze a prestazioni espandibili nella Guida per l'utente di Amazon EC2.

Famiglia di istanze T3 DAX

Le istanze T3 DAX sono il tipo di istanza per scopi generici espandibili di nuova generazione che fornisce un livello base di prestazioni della CPU con la possibilità di espandere l'utilizzo della CPU in qualsiasi momento per tutto il tempo necessario. Le istanze T3 offrono un rapporto equilibrato tra calcolo, memoria e risorse di rete e sono ideali per i carichi di lavoro con un utilizzo moderato della CPU che presentano picchi temporanei in uso. Le istanze DAX T3 sono configurate per la modalità illimitata, il che significa che possono espandersi oltre la linea di base in una finestra di 24 ore a un costo aggiuntivo. Per ulteriori informazioni sulla modalità illimitata, consulta Modalità illimitata per istanze a prestazioni espandibili nella Guida per l'utente di Amazon EC2.

Le istanze DAX T3 possono sostenere elevate prestazioni della CPU fino a quando un carico di lavoro lo richiede. Per la maggior parte dei carichi di lavoro per scopi generici, le istanze T3 offrono prestazioni elevate senza costi aggiuntivi. Il prezzo orario delle istanze T3 copre automaticamente tutti i picchi provvisori di utilizzo quando l'utilizzo medio della CPU di un'istanza T3 corrisponde o è inferiore alla base in una finestra di 24 ore.

Ad esempio, una istanza `dax.t3.small` riceve crediti in modo continuo a una velocità di 24 crediti CPU all'ora. Questa funzionalità fornisce prestazioni di base equivalenti al 20% di un core CPU (20% × 60 minuti = 12 minuti). Se l'istanza non utilizza i crediti ricevuti, questi vengono memorizzati nel saldo del credito della CPU fino a un massimo di 576 crediti CPU. Quando l'istanza `t3.small` deve espandersi a più del 20% di un core, attinge dal suo saldo di credito della CPU per gestire automaticamente questo aumento.

Mentre le istanze DAX T2 sono limitate alle prestazioni di base una volta che il saldo del credito della CPU viene portato a zero, le istanze DAX T3 possono superare la base anche quando il saldo del credito della CPU è pari a zero. Per la stragrande maggioranza dei carichi di lavoro, in cui l'utilizzo medio della CPU corrisponde o è inferiore alle prestazioni di base, il prezzo orario di base per `t3.small`copre tutte le espansioni della CPU. Se l'istanza viene eseguita con un utilizzo medio della CPU del 25% (5% al di sopra della base) in un periodo di 24 ore dopo che il saldo del credito della CPU è stato portato a zero, verranno addebitati ulteriori 11,52 centesimi (9,6 centesimi/vCPU × 1 vCPU × 5% × 24 ore). Per i dettagli sui prezzi, consulta Prezzi di Amazon DynamoDB.

Controllo degli accessi DAX

DynamoDB Accelerator (DAX) è progettato per funzionare insieme a DynamoDB in modo da aggiungere senza problemi un livello di memorizzazione nella cache alle applicazioni. Tuttavia, DAX e DynamoDB hanno meccanismi di controllo degli accessi separati. Entrambi i servizi utilizzano AWS Identity and Access Management (IAM) per implementare le rispettive politiche di sicurezza, ma i modelli di sicurezza per DAX e DynamoDB sono diversi.

Consigliamo vivamente di informarsi su entrambi i modelli di sicurezza, in modo da poter implementare misure di sicurezza adeguate per le applicazioni che utilizzano DAX.

In questa sezione sono descritti i meccanismi di controllo degli accessi forniti da DAX e sono riportate le policy IAM di esempio che possono essere personalizzare in base alle esigenze.

Con DynamoDB è possibile creare policy IAM che limitano le operazioni che un utente può eseguire sulle singole risorse DynamoDB. Ad esempio, è possibile creare un ruolo utente che consenta

all'utente solo di eseguire operazioni di sola lettura su una particolare tabella DynamoDB. Per ulteriori informazioni, consulta Identity and Access Management per Amazon DynamoDB. In confronto, il modello di sicurezza DAX si concentra sulla sicurezza dei cluster e sulla capacità del cluster di eseguire operazioni API DynamoDB per conto tuo.

> **Warning**
>
> Se attualmente vengono utilizzati ruoli e policy IAM per limitare l'accesso ai dati delle tabelle DynamoDB, l'uso di DAX può sovvertire tali policy. Ad esempio, un utente potrebbe avere accesso a una tabella DynamoDB tramite DAX ma non avere l'accesso esplicito alla stessa tabella accedendo a DynamoDB direttamente. Per ulteriori informazioni, consulta Identity and Access Management per Amazon DynamoDB.
>
> DAX non applica la separazione a livello utente sui dati in DynamoDB. Gli utenti ereditano invece le autorizzazioni della policy IAM del cluster DAX quando accedono al cluster. Pertanto, quando si accede alle tabelle DynamoDB tramite DAX, gli unici controlli degli accessi in vigore sono le autorizzazioni nella policy IAM del cluster DAX. Non sono riconosciute altre autorizzazioni.
>
> Se è necessario l'isolamento, consigliamo di creare altri cluster DAX e definire di conseguenza l'ambito della policy IAM per ciascun cluster. Ad esempio, è possibile creare più cluster DAX e consentire a ciascun cluster di accedere solo a una singola tabella.

Ruolo di servizio IAM per DAX

Quando si crea un cluster DAX, è necessario associare il cluster a un ruolo IAM. conosciuto con il termine ruolo del servizio per il cluster.

Supponiamo di voler creare un nuovo cluster DAX denominato 01. DAXCluster È possibile creare un ruolo di servizio denominato DAXServiceRole e associarlo a 01. DAXCluster La policy for DAXServiceRole definirebbe le azioni DynamoDB DAXClusterche 01 potrebbe eseguire per conto degli utenti che interagiscono con 01. DAXCluster

Quando si crea un ruolo di servizio, è necessario specificare una relazione di fiducia tra DAXServiceRole e il servizio DAX stesso. Una relazione di trust determina quali entità possono assumere un ruolo e utilizzarne le autorizzazioni. Di seguito è riportato un esempio di documento di relazione di fiducia per DAXServiceRole:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "dax.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

Questa relazione di fiducia consente a un cluster DAX di assumere il DAXServiceruolo ed eseguire chiamate API DynamoDB per tuo conto.

Le azioni dell'API DynamoDB consentite sono descritte in un documento di policy IAM, allegato a Role. DAXService Di seguito è riportato un esempio di documento di policy:

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"
            ]
        }
    ]
}
```

Questa policy consente a DAX di eseguire tutte le operazioni API DynamoDB necessarie su una tabella DynamoDB. L'operazione `dynamodb:DescribeTable` è necessaria affinché DAX mantenga i metadati sulla tabella e le altre operazioni di lettura e scrittura eseguite sugli elementi della tabella. La tabella denominata `Books` si trova nella regione us-west-2 e appartiene all'ID account AWS `123456789012`.

> **Note**
>
> DAX supporta meccanismi per evitare il problema del “confused deputy” durante l’accesso multi-servizio. Per ulteriori informazioni, consultare Problema del "confused deputy" nella Guida per l'utente di IAM.

Policy IAM per consentire l'accesso del cluster DAX

Dopo aver creato un cluster DAX, sarà necessario concedere a un utente le autorizzazioni di accesso al cluster DAX.

Ad esempio, supponiamo di voler concedere l'accesso a DAXCluster01 a un utente di nome Alice. È innanzitutto necessario creare una policy IAM (AliceAccessPolicy) che definisca i cluster DAX e le azioni dell'API DAX a cui il destinatario può accedere. Quindi devi conferire l'accesso collegando questa policy all'utente Alice.

Il seguente documento di policy fornisce al destinatario l'accesso completo su 01. DAXCluster

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
```

```
        {
            "Action": [
                "dax:*"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:dax:us-west-2:123456789012:cache/DAXCluster01"
            ]
        }
    ]
}
```

Il documento di policy consente l'accesso al cluster DAX, ma non concede alcuna autorizzazione DynamoDB. Le autorizzazioni DynamoDB sono conferite dal ruolo del servizio DAX.

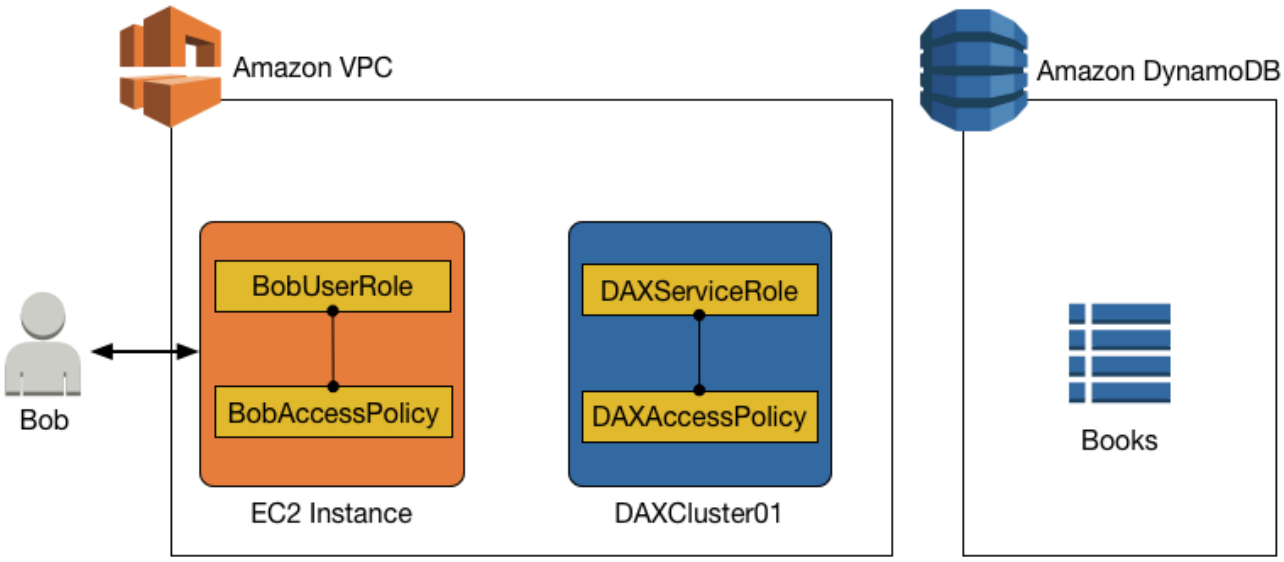
Per l'utente Alice, devi prima creare `AliceAccessPolicy` con il documento di policy mostrato in precedenza. Quindi devi collegare la policy ad Alice.

> **Note**
>
> Invece di collegare la policy a un utente, è possibile collegarla a un ruolo IAM. In questo modo, tutti gli utenti che assumono quel ruolo avranno le autorizzazioni definite nella policy.

La policy dell'utente, insieme al ruolo del servizio DAX, determina le risorse DynamoDB e le operazioni API a cui il destinatario può accedere tramite DAX.

Caso di studio: accesso a DynamoDB e DAX

Il seguente scenario può aiutare a comprendere meglio le policy IAM da utilizzare con DAX. Si farà riferimento a questo scenario nella parte restante di questa sezione. Il seguente diagramma mostra una panoramica di alto livello dello scenario.

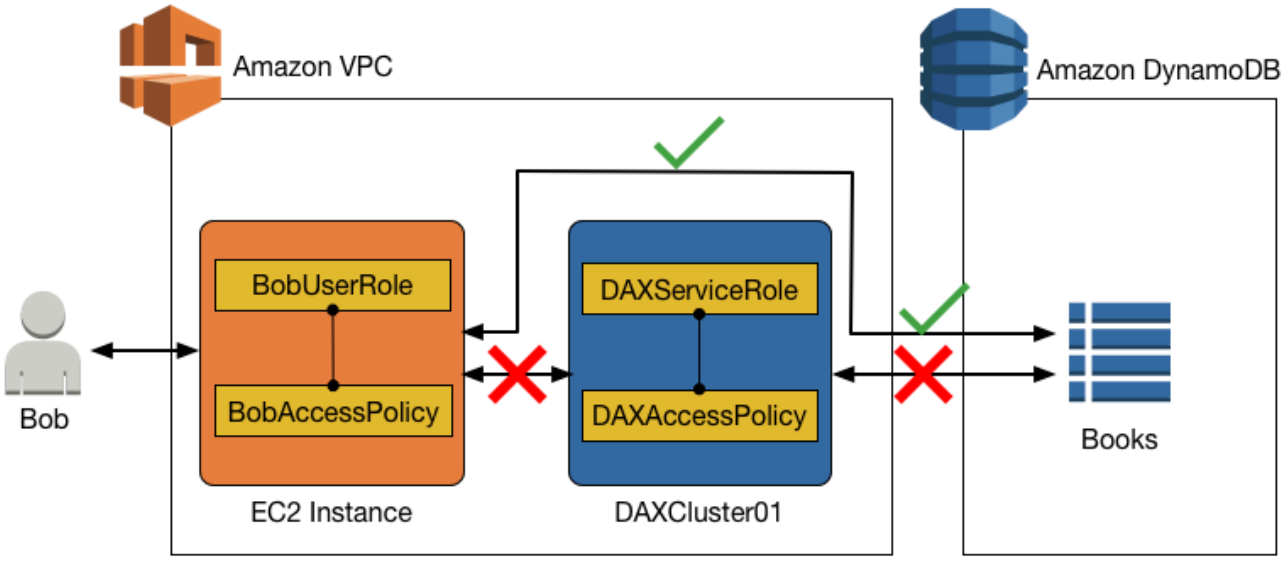


In questo scenario sono presenti le seguenti entità:

- Un utente (Bob).
- Un ruolo IAM (`BobUserRole`). Bob assume questo ruolo al runtime.
- Una policy IAM (`BobAccessPolicy`). Questa policy è collegata a `BobUserRole`. `BobAccessPolicy` definisce le risorse DynamoDB e DAX a cui `BobUserRole` può accedere.
- Un cluster DAX (`DAXCluster01`).
- Un ruolo del servizio IAM (`DAXServiceRole`). Tale ruolo consente a `DAXCluster01` di accedere a DynamoDB.
- Una policy IAM (`DAXAccessPolicy`). Questa politica è allegata a`DAXServiceRole`. `DAXAccessPolicy`definisce il APIs DynamoDB e le risorse a `DAXCluster01` cui è consentito accedere.
- Una tabella DynamoDB (`Books`).

La combinazione di dichiarazioni di policy in `BobAccessPolicy` e `DAXAccessPolicy` determina le operazioni che Bob può eseguire con la tabella `Books`. Ad esempio, Bob potrebbe essere in grado di accedere a `Books` direttamente (tramite l'endpoint DynamoDB), indirettamente (tramite il cluster DAX) o in entrambi i modi. Bob potrebbe anche essere in grado di leggere i dati da `Books`, scrivere i dati in `Books` o di eseguire entrambe le operazioni.

Accesso a DynamoDB, ma senza accesso con DAX



È possibile consentire l'accesso diretto a una tabella DynamoDB e impedire l'accesso indiretto usando un cluster DAX. Per l'accesso diretto a DynamoDB, le autorizzazioni per `BobUserRole` sono determinate da `BobAccessPolicy` (che è collegato al ruolo).

Accesso in sola lettura a DynamoDB (solo)

Bobpuò accedere a DynamoDB con`BobUserRole`. La policy IAM associata a questo ruolo (`BobAccessPolicy`) determina le tabelle DynamoDB a `BobUserRole` cui possono accedere e APIs cosa `BobUserRole` possono richiamare.

Considera il seguente documento di policy per `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"
        }
    ]
}
```

Quando questo documento è collegato a `BobAccessPolicy`, consente a `BobUserRole` di accedere all'endpoint DynamoDB ed eseguire le operazioni di sola lettura sulla tabella `Books`.

DAX non è presente in questa policy e pertanto l'accesso tramite DAX non è consentito.

Accesso in lettura e scrittura a DynamoDB (solo)

Se `BobUserRole` richiede read/write l'accesso a DynamoDB, la seguente policy funzionerebbe.

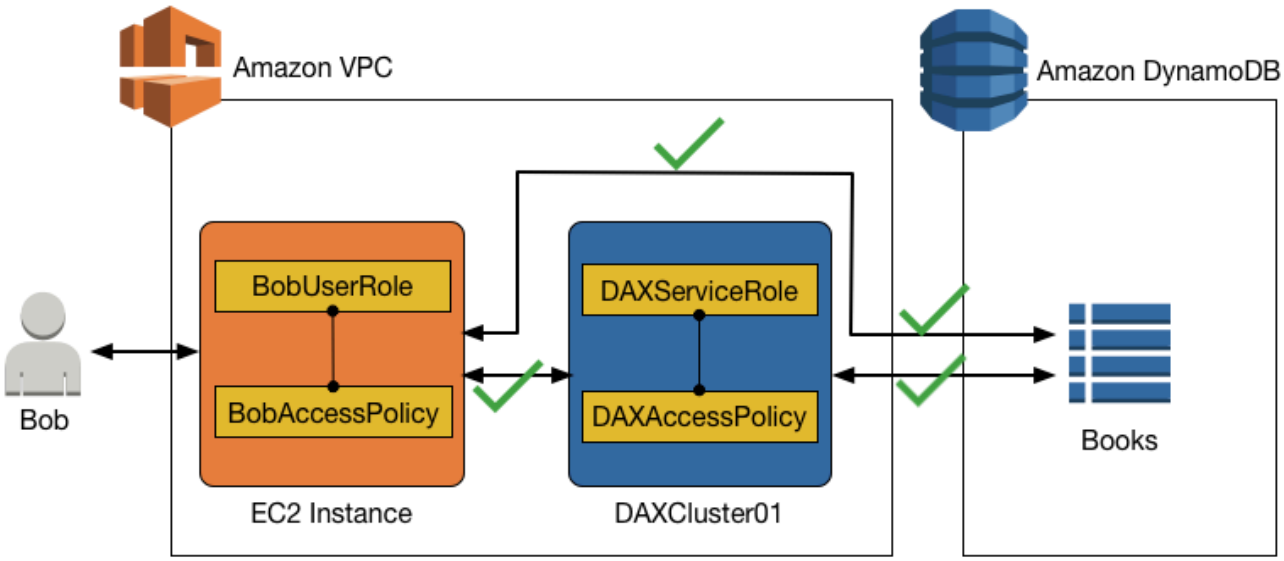
JSON

```
{
    "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"
        }
    ]
```

```
}
```

Di nuovo, DAX non è presente in questa policy e pertanto l'accesso tramite DAX non è consentito.

Accesso a DynamoDB e a DAX



Per consentire l'accesso a un cluster DAX, è necessario includere le operazioni specifiche di DAX in una policy IAM.

Le seguenti operazioni specifiche di DAX corrispondono alle controparti con nome simile dell'API DynamoDB:

- `dax:GetItem`
- `dax:BatchGetItem`
- `dax:Query`
- `dax:Scan`
- `dax:PutItem`
- `dax:UpdateItem`
- `dax:DeleteItem`
- `dax:BatchWriteItem`

- `dax:ConditionCheckItem`

Lo stesso vale per la chiave di condizione `dax:EnclosingOperation`.

Accesso in sola lettura a DynamoDB e accesso in sola lettura a DAX

Supponiamo che Bob richieda l'accesso in sola lettura alla tabella `Books` da DynamoDB e da DAX. La seguente policy (collegata a `BobUserRole`) conferisce questo accesso:

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 policy contiene una dichiarazione per DAX access (`DAXAccessStmt`) e un'altra dichiarazione per DBaccess Dynamo (). `DynamoDBAccessStmt` Le istruzioni consentono a Bob di inviare richieste `GetItem`, `BatchGetItem`, `Query` e `Scan` a `DAXCluster01`.

Tuttavia, anche il ruolo del servizio per `DAXCluster01` richiede l'accesso in sola lettura alla tabella `Books` in DynamoDB. La seguente policy IAM (collegata a `DAXServiceRole`) soddisfa questo 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"
        }
    ]
}
```

Accesso in lettura e scrittura a DynamoDB e in sola scrittura a DAX

Per un determinato ruolo utente, puoi fornire l' read/write accesso a una tabella DynamoDB, consentendo anche l'accesso in sola lettura tramite DAX.

Per Bob, la policy IAM per `BobUserRole` dovrebbe consentire le operazioni di lettura e scrittura DynamoDB sulla tabella `Books` e supportare anche le operazioni di sola lettura tramite `DAXCluster01`.

Il seguente documento di policy di esempio per `BobUserRole` conferisce questo accesso:

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"
        }
    ]
}
```

Inoltre, `DAXServiceRole` richiederebbe una policy IAM che consenta a `DAXCluster01` di eseguire operazioni di sola lettura sulla tabella `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"
        }
    ]
}
```

Read/write access to DynamoDB and read/writeaccesso a DAX

Supponiamo ora che Bob richieda read/write l'accesso alla `Books` tabella, direttamente da DynamoDB o indirettamente da. `DAXCluster01` Il seguente documento di policy, collegato a `BobAccessPolicy`, conferisce questo accesso.

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"
        }
    ]
}
```

Inoltre, `DAXServiceRole` richiederebbe una policy IAM che `DAXCluster01` consenta di eseguire read/write azioni sulla tabella. `Books`

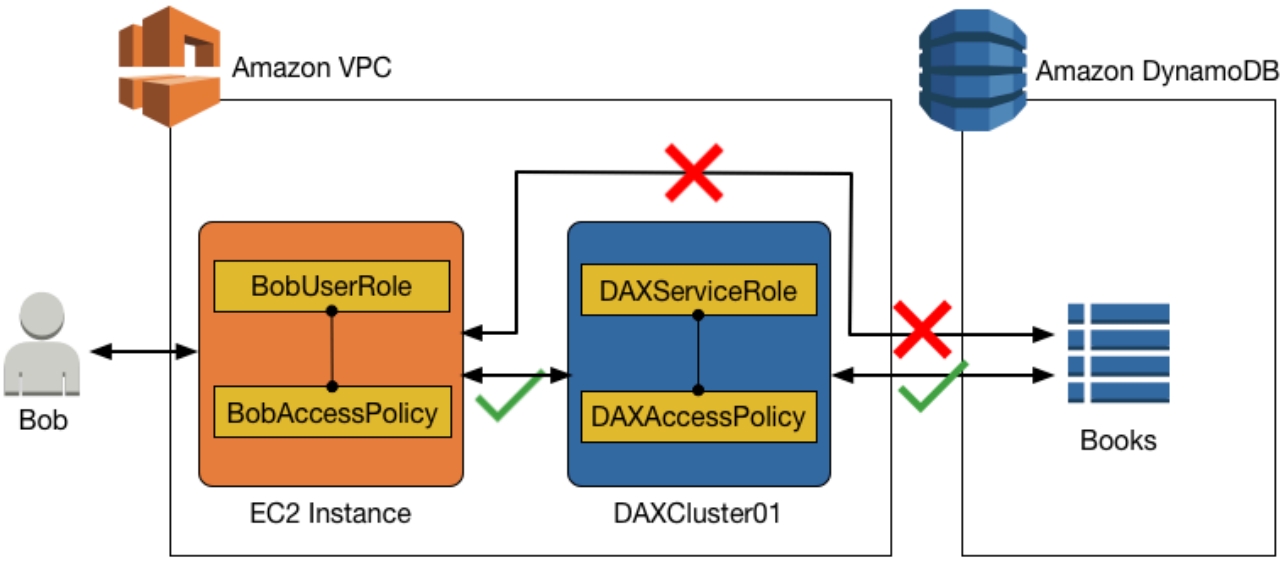
JSON

```
{
    "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"
        }
    ]
}
```

Accesso a DynamoDB tramite DAX, ma senza accesso diretto a DynamoDB

In questo scenario Bob può accedere alla tabella `Books` tramite DAX, ma non dispone dell'accesso diretto alla tabella `Books` in DynamoDB. Quindi, quando Bob ottiene l'accesso a DAX, ottiene anche l'accesso a una tabella DynamoDB a cui altrimenti potrebbe non essere in grado di accedere. Quando si configuri una policy IAM per il ruolo di servizio DAX, tenere presente che qualsiasi utente a cui viene dato l'accesso al cluster DAX tramite la policy di accesso utente ottiene l'accesso alle tabelle specificate in tale policy. In questo caso, `BobAccessPolicy` ottiene l'accesso alle tabelle specificate in `DAXAccessPolicy`.



Se stai attualmente utilizzando ruoli e policy IAM per limitare l'accesso a tabelle e dati DynamoDB, l'uso di DAX può sovvertire tali policy. Nella policy seguente, Bob dispone dell'accesso a una tabella DynamoDB tramite DAX, ma non dispone dell'accesso diretto esplicito alla stessa tabella in DynamoDB.

Il seguente documento di policy (`BobAccessPolicy`), collegato a `BobUserRole`, conferisce questo accesso.

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"
        }
    ]
}
```

In questa policy di accesso non ci sono autorizzazioni per accedere direttamente a DynamoDB.

Insieme a `BobAccessPolicy`, la seguente `DAXAccessPolicy` fornisce a `BobUserRole` l'accesso alla tabella DynamoDB `Books` anche se `BobUserRole` non può accedere direttamente alla tabella `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"
        }
    ]
}
```

Come mostra questo esempio, quando si configura il controllo di accesso per la politica di accesso degli utenti e la politica di accesso al cluster DAX, è necessario comprendere appieno l' end-to-endaccesso per garantire che venga rispettato il principio del privilegio minimo. Assicura, inoltre, che fornendo l'accesso utente a un cluster DAX, le policy di controllo degli accessi precedentemente stabilite non vengano sovvertite.

Crittografia DAX dei dati inattivi

La crittografia Amazon DynamoDB Accelerator (DAX) a riposo offre un livello aggiuntivo di protezione dei dati proteggendoli dagli accessi non autorizzati all'archiviazione sottostante. Le policy aziendali, le normative di settore e del governo e i requisiti di conformità potrebbero esigere l'uso della crittografia dei dati inattivi per proteggere i dati. Puoi usare la crittografia per aumentare la sicurezza dei dati delle applicazioni distribuite nel cloud.

Grazie alla crittografia a riposo, i dati conservati da DAX su disco vengono crittografati utilizzando lo standard di crittografia avanzata a 256 bit, noto anche come crittografia AES-256. DAX; scrive i dati sul disco durante la propagazione delle modifiche dal nodo primario alle repliche di lettura.

La crittografia DAX a riposo è integrata automaticamente con AWS Key Management Service (AWS KMS) per la gestione della singola chiave predefinita di servizio utilizzata per crittografare i cluster. Se non esiste una chiave predefinita di servizio quando crei il cluster DAX crittografato, AWS KMS crea automaticamente una nuova chiave gestita da AWS per conto tuo. da utilizzare con i cluster crittografati creati in futuro. AWS KMS combina hardware e software sicuri e a disponibilità elevata per offrire un sistema di gestione delle chiavi a misura di cloud.

Una volta crittografati i dati, DAX gestisce la decrittografia dei dati in modo trasparente e senza sacrificare le prestazioni. Non è quindi necessario modificare le applicazioni per utilizzare la crittografia.

> **Note**
>
> DAX non chiama AWS KMS per ogni singola operazione DAX. DAX usa la chiave solo all'avvio del cluster. Anche se l'accesso viene revocato, DAX può comunque accedere ai dati fino a quando il cluster non viene arrestato. Le chiavi AWS KMS specifiche dell'utente non sono supportate.

La crittografia DAX a riposo è disponibile per i tipi di nodi dei cluster seguenti.

Family	Tipo di nodo
Ottimizzato per la memoria (R4, R5 e 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

Family	Tipo di nodo
	dax.r7i.24xlarge
Uso generale (T2)	dax.t2.small dax.t2.medium
Uso generale (T3)	dax.t3.small dax.t3.medium

> **Important**
>
> La crittografia DAX a riposo non è supportata per i tipi di nodi `dax.r3.*`.

Non è possibile abilitare o disabilitare la crittografia dei dati inattivi dopo la creazione di un cluster. Se la crittografia dei dati inattivi non è stata abilitata in fase di creazione, è necessario creare nuovamente il cluster per abilitarla.

La crittografia DAX a riposo è disponibile senza costi aggiuntivi (si applicano solo i costi di utilizzo della chiave di crittografia AWS KMS). Per ulteriori informazioni sui prezzi, consulta Prezzi di Amazon DynamoDB.

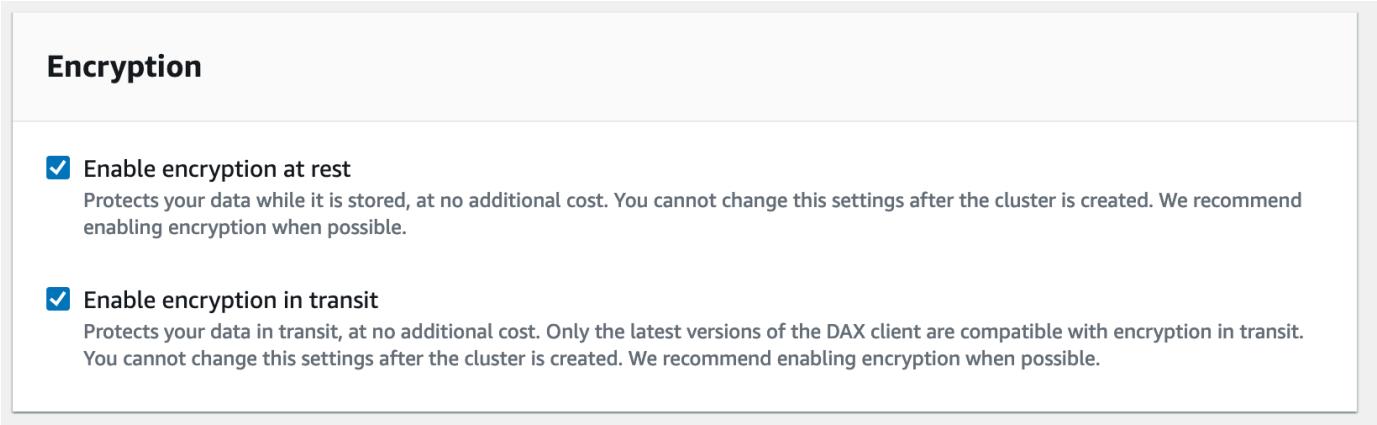
Abilitazione della crittografia dei dati inattivi mediante la Console di gestione AWS

Segui questa procedura per abilitare la crittografia DAX a riposo su una tabella utilizzando la console.

Come abilitare la crittografia DAX a riposo

1. Accedi alla console Console di gestione AWS e apri la console DynamoDB all'indirizzo https://console.aws.amazon.com/elasticache/.
2. Nel pannello di navigazione sul lato sinistro della console, in DAX, scegli Cluster.
3. Scegli Create cluster (Crea cluster).
4. In Nome cluster immettere un nome breve per il cluster. Scegli il tipo di nodo per tutti i nodi del cluster e utilizza **3** nodi come dimensione.

5. In Encryption verificare che l'opzione Enable encryption (Abilita crittografia) sia selezionata.



6. Dopo aver scelto il ruolo IAM, il gruppo di sottoreti, i gruppi di sicurezza e le impostazioni dei cluster, scegli Avvia cluster.

Per verificare che i cluster sia crittografato, controllare i dettagli nel riquadro Clusters (Cluster). La crittografia deve essere ENABLED (ABILITATA).

Crittografia DAX in transito

Amazon DynamoDB Accelerator (DAX) supporta la crittografia in transito dei dati tra la tua applicazione e il cluster DAX, consentendo di utilizzare DAX in applicazioni con requisiti di crittografia rigorosi.

Indipendentemente dal fatto che tu scelga o meno la crittografia in transito, il traffico tra la tua applicazione e il tuo cluster DAX rimane nel tuo Amazon VPC. Questo traffico viene instradato alle interfacce di rete elastiche con IP privati nel VPC collegati ai nodi del cluster. Con il VPC come limite di attendibilità, si ha un controllo significativo sulla sicurezza dei dati attraverso l'uso di strumenti standard come gruppi di sicurezza, segmentazione della sottorete con liste di controllo degli accessi di rete e traccia del flusso VPC. La crittografia DAX in transito si aggiunge a questo livello di riservatezza di base, garantendo che tutte le richieste e le risposte tra l'applicazione e il cluster siano crittografate tramite TLS (Transport Level Security) e che le connessioni al cluster possano essere autenticate mediante la verifica di un certificato cluster x509. I dati scritti su disco da DAX possono essere crittografati anche se si sceglie la crittografia a riposo durante la creazione del cluster DAX.

L'uso della crittografia in transito con DAX è molto semplice. È sufficiente selezionare questa opzione durante la creazione di un nuovo cluster e utilizzare una versione recente di uno dei client DAX nella tua applicazione. I cluster che utilizzano la crittografia in transito non supportano il traffico

non crittografato, pertanto non è possibile configurare in modo errato l'applicazione e ignorare la crittografia. Il client DAX utilizzerà il certificato x509 del cluster per autenticare l'identità del cluster quando stabilisce le connessioni, assicurando che le richieste DAX vadano dove previsto. Tutti i metodi di creazione dei cluster DAX supportano la crittografia in transito: la Console di gestione AWS, la AWS CLI, tutti gli SDK e CloudFormation.

La crittografia in transito su un cluster DAX esistente non può essere abilitata. Per utilizzare la crittografia in transito in un'applicazione DAX esistente, creare un nuovo cluster con crittografia in transito abilitata, spostare il traffico dell'applicazione su di esso, quindi eliminare il vecchio cluster.

Utilizzo di ruoli IAM collegati ai servizi per DAX

Amazon DynamoDB Accelerator (DAX) utilizza i ruoli collegati ai servizi di AWS Identity and Access Management (IAM). Un ruolo collegato al servizio è un tipo di ruolo IAM univoco collegato direttamente a DAX. I ruoli collegati ai servizi sono definiti automaticamente da DAX e includono tutte le autorizzazioni richieste dal servizio per eseguire chiamate agli altri servizi AWS per conto tuo.

Un ruolo collegato ai servizi semplifica la configurazione di DAX perché ti permette di evitare l'aggiunta manuale delle autorizzazioni necessarie. DAX definisce le autorizzazioni dei relativi ruoli collegati ai servizi e, salvo diversamente definito, solo DAX potrà assumere i propri ruoli. Le autorizzazioni definite includono policy di attendibilità e di autorizzazioni. Questa policy delle autorizzazioni non può essere collegata ad alcun'altra entità IAM.

È possibile eliminare i ruoli solo dopo aver eliminato le risorse correlate. Questa procedura protegge le risorse di DAX perché impedisce la rimozione involontaria delle autorizzazioni di accesso alle risorse.

Per informazioni sugli altri servizi che supportano i ruoli collegati ai servizi, consulta la pagina Servizi AWS supportati da IAM nella Guida per l'utente di IAM. Cerca i servizi che contengono Yes (Sì) nella colonna Service-Linked Role (Ruolo collegato ai servizi). Scegli un collegamento Yes (Sì) per visualizzare la documentazione relativa al ruolo collegato ai servizi per tale servizio.

Argomenti

- Autorizzazioni del ruolo collegato ai servizi per DAX
- Creazione di un ruolo collegato ai servizi per DAX
- Modifica di un ruolo collegato ai servizi per DAX

- Eliminazione di un ruolo collegato ai servizi per DAX

Autorizzazioni del ruolo collegato ai servizi per DAX

DAX usa il ruolo collegato ai servizi denominato `AWSServiceRoleForDAX`. Questo ruolo consente a DAX di chiamare i servizi per conto del tuo cluster DAX.

> **Important**
>
> Il ruolo collegato ai servizi `AWSServiceRoleForDAX` semplifica la configurazione e il mantenimento di un cluster DAX. Tuttavia, occorre concedere a ogni cluster l'accesso a DynamoDB prima di poterlo utilizzare. Per ulteriori informazioni, consulta Controllo degli accessi DAX.

Ai fini dell'assunzione del ruolo, il ruolo collegato ai servizi `AWSServiceRoleForDAX` considera attendibili i seguenti servizi:

- `dax.amazonaws.com`

La policy delle autorizzazioni del ruolo consente a DAX di completare le seguenti operazioni sulle risorse specificate:

- Operazioni su `ec2`:
  - `AuthorizeSecurityGroupIngress`
  - `CreateNetworkInterface`
  - `CreateSecurityGroup`
  - `DeleteNetworkInterface`
  - `DeleteSecurityGroup`
  - `DescribeAvailabilityZones`
  - `DescribeNetworkInterfaces`
  - `DescribeSecurityGroups`
  - `DescribeSubnets`
  - `DescribeVpcs`
  - `ModifyNetworkInterfaceAttribute`

- `RevokeSecurityGroupIngress`

Per consentire a un'entità IAM (come un utente, un gruppo o un ruolo) di creare, modificare o eliminare un ruolo collegato ai servizi devi configurare le relative autorizzazioni. Per ulteriori informazioni, consulta Autorizzazioni del ruolo collegato ai servizi nella Guida per l'utente di IAM.

Per consentire a un'entità IAM di creare ruoli collegati ai servizi AWSServiceRoleForDAX

Aggiungi la seguente istruzione della policy alle autorizzazioni per l'entità IAM.

```
{
    "Effect": "Allow",
    "Action": [
        "iam:CreateServiceLinkedRole"
    ],
    "Resource": "*",
    "Condition": {"StringLike": {"iam:AWSServiceName": "dax.amazonaws.com"}}
}
```

Creazione di un ruolo collegato ai servizi per DAX

Non hai bisogno di creare manualmente un ruolo collegato ai servizi. Quando crei un cluster, DAX crea il ruolo collegato ai servizi per conto tuo.

> **Important**
>
> Se utilizzavi il servizio DAX prima del 28 febbraio 2018, data da cui è disponibile il supporto dei ruoli collegati al servizio, DAX ha creato il ruolo `AWSServiceRoleForDAX` nel tuo account. Per ulteriori informazioni, consulta Visualizzazione di un nuovo ruolo nell'account AWS nella Guida per l'utente di IAM.

Se devi ricreare un ruolo collegato ai servizi che hai precedentemente eliminato, puoi utilizzare lo stesso processo per ricreare il ruolo nel tuo account. Quando crei un'istanza o un cluster, DAX crea di nuovo il ruolo collegato ai servizi per conto tuo.

Modifica di un ruolo collegato ai servizi per DAX

DAX non consente di modificare il ruolo collegato ai servizi `AWSServiceRoleForDAX`. Dopo aver creato un ruolo collegato al servizio, non potrai modificarne il nome perché varie entità potrebbero

farvi riferimento. È possibile tuttavia modificarne la descrizione utilizzando IAM. Per ulteriori informazioni, consulta la sezione Modifica di un ruolo collegato ai servizi nella Guida per l'utente di IAM.

Eliminazione di un ruolo collegato ai servizi per DAX

Se non è più necessario utilizzare una funzionalità o un servizio che richiede un ruolo collegato al servizio, ti consigliamo di eliminare il ruolo. In questo modo non sarà più presente un'entità non utilizzata che non viene monitorata e gestita attivamente. Tuttavia, prima di poter eliminare il ruolo collegato ai servizi, dovrai eliminare tutti i cluster DAX.

Pulizia di un ruolo collegato ai servizi

Prima di utilizzare IAM per eliminare un ruolo collegato ai servizi, devi innanzitutto verificare che il ruolo non abbia sessioni attive ed eliminare tutte le risorse utilizzate dal ruolo.

Per verificare se il ruolo collegato ai servizi dispone di una sessione attiva nella console IAM

1. Accedi alla Console di gestione AWSe apri la console IAM all'indirizzo https://console.aws.amazon.com/iam/.
2. Nel pannello di navigazione della console IAM seleziona Ruoli. Quindi, scegli il nome (non la casella di controllo) del ruolo AWSServiceRoleForDAX.
3. Nella pagina Riepilogo per il ruolo selezionato, scegli la scheda Consulente accessi.
4. Nella scheda Access Advisor (Consulente accessi), esamina l'attività recente per il ruolo collegato ai servizi.

> **Note**
>
> Se non si ha la certezza che DAX stia utilizzando il ruolo `AWSServiceRoleForDAX`, è possibile provare a eliminarlo. Se il servizio sta utilizzando il ruolo, l'eliminazione non andrà a buon fine e puoi visualizzare le regioni in cui il ruolo viene utilizzato. Se il ruolo è in uso, prima di poterlo eliminare devi eliminare i cluster DAX. Non puoi revocare la sessione per un ruolo collegato ai servizi.

Se vuoi rimuovere il ruolo `AWSServiceRoleForDAX`, devi prima eliminare tutti i cluster DAX.

Eliminazione di tutti i cluster DAX

Utilizza una di queste procedure per eliminare ogni cluster DAX.

Come eliminare un cluster DAX (console)

1. Apri la console DynamoDB all'indirizzo https://console.aws.amazon.com/dynamodb/.
2. Nel riquadro di navigazione, in DAX, scegli Clusters (Cluster).
3. Scegli Operazioni, quindi Elimina.
4. Nella casella Conferma eliminazione cluster scegli Elimina.

Come eliminare un cluster DAX (AWS CLI)

Consulta .delete-cluster nella Riferimento ai comandi della AWS CLI.

Come eliminare un cluster DAX (API)

Consulta .DeleteCluster nella Documentazione di riferimento delle API di Amazon DynamoDB.

Eliminazione del ruolo collegato ai servizi

Per eliminare manualmente il ruolo collegato ai servizi mediante IAM

Usa la console IAM, la CLI IAM oppure l'API IAM per eliminare il ruolo collegato ai servizi `AWSServiceRoleForDAX`. Per ulteriori informazioni, consulta Eliminazione del ruolo collegato ai servizi nella Guida per l'utente di IAM.

Accesso a DAX su più account AWS

Immagina di avere un cluster DynamoDB Accelerator (DAX) in esecuzione in AWS un account (account A) e che il cluster DAX debba essere accessibile da un'istanza Amazon Elastic Compute Cloud (EC2Amazon) in un altro account (account B). AWS In questo tutorial, esegui questa operazione avviando un' EC2istanza nell'account B con un ruolo IAM dall'account B. Quindi utilizzi le credenziali di sicurezza temporanee dell' EC2 istanza per assumere un ruolo IAM dall'account A. Infine, utilizzi le credenziali di sicurezza temporanee derivanti dall'assunzione del ruolo IAM nell'account A per effettuare chiamate alle applicazioni tramite una connessione peering Amazon VPC al cluster DAX nell'account A. Per eseguire queste attività è necessario l'accesso amministrativo in entrambi i casi. conti. AWS

> **Important**
>
> Non è possibile fare in modo che un cluster DAX acceda a una tabella DynamoDB da un account diverso.

Argomenti

- Configurazione di IAM
- Configurazione VPC
- Modifica del client DAX per consentire l'accesso multi-account

Configurazione di IAM

1. Crea un file di testo denominato `AssumeDaxRoleTrust.json` con il seguente contenuto, che EC2 consenta ad Amazon di lavorare per tuo conto.

   JSON

   ```
   {
       "Version":"2012-10-17",
       "Statement": [
           {
               "Effect": "Allow",
               "Principal": {
                   "Service": "ec2.amazonaws.com"
               },
               "Action": "sts:AssumeRole"
           }
       ]
   }
   ```

2. Nell'account B, crea un ruolo che Amazon EC2 può utilizzare per l'avvio delle istanze.

   ```
   aws iam create-role \
       --role-name AssumeDaxRole \
       --assume-role-policy-document file://AssumeDaxRoleTrust.json
   ```

3. Crea un file di testo denominato `AssumeDaxRolePolicy.json` con il seguente contenuto, che consenta al codice in esecuzione sull' EC2 istanza nell'account B di assumere un ruolo IAM nell'account A. Sostituiscilo *accountA* con l'ID effettivo dell'account A.

   JSON

   ```
   {
       "Version":"2012-10-17",
       "Statement": [
           {
               "Effect": "Allow",
               "Action": "sts:AssumeRole",
               "Resource": "arn:aws:iam::111122223333:role/DaxCrossAccountRole"
           }
       ]
   }
   ```

4. Aggiungi la policy al ruolo appena creato.

   ```
   aws iam put-role-policy \
       --role-name AssumeDaxRole \
       --policy-name AssumeDaxRolePolicy \
       --policy-document file://AssumeDaxRolePolicy.json
   ```

5. Crea un profilo di istanza per consentire alle istanze di utilizzare il ruolo.

   ```
   aws iam create-instance-profile \
       --instance-profile-name AssumeDaxInstanceProfile
   ```

6. Associa il ruolo al profilo di istanza.

   ```
   aws iam add-role-to-instance-profile \
       --instance-profile-name AssumeDaxInstanceProfile \
       --role-name AssumeDaxRole
   ```

7. Crea un file di testo denominato `DaxCrossAccountRoleTrust.json` con il contenuto seguente che consente all'account B di assumere un ruolo dell'account A. Sostituisci *accountB* con l'ID effettivo dell'account B.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:role/AssumeDaxRole"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

8. Nell'account A, crea il ruolo che l'account B può assumere.

```
aws iam create-role \
    --role-name DaxCrossAccountRole \
    --assume-role-policy-document file://DaxCrossAccountRoleTrust.json
```

9. Crea un file di testo denominato `DaxCrossAccountPolicy.json` che consente l'accesso al cluster DAX. Sostituiscilo *`dax-cluster-arn`* con l'Amazon Resource Name (ARN) corretto del tuo cluster 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. Nell'account A, aggiungi la policy al ruolo.

```
aws iam put-role-policy \
    --role-name DaxCrossAccountRole \
    --policy-name DaxCrossAccountPolicy \
    --policy-document file://DaxCrossAccountPolicy.json
```

Configurazione VPC

1. Individuare il gruppo di sottoreti del cluster DAX dell'account A. Sostituiscilo *cluster-name* con il nome del cluster DAX a cui deve accedere l'account B.

```
aws dax describe-clusters \
    --cluster-name cluster-name
    --query 'Clusters[0].SubnetGroup'
```

2. Usandolo*subnet-group*, trova il VPC del cluster.

```
aws dax describe-subnet-groups \
    --subnet-group-name subnet-group \
    --query 'SubnetGroups[0].VpcId'
```

3. Usandolo*vpc-id*, trova il CIDR del VPC.

```
aws ec2 describe-vpcs \
    --vpc vpc-id \
    --query 'Vpcs[0].CidrBlock'
```

4. Dall'account B, crea un VPC utilizzando un CIDR diverso, non sovrapposto rispetto a quello trovato nel passaggio precedente. Quindi, crea almeno una subnet. È possibile utilizzare la procedura guidata per la creazione di VPC in o in. Console di gestione AWS AWS CLI

5. Dall'account B, richiedi una connessione peering al VPC dell'account A come descritto in Creazione e accettazione di una connessione peering VPC. Dall'account A, accetta la connessione.
6. Dall'account B, trova la tabella di routing del nuovo VPC. Sostituisci *vpc-id* con l'ID del VPC che hai creato nell'account B.

```
aws ec2 describe-route-tables \
    --filters 'Name=vpc-id,Values=vpc-id' \
    --query 'RouteTables[0].RouteTableId'
```

7. Aggiungi un percorso per inviare il traffico destinato al CIDR dell'account A alla connessione peering VPC. Ricordati di sostituirli *user input placeholder* con i valori corretti per i tuoi account.

```
aws ec2 create-route \
    --route-table-id accountB-route-table-id \
    --destination-cidr accountA-vpc-cidr \
    --vpc-peering-connection-id peering-connection-id
```

8. Dall'account A, trova la tabella di routing del cluster DAX utilizzando la tabella di routing *vpc-id* che hai trovato in precedenza.

```
aws ec2 describe-route-tables \
    --filters 'Name=vpc-id, Values=accountA-vpc-id' \
    --query 'RouteTables[0].RouteTableId'
```

9. Dall'account A, aggiungi un percorso per inviare il traffico destinato al CIDR dell'account B alla connessione peering VPC. Sostituisci *user input placeholder* ognuna con i valori corretti per i tuoi account.

```
aws ec2 create-route \
    --route-table-id accountA-route-table-id \
    --destination-cidr accountB-vpc-cidr \
    --vpc-peering-connection-id peering-connection-id
```

10. Dall'account B, avvia un' EC2 istanza nel VPC che hai creato in precedenza. Assegnarle il nome `AssumeDaxInstanceProfile`. È possibile utilizzare la procedura guidata di avvio in Console di gestione AWS o in. AWS CLI Prendi nota del gruppo di sicurezza dell'istanza.
11. Dall'account A, individua il gruppo di sicurezza utilizzato dal cluster DAX. Ricordarsi di sostituirlo *cluster-name* con il nome del cluster DAX.

```
aws dax describe-clusters \
    --cluster-name cluster-name \
    --query 'Clusters[0].SecurityGroups[0].SecurityGroupIdentifier'
```

12. Aggiorna il gruppo di sicurezza del cluster DAX per consentire il traffico in entrata dal gruppo di sicurezza dell' EC2 istanza che hai creato nell'account B. Ricordati di sostituirlo *user input placeholders* con i valori corretti per i tuoi account.

```
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
```

A questo punto, un'applicazione sull' EC2 istanza dell'account B è in grado di utilizzare il profilo dell'istanza per assumere il `arn:aws:iam::`*accountA-id*`:role/DaxCrossAccountRole` ruolo e utilizzare il cluster DAX.

Modifica del client DAX per consentire l'accesso multi-account

> **Note**
>
> AWS Security Token Service (AWS STS) le credenziali sono credenziali temporanee. Alcuni client gestiscono automaticamente l'aggiornamento, mentre altri richiedono una logica aggiuntiva per aggiornare le credenziali. Consigliamo di seguire le indicazioni della documentazione appropriata.

Java

Questa sezione consente di modificare il codice client DAX esistente per consentire l'accesso DAX tra account. Se non hai già un codice client DAX, puoi trovare esempi di codice funzionanti nel tutorial Java e DAX.

1. Aggiungi le seguenti importazioni:

```
import com.amazonaws.auth.STSAssumeRoleSessionCredentialsProvider;
```

```
import com.amazonaws.services.securitytoken.AWSSecurityTokenService;
import
 com.amazonaws.services.securitytoken.AWSSecurityTokenServiceClientBuilder;
```

2. Ottieni un provider di credenziali da AWS STS e crea un oggetto client DAX. Ricordati di sostituirli *user input placeholder* con i valori corretti per i tuoi account.

```
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

Questa sezione consente di modificare il codice client DAX esistente per consentire l'accesso DAX tra account. Se non hai già un codice client DAX, puoi trovare esempi di codice funzionanti nel tutorial .NET e DAX.

1. Aggiungi il AWSSDK. SecurityToken NuGet pacchetto alla soluzione.

```
<PackageReference Include="AWSSDK.SecurityToken" Version="latest version" />
```

2. Usa i pacchetti `SecurityToken.Model` e `SecurityToken`.

```
using Amazon.SecurityToken;
using Amazon.SecurityToken.Model;
```

3. Ottieni le credenziali temporanee da `AmazonSimpleTokenService` e crea un oggetto `ClusterDaxClient`. Ricordati di sostituirli *`user input placeholder`* con i valori corretti per i tuoi account.

```
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

Questa sezione consente di modificare il codice client DAX esistente per consentire l'accesso DAX tra account. Se non disponi già del codice client DAX, puoi trovare esempi di codice funzionanti su GitHub.

1. Importa i pacchetti AWS STS e le sessioni.

```
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. Ottenere le credenziali temporanee da `AmazonSimpleTokenService` e creare un oggetto client DAX. Ricordati di sostituirli *`user input placeholder`* con i valori corretti per i tuoi account.

```
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

Questa sezione consente di modificare il codice client DAX esistente per consentire l'accesso DAX tra account. Se non hai già un codice client DAX, puoi trovare esempi di codice funzionanti nel tutorial Python e DAX.

1. Importa `boto3`.

```
import boto3
```

2. Ottieni credenziali temporanee da `sts` e crea un oggetto `AmazonDaxClient`. Ricordati di sostituirli *`user input placeholder`* con i valori corretti per i tuoi account.

```
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

Questa sezione consente di modificare il codice client DAX esistente per consentire l'accesso DAX tra account. Se non hai già un codice client DAX, puoi trovare esempi di codice funzionanti nel tutorial Node.js e DAX. Ricordati di sostituirli *`user input placeholder`* con i valori corretti per i tuoi account.

```
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);
});
```

Guida alle dimensioni del cluster DAX

Questa guida fornisce consigli per scegliere una dimensione e un tipo di nodo del cluster Amazon DynamoDB Accelerator (DAX) appropriati per la tua applicazione. Queste istruzioni consentono di valutare il traffico DAX dell'applicazione, selezionare una configurazione del cluster ed eseguirne il test.

Se hai un cluster DAX esistente e vuoi determinare se ha il numero e la dimensione appropriati dei nodi, fai riferimento a Dimensionamento di un cluster DAX.

Argomenti

- Panoramica
- Stima del traffico.
- Test di caricamento

Panoramica

È importante dimensionare il cluster DAX in modo appropriato per il carico di lavoro, sia se stai creando un nuovo cluster che se stai gestendo un cluster esistente. Con il passare del tempo e il carico di lavoro dell'applicazione cambia, quindi devi rivedere periodicamente le decisioni di dimensionamento per assicurarti che siano ancora appropriate.

Il processo in genere è costituito da queste fasi:

1. Stima del traffico. In questa fase, è possibile effettuare previsioni sul volume di traffico che l'applicazione invierà a DAX, sulla natura del traffico (operazioni di lettura e scrittura) e sulla percentuale di riscontri nella cache.

2. Test di carico. In questa fase, crei un cluster e invii il traffico rispecchiando le stime del passaggio precedente. Ripeti questo passaggio fino a trovare una configurazione del cluster adatta.
3. Monitoraggio della produzione. Durante l'utilizzo dell'applicazione DAX in produzione, è necessario monitorare il cluster per verificare continuamente che sia ancora dimensionato correttamente man mano che il carico di lavoro cambia nel tempo.

Stima del traffico.

Esistono tre fattori principali che caratterizzano un carico di lavoro DAX tipico:

- Percentuale di riscontri nella cache
- Unità di capacità in lettura (RCU) al secondo
- Unità di capacità in scrittura (WCU) al secondo

Stima della percentuale di riscontri nella cache

Se si dispone già di un cluster DAX, è possibile utilizzare i parametri di Amazon CloudWatch `ItemCacheHits` e`ItemCacheMisses` per determinare la percentuale di riscontri nella cache. La percentuale di riscontri nella cache è uguale a `ItemCacheHits` / (`ItemCacheHits` + `ItemCacheMisses`). Se il carico di lavoro include operazioni `Query` o `Scan`, è necessario esaminare anche i parametri `QueryCacheHits`, `QueryCacheMisses`, `ScanCacheHits` e `ScanCacheMisses`. La percentuale di riscontri nella cache varia da un'applicazione all'altra ed è fortemente influenzata dall'impostazione di durata (TTL, Time to Live) del cluster. La percentuale di successi tipica per le applicazioni che utilizzano DAX è 85-95%.

Stima delle unità di capacità in lettura e scrittura

Se disponi già di tabelle DynamoDB per la tua applicazione, consulta i parametri CloudWatch `ConsumedReadCapacityUnits` e `ConsumedWriteCapacityUnits`. Utilizza la statistica `Sum` e dividi per il numero di secondi del periodo.

Se hai anche un cluster DAX, tieni presente che il parametro `ConsumedReadCapacityUnits` di DynamoDB conteggia solo i mancati riscontri della cache. Quindi, per avere un'idea delle unità di capacità in lettura al secondo gestite dal cluster DAX, dividi il numero per il tasso di mancati riscontri della cache (ovvero, 1 - percentuale di riscontri nella cache).

Se non è già presente una tabella DynamoDB, consulta la documentazione sulle unità di capacità di lettura e le unità di capacità di scrittura per stimare il traffico in base al tasso stimato di richieste dell'applicazione, agli elementi a cui si accede per richiesta e alle dimensioni degli elementi.

Quando effettui le stime del traffico, pianifica la crescita futura e i picchi previsti e imprevisti per garantire che il cluster abbia lo spazio sufficiente per l'aumento del traffico.

Test di caricamento

La fase successiva alla stima del traffico consiste nel testare la configurazione del cluster sotto carico.

1. Per il test di carico iniziale, si consiglia di iniziare con il tipo di nodo `dax.r4.large`, le prestazioni fisse a minor costo e il tipo di nodo ottimizzato per la memoria.
2. Un cluster con tolleranza ai guasti richiede almeno tre nodi, distribuiti in tre zone di disponibilità. In questo caso, se una zona di disponibilità diventa non disponibile, il numero effettivo di zone di disponibilità viene ridotto di un terzo. Per il test di carico iniziale, si consiglia di iniziare con un cluster a due nodi che simula l'errore di una zona di disponibilità in un cluster a tre nodi.
3. Veicola il traffico prolungato (come stimato nel passaggio precedente) nel cluster di test per tutta la durata del test di carico.
4. Monitora le prestazioni del cluster durante il test di carico.

Idealmente, il profilo di traffico che veicoli durante il test di carico deve essere il più simile possibile al traffico reale dell'applicazione. Ciò include la distribuzione delle operazioni (ad esempio il 70% `GetItem`, il 25% `Query` e il 5% `PutItem`), il tasso di richiesta per ogni operazione, il numero di elementi a cui si accede per richiesta e la distribuzione delle dimensioni degli elementi. Per ottenere una percentuale di riscontri nella cache simile a quello previsto dalla tua applicazione, presta molta attenzione alla distribuzione delle chiavi nel traffico di test.

> **Note**
>
> Fai attenzione quando testi il carico dei tipi di nodo T2 (`dax.t2.small` e `dax.t2.medium`). I tipi di nodi T2 offrono prestazioni di CPU espandibili che variano nel tempo a seconda del saldo del credito della CPU del nodo. Un cluster DAX in esecuzione su nodi T2 potrebbe sembrare funzionare normalmente, ma se un nodo si sta espandendo oltre le prestazioni di base dell'istanza, il nodo sta spendendo il saldo del credito della CPU accumulato. Quando

il saldo di credito è basso, le prestazioni vengono gradualmente ridotte fino al livello di prestazioni di base.

Monitora il cluster DAX durante il test di carico per determinare se il tipo di nodo che stai utilizzando per il test di carico è quello corretto. Inoltre, durante un test di carico, devi monitorare il tasso di richiesta e la percentuale di riscontri nella cache per assicurarti che l'infrastruttura di test stia effettivamente veicolando la quantità di traffico desiderata.

È necessario prestare attenzione al consumo di byte di rete del tipo di istanza cluster selezionata. Il superamento della larghezza di banda di base disponibile per un'istanza Amazon EC2 indica che il cluster potrebbe non sostenere il carico di lavoro dell'applicazione e deve essere dimensionato.

Se il test di carico indica che la configurazione del cluster selezionata non può sostenere il carico di lavoro dell'applicazione, dovresti passare a un tipo di nodo più grande, soprattutto in presenza di un utilizzo elevato della CPU sul nodo primario nel cluster, tassi di espulsione elevati o utilizzo elevato della cache. Se il tasso di hit è costantemente elevato e il rapporto tra il traffico di lettura e scrittura è alto, valuta l'aggiunta di più nodi al cluster. Fai riferimento a Dimensionamento di un cluster DAX per ulteriori indicazioni su quando utilizzare un tipo di nodo più grande (dimensionamento verticale) o aggiungere più nodi (dimensionamento orizzontale).

È necessario ripetere il test di carico dopo aver apportato modifiche alla configurazione del cluster.

Modellazione dei dati per tabelle DynamoDB

Prima di immergerci nella modellazione dei dati, è importante comprendere alcuni fondamenti di DynamoDB. DynamoDB è un database NoSQL chiave-valore che consente uno schema flessibile. L'insieme di attributi dei dati oltre agli attributi chiave per ogni elemento può essere uniforme o discreto. Lo schema delle chiavi di DynamoDB ha la forma di una chiave primaria semplice in cui una chiave di partizione identifica in modo univoco un elemento o di una chiave primaria composita in cui una combinazione di chiave di partizione e chiave di ordinamento definisce in modo univoco un elemento. La chiave di partizione è sottoposta a hash per determinare la posizione fisica dei dati e recuperarli. Pertanto, è importante scegliere un attributo ad alta cardinalità e scalabile orizzontalmente come chiave di partizione per garantire una distribuzione uniforme dei dati. L'attributo sort key è facoltativo nello schema chiave e la presenza di una chiave di ordinamento consente di modellare relazioni uno-a-molti e di creare raccolte di elementi in DynamoDB. Le chiavi di ordinamento sono anche chiamate chiavi di intervallo: vengono utilizzate per ordinare gli elementi in una raccolta di articoli e consentono anche operazioni flessibili basate sull'intervallo.

Per ulteriori dettagli e best practice sullo schema delle chiavi di DynamoDB, puoi fare riferimento a quanto segue:

- the section called “Partizioni e distribuzione dei dati in DynamoDB”
- the section called “Progettazione delle chiavi di partizione”
- the section called “Progettazione della chiave di ordinamento”
- Scelta della chiave di partizione DynamoDB corretta

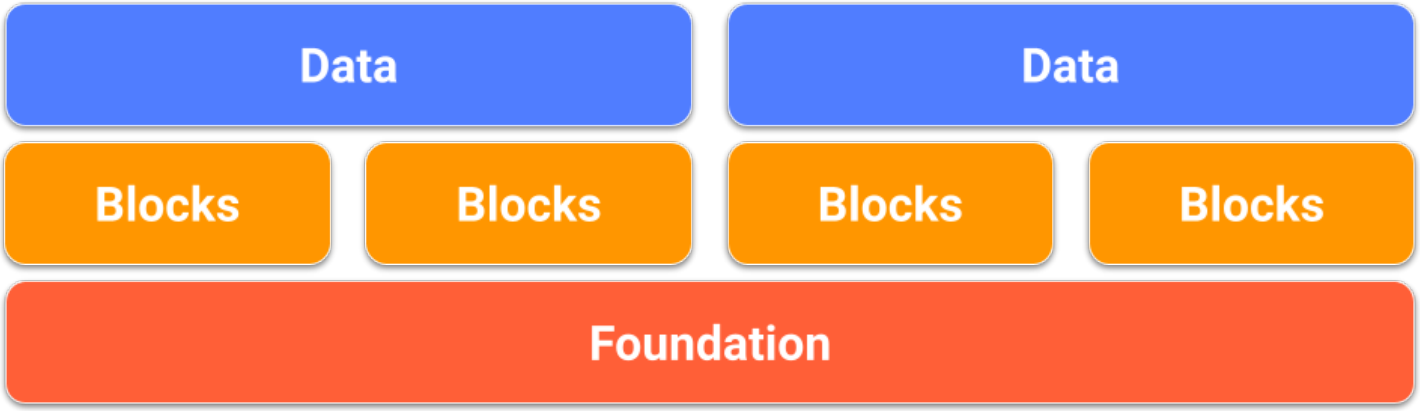
Gli indici secondari sono spesso necessari per supportare modelli di query aggiuntivi in DynamoDB. Gli indici secondari sono tabelle ombra in cui gli stessi dati sono organizzati tramite uno schema chiave diverso rispetto alla tabella di base. Un indice secondario locale (LSI) condivide la stessa chiave di partizione della tabella base e consente di disporre di una chiave di ordinamento alternativa che consente di condividere la capacità della tabella di base. Un indice secondario globale (GSI) può avere una chiave di partizione e una chiave di ordinamento diversi rispetto alla tabella di base, il che significa che la gestione della velocità di trasmissione effettiva per un GSI è indipendente dalla tabella base.

Per ulteriori dettagli sugli indici secondari e le best practice, puoi fare riferimento a quanto segue:

- the section called “Utilizzo degli indici”

- the section called “Indici secondari”

Esaminiamo ora la modellazione dei dati un po' più da vicino. Il processo di progettazione di uno schema flessibile e altamente ottimizzato su DynamoDB, o su qualsiasi database NoSQL, può essere un'abilità difficile da imparare. L'obiettivo di questo modulo è aiutarti a sviluppare un diagramma di flusso mentale per la progettazione di uno schema che ti porterà dal caso d'uso alla produzione. Inizieremo con un'introduzione alla scelta fondamentale di qualsiasi progettazione: tabella singola contro tabelle multiple. Quindi esamineremo la moltitudine di modelli di progettazione (elementi costitutivi) che possono essere utilizzati per ottenere vari risultati organizzativi o delle prestazioni per la tua applicazione. Infine, includeremo una varietà di pacchetti completi di progettazione dello schema per diversi casi d'uso e settori.



Argomenti

- Raccolte di articoli: come modellare relazioni uno a molti in DynamoDB
- Fondamenti della modellazione dei dati in DynamoDB
- Elementi costitutivi della modellazione dei dati in DynamoDB
- Pacchetti di progettazione dello schema di modellazione dei dati in DynamoDB
- Best practice per la modellazione dei dati relazionali in DynamoDB

Raccolte di articoli: come modellare relazioni uno a molti in DynamoDB

In DynamoDB, una raccolta di elementi è un gruppo di elementi che condividono lo stesso valore della chiave di partizione, il che significa che gli elementi sono correlati. Le raccolte di elementi rappresentano il meccanismo principale per modellare le relazioni uno a molti in DynamoDB. Le

raccolte di elementi possono esistere solo su tabelle o indici configurati per l'utilizzo di una chiave primaria composita.

> **Note**
>
> Le raccolte di elementi possono esistere in una tabella di base o in un indice secondario. Per ulteriori informazioni specifiche su come le raccolte di elementi interagiscono con gli indici, consulta Raccolte di elementi negli indici secondari locali.

Considerare la seguente tabella che mostra tre diversi utenti e i loro inventari all'interno di un gioco:

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

Per alcuni elementi di ogni raccolta, la chiave di ordinamento è una concatenazione composta da informazioni utilizzate per raggruppare i dati, ad esempio `inventory::armor`, `inventory::weapon` o `info`. Ogni raccolta di elementi può avere come chiave di ordinamento una combinazione diversa di questi attributi. L'utente `account1234` ha un elemento `inventory::weapons`, mentre l'utente `account1387` non ha elementi in quanto non ne ha ancora trovato nessuno. L'utente `account1138` utilizza solo due elementi per la chiave di ordinamento (poiché non ha ancora un inventario) mentre gli altri utenti ne usano tre.

DynamoDB consente di recuperare in modo selettivo gli elementi da queste raccolte di elementi per effettuare le seguenti operazioni:

- Recuperare tutti gli elementi da un determinato utente
- Recuperare un solo elemento da un determinato utente
- Recuperare tutti gli elementi di un tipo specifico appartenenti a un determinato utente

Accelerare le query organizzando i dati con le raccolte di elementi

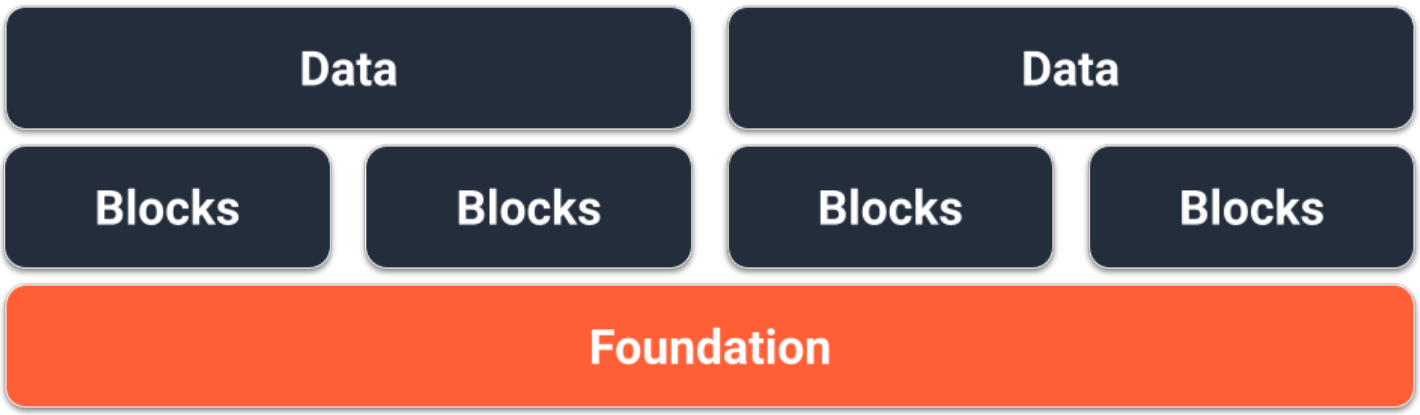
In questo esempio, ciascuno degli elementi di queste tre raccolte di elementi rappresenta un giocatore e il modello di dati che abbiamo scelto, in base ai modelli di accesso del gioco e del giocatore. Di quali dati ha bisogno il gioco? Quando ne ha bisogno? Con quale frequenza ne ha bisogno? Qual è il costo nell'operare in questo modo? Queste decisioni sulla modellazione dei dati sono state prese sulla base delle risposte a queste domande.

In questo gioco, c'è una pagina diversa presentata al giocatore per l'inventario per le armi e un'altra pagina per l'armatura. Quando il giocatore apre il proprio inventario, le armi vengono mostrate per prime perché vogliamo che la pagina venga caricata in modo estremamente veloce, mentre le pagine di inventario successive possono essere caricate in un momento successivo. Poiché ognuno di questi tipi di elementi può essere abbastanza grande in quanto il giocatore acquisisce più oggetti nel gioco, abbiamo deciso che ogni pagina dell'inventario sarebbe stata il proprio elemento nella raccolta di elementi del giocatore nel database.

Nella sezione seguente vengono fornite ulteriori informazioni su come interagire con le raccolte di elementi tramite l'operazione `Query`.

Fondamenti della modellazione dei dati in DynamoDB

Questa sezione inizia dal livello base esaminando i due tipi di progettazione tabella: tabella singola e tabelle multiple.



Base della progettazione tabella singola

Una scelta per la base dello schema DynamoDB è la progettazione tabella singola. La progettazione tabella singola è un modello che consente di archiviare più tipi (entità) di dati in una singola tabella DynamoDB. Ha lo scopo di ottimizzare i modelli di accesso ai dati, migliorare le prestazioni e ridurre i costi eliminando la necessità di mantenere più tabelle e relazioni complesse tra di esse. Ciò è possibile perché DynamoDB archivia gli elementi con la stessa chiave di partizione (nota come raccolta di elementi) sulle stesse partizioni dell'altra. In questa progettazione, tipi di dati differenti vengono archiviati come elementi nella stessa tabella e ciascun elemento è identificato da una chiave di ordinamento univoca.

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

Vantaggi

- Ubicazione dei dati per supportare query per più tipi di entità in una singola chiamata al database
- Riduce i costi finanziari e di latenza complessivi delle letture:
  - Una singola query per due elementi che totalizzano meno di 4 KB è caratterizzata da consistenza finale di 0,5 RCU

  - Due query per due elementi che totalizzano meno di 4 KB è caratterizzata da consistenza finale di 1 RCU (0,5 RCU ciascuna)
  - Il tempo richiesto per restituire due chiamate al database separate sarà in media superiore a quello di una singola chiamata
- Riduce il numero di tabelle da gestire:
  - Non è necessario mantenere le autorizzazioni per più ruoli o policy IAM
  - La gestione della capacità per la tabella viene calcolata come media tra tutte le entità, di solito risultando in un modello di consumo più prevedibile
  - Il monitoraggio richiede un numero inferiore di allarmi
  - Le chiavi di crittografia gestite dal cliente devono essere ruotate solo su una tabella
- Uniforma il traffico verso la tabella:
  - Aggregando più modelli di utilizzo nella stessa tabella, l'utilizzo complessivo tende a essere più uniforme (così come la performance di un indice azionario tende a essere più uniforme di qualsiasi titolo singolo), consentendo di raggiungere un maggiore utilizzo con tabelle con modalità assegnata

Svantaggi

- La curva di apprendimento può essere ripida a causa della progettazione paradossale rispetto ai database relazionali
- I requisiti di dati devono essere coerenti tra tutti i tipi di entità
  - I backup sono in modalità “tutto o niente”, pertanto se alcuni dati non sono mission-critical, occorre valutare la possibilità di conservarli in una tabella separata
  - La crittografia delle tabelle è condivisa tra tutti gli elementi. Per applicazioni multi-tenant con requisiti di crittografia del singolo tenant, è richiesta la crittografia lato client
  - Le tabelle con una combinazione di dati storici e dati operativi non traggono vantaggio dall'attivazione della classe di storage ad accesso infrequente. Per ulteriori informazioni, consulta Classi di tabella DynamoDB
- Tutti i dati modificati verranno propagati ai flussi DynamoDB anche se deve essere elaborato solo un sottoinsieme di entità.
  - Grazie ai filtri degli eventi Lambda, ciò non influirà sulla fatturazione quando si utilizza Lambda, ma comporterà un costo aggiuntivo quando si utilizza la Kinesis Consumer Library

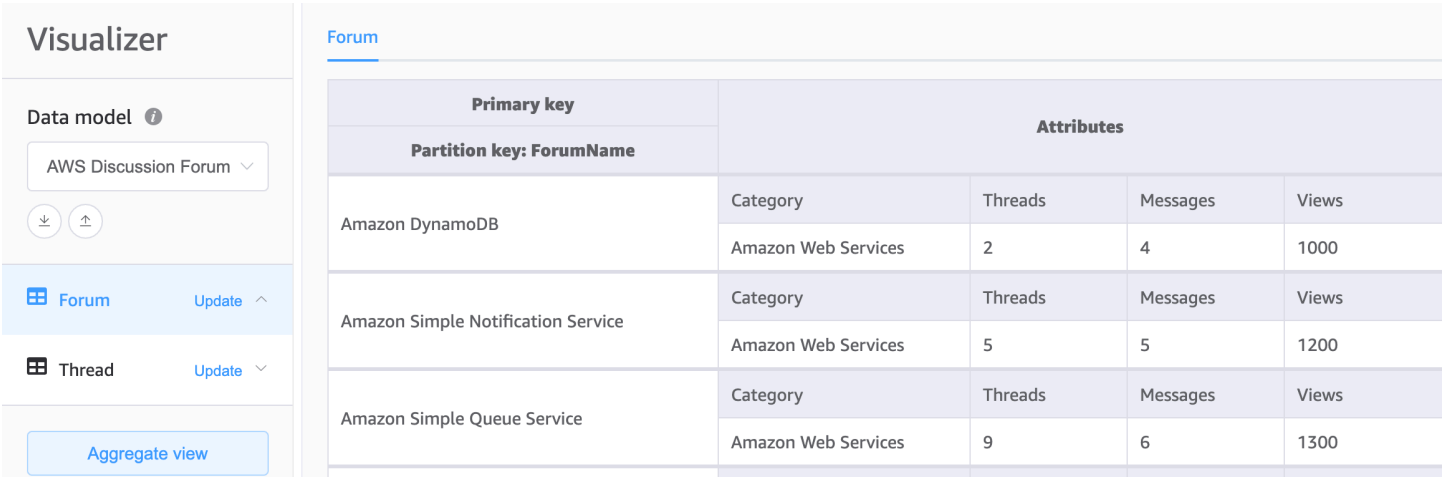
- Quando si utilizza GraphQL, la progettazione di una singola tabella sarà più difficile da implementare
- Quando si utilizzano client SDK di livello superiore come `DynamoDBMapper` o Enhanced Client di Java, può essere più difficile elaborare i risultati perché gli elementi nella stessa risposta possono essere associati a classi diverse

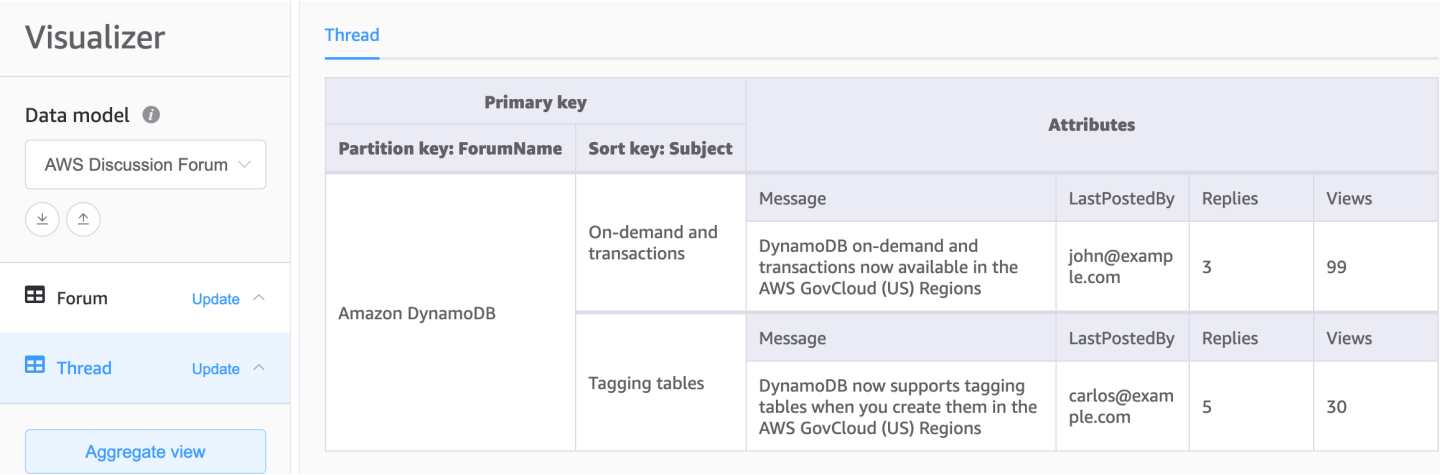
Quando usare

La progettazione a tabella singola è ideale per le applicazioni che spesso interrogano più tipi di entità insieme o devono mantenere relazioni tra diversi tipi di dati. È particolarmente efficace quando i modelli di accesso traggono vantaggio dall'ubicazione dei dati e quando si desidera ridurre al minimo il sovraccarico derivante dalla gestione di più tabelle.

Base della progettazione tabelle multiple

La seconda scelta per la base dello schema DynamoDB è la progettazione tabelle multiple. La progettazione tabelle multiple è un modello più simile a una progettazione di database tradizionale in cui si archivia un singolo tipo (entità) di dati in ogni tabella DynamoDB. I dati all'interno di ciascuna tabella saranno comunque organizzati per chiave di partizione, pertanto le prestazioni all'interno di un singolo tipo di entità saranno ottimizzate per scalabilità e prestazioni, ma le query su più tabelle devono essere eseguite in modo indipendente.





Vantaggi

- Più semplice da progettare per chi non è abituato a utilizzare la progettazione tabella singola
- Implementazione più semplice dei resolver GraphQL grazie alla mappatura di ogni resolver su una singola entità (tabella)
- Consente requisiti di dati univoci per diversi tipi di entità:
  - Backup possono essere eseguiti per le singole tabelle che sono mission critical
  - La crittografia della tabella può essere gestita per ciascuna tabella. Per applicazioni multi-tenant con requisiti di crittografia del singolo tenant, tabelle tenant separate consentono a ciascun cliente di disporre della propria chiave di crittografia
  - La classe di storage ad accesso infrequente può essere abilitata solo sulle tabelle con dati storici per ottenere il massimo vantaggio in termini di riduzione dei costi. Per ulteriori informazioni, consulta Classi di tabella DynamoDB
- Ogni tabella avrà il proprio flusso di dati di modifica. Ciò consente di progettare una funzione Lambda dedicata per ogni tipo di elemento anziché un singolo processore monolitico

Svantaggi

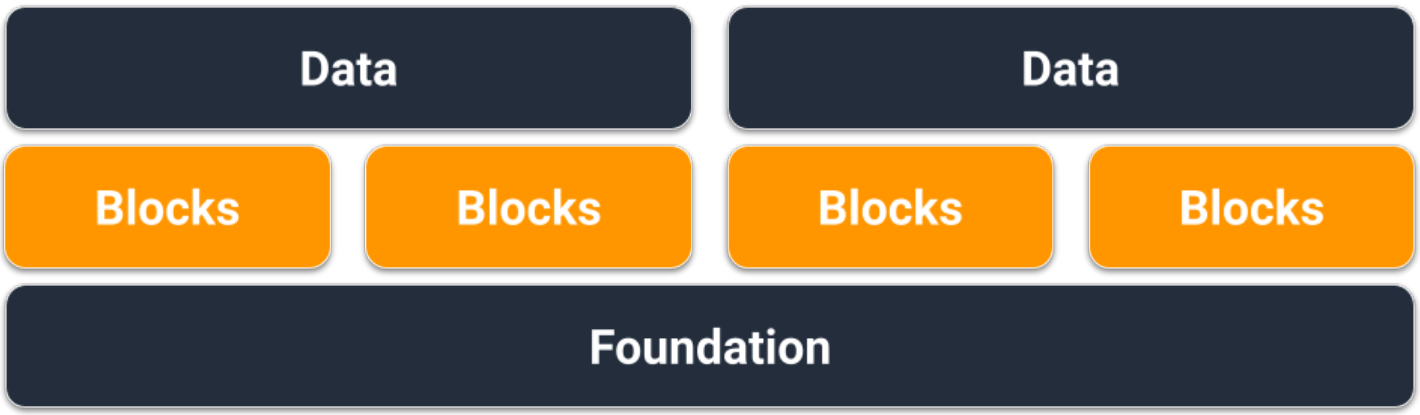
- Per modelli di accesso che richiedono dati su più tabelle, saranno necessarie più letture da DynamoDB e potrebbe essere necessario elaborare/unire i dati nel codice client.
- Il funzionamento e il monitoraggio di più tabelle richiede più allarmi CloudWatch e ciascuna tabella deve essere dimensionata in modo indipendente
- Le autorizzazioni di ciascuna tabella devono essere gestite separatamente. L'aggiunta di tabelle in futuro richiederà la modifica degli eventuali ruoli o policy IAM necessari

Quando usare

Se i modelli di accesso dell'applicazione non richiedono di interrogare più entità o tabelle insieme, la progettazione di più tabelle è un approccio valido e sufficiente.

Elementi costitutivi della modellazione dei dati in DynamoDB

In questa sezione verranno descritti i modelli di progettazione (o elementi costitutivi) che puoi utilizzare per l'applicazione.



Argomenti

- Blocco costitutivo di chiave di ordinamento composita
- Elemento costitutivo di multilocazione
- Elemento costitutivo di indice Sparse
- Elemento costitutivo di Time to Live
- Time to Live per elemento costitutivo di archiviazione
- Elemento costitutivo di partizionamento verticale
- Elemento costitutivo di partizionamento di scrittura

Blocco costitutivo di chiave di ordinamento composita

Si potrebbe pensare a NoSQL come a un database non relazionale. In definitiva, non c'è un motivo per cui le relazioni non possono essere inserite in uno schema DynamoDB; hanno semplicemente un aspetto diverso rispetto ai database relazionali e alle relative chiavi esterne. Uno dei modelli più critici che possiamo utilizzare per sviluppare una gerarchia logica dei dati in DynamoDB è una

chiave di ordinamento composita. Lo stile di progettazione più comune è quello in cui ogni livello della gerarchia (livello padre > livello figlio > livello nipote) è separato da un hashtag. Ad esempio, `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

Sebbene una chiave di partizione in DynamoDB richieda sempre il valore esatto per eseguire query sui dati, possiamo applicare alla chiave di ordinamento una condizione parziale da sinistra a destra, simile all'attraversamento di un albero binario.

Nell'esempio precedente, è disponibile un negozio di e-commerce con un carrello acquisti che deve essere mantenuto tra le sessioni utente. Ogni volta che l'utente esegue l'accesso, può scegliere di visualizzare l'intero carrello acquisti, inclusi gli elementi salvati per uso futuro. Tuttavia, al momento del pagamento, solo gli elementi nel carrello attivo devono essere caricati per l'acquisto. Poiché entrambe le `KeyConditions` richiedono esplicitamente le chiavi di ordinamento CART, i dati della lista dei desideri aggiuntivi vengono semplicemente ignorati da DynamoDB in fase di lettura. Sebbene gli elementi salvati e attivi facciano entrambi parte dello stesso carrello, devono essere gestiti in modo diverso nelle diverse parti dell'applicazione. Pertanto, l'applicazione di una `KeyCondition` al prefisso della chiave di ordinamento è il modo più ottimizzato per recuperare solo i dati necessari per ciascuna parte dell'applicazione.

Funzionalità principali di questo elemento costitutivo

- Gli elementi correlati vengono archiviati localmente tra loro per un accesso ai dati conveniente.
- Utilizzando `KeyCondition` le espressioni, i sottoinsiemi della gerarchia possono essere recuperati selettivamente, il che significa che non vengono sprecati RCUs
- Parti differenti dell'applicazione possono archiviare i relativi elementi sotto un prefisso specifico, evitando elementi sovrascritti o scritture in conflitto.

Elemento costitutivo di multilocazione

Molti clienti utilizzano DynamoDB per ospitare dati per applicazioni multi-tenant. Per questi scenari, desideriamo progettare lo schema in modo da mantenere tutti i dati di un singolo tenant nella propria partizione logica della tabella. Questo sfrutta il concetto di raccolta di elementi, che è un termine per tutti gli elementi in una tabella DynamoDB con la stessa chiave di partizione. Per ulteriori informazioni su come DynamoDB approccia la multilocazione, consulta Multitenancy on 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]

In questo esempio, gestiamo un sito di hosting di foto con potenzialmente migliaia di utenti. Inizialmente ogni utente caricherà foto solo sul proprio profilo, ma per impostazione predefinita un utente non può visualizzare le foto di qualsiasi altro utente. L'aggiunta di un ulteriore livello di isolamento all'autorizzazione della chiamata di ciascun utente all'API sarebbe ideale per garantire che i dati vengano richiesti solo dalla propria partizione, ma a livello di schema, le chiavi di partizione univoche sono sufficienti.

Funzionalità principali di questo elemento costitutivo

- La quantità di dati letta da qualsiasi utente o tenant può essere solo pari alla quantità totale di elementi nella relativa partizione
- La rimozione dei dati di un tenant a causa della chiusura di un account o di una richiesta di conformità può essere effettuata in modo discreto ed economico. Esegui semplicemente una query in cui la chiave di partizione è uguale al relativo ID tenant, quindi esegui un'operazione `DeleteItem` per ciascuna chiave primaria restituita

> **Note**
>
> Progettato pensando alla multi-tenancy, è possibile utilizzare diversi provider di chiavi di crittografia su un'unica tabella per isolare i dati in modo sicuro. L'AWS SDK di crittografia del database per Amazon DynamoDB consente di includere la crittografia lato client nei carichi di lavoro DynamoDB. Puoi eseguire la crittografia a livello di attributo, che ti consente di crittografare valori di attributo specifici prima di archiviarli nella tabella DynamoDB e di cercare attributi crittografati senza prima decrittografare l'intero database.

Elemento costitutivo di indice Sparse

A volte un modello di accesso richiede la ricerca di elementi che corrispondono a un elemento raro o a un elemento che riceve uno stato (che richiede una risposta riassegnata). Anziché eseguire regolarmente query sull'intero set di dati per questi elementi, possiamo sfruttare il fatto che gli indici secondari globali (GSI) sono scarsamente caricati con dati. Ciò significa che solo gli elementi della tabella di base che dispongono degli attributi definiti nell'indice verranno replicati nell'indice.

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

In questo esempio, possiamo vedere un caso d'uso IOT in cui ciascun dispositivo sul campo segnala periodicamente uno stato. Per la maggior parte dei report, ci aspettiamo che il dispositivo segnali tutto, ma a volte può essere presente un guasto che deve essere riassegnato a un tecnico di

riparazione. Per i report con un'escalation, l'attributo `EscalatedTo` viene aggiunto all'elemento, ma non è altrimenti presente. Il GSI in questo esempio è partizionato da `EscalatedTo` e, poiché il GSI porta con sé chiavi della tabella di base, possiamo ancora vedere quale DeviceID ha segnalato il guasto e a che ora.

Sebbene le letture siano più economiche delle scritture in DynamoDB, gli indici Sparse sono uno strumento molto potente per casi d'uso in cui le istanze di un tipo di elemento specifico sono rare, ma le letture per trovarle sono comuni.

Funzionalità principali di questo elemento costitutivo

- I costi di scrittura e archiviazione per il GSI sparso si applicano solo agli elementi che corrispondono allo schema di chiave, quindi il costo del GSI può essere sostanzialmente inferiore rispetto ad altri in cui sono replicati tutti gli elementi GSIs
- Una chiave di ordinamento composita può ancora essere utilizzata per restringere ulteriormente gli elementi che corrispondono alla query desiderata. Ad esempio, è possibile utilizzare un timestamp per la chiave di ordinamento per visualizzare solo i guasti segnalati negli ultimi X minuti (`SK > 5 minutes ago, ScanIndexForward: False`)

Elemento costitutivo di Time to Live

La maggior parte dei dati ha una certa durata per cui può essere considerato utile mantenerli in un datastore primario. Per facilitare l'uscita dei dati da DynamoDB, è disponibile una funzionalità chiamata Time to Live (TTL). La funzionalità TTL consente di definire un attributo specifico a livello di tabella che deve essere monitorato per gli elementi con un timestamp epoch (che appartiene al passato). Ciò consente di eliminare gratuitamente i record scaduti dalla tabella.

> **Note**
>
> Se si utilizzano le Tabelle globali versione 2019.11.21 (Corrente) e si utilizza anche la funzionalità Time to Live, DynamoDB replica le eliminazioni TTL in tutte le tabelle di replica. L'eliminazione TTL iniziale non consuma capacità di scrittura nella regione in cui si verifica la scadenza del TTL. Tuttavia, l'eliminazione TTL replicata nelle tabelle di replica consuma capacità di scrittura replicata in ciascuna regione di replica e verranno addebitati i costi applicabili.

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

In questo esempio, è disponibile un'applicazione progettata per consentire a un utente di creare messaggi di breve durata. Quando un messaggio viene creato in DynamoDB, l'attributo TTL viene impostato su una data sette giorni nel futuro dal codice dell'applicazione. In circa sette giorni, DynamoDB rileverà che il timestamp di epoca di questi elementi appartiene al passato e li eliminerà.

Poiché le eliminazioni effettuate dal TTL sono gratuite, si consiglia di utilizzare questa funzionalità per rimuovere i dati storici dalla tabella. Ciò ridurrà le spese mensili di archiviazione complessive e i costi delle letture utente poiché le query devono recuperare una quantità minore di dati. Sebbene TTL sia abilitato a livello di tabella, l'utente deve decidere per quali elementi o entità creare un attributo TTL e fino a quando nel futuro impostare il timestamp di epoca.

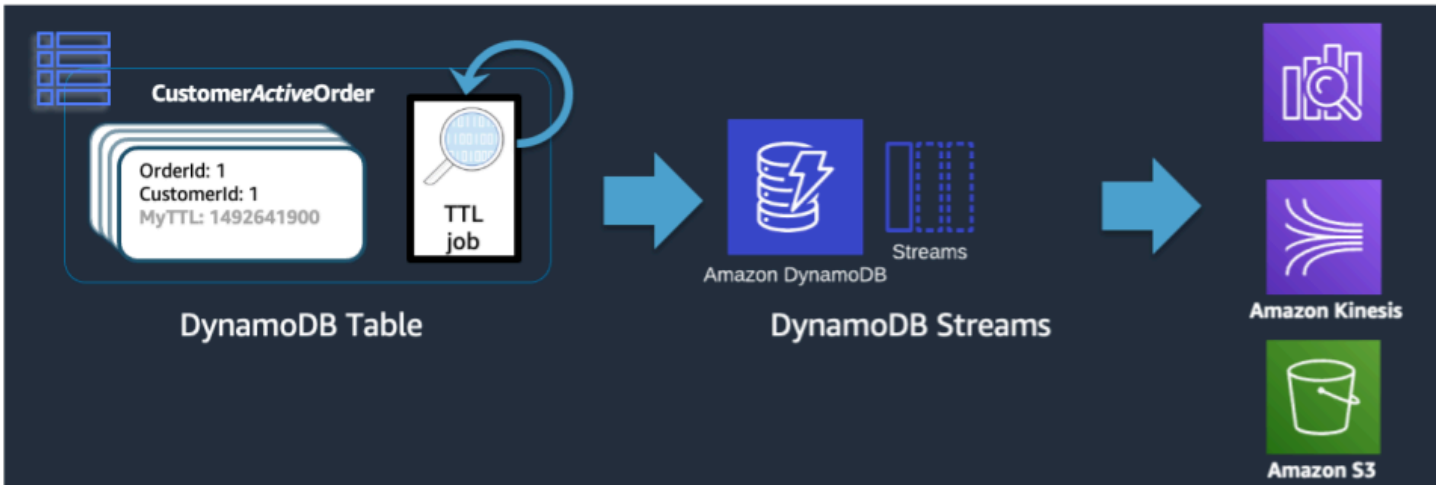
Funzionalità principali di questo elemento costitutivo

- Eliminazioni TTL vengono eseguite dietro le quinte senza ripercussioni sulle prestazioni della tabella
- TTL è un processo asincrono che viene eseguito all'incirca ogni sei ore, ma l'eliminazione di un record scaduto può richiedere oltre 48 ore
  - Non fare affidamento sulle eliminazioni TTL per casi d'uso come record di blocco o gestione dello stato, se i dati obsoleti devono essere eliminati in meno di 48 ore
- Puoi assegnare all'attributo TTL un nome attributo valido, ma il valore deve essere un tipo numero

Time to Live per elemento costitutivo di archiviazione

Sebbene TTL sia uno strumento efficace per eliminare dati meno recenti da DynamoDB, molti casi d'uso richiedono che un archivio dei dati venga mantenuto per un periodo di tempo più lungo rispetto

al datastore principale. In questo caso, è possibile sfruttare l'eliminazione temporizzata dei record di TTL per inviare i record scaduti in un datastore a lungo termine.



Quando un'eliminazione TTL viene eseguita da DynamoDB, viene ancora inviata nel flusso DynamoDB come un evento `Delete`. Tuttavia, quando TTL DynamoDB è quello che esegue l'eliminazione, nel record di flusso è presente un attributo di `principal:dynamodb`. Utilizzando un abbonato Lambda a DynamoDB Stream, possiamo applicare un filtro eventi solo per l'attributo principale di DynamoDB e sapere che tutti i record che corrispondono a quel filtro devono essere inviati a un archivio come Amazon Glacier.

Funzionalità principali di questo elemento costitutivo

- Una volta che le letture a bassa latenza di DynamoDB non sono più necessarie per gli elementi storici, la loro migrazione a un servizio di storage più freddo come Amazon Glacier può ridurre significativamente i costi di storage, soddisfacendo al contempo le esigenze di conformità dei dati del tuo caso d'uso
- Se i dati vengono mantenuti in Amazon S3, è possibile utilizzare strumenti di analisi economici come Amazon Athena o Redshift Spectrum per eseguire l'analisi storica dei dati

Elemento costitutivo di partizionamento verticale

Gli utenti che hanno familiarità con un database di modelli di documenti sono a conoscenza dell'idea di archiviare tutti i dati correlati all'interno di un singolo documento JSON. Sebbene DynamoDB supporti i tipi di dati JSON, non supporta l'esecuzione di `KeyConditions` su JSON annidati. Poiché `KeyConditions` sono proprio i fattori che determinano la quantità di dati letti dal disco e il numero effettivo di dati consumati da RCUs una query, ciò può comportare inefficienze su larga scala.
Per ottimizzare meglio le scritture e le letture di DynamoDB, si consiglia di suddividere le singole

entità del documento in singoli elementi DynamoDB, operazione nota anche come partizionamento verticale.

```
{
        ”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

Il partizionamento verticale, come mostrato in precedenza, è un esempio chiave di progettazione tabella singola, ma può anche essere implementato su più tabelle, se lo si desidera. Poiché DynamoDB addebita le scritture in incrementi di 1 KB, devi partizionare idealmente il documento in un modo che generi inferiori a 1 KB.

Funzionalità principali di questo elemento costitutivo

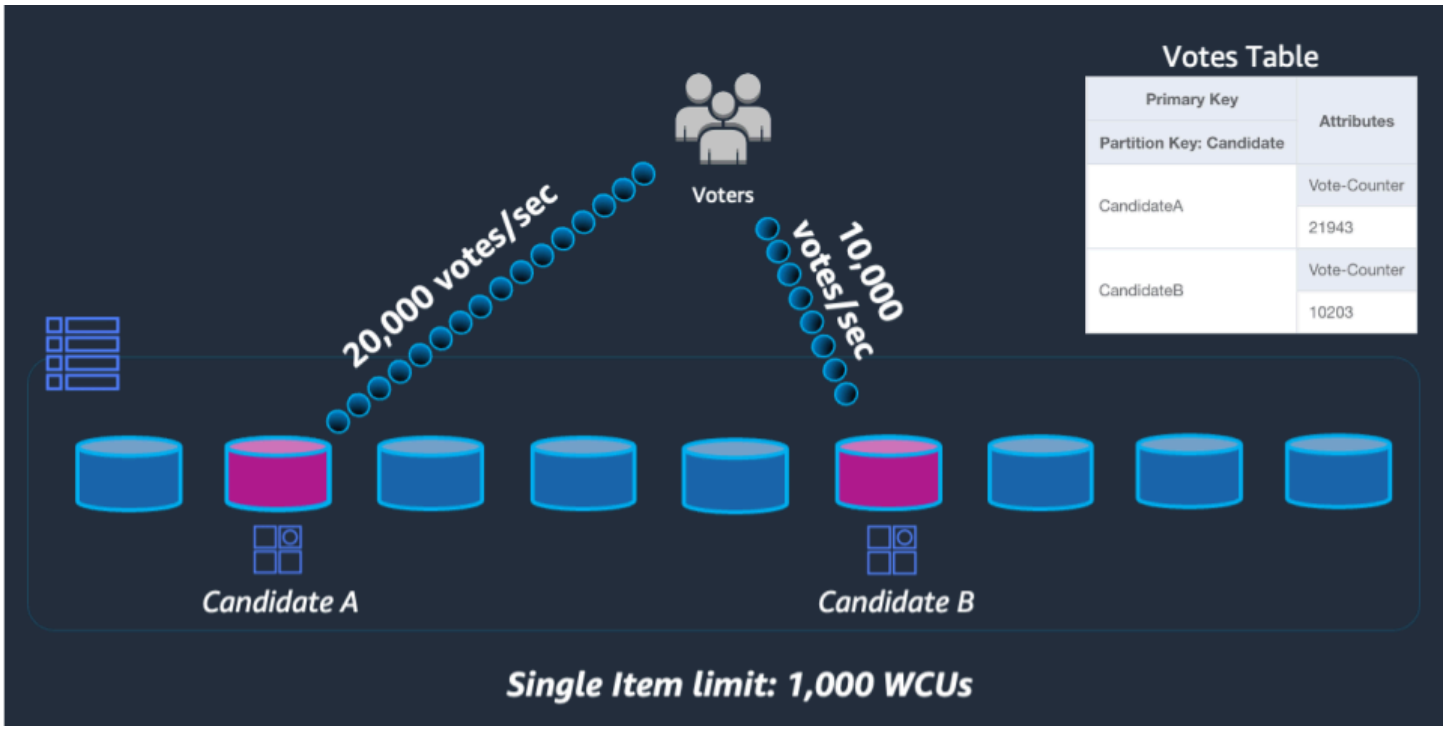
- Una gerarchia di relazioni dei dati viene mantenuta tramite prefissi delle chiavi di ordinamento, in modo che la singola struttura del documento possa essere ricostruita lato client, se necessario
- I singoli componenti della struttura di dati possono essere aggiornati in maniera indipendente, determinando piccoli aggiornamenti elemento di solo 1 WCU
- Utilizzando la chiave di ordinamento `BeginsWith`, l'applicazione può recuperare dati simili in una singola query, aggregando i costi di lettura per ridurre il costo/la latenza totale
- I documenti di grandi dimensioni possono facilmente superare il limite di dimensioni del singolo elemento di 400 KB in DynamoDB e il partizionamento verticale consente di aggirare questo limite

Elemento costitutivo di partizionamento di scrittura

Uno dei pochi limiti fissi implementati da DynamoDB è la limitazione della velocità di trasmissione effettiva che una singola partizione fisica può mantenere al secondo (non necessariamente una singola chiave di partizione). Questi limiti sono attualmente:

- 1.000 WCU (o 1.000 <=1 KB elementi scritti al secondo) e 3.000 RCU (o 3.000 <=4 KB letture al secondo) fortemente coerenti o
- 6000 <=4 KB letture al secondo a consistenza finale

Nel caso in cui le richieste rispetto alla tabella superino uno di questi limiti, un errore di `ThroughputExceededException`, più comunemente noto come limitazione (della larghezza di banda della rete), viene restituito all'SDK client. I casi d'uso che richiedono operazioni di lettura oltre tale limite saranno per lo più gestiti al meglio posizionando una cache di lettura davanti a DynamoDB, ma le operazioni di scrittura richiedono una progettazione a livello di schema nota come partizionamento di scrittura.



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

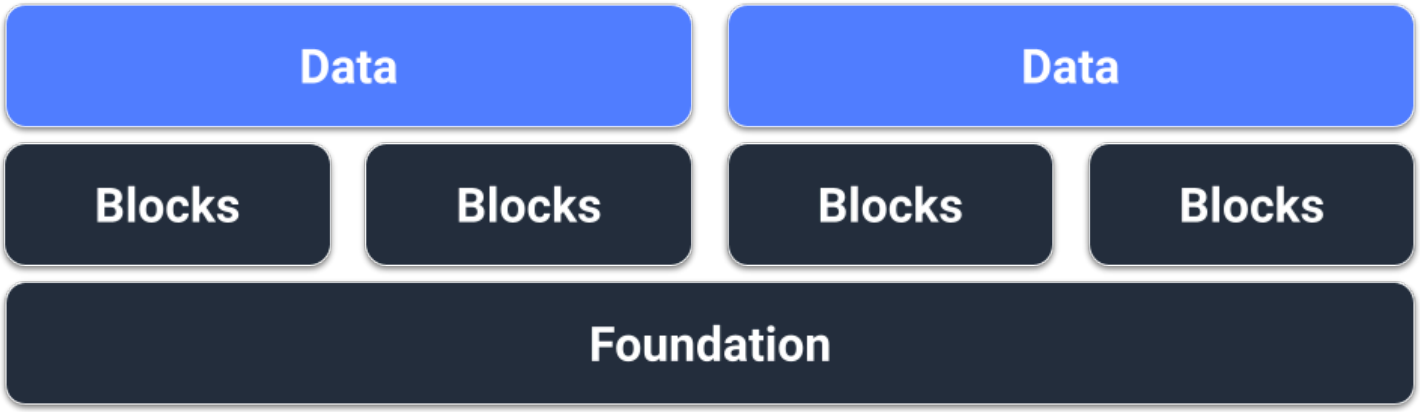
Per risolvere questo problema, un numero intero casuale verrà aggiunto alla fine della chiave di partizione per ogni partecipante nel codice `UpdateItem` dell'applicazione. L'intervallo del generatore di numeri interi casuali dovrà avere un limite superiore che corrisponde o supera la quantità prevista di scritture al secondo per un determinato partecipante diviso 1000. Per supportare 20.000 voti al secondo, l'aspetto è simile a rand (0,19). Ora che i dati sono archiviati in partizioni logiche separate, devono essere ricombinati in fase di lettura. Poiché i totali dei voti non devono essere in tempo reale, una funzione Lambda pianificata per leggere tutte le partizioni di voto ogni X minuti può eseguire un'aggregazione occasionale per ogni partecipante e riscriverla in un singolo record totale di voti per le letture dal vivo.

Funzionalità principali di questo elemento costitutivo

- Per casi d'uso con una velocità di trasmissione effettiva di scrittura estremamente elevata per una determinata chiave di partizione che non può essere evitata, le operazioni di scrittura possono essere distribuite artificialmente su più partizioni DynamoDB
- GSIs anche con una chiave di partizione a bassa cardinalità dovrebbe utilizzare questo schema, poiché la limitazione su un GSI applicherà una contropressione alle operazioni di scrittura sulla tabella di base

Pacchetti di progettazione dello schema di modellazione dei dati in DynamoDB

Informazioni sui pacchetti di progettazione degli schemi di modellazione dei dati per DynamoDB, comprensive di casi d’uso, modelli di accesso e progetti di schemi finali per social network, profili di gioco, gestione dei reclami, pagamenti ricorrenti, stato dei dispositivi e negozi online.



Prerequisiti

Prima di tentare di progettare lo schema per DynamoDB, occorre innanzitutto raccogliere alcuni dati dei prerequisiti sul caso d'uso che lo schema deve supportare. A differenza dei database relazionali, DynamoDB è condiviso per impostazione predefinita, a indicare che i dati risiederanno su più server dietro le quinte, pertanto è importante progettare in base all'ubicazione dei dati. Per ogni progettazione dello schema, occorre assemblare il seguente elenco:

- Elenco di entità (diagramma ER)
- Volumi e velocità di trasmissione effettiva stimati per ciascuna entità
- Modelli di accesso che devono essere supportati (query e scritture)
- Requisiti di conservazione dei dati

Argomenti

- Progettazione dello schema di social network in DynamoDB
- Progettazione dello schema del profilo di gioco in DynamoDB
- Progettazione dello schema del sistema di gestione dei reclami in DynamoDB
- Progettazione dello schema dei pagamenti ricorrenti in DynamoDB
- Monitoraggio degli aggiornamenti dello stato dei dispositivi in DynamoDB
- Utilizzo di DynamoDB come archivio dati per un negozio online

Progettazione dello schema di social network in DynamoDB

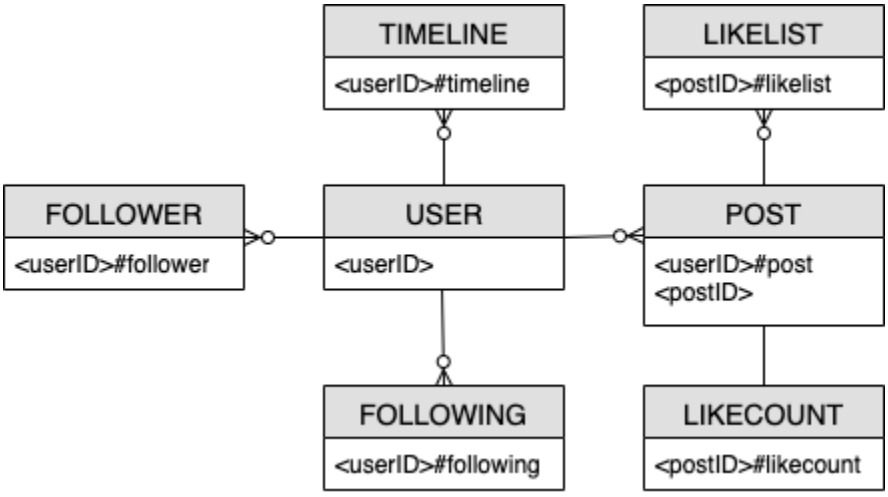
Caso d'uso aziendale di social network

In questo caso d'uso viene descritto l'utilizzo di DynamoDB come un social network. Un social network è un servizio online che consente a diversi utenti di interagire tra loro. Il social network che verrà progettato consentirà all'utente di visualizzare una sequenza temporale composta da post, follower, utenti seguiti e post scritti dagli utenti seguiti. I modelli di accesso per questo schema sono:

- Ottenere informazioni utente per un userID specifico
- Ottenere l'elenco di follower per un userID specifico
- Ottenere l'elenco di follower per un userID specifico
- Ottenere l'elenco di post per un userID specifico
- Ottenere l'elenco di utenti a cui piace il post per un postID specifico
- Ottenere il numero di like per un postID specifico
- Ottenere la sequenza temporale per un userID specifico

Diagramma delle relazioni tra entità del social network

Questo è il diagramma delle relazioni tra entità (ERD) che useremo per la progettazione dello schema di social network.



Modelli di accesso di social network

Questi sono i modelli di accesso che creeremo per la progettazione dello schema di social network.

- `getUserInfoByUserID`
- `getFollowerListByUserID`
- `getFollowingListByUserID`
- `getPostListByUserID`
- `getUserLikesByPostID`
- `getLikeCountByPostID`
- `getTimelineByUserID`

Evoluzione della progettazione dello schema di social network

DynamoDB è un database NoSQL, quindi non consente di eseguire un join, ovvero un'operazione che combina dati di più database. I clienti che non hanno familiarità con DynamoDB potrebbero applicare filosofie di progettazione del sistema di gestione dei database relazionali (RDBMS) (come la creazione di una tabella per ogni entità) a DynamoDB quando non è necessario. Lo scopo della progettazione tabella singola di DynamoDB è scrivere dati in un formato pre-unito in base al modello di accesso dell'applicazione e quindi utilizzare immediatamente i dati senza calcoli aggiuntivi. Per ulteriori informazioni, consulta Progettazione tabella singola e progettazione tabelle multiple in DynamoDB.

Ora illustreremo come intendiamo sviluppare la progettazione dello schema per gestire tutti i modelli di accesso.

Fase 1: Gestire il modello di accesso 1 (**`getUserInfoByUserID`**)

Per ottenere le informazioni di un determinato utente, devi eseguire una `Query` della tabella di base con una condizione chiave di `PK=<userID>`. L'operazione di interrogazione consente di impaginare i risultati, il che può essere utile quando un utente ha molti follower. Per ulteriori informazioni su Query, consulta Esecuzione di query in DynamoDB.

In questo esempio, vengono monitorati due tipi di dati per gli utenti: il "conteggio" e le "informazioni". Il "conteggio" di un utente indica quanti follower ha, quanti utenti sta seguendo e quanti post ha creato. Le "informazioni" di un utente indicano le informazioni personali, come il nome.

Questi due tipi di dati sono rappresentati dai due elementi sottostanti. L'elemento che contiene "conteggio" nella sua chiave di ordinamento (SK) è più probabile che cambi rispetto all'elemento con "informazioni". DynamoDB considera le dimensioni elemento così come appaiono prima e dopo l'aggiornamento e la velocità di trasmissione effettiva assegnata consumata rifletterà il più grande

di questi valori. Anche se aggiorni solo un sottoinsieme di attributi dell'elemento, `UpdateItem` utilizza comunque la quantità completa della velocità di trasmissione effettiva assegnata (il valore maggiore tra le dimensioni "prima" e "dopo"). Puoi ottenere gli elementi tramite una singola operazione `Query` e utilizzare `UpdateItem` per aggiungere o sottrarre attributi numerici esistenti.

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://....	...

Fase 2: Gestire il modello di accesso 2 (**`getFollowerListByUserID`**)

Per ottenere un elenco di utenti che seguono un determinato utente, occorre eseguire una `Query` della tabella di base con una condizione chiave di `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				

Fase 3: Gestire il modello di accesso 3 (**`getFollowingListByUserID`**)

Per ottenere un elenco di utenti seguiti da un determinato utente, occorre eseguire una `Query` della tabella di base con una condizione chiave di `PK=<userID>#following`. Quindi è possibile utilizzare un'operazione `TransactWriteItems` per raggruppare più richieste e procedere come segue:

- Aggiungi l'Utente A all'elenco di follower dell'Utente B, quindi incrementa il conteggio di follower dell'Utente B di uno.
- Aggiungi l'Utente B all'elenco di follower dell'Utente A, quindi incrementa il conteggio di follower dell'Utente A di 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				

Fase 4: Gestire il modello di accesso 4 (**`getPostListByUserID`**)

Per ottenere un elenco di post creati da un determinato utente, devi eseguire una `Query` della tabella di base con una condizione chiave di `PK=<userID>#post`. Una cosa importante che occorre notare qui è che i postID di un utente devono essere incrementali: il secondo valore postID deve essere maggiore del primo valore postID (poiché gli utenti desiderano vedere i post in modo ordinato). A questo scopo, genera i postID in base a un valore temporale come un ULID (Universally Unique Lexicographically Sortable Identifier).

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	

Fase 5: Gestire il modello di accesso 5 (**`getUserLikesByPostID`**)

Per ottenere un elenco di utenti che hanno messo like al post di un determinato utente, devi eseguire una `Query` della tabella di base con una condizione chiave di `PK=<postID>#likelist`. Questo approccio è lo stesso modello utilizzato per recuperare gli elenchi di follower e di utenti seguiti nel modello di accesso 2 (`getFollowerListByUserID`) e nel modello di accesso 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				

Fase 6: Gestire il modello di accesso 6 (**`getLikeCountByPostID`**)

Per ottenere un conteggio di like per un determinato post, occorre eseguire un'operazione `GetItem` sulla tabella di base con una condizione chiave di `PK=<postID>#likecount`. Questo modello di accesso può causare problemi di limitazione (della larghezza di banda della rete) ogni volta che un utente con molti follower (ad esempio, una celebrità) crea un post poiché la limitazione (della larghezza di banda della rete) si verifica quando la velocità di trasmissione effettiva di una partizione supera i 1000 WCU al secondo. Questo problema non è una conseguenza di DynamoDB, ma appare solo in DynamoDB poiché si trova alla fine dello stack software.

Occorre valutare se è davvero essenziale che tutti gli utenti visualizzino il conteggio dei like contemporaneamente o se questo può avvenire gradualmente nel tempo. In generale, il conteggio dei like di un post non deve essere immediatamente accurato al 100%. È possibile implementare

questa strategia inserendo una coda tra l'applicazione e DynamoDB in modo che gli aggiornamenti avvengano periodicamente.

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			

Fase 7: Gestire il modello di accesso 7 (**`getTimelineByUserID`**)

Per ottenere la sequenza temporale per un determinato post, occorre eseguire un'operazione `Query` sulla tabella di base con una condizione chiave di `PK=<userID>#timeline`. Consideriamo uno scenario in cui i follower di un utente devono visualizzare i loro post in modo sincrono. Ogni volta che un utente scrive un post, il relativo elenco di follower viene letto e i valori userID e postID vengono inseriti lentamente nella chiave della sequenza temporale di tutti i suoi follower. Quindi, all'avvio dell'applicazione, puoi leggere la chiave della sequenza temporale con l'operazione `Query` e riempire la schermata della sequenza temporale con una combinazione di userID e postID utilizzando l'operazione `BatchGetItem` per tutti i nuovi elementi. Non è possibile leggere la sequenza temporale con una chiamata API, ma questa è una soluzione più conveniente se i post possono essere modificati frequentemente.

Poiché la sequenza temporale visualizza i post recenti, occorre un modo per cancellare quelli vecchi. Anziché usare WCU per eliminarli, puoi utilizzare la funzionalità TTL di DynamoDB per eseguire questa operazione gratuitamente.

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			

Tutti i modelli di accesso e il modo in cui la progettazione dello schema li affronta sono riassunti nella tabella seguente:

Modello di accesso	Tabella di base/GSI/LSI	Operazione	Valore della chiave di partizione	Valore della chiave di ordinamento	Altre condizioni/ filtri
getUserInfoByUserID	Tabella di base	Query	PK=<userID>		
getFollowerListByUserID	Tabella di base	Query	PK=<userID>#follower		
getFollowingListByUserID	Tabella di base	Query	PK=<userID>#following		
getPostListByUserID	Tabella di base	Query	PK=<userID>#post		
getUserLikesByPostID	Tabella di base	Query	PK=<postID>#likelist		
getLikeCountByPostID	Tabella di base	GetItem	PK=<postID>#likecount		
getTimelineByUserID	Tabella di base	Query	PK=<userID>#timeline		

Schema finale del social network

Di seguito è riportata la progettazione dello schema finale. Per scaricare questa progettazione dello schema come un file JSON, consulta Esempi di DynamoDB su GitHub.

Tabella di 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			

Utilizzo di NoSQL Workbench con questa progettazione dello schema

Puoi importare questo schema finale in NoSQL Workbench, uno strumento visivo che fornisce funzionalità di modellazione dei dati, visualizzazione dei dati e sviluppo di query per DynamoDB, per esplorare e modificare ulteriormente il tuo nuovo progetto. Per iniziare, segui queste fasi:

1. Scarica NoSQL Workbench. Per ulteriori informazioni, consulta the section called “Scarica”.
2. Scarica il file dello schema JSON elencato in precedenza, che si trova già nel formato del modello NoSQL Workbench.
3. Importa il file dello schema JSON in NoSQL Workbench. Per ulteriori informazioni, consulta the section called “Importazione di un modello esistente”.

4. Dopo che è stato importato in NOSQL Workbench, puoi modificare il modello di dati. Per ulteriori informazioni, consulta the section called “Modifica di un modello esistente”.
5. Per visualizzare il tuo modello di dati, aggiungere dati di esempio o importare dati di esempio da un file CSV, utilizza la funzionalità Data Visualizer di NoSQL Workbench.

Progettazione dello schema del profilo di gioco in DynamoDB

Caso d'uso aziendale profilo di gioco

In questo caso d'uso viene descritto l'utilizzo di DynamoDB per archiviare i profili dei giocatori per un sistema di gioco. Gli utenti (in questo caso, i giocatori) devono creare profili prima di poter interagire con molti giochi moderni, in particolare quelli online. I profili di gioco includono in genere i seguenti:

- Informazioni di base come il nome utente
- Dati di gioco come elementi e apparecchiature
- Record di gioco come operazioni e attività
- Informazioni social come elenchi di amici

Per soddisfare i requisiti granulari di accesso alle query di dati per questa applicazione, le chiavi primarie (chiave di partizione e chiave di ordinamento) utilizzeranno nomi generici (PK e SK) in modo da poter essere sovraccaricate con vari tipi di valori, come descritto di seguito.

I modelli di accesso per questa progettazione dello schema sono:

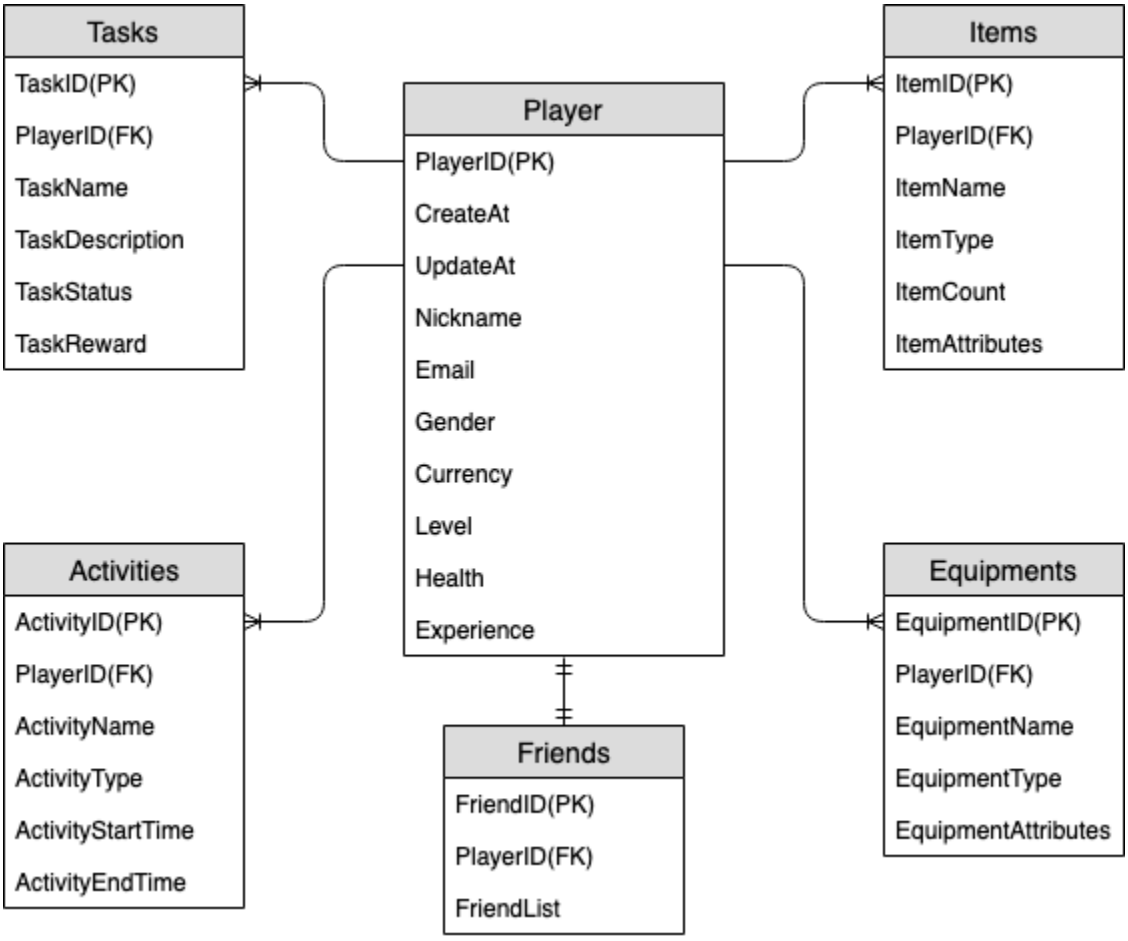
- Ottenere l'elenco di amici di un utente
- Ottenere tutte le informazioni di un giocatore
- Ottenere l'elenco di elementi di un utente
- Ottenere un elemento specifico dall'elenco di elementi dell'utente
- Aggiornare il personaggio di un utente
- Aggiornare il conteggio elementi per un utente

Le dimensioni del profilo di gioco possono variare in base ai giochi. La compressione di valori di attributi large permette la loro archiviazione nei limiti degli elementi in DynamoDB, riducendo così i costi. La strategia di gestione della velocità di trasmissione effettiva dipende da vari fattori quali: il numero di giocatori, il numero di partite giocate al secondo e la stagionalità del carico di lavoro. In

genere, per un gioco appena lanciato, il numero di giocatori e il livello di popolarità sono sconosciuti, pertanto inizieremo con la modalità di velocità di trasmissione effettiva on demand.

Diagramma delle relazioni tra entità del profilo di gioco

Questo è il diagramma delle relazioni tra entità (ERD) utilizzato per la progettazione dello schema del profilo di gioco.



Modelli di accesso del profilo di gioco

Questi sono i modelli di accesso che creeremo per la progettazione dello schema di social network.

- `getPlayerFriends`
- `getPlayerAllProfile`
- `getPlayerAllItems`
- `getPlayerSpecificItem`
- `updateCharacterAttributes`

- `updateItemCount`

Evoluzione della progettazione dello schema del profilo di gioco

Dall'ERD precedente si evince che si tratta di un tipo di relazione uno a molti della modellazione dei dati. In DynamoDB, i modelli di dati uno a molti possono essere organizzati in raccolte di elementi. Ciò differisce dai database relazionali tradizionali in cui più tabelle vengono create e collegate tramite chiavi esterne. Una raccolta di elementi è un gruppo di elementi che condividono lo stesso valore della chiave di partizione ma hanno valori di chiave di ordinamento differenti. All'interno di una raccolta di elementi, ogni elemento dispone di un valore di chiave di ordinamento univoco che lo distingue dagli altri elementi. Tenendo questo a mente, utilizziamo il modello seguente per i valori `HASH` e `RANGE` per ogni tipo di entità.

Per iniziare, utilizziamo nomi generici come `PK` e `SK` per archiviare diversi tipi di entità nella stessa tabella in modo da rendere il modello orientato al futuro. Per una migliore leggibilità, possiamo includere prefissi per indicare il tipo di dati o includere un attributo arbitrario chiamato `Entity_type` o `Type`. Nell'esempio corrente, utilizziamo una stringa che inizia con `player` per archiviare `player_ID` come `PK`; utilizziamo `entity name#` come il prefisso di `SK` e aggiungiamo un attributo `Type` per indicare il tipo di entità di questo dato. Questo consente di supportare l'archiviazione di più tipi di entità in futuro e di utilizzare tecnologie avanzate come Overload di GSI e GSI Sparse per soddisfare più modelli di accesso.

Iniziamo implementando i modelli di accesso. I modelli di accesso come l'aggiunta di giocatori e l'aggiunta di apparecchiature possono essere realizzati tramite l'operazione `PutItem`, pertanto possiamo ignorarli. In questo documento ci concentreremo sui modelli di accesso tipici elencati in precedenza.

Fase 1: Gestire il modello di accesso 1 (**`getPlayerFriends`**)

Gestiamo il modello di accesso 1 (`getPlayerFriends`) con questa fase. Nella progettazione attuale, l'amicizia è semplice e il numero di amici nel gioco è limitato. Per semplicità, utilizziamo un tipo di dati di elenco per archiviare gli elenchi di amici (modellazione 1:1). In questa progettazione, utilizziamo `GetItem` per soddisfare questo modello di accesso. Nell'operazione `GetItem`, forniamo esplicitamente il valore della chiave di partizione e della chiave di ordinamento per ottenere un elemento specifico.

Tuttavia, se in un gioco sono presenti molti amici e le relazioni tra di loro sono complesse (ad esempio, le amicizie sono bidirezionali con una componente di invito e di accettazione),

è necessario utilizzare una relazione molti a molti per archiviare ogni amico individualmente, in modo da dimensionare fino a un elenco di amici di dimensioni illimitate. Se l'evoluzione dell'amicizia implica l'utilizzo di più elementi contemporaneamente, le transazioni DynamoDB possono essere utilizzate per raggruppare più azioni e inviarle come un'unica operazione `TransactWriteItems` o `TransactGetItems` tutto o niente.

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"}}}]

Fase 2: Gestire i modelli di accesso 2 (**`getPlayerAllProfile`**), 3 (**`getPlayerAllItems`**) e 4 (**`getPlayerSpecificItem`**)

Usiamo questa fase per gestire i modelli di accesso 2 (`getPlayerAllProfile`), 3 (`getPlayerAllItems`) e 4 (`getPlayerSpecificItem`). Ciò che accomuna questi tre modelli di accesso è una query di intervallo che utilizza l'operazione `Query`. A seconda dell'ambito della query, vengono usate Condizione chiave ed Espressioni filtro, comunemente utilizzate nello sviluppo pratico.

Nell'operazione Query, forniamo un singolo valore per Partition Key e otteniamo tutti gli elementi con tale valore Partition Key. Il modello di accesso 2 (`getPlayerAllProfile`) è implementato in questo modo. Facoltativamente, possiamo aggiungere un'espressione di condizione della chiave di ordinamento, ovvero una stringa che determina gli elementi da leggere dalla tabella. Il modello di accesso 3 (`getPlayerAllItems`) viene implementato aggiungendo la chiave di condizione della chiave di ordinamento begins_with `ITEMS#`. Inoltre, al fine di semplificare lo sviluppo del lato applicativo, possiamo utilizzare le espressioni di filtro per implementare il modello di accesso 4 (`getPlayerSpecificItem`).

Di seguito è riportato un esempio di pseudocodice che utilizza un'espressione di filtro che filtra gli elementi della categoria `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"}}}
	ITEMS#003	Type	ItemName	ItemType	ItemCount	ItemAttributes
		Item	Sword of the Dragon	Weapon	1	{"M":{"ATK":{"N":"100"},"DEF":{"N":"50"}}}

> **Note**
>
> Un'espressione di filtro viene applicata al termine di un'operazione Query, ma prima che i risultati vengano restituiti al client. Pertanto, una Query consuma la stessa quantità di capacità in lettura, a prescindere che sia presente un'espressione di filtro.

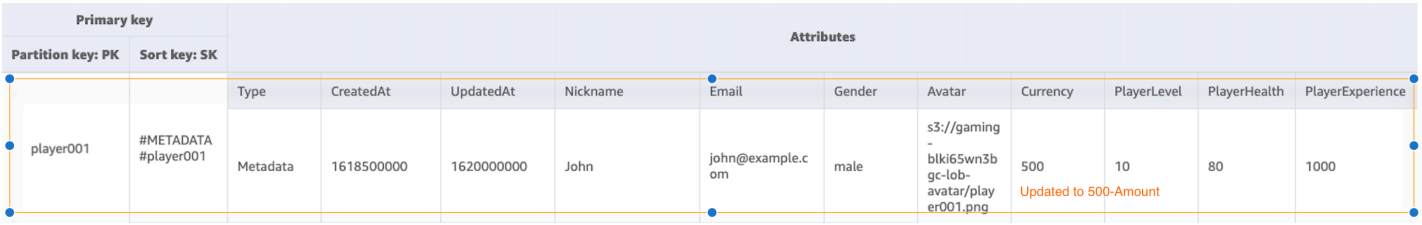
Se il modello di accesso consiste nell'eseguire query di un set di dati di grandi dimensioni e filtrare una grande quantità di dati per conservare solo un piccolo sottoinsieme di dati, l'approccio appropriato è progettare la Partition Key e la chiave di ordinamento DynamoDB in modo più efficace. Nell'esempio precedente, per ottenere un determinato `ItemType`, se sono presenti molti oggetti per ciascun giocatore e la richiesta di un determinato `ItemType` è un modello di accesso tipico, sarebbe più efficiente inserire `ItemType` in SK come una chiave composita. L'aspetto del modello di dati è simile a: `ITEMS#ItemType#ItemId`.

Fase 3: Gestire i modelli di accesso 5 (**`updateCharacterAttributes`**) e 6 (**`updateItemCount`**)

Utilizziamo questa fase per gestire i modelli di accesso 5 (`updateCharacterAttributes`) e 6 (`updateItemCount`). Quando il giocatore deve modificare il personaggio, ad esempio riducendo la valuta o modificando la quantità di una determinata arma nei suoi elementi, utilizza `UpdateItem` per implementare questi modelli di accesso. Per aggiornare la valuta di un giocatore ma garantire che non scenda mai al di sotto di un importo minimo, possiamo aggiungere the section called “Esempio di CLI” per ridurre il saldo solo se è maggiore o uguale all'importo minimo. Di seguito è riportato un esempio di pseudocodice:

```
UpdateExpression: "SET currency = currency - :amount"
```

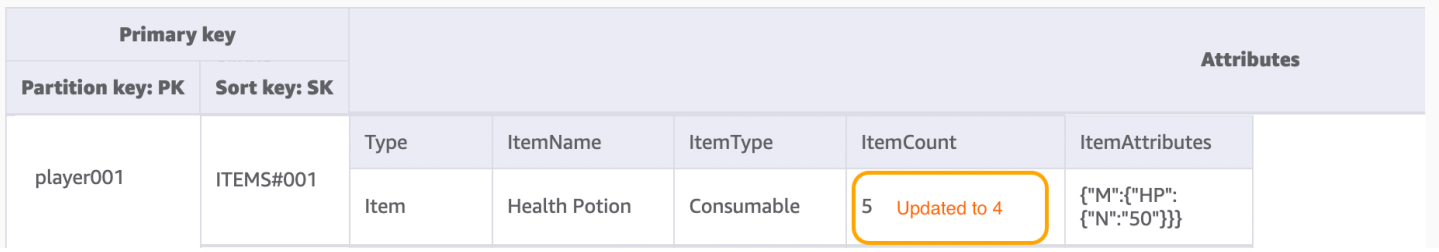
```
ConditionExpression: "currency >= :minAmount"
```



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

Durante lo sviluppo con DynamoDB e l'utilizzo di contatori atomici per diminuire l'inventario, possiamo garantire l'idempotenza utilizzando il blocco ottimistico. Di seguito è riportato un esempio di pseudocodice per contatori atomici:

```
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"}}}

Inoltre, in uno scenario in cui il giocatore acquista un elemento con valuta, l'intero processo deve detrarre la valuta e aggiungere contemporaneamente un elemento. Possiamo usare transazioni DynamoDB per raggruppare più azioni e inviarle come una singola operazione `TransactWriteItems` o `TransactGetItems` tutto o niente. `TransactWriteItems` è un'operazione di scrittura sincrona e idempotente che raggruppa fino a 100 azioni di scrittura in una singola operazione tutto o niente. Le operazioni vengono completate in modo atomico, in maniera tale che abbiano tutte esito positivo oppure abbiano tutte esito negativo. Le transazioni consentono di eliminare il rischio di duplicazione o di scomparsa della valuta. Per ulteriori informazioni sulle transazioni, consulta Esempio di transazioni di DynamoDB.

Tutti i modelli di accesso e il modo in cui la progettazione dello schema li affronta sono riassunti nella tabella seguente:

Modello di accesso	Tabella di base/GSI/LSI	Operazione	Valore della chiave di partizione	Valore della chiave di ordinamento	Altre condizioni/ filtri
getPlayer Friends	Tabella di base	GetItem	PK=PlayerID	SK="FRIENDS#playerID"	
getPlayer AllProfile	Tabella di base	Query	PK=PlayerID		
getPlayer AllItems	Tabella di base	Query	PK=PlayerID	SK begins_with "ITEMS#"	
getPlayer SpecificItem	Tabella di base	Query	PK=PlayerID	SK begins_with "ITEMS#"	filterExpression: "ItemType = :itemType" expressionAttributeValues: { ":itemType": "Weapon" }
updateCharacterAttributes	Tabella di base	UpdateItem	PK=PlayerID	SK="#METADATA#playerID"	UpdateExpression: "SET currency = currency - :amount" ConditionExpression: "currency >= :minAmount"
updateItemCount	Tabella di base	UpdateItem	PK=PlayerID	SK ="ITEMS#ItemID"	update-expression: "SET ItemCount

Modello di accesso	Tabella di base/GSI/LSI	Operazione	Valore della chiave di partizione	Valore della chiave di ordinamento	Altre condizioni/ filtri
					= ItemCount - :incr" expression-attribute-values : '{":incr": {"N":"1"}}'

Schema finale del profilo di gioco

Di seguito è riportata la progettazione dello schema finale. Per scaricare questa progettazione dello schema come un file JSON, consulta Esempi di DynamoDB su GitHub.

Tabella di 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"}}}						

Utilizzo di NoSQL Workbench con questa progettazione dello schema

Puoi importare questo schema finale in NoSQL Workbench, uno strumento visivo che fornisce funzionalità di modellazione dei dati, visualizzazione dei dati e sviluppo di query per DynamoDB, per esplorare e modificare ulteriormente il tuo nuovo progetto. Per iniziare, segui queste fasi:

1. Scarica NoSQL Workbench. Per ulteriori informazioni, consulta the section called “Scarica”.
2. Scarica il file dello schema JSON elencato in precedenza, che si trova già nel formato del modello NoSQL Workbench.
3. Importa il file dello schema JSON in NoSQL Workbench. Per ulteriori informazioni, consulta the section called “Importazione di un modello esistente”.
4. Dopo che è stato importato in NOSQL Workbench, puoi modificare il modello di dati. Per ulteriori informazioni, consulta the section called “Modifica di un modello esistente”.
5. Per visualizzare il tuo modello di dati, aggiungere dati di esempio o importare dati di esempio da un file CSV, utilizza la funzionalità Data Visualizer di NoSQL Workbench.

Progettazione dello schema del sistema di gestione dei reclami in DynamoDB

Caso d'uso aziendale del sistema di gestione dei reclami

DynamoDB è un database ideale per un sistema di gestione dei reclami (o contact center) in quanto la maggior parte dei modelli di accesso ad essi associati sarebbero ricerche transazionali basate su valori chiave. I modelli di accesso tipici in questo scenario sarebbero i seguenti:

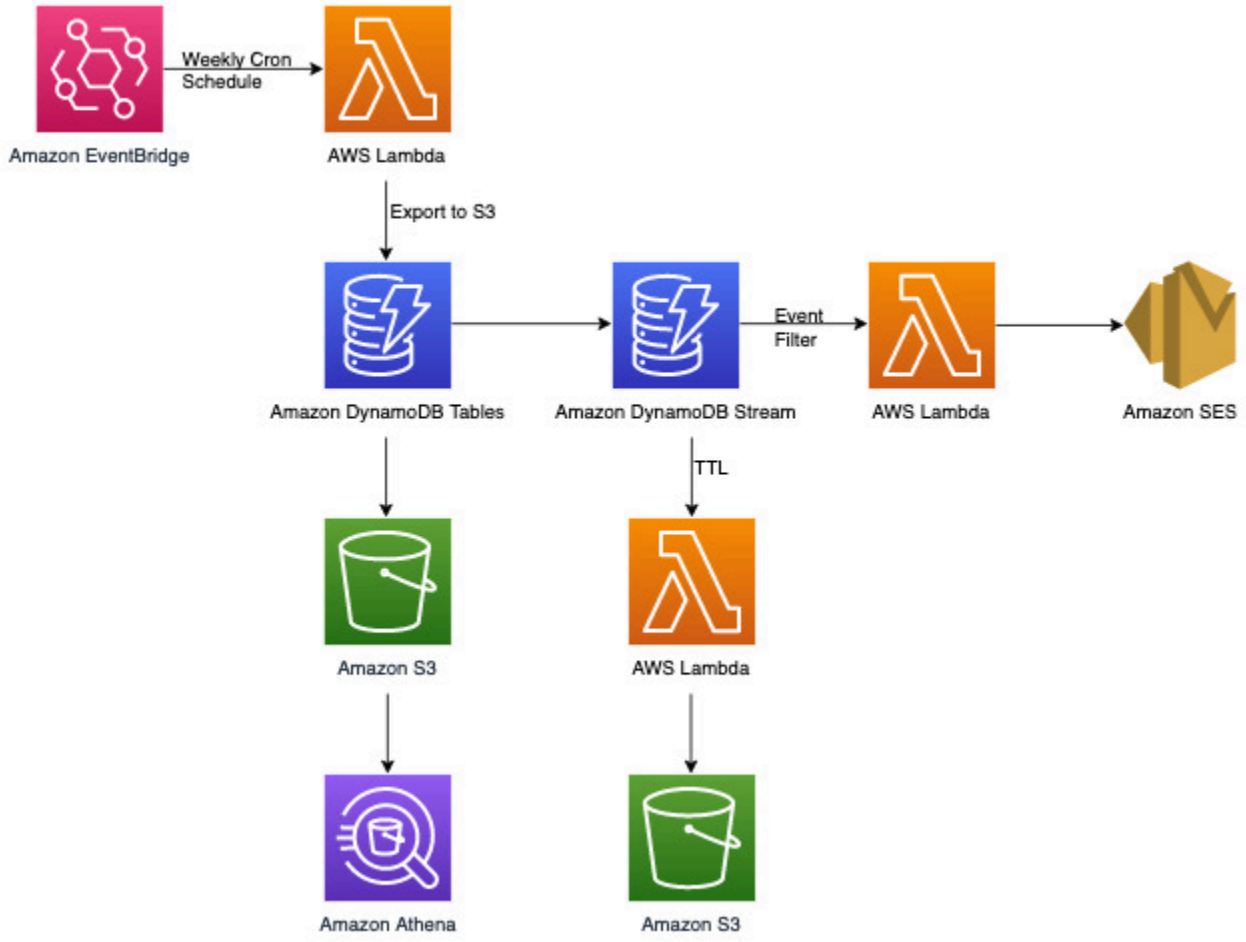
- Crea e aggiorna i reclami
- Inoltra un reclamo
- Crea e leggi commenti su un reclamo
- Ricevi tutti i reclami di un cliente
- Ottieni tutti i commenti di un agente e ottieni tutte le escalation

Alcuni commenti possono contenere allegati che descrivono il reclamo o la soluzione. Sebbene questi siano tutti modelli di accesso fondamentali, possono esserci requisiti aggiuntivi, come l'invio di notifiche quando viene aggiunto un nuovo commento a un reclamo o l'esecuzione di interrogazioni analitiche per determinare la distribuzione dei reclami in base alla gravità (o alle prestazioni degli

agenti) a settimana. Un ulteriore requisito relativo alla gestione del ciclo di vita o alla conformità sarebbe quello di archiviare i dati dei reclami dopo tre anni dalla registrazione del reclamo.

Diagramma dell'architettura del sistema di gestione dei reclami

Nel contenuto seguente viene illustrato il diagramma dell'architettura del sistema di gestione dei reclami. Questo diagramma mostra integrazioni di diversi Servizio AWS utilizzate dal sistema di gestione dei reclami.



Oltre ai modelli di accesso transazionale chiave-valore che tratteremo più avanti nella sezione sulla modellazione dei dati di DynamoDB, abbiamo tre requisiti non transazionali. Il diagramma dell'architettura riportato sopra può essere suddiviso nei seguenti tre flussi di lavoro:

1. Invia una notifica quando un nuovo commento viene aggiunto a un reclamo
2. Esegui interrogazioni analitiche sui dati settimanali
3. Archivia dati più vecchi di tre anni

Diamo un'occhiata più approfondita a ciascuno di essi.

Invia una notifica quando un nuovo commento viene aggiunto a un reclamo

Possiamo utilizzare il flusso di lavoro seguente per soddisfare questo requisito:

DynamoDB Streams è un meccanismo di acquisizione dei dati di modifica per registrare tutte le attività di scrittura sulle tabelle DynamoDB. È possibile configurare le funzioni Lambda in modo che si attivino su alcune o tutte queste modifiche. Un filtro eventi può essere configurato sui trigger Lambda per filtrare gli eventi non pertinenti al caso d'uso. In questo caso, possiamo utilizzare un filtro per attivare Lambda solo quando viene aggiunto un nuovo commento e inviare una notifica agli ID e-mail pertinenti che possono essere recuperati da AWS Secrets Manager o qualsiasi altro archivio di credenziali.

Esegui interrogazioni analitiche sui dati settimanali

DynamoDB è adatto per carichi di lavoro incentrati principalmente sull'elaborazione transazionale online (OLTP). Per gli altri modelli di accesso del 10-20% con requisiti analitici, i dati possono essere esportati in S3 con la funzionalità gestita Esporta su Amazon S3 senza alcun impatto sul traffico in tempo reale sulla tabella DynamoDB. Dai un'occhiata a questo flusso di lavoro qui sotto:

Amazon EventBridgepuò essere usato per innescare AWS Lambda nei tempi previsti: consente di configurare un'espressione cron per l'invocazione periodica di Lambda. Lambda può richiamare l'API `ExportToS3` e archiviare i dati di DynamoDB in S3. È quindi possibile accedere a questi dati S3 da un motore SQL come Amazon Athena per eseguire query analitiche sui dati DynamoDB senza influire sul carico di lavoro transazionale in tempo reale sulla tabella. Un esempio di query su Athena per trovare il numero di reclami per livello di gravità sarebbe il seguente:

```
SELECT Item.severity.S as "Severity", COUNT(Item) as "Count"
FROM "complaint_management"."data"
WHERE NOT Item.severity.S = ''
GROUP BY Item.severity.S ;
```

Questa query Athena restituisce il seguente risultato:

Results (3)

Search rows

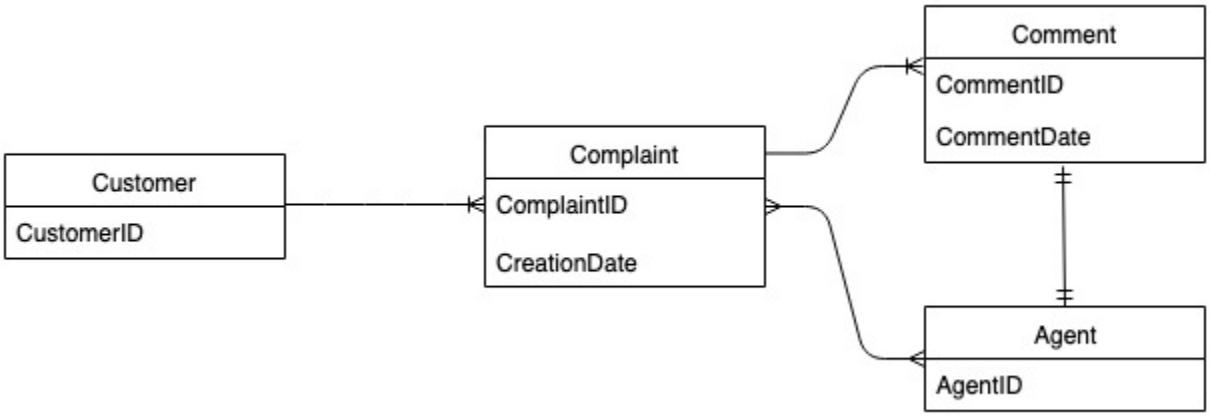
#	Severity	Count
1	P3	1
2	P2	2
3	P1	1

Archivia dati più vecchi di tre anni

Puoi sfruttare la funzionalità di DynamoDB Time to Live (TTL) per eliminare i dati obsoleti dalla tabella DynamoDB senza costi aggiuntivi (tranne nel caso delle repliche delle tabelle globali per la versione 2019.11.21 (attuale), in cui le eliminazioni TTL replicate in altre regioni consumano capacità di scrittura). Questi dati vengono visualizzati e possono essere utilizzati da DynamoDB Streams per essere archiviati in Amazon S3. Il flusso di lavoro per questo requisito è il seguente:

Diagramma delle relazioni tra entità del sistema di gestione dei reclami

Questo è il diagramma delle relazioni tra entità (ERD) che useremo per la progettazione dello schema di gestione dei reclami.



Modelli di accesso al sistema di gestione dei reclami

Questi sono i modelli di accesso che creeremo per la progettazione dello schema di gestione dei reclami.

1. createComplaint
2. updateComplaint
3. updateSeveritybyComplaintID
4. getComplaintByComplaintID
5. addCommentByComplaintID
6. getAllCommentsByComplaintID
7. getLatestCommentByComplaintID
8. getAComplaintbyCustomerIDAndComplaintID
9. getAllComplaintsByCustomerID
10. escalateComplaintByComplaintID
11. getAllEscalatedComplaints
12. getEscalatedComplaintsByAgentID (order from newest to oldest)
13. getCommentsByAgentID (between two dates)

Evoluzione della progettazione dello schema del sistema di gestione dei reclami

Poiché si tratta di un sistema di gestione dei reclami, la maggior parte dei modelli di accesso ruota attorno a un reclamo come entità principale. Essendo `ComplaintID` altamente importante garantirà una distribuzione uniforme dei dati nelle partizioni sottostanti ed è anche il criterio di ricerca più comune per i nostri modelli di accesso identificati. Pertanto, `ComplaintID` è un buon candidato per la chiave di partizione in questo set di dati.

Fase 1: gestire i modelli di accesso 1 (**`createComplaint`**), 2 (**`updateComplaint`**), 3 (**`updateSeveritybyComplaintID`**) e 4 (**`getComplaintByComplaintID`**)

Possiamo utilizzare una chiave di ordinamento generica denominata "metadati" (o "AA") per archiviare informazioni specifiche sul reclamo, ad esempio `CustomerID`, `State` ,`Severity` e`CreationDate`. Utilizziamo operazioni singleton con `PK=ComplaintID` e `SK="metadata"` per effettuare le seguenti operazioni:

1. `PutItem` per creare un nuovo reclamo
2. `UpdateItem` per aggiornare la gravità o altri campi nei metadati del reclamo
3. `GetItem` per recuperare i metadati per il reclamo

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>

Fase 2: Gestire il modello di accesso 5 (**`addCommentByComplaintID`**)

Questo modello di accesso richiede un modello di relazione uno-a-molti tra un reclamo e i commenti sul reclamo. Useremo qui la tecnica di partizionamento verticale per utilizzare una chiave di ordinamento e creare una raccolta di elementi con diversi tipi di dati. Se esaminiamo i modelli di accesso 6 (`getAllCommentsByComplaintID`) e 7 (`getLatestCommentByComplaintID`), sappiamo che i commenti dovranno essere ordinati per tempo. Possiamo anche ricevere più commenti contemporaneamente in modo da poter utilizzare la tecnica della chiave di ordinamento composita per aggiungere tempo e `CommentID` nell'attributo sort key.

Altre opzioni per gestire tali possibili collisioni di commenti sarebbero aumentare la granularità del timestamp o aggiungere un numero incrementale come suffisso invece di utilizzare `Comment_ID`.

In questo caso, prefisseremo il valore della chiave di ordinamento per gli elementi corrispondenti ai commenti con “comm#” per abilitare le operazioni basate sull'intervallo.

Dobbiamo inoltre garantire che `currentState` nei metadati del reclamo rifletta lo stato in cui viene aggiunto un nuovo commento. L'aggiunta di un commento potrebbe indicare che il reclamo è stato assegnato a un agente o è stato risolto e così via. Per raggruppare l’aggiunta di commenti e l’aggiornamento dello stato corrente nei metadati del reclamo, in modalità “tutto o niente”, verrà utilizzata l’API TransactWriteItems. Lo stato della tabella risultante ora è simile al seguente:

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>

Aggiungiamo altri dati nella tabella e aggiungiamo anche `ComplaintID` come campo separato dal nostro PK per rendere il modello a prova di futuro nel caso in cui avessimo bisogno di indici aggiuntivi su `ComplaintID`. Tieni inoltre presente che alcuni commenti potrebbero contenere allegati che verranno archiviati in Amazon Simple Storage Service e ne manterremo i riferimenti o gli URL solo in DynamoDB. È consigliabile mantenere il database transazionale il più snello possibile per ottimizzare costi e prestazioni. L'aspetto dei dati è ora il seguente:

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>

Fase 3: Gestire i modelli di accesso 6 (**`getAllCommentsByComplaintID`**) e 7 (**`getLatestCommentByComplaintID`**)

Per ottenere tutti i commenti relativi a un reclamo, possiamo utilizzare l'operazione `query` con la condizione `begins_with` sulla chiave di ordinamento. Invece di consumare capacità di lettura aggiuntiva per leggere i metadati e quindi avere il sovraccarico di filtrare i risultati pertinenti, una condizione chiave di ordinamento come questa ci aiuta a leggere solo ciò di cui abbiamo bisogno. Ad esempio, un'operazione di query con PK=`Complaint123` e SK begins_with `comm#` restituirebbe quanto segue saltando la voce dei metadati:

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>

Poiché abbiamo bisogno dell'ultimo commento per un reclamo nel modello 7 (`getLatestCommentByComplaintID`), utilizziamo due parametri di interrogazione aggiuntivi:

1. `ScanIndexForward` deve essere impostato su False per ottenere risultati ordinati in ordine decrescente
2. `Limit` deve essere impostato su 1 per ottenere l'ultimo commento (solo uno)

Simile allo schema di accesso 6 (`getAllCommentsByComplaintID`), saltiamo l'immissione dei metadati utilizzando `begins_with comm#` come condizione chiave di ordinamento. Ora, puoi eseguire il modello di accesso 7 su questa progettazione usando l'operazione query con `PK=Complaint123` e `SK=begins_with comm#`, `ScanIndexForward=False` e `Limit` 1. Verrà restituito come risultato l'elemento mirato seguente:

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>

Aggiungiamo altri dati fittizi alla tabella.

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>

Fase 4: Gestire i modelli di accesso 8 (**`getAComplaintbyCustomerIDAndComplaintID`**) e 9 (**`getAllComplaintsByCustomerID`**)

I modelli di accesso 8 (`getAComplaintbyCustomerIDAndComplaintID`) e 9 (`getAllComplaintsByCustomerID`) introducono nuovi criteri di ricerca: `CustomerID`. Recuperarlo dalla tabella esistente richiede una costosa `Scan` per leggere tutti i dati e quindi filtrare gli elementi pertinenti per il `CustomerID` in questione. Possiamo rendere questa ricerca più

efficiente creando un indice secondario globale (GSI) con `CustomerID` come chiave di partizione. Tenendo presente la relazione uno-a-molti tra cliente e reclami, nonché il modello di accesso 9 (`getAllComplaintsByCustomerID`),`ComplaintID` sarebbe il candidato giusto per la chiave di ordinamento.

I dati del GSI sarebbero i seguenti:

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>

Una query di esempio su questo GSI per il modello di accesso 8 (`getAComplaintbyCustomerIDAndComplaintID`) sarebbe: `customer_id=custXYZ`, `sort key=Complaint1321`. Il risultato sarebbe:

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>
	Complaint0987	PK	SK	current_state	creation_time	severity	complaint_description
		Complaint0987	metadata	assigned	2023-06-10T12:30:08	P3	<description text>
custXYZ							
	Complaint1321	PK	SK	current_state	creation_time	severity	complaint_description
		Complaint1321	metadata	investigating	2023-05-10T15:58:00	P2	<descr_text>

Per ricevere tutti i reclami di un cliente in merito al modello di accesso 9 (`getAllComplaintsByCustomerID`), la domanda sul GSI sarebbe: `customer_id=custXYZ` come condizione della chiave di partizione. Il risultato sarebbe:

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>

Fase 5: Gestire il modello di accesso 10 (**`escalateComplaintByComplaintID`**)

Questo accesso introduce l'aspetto dell'escalation. Per inoltrare un reclamo, possiamo usare `UpdateItem` per aggiungere attributi come `escalated_to` e `escalation_time` all'elemento dei metadati del reclamo esistente. DynamoDB offre una progettazione flessibile dello schema, il che significa che un insieme di attributi non chiave può essere uniforme o discreto tra diversi elementi. Un esempio è fornito di seguito.

`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

Fase 6: Gestire i modelli di accesso 11 (**`getAllEscalatedComplaints`**) e 12 (**`getEscalatedComplaintsByAgentID`**)

Si prevede che solo alcuni reclami vengano estrapolati dall'intero set di dati. Pertanto, la creazione di un indice degli attributi relativi all'escalation porterebbe a ricerche efficienti e a uno storage GSI conveniente. Possiamo farlo sfruttando la tecnica dell'indice sparso. Il GSI con chiave di partizione come `escalated_to` e chiave di ordinamento come `escalation_time` sarebbe simile a questo:

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>

Per ricevere tutti i reclami più elevati relativi allo schema di accesso 11 (`getAllEscalatedComplaints`), eseguiamo semplicemente la scansione di questo GSI. Tieni presente che questa scansione sarà efficiente ed economica a causa delle dimensioni del GSI. Per ricevere reclami più elevati per un agente specifico (shema di accesso 12 (`getEscalatedComplaintsByAgentID`)), la chiave di partizione sarebbe `escalated_to=agentID` e abbiamo impostato `ScanIndexForward` su `False` per ordinare dal più recente al meno recente.

Fase 7: Gestire il modello di accesso 13 (**`getCommentsByAgentID`**)

Per l'ultimo schema di accesso, dobbiamo eseguire una ricerca in base a una nuova dimensione: `AgentID`. Abbiamo anche bisogno di un ordine temporale per leggere i commenti tra due date, quindi creiamo un GSI con `agent_id` come chiave di partizione e `comm_date` come chiave di ordinamento. I dati di questo GSI saranno simili ai seguenti:

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"]

Una domanda di esempio su questo GSI potrebbe essere `partition key agentID=AgentA` e `sort key=comm_date between (2023-04-30T12:30:00, 2023-05-01T09:00:00)`, il cui risultato è:

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"]

Tutti i modelli di accesso e il modo in cui la progettazione dello schema li affronta sono riassunti nella tabella seguente:

Modello di accesso	Tabella di base/GSI/LSI	Operazione	Valore della chiave di partizione	Valore della chiave di ordinamento	Altre condizioni/ filtri
createComplaint	Tabella di base	PutItem	PK=complaint_id	SK=metadata	
updateComplaint	Tabella di base	UpdateItem	PK=complaint_id	SK=metadata	
updateSeveritybyComplaintID	Tabella di base	UpdateItem	PK=complaint_id	SK=metadata	
getComplaintByComplaintID	Tabella di base	GetItem	PK=complaint_id	SK=metadata	

Modello di accesso	Tabella di base/GSI/LSI	Operazione	Valore della chiave di partizione	Valore della chiave di ordinamento	Altre condizioni/ filtri
addCommentByComplaintID	Tabella di base	TransactWriteItems	PK=complaint_id	SK=metadata, SK=comm#comm_date#comm_id	
getAllCommentsByComplaintID	Tabella di base	Query	PK=complaint_id	SK begins_with "comm#"	
getLatestCommentByComplaintID	Tabella di base	Query	PK=complaint_id	SK begins_with "comm#"	scan_index_forward=False, Limit 1
getAComplaintbyCustomerIDAndComplaintID	Customer_complaint_GSI	Query	customer_id=customer_id	complaint_id = complaint_id	
getAllComplaintsByCustomerID	Customer_complaint_GSI	Query	customer_id=customer_id	N/D	
escalateComplaintByComplaintID	Tabella di base	UpdateItem	PK=complaint_id	SK=metadata	
getAllEscalatedComplaints	Escalations_GSI	Scan	N/D	N/D	

Modello di accesso	Tabella di base/GSI/LSI	Operazione	Valore della chiave di partizione	Valore della chiave di ordinamento	Altre condizioni/ filtri
getEscalatedComplaintsByAgentID (order from newest to oldest)	Escalations_GSI	Query	escalated_to=agent_id	N/D	scan_index_forward=False
getCommentsByAgentID (between two dates)	Agents_Comments_GSI	Query	agent_id=agent_id	SK between (date1, date2)	

Schema finale del sistema di gestione dei reclami

Di seguito sono riportate le progettazioni dello schema finale. Per scaricare questa progettazione dello schema come un file JSON, consulta Esempi di DynamoDB su GitHub.

Tabella di 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"]

Utilizzo di NoSQL Workbench con questa progettazione dello schema

Puoi importare questo schema finale in NoSQL Workbench, uno strumento visivo che fornisce funzionalità di modellazione dei dati, visualizzazione dei dati e sviluppo di query per DynamoDB, per esplorare e modificare ulteriormente il tuo nuovo progetto. Per iniziare, segui queste fasi:

1. Scarica NoSQL Workbench. Per ulteriori informazioni, consulta the section called “Scarica”.
2. Scarica il file dello schema JSON elencato in precedenza, che si trova già nel formato del modello NoSQL Workbench.
3. Importa il file dello schema JSON in NoSQL Workbench. Per ulteriori informazioni, consulta the section called “Importazione di un modello esistente”.
4. Dopo che è stato importato in NOSQL Workbench, puoi modificare il modello di dati. Per ulteriori informazioni, consulta the section called “Modifica di un modello esistente”.
5. Per visualizzare il tuo modello di dati, aggiungere dati di esempio o importare dati di esempio da un file CSV, utilizza la funzionalità Data Visualizer di NoSQL Workbench.

Progettazione dello schema dei pagamenti ricorrenti in DynamoDB

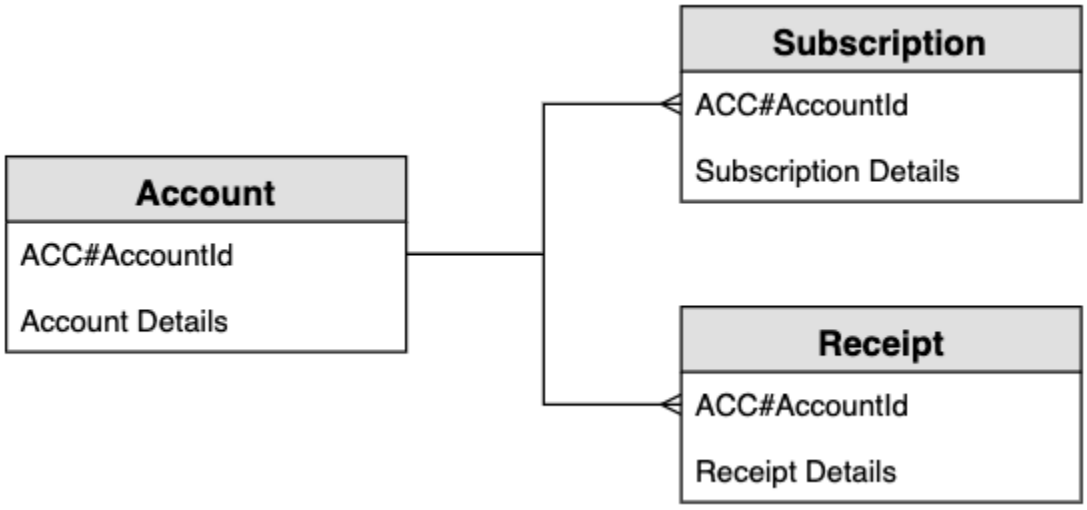
Caso d'uso aziendale dei pagamenti ricorrenti

Questo caso d'uso parla dell'utilizzo di DynamoDB per implementare un sistema di pagamenti ricorrenti. Il modello di dati ha le seguenti entità: account, sottoscrizioni e ricevute. Le specifiche del nostro caso d'uso includono quanto segue:

- Ciascun account può avere più sottoscrizioni
- La sottoscrizione ha un `NextPaymentDate` quando deve essere effettuato il prossimo pagamento e un `NextReminderDate` quando viene inviato un promemoria via e-mail al cliente
- C'è un articolo per sottoscrizione che viene archiviato e aggiornato al momento dell'elaborazione del pagamento (la dimensione media degli articoli è di circa 1 KB e la velocità effettiva dipende dal numero di account e sottoscrizioni)
- Il processore del pagamento creerà anche una ricevuta come parte del processo, che è memorizzato nella tabella ed è impostato per scadere dopo un certo periodo di tempo utilizzando unattributo TTL

Diagramma delle relazioni tra le entità dei pagamenti ricorrenti

Questo è il diagramma delle relazioni tra entità (ERD) che useremo per la progettazione dello schema di gestione dei pagamenti ricorrenti.



Schemi di accesso ai sistemi di pagamento ricorrenti

Questi sono i modelli di accesso che creeremo per la progettazione dello schema del sistema di pagamenti ricorrenti.

1. `createSubscription`
2. `createReceipt`
3. `updateSubscription`
4. `getDueRemindersByDate`
5. `getDuePaymentsByDate`
6. `getSubscriptionsByAccount`
7. `getReceiptsByAccount`

Progettazione dello schema dei pagamenti ricorrenti

I nomi generici PK e SK vengono utilizzati per gli attributi chiave per consentire la memorizzazione di diversi tipi di entità nella stessa tabella, ad esempio le entità account, sottoscrizione e ricevuta. L'utente crea innanzitutto un abbonamento, in cui accetta di pagare un importo lo stesso giorno di ogni mese in cambio di un prodotto. Possono scegliere in quale giorno del mese elaborare il pagamento. C'è anche un promemoria che verrà inviato prima dell'elaborazione del pagamento. L'applicazione funziona con due processi batch eseguiti ogni giorno: un processo batch invia i promemoria con scadenza quel giorno e l'altro processo batch elabora i pagamenti in scadenza quel giorno.

Fase 1: Gestire il modello di accesso 1 (**`createSubscription`**)

Il modello di accesso 1 (`createSubscription`) viene utilizzato per creare inizialmente l'abbonamento e vengono impostati i dettagli, tra cui SKU, `NextPaymentDate`, `NextReminderDate` e `PaymentDetails`. Questo passaggio mostra lo stato della tabella per un solo account con un abbonamento. Possono esserci più abbonamenti nella raccolta di articoli, quindi si tratta di una relazione uno a molti.

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

Fase 2: Gestire i modelli di accesso 2 (**`createReceipt`**) e 3 (**`updateSubscription`**)

Il modello di accesso 2 (`createReceipt`) viene utilizzato per creare l'articolo della ricevuta. Dopo l'elaborazione del pagamento ogni mese, il processore di pagamento riscriverà una ricevuta nella tabella di base. Possono esserci più abbonamenti nella raccolta di articoli, quindi si tratta di una relazione uno a molti. Il processore di pagamento aggiornerà anche l'elemento di abbonamento (Modello di accesso 3 (`updateSubscription`)) da aggiornare per `NextReminderDate`o il `NextPaymentDate` per il prossimo mese.

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

Fase 3: Gestire il modello di accesso 4 (**`getDueRemindersByDate`**)

L'applicazione elabora i promemoria per il pagamento in batch per il giorno corrente. Pertanto l'applicazione deve accedere agli abbonamenti su una dimensione diversa: data anziché account. Questo è un buon caso d'uso per l'indice secondario globale (GSI). In questo passaggio aggiungiamo l'indice `GSI-1`, che utilizza `NextReminderDate` come chiave di partizione GSI. Non è necessario replicare tutti gli articoli. Questo GSI è un indice sparso e gli articoli delle ricevute non vengono replicati. Inoltre, non è necessario proiettare tutti gli attributi: è sufficiente includere un sottoinsieme degli attributi. L'immagine sotto mostra lo schema di `GSI-1` e fornisce le informazioni necessarie all'applicazione per inviare l'e-mail di promemoria.

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

Fase 4: Gestire il modello di accesso 5 (**`getDuePaymentsByDate`**)

L'applicazione elabora i pagamenti in batch per il giorno corrente nello stesso modo in cui elabora promemoria. In questo passaggio aggiungiamo `GSI-2`, che utilizza `NextPaymentDate` come chiave di partizione GSI. Non è necessario replicare tutti gli articoli. Questo GSI è un indice sparso e gli articoli delle ricevute non vengono replicati. L'immagine sotto mostra lo schema di `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"}}

Fase 5: Gestire i modelli di accesso 6 (**`getSubscriptionsByAccount`**) e 7 (**`getReceiptsByAccount`**)

L'applicazione può recuperare tutti gli abbonamenti per un account utilizzando una query nella tabella di base destinata all'identificatore dell'account (PK) e utilizza l'operatore di intervallo per ottenere tutti gli articoli in cui SK inizia con “SUB#”. L'applicazione può anche utilizzare la stessa struttura di query per recuperare tutte le ricevute utilizzando un operatore di intervallo per ottenere tutti gli articoli in cui SK inizia con “REC#”. Questo ci consente di soddisfare i modelli di accesso 6 (`getSubscriptionsByAccount`) e 7 (`getReceiptsByAccount`). L'applicazione utilizza questi modelli di accesso in modo che l'utente possa vedere i propri abbonamenti attuali e le ricevute passate degli ultimi sei mesi. Non vi è alcuna modifica allo schema della tabella in questo passaggio e possiamo vedere di seguito come indirizziamo solo gli elementi dell'abbonamento nel modello di accesso 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	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

Tutti i modelli di accesso e il modo in cui la progettazione dello schema li affronta sono riassunti nella tabella seguente:

Modello di accesso	Tabella di base/ GSI/LSI	Operazione	Valore della chiave di partizione	Valore della chiave di ordinamento
createSubscription	Tabella di base	PutItem	ACC#account_id	SUB#<SUBID>#SKU<SKUID>
createReceipt	Tabella di base	PutItem	ACC#account_id	REC#<ReceiptDate>#SKU<SKUID>

Modello di accesso	Tabella di base/ GSI/LSI	Operazione	Valore della chiave di partizione	Valore della chiave di ordinamento
updateSubscription	Tabella di base	UpdateItem	ACC#account_id	SUB#<SUBID>#SKU<SKUID>
getDueRemindersByDate	GSI-1	Query	<NextReminderDate>	
getDuePaymentsByDate	GSI-2	Query	<NextPaymentDate>	
getSubscriptionsByAccount	Tabella di base	Query	ACC#account_id	SK begins_with “SUB#”
getReceiptsByAccount	Tabella di base	Query	ACC#account_id	SK begins_with “REC#”

Schema finale dei pagamenti ricorrenti

Di seguito sono riportate le progettazioni dello schema finale. Per scaricare questa progettazione dello schema come un file JSON, consulta Esempi di DynamoDB su GitHub.

Tabella di 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					
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

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"}}

Utilizzo di NoSQL Workbench con questa progettazione dello schema

Puoi importare questo schema finale in NoSQL Workbench, uno strumento visivo che fornisce funzionalità di modellazione dei dati, visualizzazione dei dati e sviluppo di query per DynamoDB, per esplorare e modificare ulteriormente il tuo nuovo progetto. Per iniziare, segui queste fasi:

1. Scarica NoSQL Workbench. Per ulteriori informazioni, consulta the section called “Scarica”.
2. Scarica il file dello schema JSON elencato in precedenza, che si trova già nel formato del modello NoSQL Workbench.
3. Importa il file dello schema JSON in NoSQL Workbench. Per ulteriori informazioni, consulta the section called “Importazione di un modello esistente”.
4. Dopo che è stato importato in NOSQL Workbench, puoi modificare il modello di dati. Per ulteriori informazioni, consulta the section called “Modifica di un modello esistente”.
5. Per visualizzare il tuo modello di dati, aggiungere dati di esempio o importare dati di esempio da un file CSV, utilizza la funzionalità Data Visualizer di NoSQL Workbench.

Monitoraggio degli aggiornamenti dello stato dei dispositivi in DynamoDB

In questo caso d'uso viene descritto l'utilizzo di DynamoDB per monitorare gli aggiornamenti dello stato del dispositivo (o le modifiche allo stato del dispositivo) in DynamoDB.

Caso d'uso

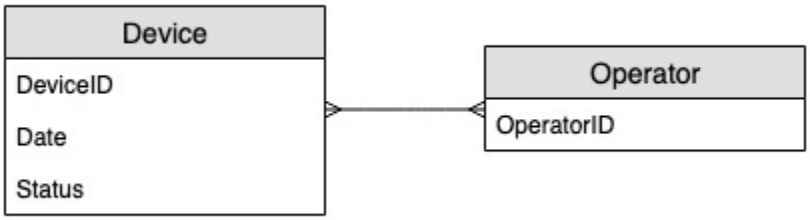
Nei casi d'uso dell'IoT (ad esempio una fabbrica intelligente) molti dispositivi devono essere monitorati dagli operatori e inviano periodicamente il proprio stato o i log a un sistema di monitoraggio. Quando si verifica un problema con un dispositivo, lo stato del dispositivo cambia da normale ad avviso. Esistono diversi livelli o stati di log a seconda della gravità e del tipo di comportamento anomalo nel dispositivo. Il sistema incarica quindi un operatore di controllare il dispositivo e, se necessario, può segnalare il problema al proprio supervisore.

Alcuni modelli di accesso tipici per questo sistema includono:

- Crea una voce di log per un dispositivo
- Ottieni tutti i log per uno stato specifico del dispositivo, mostrando per primi i log più recenti
- Ottieni tutti i log di un determinato operatore tra due date
- Ottieni tutti i log inoltrati per un determinato supervisore
- Ottieni tutti i log inoltrati con uno stato specifico del dispositivo per un determinato supervisore
- Ottieni tutti i log inoltrati con uno stato specifico del dispositivo per un determinato supervisore e per una data specifica

Diagramma delle relazioni tra entità

Questo è il diagramma di relazione delle entità (ERD) ch useremo per monitorare gli aggiornamenti dello stato dei dispositivi.



Modelli di accesso

Questi sono i modelli di accesso che prenderemo in considerazione per monitorare gli aggiornamenti dello stato dei dispositivi.

1. `createLogEntryForSpecificDevice`
2. `getLogsForSpecificDevice`
3. `getWarningLogsForSpecificDevice`
4. `getLogsForOperatorBetweenTwoDates`
5. `getEscalatedLogsForSupervisor`
6. `getEscalatedLogsWithSpecificStatusForSupervisor`
7. `getEscalatedLogsWithSpecificStatusForSupervisorForDate`

Evoluzione della progettazione dello schema

Fase 1: Gestire i modelli di accesso 1 (**`createLogEntryForSpecificDevice`**) e 2 (**`getLogsForSpecificDevice`**)

L'unità di dimensionamento per un sistema di tracciamento dei dispositivi sarebbe costituita dai singoli dispositivi. In questo sistema, `deviceID` identifica in modo univoco un dispositivo. Questo rende `deviceID` un buon candidato per la chiave di partizione. Ogni dispositivo invia informazioni al sistema di tracciamento periodicamente (ad esempio ogni cinque minuti circa). Questo ordinamento rende la data un criterio di ordinamento logico e quindi la chiave di ordinamento. I dati di esempio in questo caso apparirebbero simili a:

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

Per recuperare le voci di log per un dispositivo specifico, possiamo eseguire un'operazione di query con la chiave di partizione `DeviceID="d#12345"`.

Fase 2: Gestire il modello di accesso 3 (**`getWarningLogsForSpecificDevice`**)

Poiché `State` è un attributo non chiave, affrontare il pattern di accesso 3 con lo schema corrente richiederebbe un'espressione di filtro. In DynamoDB, le espressioni di filtro vengono applicate dopo la lettura dei dati utilizzando le espressioni delle condizioni chiave. Ad esempio, se si dovessero recuperare i registri dei log per `d#12345`, l'operazione di query con chiave di partizione `DeviceID="d#12345"` leggerà quattro elementi dalla tabella precedente e poi filtrerà l'unico elemento senza lo stato di avviso. Questo approccio non è efficiente su larga scala. Un'espressione di filtro può essere un buon modo per escludere gli elementi interrogati se il rapporto tra gli elementi esclusi è basso o la query viene eseguita di rado. Tuttavia, nei casi in cui molti elementi vengono recuperati da una tabella e la maggior parte degli elementi viene filtrata, possiamo continuare a sviluppare il design della tabella in modo che funzioni in modo più efficiente.

Cambiamo come gestire questo modello di accesso utilizzando chiavi di ordinamento composte. È possibile importare dati di esempio da DeviceStateLog_3.json in cui la chiave di ordinamento viene modificata in `State#Date`. Questa chiave di ordinamento è la composizione degli attributi `State`, `#` e `Date`. In questo caso, # viene usato come un delimitatore. L'aspetto dei dati è ora il seguente:

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

Per recuperare solo i log degli avvisi per un dispositivo, la query diventa più mirata con questo schema. La condizione chiave per la query utilizza la chiave di partizione `DeviceID="d#12345"` e la chiave di ordinamento `State#Date begins_with “WARNING”`. Questa interrogazione leggerà solo i tre elementi pertinenti con lo stato di avviso.

Fase 3: Gestire il modello di accesso 4 (**`getLogsForOperatorBetweenTwoDates`**)

È possibile importare DeviceStateLog_4.jsonD dove l'attributo `Operator` è stato aggiunto alla tabella `DeviceStateLog`con dati di esempio.

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

Poiché `Operator` attualmente non è una chiave di partizione, non è possibile eseguire una ricerca diretta chiave-valore su questa tabella basata su `OperatorID`. Dovremo creare una nuova

collezione di articoli con un indice secondario globale su `OperatorID`. Il modello di accesso richiede una ricerca basata sulle date, quindi Data è l'attributo chiave di ordinamento per indice secondario globale (GSI). Ecco come appare ora il 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

Per il modello di accesso 4 (`getLogsForOperatorBetweenTwoDates`), puoi interrogare questo GSI con la chiave di partizione `OperatorID=Liz` e la chiave di ordinamento `Date` fra `2020-04-11T05:58:00` e `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

Fase 4: Gestire i modelli di accesso 5 (**`getEscalatedLogsForSupervisor`**), 6 (**`getEscalatedLogsWithSpecificStatusForSupervisor`**) e 7 (**`getEscalatedLogsWithSpecificStatusForSupervisorForDate`**)

Useremo un indice sparso per affrontare questi modelli di accesso.

Per impostazione predefinita, gli indici secondari globali sono sparsi, quindi solo gli elementi della tabella di base che contengono gli attributi della chiave primaria dell'indice verranno effettivamente visualizzati nell'indice. Questo è un altro modo per escludere gli elementi che non sono rilevanti per il modello di accesso da modellare.

È possibile importare DeviceStateLog_6.json dove l'attributo`EscalatedTo` è stato aggiunto alla tabella `DeviceStateLog` con dati di esempio. Come accennato in precedenza, non tutti i log vengono inoltrati a un supervisore.

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

È ora possibile creare un nuovo GSI in cui `EscalatedTo` è la chiave di partizione e `State#Date` è la chiave di ordinamento. Nota che solo gli elementi che hanno entrambi gli attributi `EscalatedTo` e `State#Date` vengono visualizzati nell'indice.

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

Gli altri modelli di accesso sono riassunti come segue:

Tutti i modelli di accesso e il modo in cui la progettazione dello schema li affronta sono riassunti nella tabella seguente:

Modello di accesso	Tabella di base/GSI/LSI	Operazione	Valore della chiave di partizione	Valore della chiave di ordinamento	Altre condizioni/ filtri
createLogEntryForSpecificDevice	Tabella di base	PutItem	DeviceID= deviceId	State#Date=state#date	
getLogsForSpecificDevice	Tabella di base	Query	DeviceID= deviceId	State#Date begins_with "state1#"	ScanIndexForward = Falso
getWarningLogsForSpecificDevice	Tabella di base	Query	DeviceID= deviceId	State#Date begins_with "WARNING"	
getLogsForOperatorBetweenTwoDates	GSI-1	Query	Operator= operatorName	Data compresa tra data1 e data2	
getEscalatedLogsForSupervisor	GSI-2	Query	EscalatedTo=supervisorName		

Modello di accesso	Tabella di base/GSI/LSI	Operazione	Valore della chiave di partizione	Valore della chiave di ordinamento	Altre condizioni/ filtri
getEscalatedLogsWithSpecificStatusForSupervisor	GSI-2	Query	EscalatedTo=supervisorName	State#Date begins_with "state1#"	
getEscalatedLogsWithSpecificStatusForSupervisorForDate	GSI-2	Query	EscalatedTo=supervisorName	State#Date begins_with "state1#date1"	

Schema finale

Di seguito sono riportate le progettazioni dello schema finale. Per scaricare questa progettazione dello schema come un file JSON, consulta Esempi di DynamoDB su GitHub.

Tabella di 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

Utilizzo di NoSQL Workbench con questa progettazione dello schema

Puoi importare questo schema finale in NoSQL Workbench, uno strumento visivo che fornisce funzionalità di modellazione dei dati, visualizzazione dei dati e sviluppo di query per DynamoDB, per esplorare e modificare ulteriormente il tuo nuovo progetto. Per iniziare, segui queste fasi:

1. Scarica NoSQL Workbench. Per ulteriori informazioni, consulta the section called “Scarica”.
2. Scarica il file dello schema JSON elencato in precedenza, che si trova già nel formato del modello NoSQL Workbench.
3. Importa il file dello schema JSON in NoSQL Workbench. Per ulteriori informazioni, consulta the section called “Importazione di un modello esistente”.
4. Dopo che è stato importato in NOSQL Workbench, puoi modificare il modello di dati. Per ulteriori informazioni, consulta the section called “Modifica di un modello esistente”.
5. Per visualizzare il tuo modello di dati, aggiungere dati di esempio o importare dati di esempio da un file CSV, utilizza la funzionalità Data Visualizer di NoSQL Workbench.

Utilizzo di DynamoDB come archivio dati per un negozio online

In questo caso di utilizzo si parla di usare DynamoDB come archivio dati per un negozio online (o e-store).

Caso d'uso

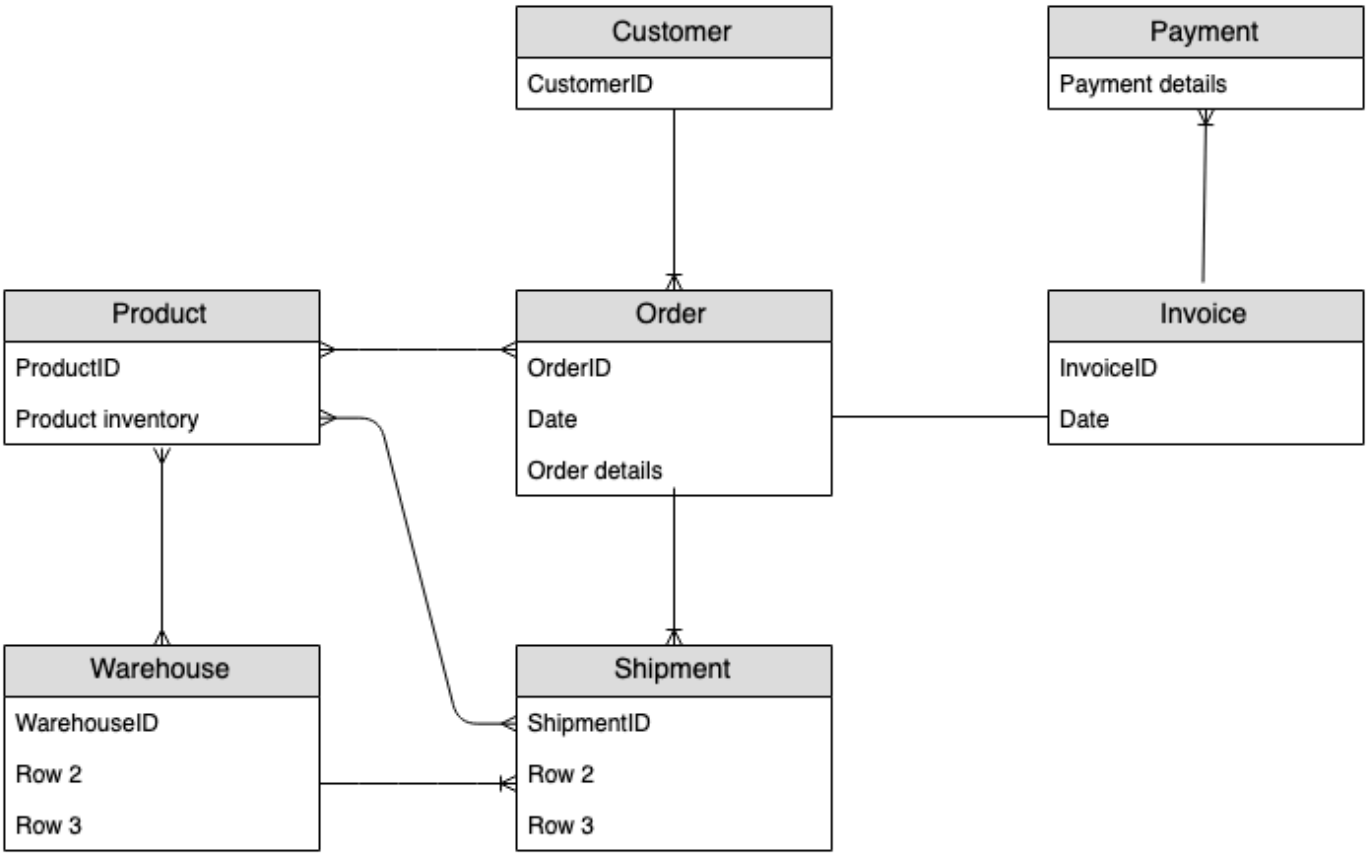
Un negozio online consente agli utenti di sfogliare diversi prodotti e alla fine di acquistarli. In base alla fattura generata, un cliente può pagare utilizzando un codice sconto o una carta regalo e poi pagare l'importo residuo con una carta di credito. I prodotti acquistati verranno ritirati da uno dei diversi magazzini e spediti all'indirizzo fornito. I modelli di accesso tipici per un negozio online includono:

- Ottieni cliente per un determinato customerId

- Ottieni il prodotto per un determinato productId
- Ottieni il magazzino per un determinato warehouseId
- Ottieni un inventario di prodotti per tutti i magazzini tramite un productId
- Ottieni l'ordine per un determinato orderId
- Ottieni tutti i prodotti per un determinato orderId
- Ottieni la fattura per un determinato orderId
- Ottieni tutte le spedizioni per un determinato orderId
- Ottieni tutti gli ordini per un determinato productId per un determinato intervallo di date
- Ottieni la fattura per un determinato invoiceId
- Ottieni tutti i pagamenti per un determinato invoiceId
- Ottieni i dettagli della spedizione per un determinato shipmentId
- Ottieni tutte le spedizioni per un determinato warehouseId
- Ottieni l'inventario di tutti i prodotti per un determinato warehouseId
- Ottieni tutte le fatture per un determinato customerId per un determinato intervallo di date
- Ottieni tutti i prodotti per un determinato customerId per un determinato intervallo di date

Diagramma delle relazioni tra entità

Questo è il diagramma delle relazioni tra entità (ERD) che useremo per la progettazione del modello DynamoDB come archivio dati per un negozio online.



Modelli di accesso

Questi sono i modelli di accesso da considerare quando si usa DynamoDB come archivio dati per un negozio online.

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

Evoluzione della progettazione dello schema

Usando NoSQL Workbench per DynamoDB, importa Onlineshop_1.json per la creazione di un nuovo modello di dati chiamato AnOnlineShope una nuova tabella chiamata OnlineShop. Nota che usiamo i nomi generici PK e SK per la chiave di partizione e la chiave di ordinamento. Questa è una pratica utilizzata per memorizzare diversi tipi di entità nella stessa tabella.

Fase 1: Gestire il modello di accesso 1 (**getCustomerByCustomerId**)

Importa AnOnlineShop_2.json per gestire il modello di accesso 1 (getCustomerByCustomerId). Alcune entità non hanno relazioni con altre entità, quindi useremo lo stesso valore di PK e SK per loro. Nei dati di esempio, nota che le chiavi utilizzano un prefisso c#per distinguere customerId da altre entità che verranno aggiunte in seguito. Questa pratica viene ripetuta anche per altre entità.

Per risolvere questo modello di accesso, un'operazione GetItem può essere utilizzata con PK=customerId e SK=customerId.

Fase 2: Gestire il modello di accesso 2 (**getProductByProductId**)

Importa OnlineShop_3.json per risolvere il modello di accesso 2 (getProductByProductId) per l'entità product. Le entità del prodotto sono precedute da p# e lo stesso attributo chiave di ordinamento è stato utilizzato per memorizzare customerID così come productID. Denominazione generica e partizionamento verticale ci consentono di creare tali collezioni di articoli per un design efficace di una tabella singola.

Per risolvere questo modello di accesso, un'operazione GetItem può essere utilizzata con PK=productId e SK=productId.

Fase 3: Gestire il modello di accesso 3 (**getWarehouseByWarehouseId**)

Importa AnOnlineShop_4.json per risolvere il modello di accesso 3 (getWarehouseByWarehouseId) per l'entità warehouse. Al momento abbiamo le entità

`customer`, `product` e `warehouse` aggiunte alla stessa tabella. Si distinguono utilizzando i prefissi e l'attributo `EntityType`. Un attributo di tipo (o denominazione del prefisso) migliora la leggibilità del modello. La leggibilità ne risentirebbe se memorizzassimo semplicemente ID alfanumerici per diverse entità nello stesso attributo. Sarebbe difficile distinguere un'entità dall'altra in assenza di questi identificatori.

Per risolvere questo modello di accesso, un'operazione `GetItem` può essere utilizzata con `PK=warehouseId` e `SK=warehouseId`.

Tabella di 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"}}	

Fase 4: Gestire il modello di accesso 4 (**`getProductInventoryByProductId`**)

Importa AnOnlineShop_5.json per risolvere il modello di accesso 4 (`getProductInventoryByProductId`). L'entità `warehouseItem` viene utilizzata per tenere traccia del numero di prodotti in ogni magazzino. Questo articolo viene normalmente aggiornato quando un prodotto viene aggiunto o rimosso da un magazzino. Come si vede nell'ERD, esiste una relazione molti a molti tra `product` e `warehouse`. Ecco, la relazione uno-a-molti da `product` a `warehouse` è modellata come `warehouseItem`. Più tardi, la relazione uno-a-molti da `warehouse` a `product` sarà anch'essa modellata.

Il pattern di accesso 4 può essere risolto con una query su `PK=ProductId` e `SK begins_with "w#"`.

Per ulteriori informazioni su `begins_with()` e altre espressioni che possono essere applicate alle chiavi di ordinamento, vedi Espressioni delle condizioni chiave.

Tabella di 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"}}	

Fase 5: Gestire i modelli di accesso 5 (**`getOrderDetailsByOrderId`**) e 6 (**`getProductByOrderId`**)

Aggiungi ancora altri elementi `customer`, `product` e `warehouse` alla tabella importando AnOnlineShop_6.json. Quindi, importa AnOnlineShop_7.json per creare una collezione di oggetti per `order` che può risolvere i modelli di accesso 5 (`getOrderDetailsByOrderId`) e 6 (`getProductByOrderId`). Puoi vedere la relazione uno-a-molti tra `order` e `product` modellata come entità orderItem.

Per risolvere il modello di accesso 5 (`getOrderDetailsByOrderId`), esegui una query della tabella con `PK=orderId`. Questo fornirà tutte le informazioni sull'ordine, tra cui `customerId` e i prodotti ordinati.

Tabella di 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

Per risolvere il modello di accesso 6 (`getProductByOrderId`), dobbiamo leggere i prodotti solo in `order`. Interroga la tabella con `PK=orderId` e `SK begins_with "p#"` per realizzarlo.

Tabella di 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

Fase 6: Gestire il modello di accesso 7 (**`getInvoiceByOrderId`**)

Importa OnlineShop_8.json per aggiungere un'entità `invoice` alla raccolta di elementi dell'ordineper gestire il pattern di accesso 7 (`getInvoiceByOrderId`). Per risolvere questo modello di accesso, puoi usare un'operazione di query con `PK=orderId` e `SK begins_with "i#"`.

Tabella di 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

Fase 7: Gestire il modello di accesso 8 (**`getShipmentByOrderId`**)

Importa AnOnlineShop_9.json per aggiungere entità `shipment` alla raccolta di articoli dell'ordine per soddisfare il modello di accesso 8 (`getShipmentByOrderId`). Stiamo estendendo lo stesso modello partizionato verticalmente aggiungendo più tipi di entità nel design a tabella singola. Da notare come la raccolta di articoli dell'ordine contenga le diverse relazioni che un'entità `order` ha con le entità `shipment`, `orderItem` e `invoice`.

Per ricevere spedizioni entro la data `orderId`, è possibile eseguire un'operazione di query con `PK=orderId` e `SK begins_with "sh#"`.

Tabella di 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

Fase 8: Gestire il modello di accesso 9 (**`getOrderByProductIdForDateRange`**)

Abbiamo creato una raccolta degli articoli dell'ordine nel passaggio precedente. Questo modello di accesso ha nuove dimensioni di ricerca (`ProductID` e `Date`) che richiedono la scansione dell'intera tabella e il filtraggio dei record pertinenti per recuperare gli elementi mirati. Per risolvere questo modello di accesso, dovremo creare un indice secondario globale (GSI). Importa AnOnlineShop_10.json per creare una nuova raccolta di articoli utilizzando il GSI che consente di recuperare i dati `orderItem` provenienti da diverse raccolte di articoli dell'ordine. I dati ora hanno `GSI1-PK` e `GSI1-SK` che saranno rispettivamente la chiave di partizione e la chiave di ordinamento di `GSI1`.

DynamoDB popola automaticamente gli elementi che contengono gli attributi chiave di un GSI dalla tabella al GSI. Non è necessario inserire manualmente inserimenti aggiuntivi nel GSI.

Per risolvere il modello di accesso 9, esegui una query su `GSI1` con `GSI1-PK=productId` e `GSI1SK between (date1, date2)`.

Tabella di 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

Fase 9: Gestire i modelli di accesso 10 (**`getInvoiceByInvoiceId`**) e 11 (**`getPaymentByInvoiceId`**)

Importa AnOnlineShop_11.json per affrontare i modelli di accesso 10 (`getInvoiceByInvoiceId`) e 11 (`getPaymentByInvoiceId`), entrambi correlati a `invoice`. Anche se si tratta di due modelli di accesso diversi, vengono realizzati utilizzando la stessa condizione chiave. `Payments` sono definiti come un attributo con il tipo di dati della mappa su entità `invoice`.

> **Note**
>
> `GSI1-PK` e `GSI1-SK` è sovraccarico per memorizzare informazioni su diverse entità in modo che più modelli di accesso possano essere serviti dallo stesso GSI. Per ulteriori informazioni sul sovraccarico di GSI, vedi Overload degli indici secondari globali in DynamoDB.

Per risolvere il modello di accesso 10 e 11, esegui una query su `GSI1` con `GSI1-PK=invoiceId` e `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

Fase 10: Gestire i modelli di accesso 12 (**`getShipmentDetailsByShipmentId`**) e 13 (**`getShipmentByWarehouseId`**)

Importa AnOnlineShop_12.json per affrontare i modelli di accesso 12 (`getShipmentDetailsByShipmentId`) e 13 (`getShipmentByWarehouseId`).

Da notare che le entità `shipmentItem` vengono aggiunte alla raccolta di elementi dell'ordine sulla tabella di base per poter recuperare tutti i dettagli su un ordine in un'unica operazione di query.

Tabella di 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				

Le chiavi `GSI1` di partizione e ordinamento sono già state utilizzate per modellare una relazione uno a molti tra `shipment` e `shipmentItem`. Per risolvere il modello di accesso 12 (`getShipmentDetailsByShipmentId`), esegui una query su `GSI1` con `GSI1-PK=shipmentId` e `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
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

Dovremo creare un altro GSI (`GSI2`) per la progettazione della nuova relazione uno a molti tra `warehouse` e `shipment` per il modello di accesso 13 (`getShipmentByWarehouseId`). Per

risolvere questo modello di accesso, esegui una query su `GSI2` con `GSI2-PK=warehouseId` e `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

Fase 11: Gestire i modelli di accesso 14 (**`getProductInventoryByWarehouseId`**) 15 (**`getInvoiceByCustomerIdForDateRange`**) e 16 (**`getProductsByCustomerIdForDateRange`**)

Importa AnOnlineShop_13.json per aggiungere dati relativi al prossimo set di modelli di accesso. Per risolvere il modello di accesso 14 (`getProductInventoryByWarehouseId`), esegui una query su `GSI2` con `GSI2-PK=warehouseId` e `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	

Per risolvere il modello di accesso 15 (`getInvoiceByCustomerIdForDateRange`), esegui una query su `GSI2` con `GSI2-PK=customerId` e `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	

Per risolvere il modello di accesso 16 (`getProductsByCustomerIdForDateRange`), esegui una query su `GSI2` con `GSI2-PK=customerId` e `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

In NoSQL Workbench, i facet rappresentano pattern di accesso ai dati differenti di un'applicazione per DynamoDB. I facet consentono di visualizzare un sottoinsieme di dati in una tabella, senza dover visualizzare record che non soddisfano i vincoli del facet. I facet sono considerati uno strumento di modellazione visiva dei dati e non esistono come costrutto utilizzabile in DynamoDB, in quanto sono puramente un aiuto alla modellazione dei modelli di accesso.

Importa AnOnlineShop_facets.json per vedere le sfaccettature relative a questo caso d'uso.

Tutti i modelli di accesso e il modo in cui la progettazione dello schema li affronta sono riassunti nella tabella seguente:

Modello di accesso	Tabella di base/ GSI/LSI	Operazione	Valore della chiave di partizione	Valore della chiave di ordinamento
getCustomerByCustomerId	Tabella di base	GetItem	PK=customerId	SK=customerId
getProductByProductId	Tabella di base	GetItem	PK=productid	SK=productId
getWarehouseByWarehouseId	Tabella di base	GetItem	PK=warehouseId	SK=warehouseId
getProductInventoryByProductId	Tabella di base	Query	PK=productId	SK begins_with "w#"
getOrderDetailsByOrderId	Tabella di base	Query	PK=orderId	
getProductByOrderId	Tabella di base	Query	PK=orderId	SK begins_with "p#"
getInvoiceByOrderId	Tabella di base	Query	PK=orderId	SK begins_with "i#"
getShipmentByOrderId	Tabella di base	Query	PK=orderId	SK begins_with "sh#"
getOrderByProductIdForDateRange	GSI1	Query	PK=productId	SK tra date1 e date2
getInvoiceByInvoiceId	GSI1	Query	PK=invoiceId	SK=invoiceId

Modello di accesso	Tabella di base/ GSI/LSI	Operazione	Valore della chiave di partizione	Valore della chiave di ordinamento
getPaymentByInvoiceId	GSI1	Query	PK=invoiceId	SK=invoiceId
getShipmentDetailsByShipmentId	GSI1	Query	PK=shipmentId	SK=shipmentId
getShipmentByWarehouseId	GSI2	Query	PK=warehouseId	SK begins_with "sh#"
getProductInventoryByWarehouseId	GSI2	Query	PK=warehouseId	SK begins_with "p#"
getInvoiceByCustomerIdForDateRange	GSI2	Query	PK=customerId	SK tra i#date1 e i#date2
getProductsByCustomerIdForDateRange	GSI2	Query	PK=customerId	SK between p#date1 and p#date2

Schema finale per negozio online

Di seguito sono riportate le progettazioni dello schema finale. Per scaricare questa progettazione dello schema come un file JSON, consulta Esempi di progettazione DynamoDB su GitHub.

Tabella di 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**								
	p#12345	PK	SK	EntityType	Quantity				
		p#12345	w#12345	warehouseItem	50				
	p#99887	PK	SK	EntityType	Quantity				
		p#99887	w#12345	warehouseItem	4				
w#12345	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

Utilizzo di NoSQL Workbench con questa progettazione dello schema

Puoi importare questo schema finale in NoSQL Workbench, uno strumento visivo che fornisce funzionalità di modellazione dei dati, visualizzazione dei dati e sviluppo di query per DynamoDB, per esplorare e modificare ulteriormente il tuo nuovo progetto. Per iniziare, segui queste fasi:

1. Scarica NoSQL Workbench. Per ulteriori informazioni, consulta the section called “Scarica”.
2. Scarica il file dello schema JSON elencato in precedenza, che si trova già nel formato del modello NoSQL Workbench.
3. Importa il file dello schema JSON in NoSQL Workbench. Per ulteriori informazioni, consulta the section called “Importazione di un modello esistente”.
4. Dopo che è stato importato in NOSQL Workbench, puoi modificare il modello di dati. Per ulteriori informazioni, consulta the section called “Modifica di un modello esistente”.
5. Per visualizzare il tuo modello di dati, aggiungere dati di esempio o importare dati di esempio da un file CSV, utilizza la funzionalità Data Visualizer di NoSQL Workbench.

Best practice per la modellazione dei dati relazionali in DynamoDB

Questa sezione fornisce best practice per la modellazione dei dati relazionali in Amazon DynamoDB. Innanzitutto, presentiamo i concetti tradizionali di modellazione dei dati. Quindi, descriviamo i vantaggi dell'utilizzo di DynamoDB rispetto ai tradizionali sistemi di gestione di database relazionali e in che modo elimina la necessità di operazioni JOIN e riduce il sovraccarico.

Illustriamo quindi come progettare una tabella DynamoDB dimensionabile in modo efficiente. Infine, forniamo un esempio di come modellare i dati relazionali in DynamoDB.

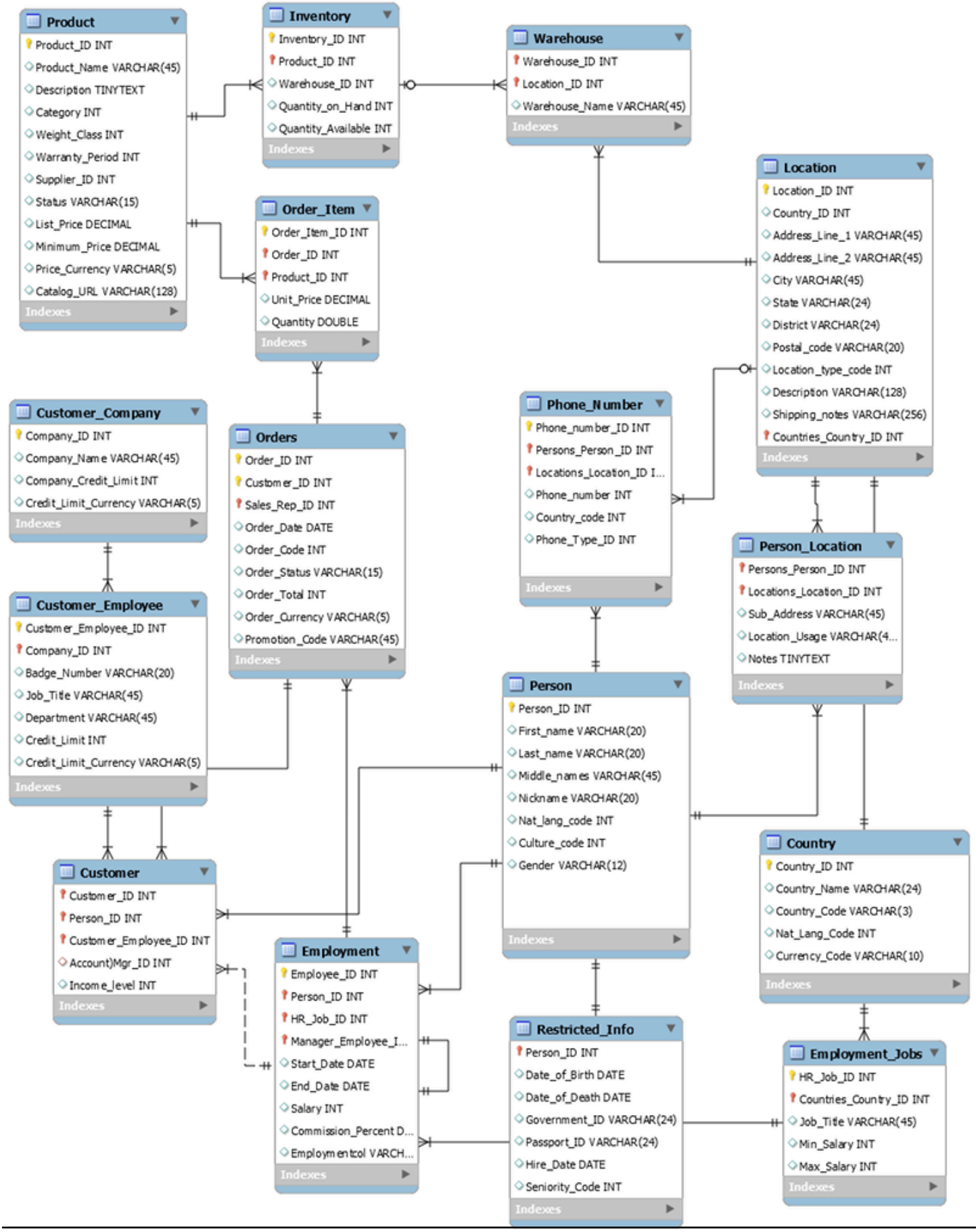
Argomenti

- Modelli di database relazionali tradizionali
- In che modo DynamoDB elimina la necessità di operazioni JOIN
- In che modo le transazioni DynamoDB eliminano il sovraccarico del processo di scrittura
- Fase iniziale per la modellazione dei dati relazionali in DynamoDB
- Esempio di modellazione dei dati relazionali in DynamoDB

Modelli di database relazionali tradizionali

I sistemi di gestione dei database relazionali (RDBMS) tradizionali archiviano i dati in una struttura relazionale normalizzata. L'obiettivo del modello di dati relazionali è ridurre la duplicazione dei dati (attraverso la normalizzazione) per supportare l'integrità referenziale e ridurre le anomalie dei dati.

Lo schema seguente è un esempio di modello di dati relazionale per un'applicazione generica di immissione degli ordini. L'applicazione supporta uno schema delle risorse umane che supporta i sistemi di supporto operativi e aziendali di un produttore ipotetico.


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

In quanto servizio di database non relazionale, DynamoDB offre molti vantaggi rispetto ai tradizionali sistemi di gestione di database relazionali.

In che modo DynamoDB elimina la necessità di operazioni JOIN

Un RDBMS utilizza un linguaggio SQL (Structure Query Language) per restituire i dati all'applicazione. A causa della normalizzazione del modello di dati, tali query richiedono in genere l'uso dell'operatore `JOIN` per combinare i dati di una o più tabelle.

Ad esempio, per generare un elenco di articoli dell'ordine d'acquisto ordinati in base alla quantità disponibile in tutti i magazzini che possono spedire tale articolo, puoi eseguire la query SQL seguente nello schema precedente.

```
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
```

Le query SQL di questo tipo possono fornire un'API flessibile per accedere ai dati, ma richiedono un alto livello di elaborazione. Ogni join nella query aumenta la complessità del runtime della query perché i dati di ogni tabella devono essere suddivisi in fasi e quindi assemblati per restituire il set di risultati.

Altri fattori che possono influire sul tempo impiegato per l'esecuzione delle query sono la dimensione delle tabelle e la presenza di indici nelle colonne da unire. La query precedente avvia query complesse in diverse tabelle e poi ordina il set di risultati.

L'eliminazione della necessità di `JOIN`s è la base della modellazione dei dati NoSQL. Ecco perché abbiamo creato DynamoDB per supportare Amazon.com e perché DynamoDB può offrire prestazioni costanti a larga scala. Data la complessità di runtime delle query SQL e di `JOIN`s le prestazioni RBDMS non sono costanti su larga scala. Ciò causa problemi di prestazioni man mano che le applicazioni dei clienti aumentano.

Sebbene la normalizzazione dei dati riduca la quantità di dati archiviati su disco, spesso le risorse più limitate che influiscono sulle prestazioni sono il tempo della CPU e la latenza della rete.

DynamoDB è progettato per ridurre al minimo entrambi i vincoli eliminando `JOIN`s (e incoraggiando la denormalizzazione dei dati) e ottimizzando l'architettura del database per rispondere

completamente a una query dell'applicazione con una singola richiesta a un elemento. Queste qualità consentono a DynamoDB di fornire prestazioni a una cifra in millisecondi su qualsiasi scala. Questo perché la complessità del runtime delle operazioni DynamoDB è costante, indipendentemente dalla dimensione dei dati, per i modelli di accesso comuni.

In che modo le transazioni DynamoDB eliminano il sovraccarico del processo di scrittura

Un altro fattore che può rallentare un RDBMS è l'uso di transazioni per scrivere in uno schema normalizzato. Come indicato nell'esempio, le strutture dei dati relazionali utilizzate dalla maggior parte delle applicazioni (OLTP) devono essere suddivise e distribuite in più tabelle logiche quando sono archiviate in un sistema RDBMS.

Quindi, un framework delle transazioni conforme ad ACID è necessario per evitare race condition e problemi di integrità dei dati che possono verificarsi se un'applicazione cerca di leggere un oggetto che sta per essere scritto. Un tale framework di transazioni, se abbinato a uno schema relazionale, può aggiungere un sovraccarico significativo al processo di scrittura.

L'implementazione delle transazioni in DynamoDB impedisce i problemi di scalabilità comuni riscontrati con un RDBMS. DynamoDB esegue questa operazione emettendo una transazione come singola chiamata API e limitando il numero di elementi a cui è possibile accedere in quella singola transazione. Le transazioni di lunga durata possono causare problemi operativi mantenendo i dati bloccati per un lungo periodo di tempo o in modo permanente, poiché la transazione non viene mai chiusa.

Per prevenire tali problemi in DynamoDB, le transazioni sono state implementate con due operazioni API distinte: `TransactWriteItems` e `TransactGetItems`. Queste operazioni API non hanno una semantica di inizio e fine comune in un RDBMS. Inoltre, DynamoDB ha un limite di accesso di 100 elementi in una transazione per prevenire allo stesso modo transazioni di lunga durata. Per ulteriori informazioni sulle transazioni DynamoDB, consulta Utilizzo delle transazioni.

Per questi motivi, se la tua azienda ha bisogno di una risposta a bassa latenza alle query a traffico elevato, lo sfruttamento di un sistema NoSQL ha senso dal punto di vista tecnico ed economico. Amazon DynamoDB aiuta a risolvere i problemi che limitano la scalabilità del sistema relazionale evitandoli.

Le prestazioni di un RDBMS in genere non sono sufficientemente scalabili per i seguenti motivi:

- Utilizza join costosi per riassemblare le visualizzazioni richieste dei risultati delle query.

- Normalizza i dati e li archivia in tabelle multiple che richiedono query multiple da scrivere nel disco.
- In genere ha i costi di prestazione di un sistema delle transazioni conforme ad ACID.

DynamoDB si ridimensiona bene per questi motivi:

- La flessibilità dello schema permette a DynamoDB di archiviare i dati gerarchici complessi in un singolo elemento.
- La progettazione della chiave composita gli permette di archiviare voci assieme nella stessa tabella.
- Le transazioni vengono eseguite in un'unica operazione. Il numero massimo di elementi a cui è possibile accedere è 100, per evitare operazioni di lunga durata.

Le query sull'archivio dei dati diventano molto più semplici, spesso come segue:

```
SELECT * FROM Table_X WHERE Attribute_Y = "somevalue"
```

DynamoDB lavora molto meno per fornire i dati richiesti rispetto a RDBMS dell'esempio precedente.

Fase iniziale per la modellazione dei dati relazionali in DynamoDB

> **Important**
>
> La progettazione di un NoSQL richiede un approccio diverso rispetto alla progettazione di un RDBMS. Per un sistema RDBMS puoi creare un modello di dati normalizzati senza pensare ai modelli di accesso. Puoi estenderlo in seguito quando si presentano nuovi quesiti e requisiti di query. Per contro, in Amazon DynamoDB, non si inizia a progettare lo schema finché non si sa a quali domande si deve rispondere. È assolutamente essenziale capire da subito i problemi di business e i casi d'uso dell'applicazione.

Per iniziare a progettare una tabella DynamoDB in grado di dimensionare in maniera efficace, è necessario prima seguire alcune procedure per identificare i modelli di accesso richiesti dai sistemi di operations e business support (OSS/BSS) che deve supportare.

- Per le nuove applicazioni, revisiona le storie degli utenti sulle attività e obiettivi. Documenta i vari casi d'uso che identifichi e analizza i modelli d'accesso che necessitano.

- Per le applicazioni esistenti, analizza i log di query per scoprire come il sistema viene utilizzato al momento e quali sono i modelli della chiave di accesso.

Dopo aver completato il processo, dovresti avere un elenco che può sembrare a quanto segue.

	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

In un'applicazione reale, l'elenco può essere molto più lungo. Ma questo elenco rappresenta il livello di complessità del modello di query che puoi trovare in un ambiente di produzione.

Un approccio comune alla progettazione dello schema di DynamoDB è identificare le entità a livello di applicazione e utilizzare la denormalizzazione e l'aggregazione delle chiavi composite per ridurre la complessità delle query.

In DynamoDB, ciò significa utilizzare chiavi di ordinamento composite, indici secondari globali sottoposti a overload, tabelle/indici partizionati e altri modelli di progettazione. Puoi utilizzare questi elementi per strutturare i dati in modo che un'applicazione possa recuperare ciò di cui hai bisogno per un determinato modello di accesso utilizzando una query singola in una tabella o indice. Il modello primario che puoi utilizzare per modellare lo schema normalizzato visualizzato in Modellazione relazionale è il modello dell'elenco di adiacenza. Altri modelli utilizzati in questa progettazione possono includere il partizionamento di scrittura dell'indice secondario globale, l'overload dell'indice secondario globale, le chiavi composite e le aggregazioni materializzate.

> **Important**
>
> In generale, è necessario mantenere il minor numero possibile di tabelle in un'applicazione DynamoDB. Le eccezioni sono i casi in cui sono coinvolti dati di serie temporali a volumi elevati o i set di dati che hanno modelli di accesso molto diversi. Una tabella singola con

indici invertiti possono solitamente abilitare query semplici per creare ed estrarre strutture di dati gerarchici complesse richieste dalla tua applicazione.

Per utilizzare NoSQL Workbench per DynamoDB per aiutarti a visualizzare la progettazione delle chiavi di partizione, consulta Creazione di modelli di dati con NoSQL Workbench.

Esempio di modellazione dei dati relazionali in DynamoDB

Questo esempio descrive come modellare i dati relazionali in Amazon DynamoDB. Una progettazione di tabella DynamoDB corrisponde allo schema della voce dell'ordine relazionale visualizzato in Modellazione relazionale. Segue l'Modello di progettazione elenchi di adiacenza, che è un modo comune di rappresentare le strutture di dati relazionali in DynamoDB.

Il modello di progettazione richiede che tu definisca un set di tipi di entità che solitamente sono collegati a varie tabelle nello schema relazionale. Le voci di entità vengono aggiunte alla tabella utilizzando una chiave primaria composta (partizione e ordinamento). La chiave di partizione di queste voci di entità è l'attributo che identifica in modo univoco la voce e al quale ci si riferisce solitamente come PK in tutte le voci. L'attributo di chiave di ordinamento contiene un valore di attributo che puoi utilizzare per un indice invertito o un indice secondario globale. Solitamente viene chiamato SK.

Definisci le seguenti entità che supportano lo schema della voce dell'ordine relazionale:

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

Dopo aver aggiunto queste voci di entità alla tabella, puoi definire le relazioni tra loro aggiungendo voci edge alle partizioni della voce di entità. La tabella seguente dimostra questa fase.
Tabella esempio che mostra le relazioni tra voci di entità.

In questo esempio, le partizioni `Employee`, `Order` e `Product Entity` sulla tabella hanno voci edge addizionali che contengono puntatori ad altre voci di entità sulla tabella. In seguito, definisci alcuni indici secondari globali (GSI) per supportare tutti i modelli di accesso definiti in precedenza. Le voci di entità non utilizzano tutte lo stesso tipo di valore per la chiave primaria o l'attributo della chiave di ordinamento. Ciò che è necessario è avere la chiave primaria e gli attributi della chiave di ordinamento presenti per essere inseriti nella tabella.

Il fatto che alcune di queste entità utilizzano nomi propri e altre utilizzano altre ID entità come valori della chiave di ordinamento permette allo stesso indice secondario globale di supportare molteplici tipi di query. Questa tecnica si chiama overload di GSI. In effetti elimina il limite di default di 20 indici secondari globali per le tabelle che contengono più tipi di elementi. Questo viene mostrato nel seguente diagramma come GSI 1.
Tabella di esempio che mostra gli indici secondari globali che supportano query multiple.

GSI 2 è progettato per supportare un modello di accesso all'applicazione abbastanza comune, che corrisponde all'inserimento di tutte le voci che hanno un certo stato sulla tabella. Per una tabella grande con una distribuzione non uniforme di voci negli stati disponibili, questo modello di accesso può risultare in un tasto di scelta rapida, a meno che le voci siano distribuite su più di una partizione logica sulla quale si può eseguire una query in parallelo. Il modello di progettazione si chiama `write sharding`.

Per ottenere ciò per GSI 2, l'applicazione aggiunge l'attributo della chiave primaria a ogni voce di ordine. Lo popola con un numero casuale in un intervallo 0N, dove N può essere calcolato generale utilizzando la formula riportata di seguito (a meno che ci sia una ragione specifica per non farlo).

```
ItemsPerRCU = 4KB / AvgItemSize

PartitionMaxReadRate = 3K * ItemsPerRCU

N = MaxRequiredIO / PartitionMaxReadRate
```

Ad esempio, presumi di aspettarti quanto segue:

- Fino a due milioni di ordini saranno presenti nel sistema, aumentando fino a tre milioni in cinque anni.

- Fino al 20 per cento di questi ordini sarà in uno stato OPEN in qualsiasi momento.
- Il record medio di un ordine è 100 byte, con tre record `OrderItem` nella partizione dell'ordine con circa 50 byte ciascuno, dando dimensioni medie dell'entità dell'ordine di 250 byte.

Per quella tabella, il calcolo del fattore N sembrerebbe a quanto segue.

```
ItemsPerRCU = 4KB / 250B = 16

PartitionMaxReadRate = 3K * 16 = 48K

N = (0.2 * 3M) / 48K = 13
```

In questo caso, devi distribuire tutti gli ordini in circa 13 partizioni logiche su GSI 2 per assicurare che una lettura di tutte le voci `Order` con uno stato `OPEN` non provochi una partizione hot sul livello di storage fisico. È buona norma aumentare questo numero per permettere anomalie nel set di dati. Quindi un modello che utilizza `N = 15` probabilmente va bene. Come indicato prima, ciò si ottiene aggiungendo il valore casuale 0-N all'attributo GSI 2 PK di ogni record `Order` e `OrderItem` inserito nella tabella.

Questa suddivisione presume che il modello di accesso che richiede la raccolta di tutte le fatture `OPEN` accada molto raramente permettendoti di utilizzare la capacità di ottimizzazione per soddisfare la richiesta. Puoi eseguire una query sull'indice secondario globale seguente utilizzando una condizione di chiave di ordinamento per `State` e `Date Range` per produrre un sottoinsieme di tutti gli `Orders` in un determinato stato quando necessario.
Tabella di esempio che mostra la chiave primaria di GSI 2 e gli attributi previsti.

In questo esempio, le voci sono distribuite casualmente nelle 15 partizioni logiche. Questa struttura funziona perché il modello di accesso richiede un ampio numero di voci da recuperare. Quindi è improbabile che qualsiasi dei 15 thread daranno set di risultati vuoti che potrebbero rappresentare potenzialmente una capacità sprecata. Una query utilizza sempre una unità di capacità di lettura (RCU) o una unità di capacità di scrittura (WCU) anche quando non ci sono risultati e nessun dato viene scritto.

Se il modello di accesso richiede una query ad alta velocità in questo indice secondario globale che produce un set di risultati sparse, è probabilmente meglio utilizzare un algoritmo hash per distribuire le voci piuttosto che un modello casuale. In questo caso, è possibile selezionare un attributo conosciuto quando viene eseguita la query al runtime ed eseguire l'hash di quell'attributo in

uno spazio di chiave 0-14 quando vengono inseriti gli elementi. Possono poi essere letti in maniera efficace dall'indice secondario globale.

Infine, puoi rivedere i modelli di accesso definiti in precedenza. Di seguito è riportato l'elenco dei modelli di accesso e delle condizioni di query che utilizzerai con la nuova versione DynamoDB dell'applicazione per adattarli.

N. S.	Modelli di accesso	Condizioni di query
1	Ricerca dei dettagli dei dipendenti tramite ID dipendente	Chiave primaria sulla tabella, ID="HR-EMPLOYEE"
2	Esecuzione di query sui dettagli dei dipendenti tramite Nome dipendente	Utilizzare GSI-1, PK="Emplo yee Name"
3	Ottenimento dei soli dettagli del lavoro corrente di un dipendente	Chiave primaria sulla tabella, PK=HR-EMPLOYEE-1, SK inizia con "JH"
4	Ottenimento di ordini per un cliente per un intervallo di date	Utilizzare GSI-1, PK=CUSTOMER1, SK="STATUS-DATE", per ogni StatusCode
5	Mostra tutti gli ordini in stato OPEN per un intervallo di date per tutti i clienti	Utilizzare GSI-2, PK=query in parallelo per l'intervallo [0..N], SK tra OPEN-Date1 e OPEN-Date2
6	Tutti i dipendenti assunti di recente	Utilizzare GSI-1, PK="HR-CO NFIDENTIAL', SK > date1
7	Trova tutti i dipendenti in un magazzino specifico	Utilizzare GSI-1, PK=WAREHOUSE1
8	Ottenimento di tutti gli Orderitems per un prodotto,	Utilizzare GSI-1, PK=PRODUCT1

N. S.	Modelli di accesso	Condizioni di query
	inclusi gli inventari della posizione del magazzino	
9	Ottenimento dei clienti per rappresentante account	Utilizzare GSI-1, PK=ACCOUNT-REP
10	Ottenimento di ordini per rappresentante account e data	Utilizzare GSI-1, PK=ACCOUNT-REP, SK="STATUS-DATE", per ogni StatusCode
11	Ottenimento di tutti i dipendenti con una mansione specifica	Utilizza GSI-1, PK=JOBTITLE
12	Ottenimento dell'inventario per prodotto e magazzino	Chiave primaria sulla tabella, PK=OE-PRODUCT1,SK=PRODUCT1
13	Ottenimento dell'inventario totale dei prodotti	Chiave primaria sulla tabella, PK=OE-PRODUCT1,SK=PRODUCT1
14	Ottenimento dei rappresentanti dell'account classificati in base al totale degli ordini e al periodo di vendita	Utilizzare GSI-1, PK=YYYY-Q1, scanIndexForward=False

Migrazione da un database relazionale a DynamoDB

La migrazione di un database relazionale in DynamoDB richiede un'attenta pianificazione per garantire un esito positivo. Questa guida è utile per comprendere come funziona questo processo, quali sono gli strumenti a disposizione e come valutare quindi le potenziali strategie di migrazione e selezionarne una più adatta alle proprie esigenze.

Argomenti

- Motivi per migrare a DynamoDB
- Considerazioni sulle tempistiche di migrazione di un database relazionale a DynamoDB
- Come funziona una migrazione a DynamoDB
- Strumenti per la migrazione a DynamoDB
- Scelta della strategia appropriata per la migrazione a DynamoDB
- Esecuzione di una migrazione offline a DynamoDB
- Esecuzione di una migrazione ibrida a DynamoDB
- Esecuzione di una migrazione online a DynamoDB migrando ogni tabella 1:1
- Esecuzione di una migrazione online a DynamoDB con una tabella di staging personalizzata

Motivi per migrare a DynamoDB

La migrazione ad Amazon DynamoDB offre una serie di vantaggi interessanti per aziende e organizzazioni. Ecco alcuni vantaggi chiave che rendono DynamoDB una scelta interessante per la migrazione dei database:

- Scalabilità: DynamoDB è progettato per gestire carichi di lavoro enormi e scalare in modo fluido per adattarsi a volumi e traffico di dati in crescita. Con DynamoDB, è possibile aumentare o ridurre verticalmente un database in base alla richiesta in modo semplice, assicurando che le applicazioni siano in grado di gestire picchi improvvisi di traffico senza compromettere le prestazioni.
- Prestazioni: DynamoDB offre un accesso ai dati a bassa latenza, permettendo alle applicazioni di recuperare ed elaborare i dati con una velocità eccezionale. La sua architettura distribuita garantisce che le operazioni di lettura e scrittura siano distribuite su più nodi, offrendo tempi di risposta coerenti con risposte in pochi millisecondi anche a frequenze di richieste elevate.
- Completamente gestito: DynamoDB è un servizio completamente gestito fornito da AWS. Ciò significa che AWS gestisce gli aspetti operativi della gestione del database, tra cui il provisioning,

la configurazione, l'applicazione di patch, i backup e la scalabilità. Ciò consente di concentrarsi maggiormente sullo sviluppo delle applicazioni e meno sulle attività di amministrazione del database.

- Architettura serverless: DynamoDB supporta un modello serverless, noto come DynamoDB on demand, in cui il pagamento avviene solo per le effettive richieste di lettura e scrittura effettuate dall'applicazione senza richiedere il provisioning anticipato della capacità. Questo pay-per-request modello offre efficienza in termini di costi e spese operative minime, in quanto si pagano solo le risorse utilizzate senza la necessità di fornire e monitorare la capacità.
- Flessibilità NoSQL: a differenza dei database relazionali tradizionali, DynamoDB segue un modello di dati NoSQL, che offre flessibilità nella progettazione dello schema. Con DynamoDB, è possibile archiviare dati strutturati, semistrutturati e non strutturati, rendendoli ideali per la gestione di tipi di dati diversi e in evoluzione. Questa flessibilità consente cicli di sviluppo più rapidi e un adattamento più semplice alle mutevoli esigenze aziendali.
- Disponibilità e durabilità elevate: DynamoDB replica i dati su più zone di disponibilità all'interno di una Regione, garantendo elevata disponibilità e durabilità dei dati. Gestisce automaticamente la replica, il failover e il recupero, riducendo al minimo il rischio di perdita dei dati o interruzioni del servizio. DynamoDB offre un accordo sul livello di servizio (SLA) di disponibilità fino al 99,999%.
- Sicurezza e conformità: DynamoDB si integra con AWS Identity and Access Management per il controllo granulare degli accessi. Fornisce la crittografia dei dati a riposo e in transito, garantendo la sicurezza dei dati. DynamoDB aderisce inoltre a vari standard di conformità, tra cui HIPAA, PCI DSS e GDPR, che consentono di soddisfare i requisiti normativi.
- Integrazione con l' AWS ecosistema: come parte dell' AWS ecosistema, DynamoDB si integra perfettamente con AWS altri servizi, come, e. AWS Lambda CloudFormation AWS AppSync Questa integrazione consente di creare architetture serverless, sfruttare l'infrastruttura come codice e creare applicazioni basate sui dati in tempo reale.

Considerazioni sulle tempistiche di migrazione di un database relazionale a DynamoDB

I sistemi di gestione dei database relazionali e i database NoSQL hanno diversi vantaggi e svantaggi. Queste differenze rendono la progettazione del database diversa tra i due sistemi.

Tipo di attività	Database relazionale	Database NoSQL
Esecuzione di query sul database	Nei database relazionali è possibile eseguire query sui dati in modo flessibile, ma le query sono relativam ente costose e non scalano bene in situazioni di traffico elevato (consulta Fase iniziale per la modellazione dei dati relazionali in DynamoDB). Un'applicazione di database relazionale può implementare la logica aziendale nelle stored procedure, nelle sottoquery SQL, nelle query di aggiornam ento in blocco e nelle query di aggregazione.	In un database NoSQL come DynamoDB, le query possono essere eseguite in modo efficiente in un numero limitato di modi, al di fuori dei quali possono essere costose e lente. Le scritture su DynamoDB sono singleton. La logica di business dell'appl icazione che in precedenza veniva eseguita nelle stored procedure deve essere rifattori zzata per essere eseguita all'esterno di DynamoDB in codice personalizzato eseguito su un host come Amazon Amazon o. EC2 AWS Lambda
Progettazione del database	La progettazione viene effettuata tenendo a mente la flessibilità senza doversi preoccupare dei dettagli dell'implementazione o delle prestazioni. L'ottimizzazione delle query in genere non ha ripercussioni sulla progettaz ione dello schema, ma la normalizzazione è importante.	Lo schema viene progettat o in maniera specifica per fare in modo che le query più comuni e importanti siano il più possibile veloci e non costose. Le tue strutture di dati sono programmate per i requisiti specifici dei casi d'uso del tuo business.

La progettazione per un database NoSQL richiede una mentalità diversa rispetto alla progettazione per un sistema di gestione di database relazionali (RDBMS, Relational Database Management System). Per un sistema RDBMS puoi creare un modello di dati normalizzati senza pensare ai modelli di accesso. Puoi estenderlo in seguito quando si presentano nuovi quesiti e requisiti di query. Puoi organizzare ogni tipo di dati nella sua tabella.

Con la progettazione NoSQL è possibile progettare lo schema per DynamoDB quando si è a conoscenza delle domande alle quali si deve rispondere. È essenziale capire da subito i problemi di business e i modelli di lettura e scrittura dell'applicazione. In un'applicazione DynamoDB è inoltre necessario cercare di mantenere il minor numero possibile di tabelle. Un numero inferiore di tabelle rende le cose più scalabili, richiede una minore gestione delle autorizzazioni e riduce il sovraccarico dell'applicazione DynamoDB. Può anche aiutare a mantenere i costi di backup complessivi più bassi.

L'attività di modellare i dati relazionali per DynamoDB e creare una nuova versione dell'applicazione front-end costituisce un argomento a parte. Questa guida presuppone la disponibilità della nuova versione dell'applicazione creata per utilizzare DynamoDB, ma è comunque necessario determinare il modo migliore per migrare e sincronizzare i dati storici durante la conversione.

Considerazioni sul dimensionamento

La dimensione massima di ogni elemento (riga) che viene archiviato in una tabella DynamoDB è 400 KB. Per ulteriori informazioni, consulta the section called "Quote". La dimensione dell'elemento è determinata dalla dimensione totale di tutti i nomi e i valori degli attributi in un elemento. Per ulteriori informazioni, consulta the section called "Dimensioni e formati degli elementi".

Se l'applicazione ha bisogno di archiviare più dati in un elemento rispetto al limite di dimensione consentito da DynamoDB, suddividi l'elemento in una raccolta di elementi, comprimi i dati dell'elemento o archivia l'elemento come un oggetto in Amazon Simple Storage Service (Amazon S3) archiviando l'identificatore dell'oggetto Amazon S3 nell'elemento DynamoDB. Per informazioni, consulta the section called "Elementi di grandi dimensioni". Il costo dell'aggiornamento di un elemento si basa sulla dimensione completa dell'elemento. Per i carichi di lavoro che richiedono aggiornamenti frequenti degli elementi esistenti, l'aggiornamento di elementi di piccole dimensioni di uno o due KB avrà un costo inferiore rispetto agli elementi più grandi. Per ulteriori informazioni sulle raccolte di elementi, consulta the section called "Utilizzo di Raccolte di elementi".

Per la scelta degli attributi delle chiavi di partizione e ordinamento, le altre impostazioni della tabella, la dimensione e la struttura degli elementi e se creare indici secondari, assicurati di consultare la documentazione DynamoDB Modeling documentation e la guida per the section called "Ottimizzazione dei costi". Testa il piano di migrazione in modo che la tua soluzione DynamoDB sia efficiente in termini di costi e si adatti alle funzionalità e ai limiti di DynamoDB.

Come funziona una migrazione a DynamoDB

Prima di esaminare gli strumenti di migrazione a disposizione, considera come le scritture vengono elaborate da DynamoDB.

L'operazione di scrittura predefinita e più comune è una singola operazione API `PutItem`. È possibile eseguire un'operazione `PutItem` in un ciclo per elaborare set di dati. DynamoDB supporta connessioni simultanee praticamente illimitate, quindi supponendo che sia possibile configurare ed eseguire una routine di caricamento multithread di massa MapReduce come o Spark, la velocità di scrittura è limitata solo dalla capacità della tabella di destinazione (anch'essa generalmente illimitata).

Quando si caricano dati in DynamoDB, è importante comprendere la velocità di scrittura dello strumento di caricamento. Se gli elementi (righe) in caricamento hanno una dimensione pari o inferiore a 1 KB, questa velocità è semplicemente il numero di elementi al secondo. È quindi possibile dotare la tabella di destinazione di una quantità sufficiente di WCU (Write Capacity Units, unità di capacità di scrittura) per gestire questa frequenza. Se lo strumento di caricamento supera la capacità allocata in un dato secondo, le richieste aggiuntive potrebbero essere sottoposte a limitazione (della larghezza di banda della rete) o rifiutate del tutto. Puoi verificare la presenza di accelerazioni nei CloudWatch grafici che si trovano nella scheda di monitoraggio della console DynamoDB.

La seconda operazione che può essere eseguita è con un'API correlata denominata `BatchWriteItem`. `BatchWriteItem` consente di combinare fino a 25 richieste di scrittura in un'unica chiamata API. Queste vengono ricevute dal servizio ed elaborate come richieste `PutItem` separate nella tabella. Attualmente, scegliendo`BatchWriteItem`, non avrai il vantaggio dei nuovi tentativi automatici inclusi nell' AWS SDK quando effettui chiamate singleton con. `PutItem` Pertanto, in caso di errori come le eccezioni di limitazione (della larghezza di banda della rete), sarà necessario cercare l'elenco di tutte le scritture non andate a buon fine nella chiamata di risposta a `BatchWriteItem`. Per ulteriori informazioni sulla gestione degli avvisi di limitazione nel caso in cui questi vengano rilevati nelle tabelle di limitazione, consulta. CloudWatch the section called "Throttling"

Il terzo tipo di importazione dei dati è possibile con la funzionalità DynamoDB Import from S3. Questa funzionalità consente di mettere in staging un set di dati di grandi dimensioni in Amazon S3 e di chiedere a DynamoDB di importare automaticamente i dati in una nuova tabella. L'importazione non è istantanea e richiederà tempo in modo proporzionale alle dimensioni del set di dati. Tuttavia, è comodo in quanto non richiede una piattaforma ETL o un codice DynamoDB personalizzato. DynamoDB carica i dati in una nuova tabella creata dall'importazione. Al momento non consente di caricare dati in una tabella esistente. DynamoDB importa i dati così come sono, senza trasformazioni.

Analogamente a `PutItem`, richiede un processo a monte e scrive i dati nel formato scelto in un bucket Amazon S3.

Strumenti per la migrazione a DynamoDB

Esistono diversi strumenti di migrazione ed ETL comuni che è possibile utilizzare per migrare i dati in DynamoDB.

Amazon offre una serie di strumenti di dati che possono essere utilizzati durante la migrazione, tra cui AWS Database Migration Service (DMS), AWS Glue, Amazon EMR e Streaming gestito da Amazon per Apache Kafka. Tutti questi strumenti possono essere utilizzati per eseguire una migrazione con tempi di inattività e possono sfruttare le funzionalità di Change Data Capture (CDC) del database relazionale per supportare le migrazioni online. Nella scelta di uno strumento, è utile considerare il set di competenze e l'esperienza dell'organizzazione con ogni strumento, oltre alle funzionalità, alle prestazioni e al costo di ciascuno di essi.

Molti clienti scelgono di scrivere i propri script e processi di migrazione per creare trasformazioni dei dati personalizzate per il processo di migrazione. Se si prevede di gestire una tabella DynamoDB ad alto volume con traffico di scrittura intenso o processi regolari di caricamento in blocco di grandi dimensioni, si potrebbe voler scrivere in prima persona il codice di migrazione per acquisire maggiore familiarità con il comportamento di DynamoDB in presenza di un traffico di scrittura intenso. Scenari come la gestione della limitazione (della larghezza di banda della rete) e il provisioning efficiente delle tabelle possono essere sperimentati nelle prime fasi del progetto, quando si esegue una migrazione pratica.

Scelta della strategia appropriata per la migrazione a DynamoDB

Un'applicazione di database relazionale di grandi dimensioni può estendersi su un centinaio o più tabelle e supportare diverse funzioni applicative. Quando si predispone una migrazione su larga scala, occorre prendere in considerazione la possibilità di suddividere l'applicazione in componenti o microservizi più piccoli e di migrare un piccolo set di tabelle alla volta. È quindi possibile migrare componenti aggiuntivi su DynamoDB a ondate.

Nella selezione di una strategia di migrazione, diversi fattori possono indirizzare l'utente verso una soluzione o un'altra. Si presentano queste opzioni in un albero decisionale per semplificare le opzioni a disposizione in base ai requisiti e alle risorse disponibili. I concetti sono qui brevemente menzionati (ma verranno trattati più approfonditamente più avanti nella guida):

- Migrazione offline: se l'applicazione è in grado di tollerare tempi di inattività durante la migrazione, questo semplificherà il processo di migrazione.
- Migrazione ibrida: questo approccio consente un tempo di attività parziale durante la migrazione, ad esempio consentendo le letture ma non le scritture, oppure consente letture e inserimenti ma non aggiornamenti ed eliminazioni.
- Migrazione online: la migrazione delle applicazioni che richiedono l'assenza di tempi di inattività durante la migrazione è meno semplice e può richiedere una pianificazione significativa e uno sviluppo personalizzato. Una decisione chiave riguarda la stima e la ponderazione dei costi legati alla creazione di un processo di migrazione personalizzato rispetto al costo per l'azienda derivante dai tempi di inattività durante la fase di conversione.

Se	And	Allora
È possibile disattivare l'applicazione per qualche tempo durante una finestra di manutenzione per eseguire la migrazione dei dati. È una migrazione offline.		Utilizzare AWS DMS ed eseguire una migrazione offline utilizzando un'attività a pieno carico. Se lo desideri, preforma i dati di origine con un codice SQL `VIEW`.
È possibile eseguire l'applicazione in modalità di sola lettura durante la migrazione. È una migrazione ibrida.		Disabilita le scritture all'interno dell'applicazione o del database di origine. Utilizza AWS DMS ed esegui una migrazione offline utilizzando un'attività a pieno carico.
Durante la migrazione è possibile eseguire l'applicazione con letture e inserimenti di nuovi record, ma senza aggiornamenti o eliminazioni. È una migrazione ibrida.	Si hanno competenze nello sviluppo di applicazioni e l'app relazionale esistente può essere aggiornata per eseguire doppie scritture, anche su DynamoDB, per tutti i nuovi record	Utilizza AWS DMS ed esegui una migrazione offline utilizzando un'attività a pieno carico. Contemporaneamente implementa una versione dell'app esistente che consenta la lettura e l'esecuzione di doppie scritture.

Se	And	Allora
È necessaria una migrazione con tempi di inattività minimi. È una migrazione online.	• Si stanno migrando le tabelle di origine 1:1 in DynamoDB senza modifiche importanti allo schema.	Utilizzare AWS DMS per eseguire una migrazione dei dati online. Esegui un'attività di caricamento in blocco seguita da un'attività di sincronizzazione CDC.
È necessaria una migrazione con tempi di inattività minimi. È una migrazione online.	• Si stanno combinando le tabelle di origine in un numero inferiore di tabelle DynamoDB seguendo la filosofia dello schema impilato o della tabella singola. • Si dispone di competenze per lo sviluppo di database in backend e capacità di riserva sull'host SQL.	Crea la tabella NoSQL-ready all'interno del database SQL. Compilalo e sincronizzalo utilizzando JOINs,, UNIONs, trigger VIEWs, stored procedure.
È necessaria una migrazione con tempi di inattività minimi. È una migrazione online.	• Si stanno combinando le tabelle di origine in un numero inferiore di tabelle DynamoDB seguendo la filosofia della tabella singola. Ad esempio: • Non si dispone di competenze per lo sviluppo di database di backend e di capacità di riserva sull'host SQL.	Prendi in considerazione gli approcci di migrazione ibridi o offline.

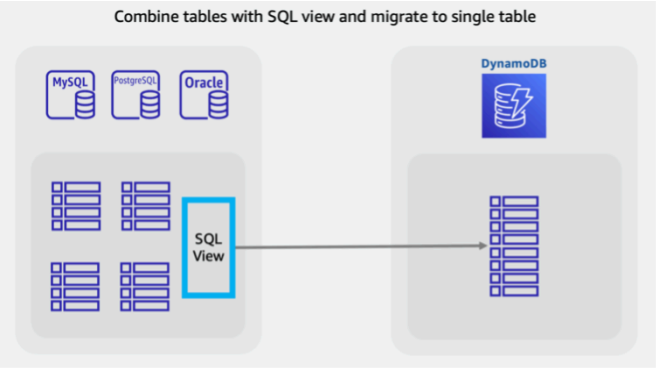
Se	And	Allora
È necessaria una migrazione con tempi di inattività minimi. È una migrazione online.	È possibile saltare la migrazione dei dati storici delle transazioni oppure archiviarli in Amazon S3 anziché migrarli. È necessario migrare solo alcune piccole tabelle statiche.	Scrivi uno script o usa qualsiasi strumento ETL per migrare le tabelle. Se lo desideri, preforma i dati di origine con un codice SQL `VIEW`.

Esecuzione di una migrazione offline a DynamoDB

Le migrazioni offline sono adatte quando è possibile un tempo di inattività per eseguire la migrazione. I database relazionali in genere richiedono almeno qualche tempo di inattività ogni mese per la manutenzione e l'applicazione delle patch, oppure possono richiedere tempi di inattività più lunghi per gli aggiornamenti hardware o gli aggiornamenti delle versioni principali.

Amazon S3 può essere utilizzato come area di staging durante una migrazione. I dati archiviati in formato CSV (valori separati da virgola) o DynamoDB JSON possono essere importati automaticamente in una nuova tabella DynamoDB utilizzando la funzionalità di importazione in DynamoDB da S3.

Si potrebbe voler combinare le tabelle per sfruttare modelli di accesso NoSQL unici (ad esempio, trasformando quattro tabelle legacy in un'unica tabella DynamoDB). Una singola richiesta di documento chiave-valore o una query per una raccolta di elementi preraggruppati di solito restituisce una latenza migliore rispetto a un database SQL che esegue una query join multi-tabella. Tuttavia, ciò rende l'attività di migrazione più difficile. Una query view SQL potrebbe essere adatta all'interno del database di origine per preparare un singolo set di dati che rappresenti tutte e quattro le tabelle in un unico set.



Questa visualizzazione consente di creare tabelle `JOIN` in una forma denormalizzata oppure può mantenere le entità normalizzate e impilare le tabelle utilizzando un query `UNION` SQL. Le decisioni chiave relative alla ridefinizione dei dati relazionali sono trattate in questo video. Per le migrazioni offline, l'utilizzo di una vista per combinare le tabelle è un ottimo modo per modellare i dati per uno schema a tabella singola di DynamoDB.

Pianifica

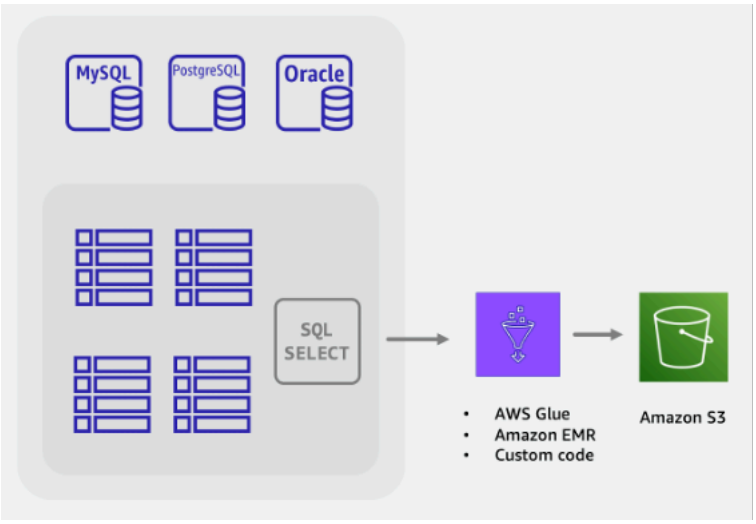
Esecuzione di una migrazione offline con Amazon S3

Strumenti

- Un processo ETL per estrarre e trasformare i dati SQL e archiviarli in un bucket S3, ad esempio:
  - AWS Database Migration Service, un servizio in grado di caricare in blocco i dati storici e di elaborare anche i record del CDC per sincronizzare le tabelle di origine e di destinazione.
  - AWS Glue
  - Amazon EMR
  - Il proprio codice personalizzato
- La funzionalità di importazione in DynamoDB da S3

Fasi di migrazione offline:

1. Crea un job ETL in grado di eseguire query sul database SQL, trasformare i dati della tabella in formato DynamoDB JSON o CSV e salvarli in un bucket S3.



2. La funzionalità di importazione in DynamoDB da S3 viene invocata per creare una nuova tabella e caricare automaticamente i dati dal bucket S3.

La migrazione completamente offline è semplice e diretta, ma potrebbe non essere apprezzata dai proprietari e dagli utenti delle applicazioni. Gli utenti trarrebbero vantaggio se l'applicazione potesse fornire livelli di servizio ridotti durante la migrazione, anziché non fornire alcun servizio.

È possibile aggiungere funzionalità per disabilitare le scritture durante la migrazione offline, consentendo al contempo alle letture di continuare normalmente. Gli utenti delle applicazioni possono comunque sfogliare i dati esistenti ed eseguire quey in modo sicuro durante la migrazione dei dati relazionali. Se questo è l'obiettivo, prosegui la lettura per ulteriori informazioni sulle migrazioni ibride.

Esecuzione di una migrazione ibrida a DynamoDB

Sebbene tutte le applicazioni di database eseguano operazioni di lettura e scrittura, è necessario considerare i tipi di operazioni di scrittura eseguite quando si pianifica una migrazione ibrida o online. Le scritture sul database possono essere classificate in tre bucket: inserimenti, aggiornamenti ed eliminazioni. Alcune applicazioni potrebbero non richiedere l'elaborazione immediata delle eliminazioni. Ad esempio, queste applicazioni possono rimandare le eliminazioni a un processo di pulizia in blocco alla fine del mese. La migrazione di questi tipi di applicazioni può essere più semplice e al contempo consentire tempi di attività parziali.

Piano

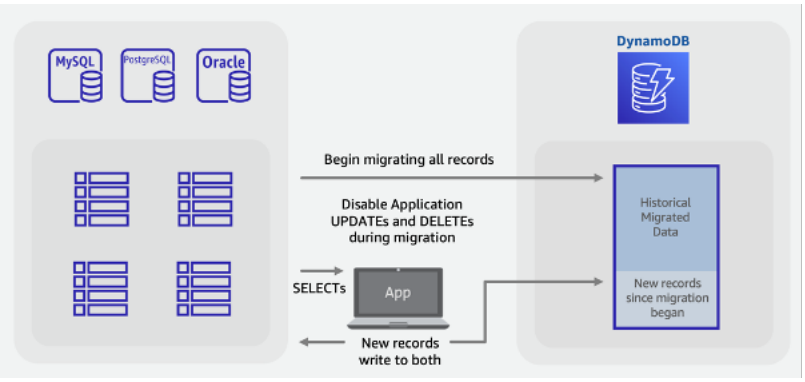
Esegui una online/offline migrazione ibrida con doppia scrittura delle applicazioni

Strumenti

- Un processo ETL per estrarre e trasformare i dati SQL e archiviarli in un bucket S3, ad esempio:
  - AWS DMS
  - AWS Glue
  - Amazon EMR
  - Il proprio codice personalizzato

Fasi della migrazione ibrida:

1. Crea la tabella DynamoDB. Questa tabella riceverà sia dati storici in blocco che i nuovi dati in tempo reale.
2. Crea una versione dell'applicazione legacy con le eliminazioni e gli aggiornamenti disabilitati mentre esegui tutti gli inserimenti come doppia scrittura sia nel database SQL che in DynamoDB.
3. Iniziate il lavoro o l' AWS DMS attività ETL per completare i dati esistenti e distribuite contemporaneamente la nuova versione dell'applicazione
4. Una volta completato il processo di riempimento, DynamoDB disporrà di tutti i record esistenti e nuovi e sarà pronto per la conversione dell'applicazione.



> **Note**
>
> Il processo di riempimento scrive direttamente da SQL a DynamoDB. Non è possibile utilizzare la funzionalità di importazione da S3 come nell'esempio di migrazione offline, poiché tale funzionalità crea una nuova tabella che sarà attiva solo dopo il caricamento dei dati da parte di DynamoDB.

Esecuzione di una migrazione online a DynamoDB migrando ogni tabella 1:1

Molti database relazionali dispongono di una funzionalità denominata Change Data Capture (CDC), in cui il database consente agli utenti di richiedere un elenco delle modifiche a una tabella avvenute prima o dopo un determinato momento. Il CDC utilizza i log interni per abilitare questa funzionalità e non richiede che la tabella presenti colonne di timestamp per funzionare.

Nell'esecuzione della migrazione di uno schema di tabelle SQL a un database NoSQL, si potrebbe voler combinare e rimodellare i dati in un numero inferiore di tabelle. In questo modo sarà possibile raccogliere i dati in un'unica ubicazione ed evitare di dover unire manualmente i dati correlati in operazioni di lettura in più fasi. Tuttavia, il modellamento dei dati in un'unica tabella non è sempre necessario e a volte le tabelle vengono migrate 1:1 in DynamoDB. Queste migrazioni 1:1 sono meno complicate in quanto è possibile sfruttare la funzionalità CDC del database di origine, utilizzando strumenti ETL comuni che supportano questo tipo di migrazione. I dati per ogni riga possono comunque essere trasformati in nuovi formati, ma l'ambito di ogni tabella rimane lo stesso.

Prendi in considerazione la migrazione delle tabelle SQL 1:1 in DynamoDB, con l'avvertenza che DynamoDB non supporta query join lato server. Sarà necessario aggiungere logica all'applicazione per combinare i dati di più tabelle.

Pianifica

Esegui una migrazione online di ogni tabella in DynamoDB utilizzando AWS DMS

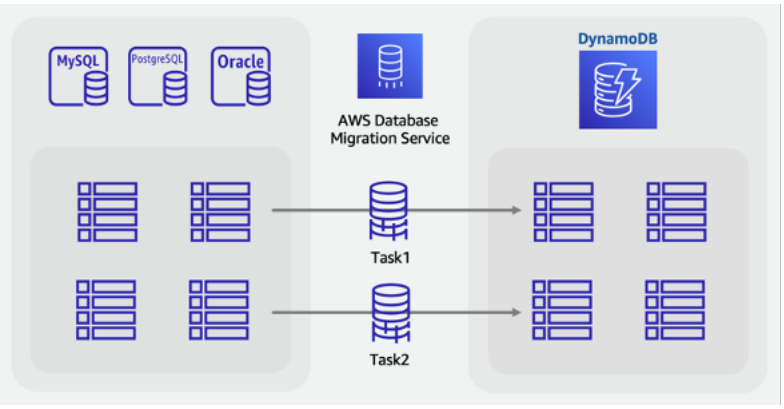
Strumenti

- AWS DMS (DMS)

Fasi di migrazione online:

1. Identifica nello schema di origine le tabelle che verranno migrate
2. Crea lo stesso numero di tabelle in DynamoDB con la stessa struttura di chiave dell'origine
3. Crea un server di replica AWS DMS e configura gli endpoint di origine e di destinazione
4. Definisci le trasformazioni necessarie per riga (ad esempio colonne concatenate o conversione delle date nel formato di stringa ISO-8601)
5. Crea un'attività di migrazione per ogni tabella per Pieno carico e Change Data Capture
6. Monitora queste attività fino all'inizio della fase di replica in corso

7. A questo punto, è possibile eseguire qualsiasi verifica di convalida e quindi far passare gli utenti all'applicazione che legge e scrive su DynamoDB.



Esecuzione di una migrazione online a DynamoDB con una tabella di staging personalizzata

Come nello scenario di migrazione offline riportato sopra, è possibile scegliere di combinare le tabelle per sfruttare modelli di accesso NoSQL unici (ad esempio, trasformando quattro tabelle legacy in un'unica tabella DynamoDB). Una query `VIEW` SQL potrebbe essere adatta all'interno del database di origine per preparare un singolo set di dati che rappresenti tutte e quattro le tabelle in un unico set.

Tuttavia, per le migrazioni online con dati in tempo reale e mutevoli, non è possibile sfruttare le funzionalità CDC in quanto non sono supportate per le query `VIEW`. Se le tabelle includono una colonna di timestamp dell'ultimo aggiornamento e questa è incorporata nella query `VIEW`, è possibile creare un processo ETL personalizzato che la utilizzi per eseguire un caricamento in blocco con sincronizzazione.

Un nuovo approccio a questa sfida consiste nell'utilizzare funzionalità SQL standard come query view, stored procedure e trigger per creare una nuova tabella SQL nel formato DynamoDB NoSQL finale desiderato.

Se il server di database dispone di capacità di riserva, è possibile creare questa singola tabella di staging prima dell'inizio della migrazione. A tale scopo, è necessario scrivere una stored procedure che legga le tabelle esistenti, trasformi i dati in base alle esigenze e scriva nella nuova tabella di staging. È possibile aggiungere un set di trigger per replicare le modifiche nelle tabelle nella tabella di staging in tempo reale. Se i trigger non sono consentiti in base alla policy aziendale, le modifiche

alle stored procedure possono ottenere lo stesso risultato. È necessario aggiungere alcune righe di codice alle procedure che scrivono dati per scrivere le stesse modifiche anche nella tabella di staging.

La disponibilità di questa tabella di staging completamente sincronizzata con le tabelle delle applicazioni precedenti offrirà un ottimo punto di partenza per una migrazione in tempo reale. Gli strumenti che utilizzano il database CDC per eseguire migrazioni in tempo reale, ad esempio AWS DMS, possono ora essere utilizzati su questa tabella. Un vantaggio di questo approccio è che utilizza le competenze e le funzionalità SQL ben note disponibili nel motore di database relazionale.

Pianifica

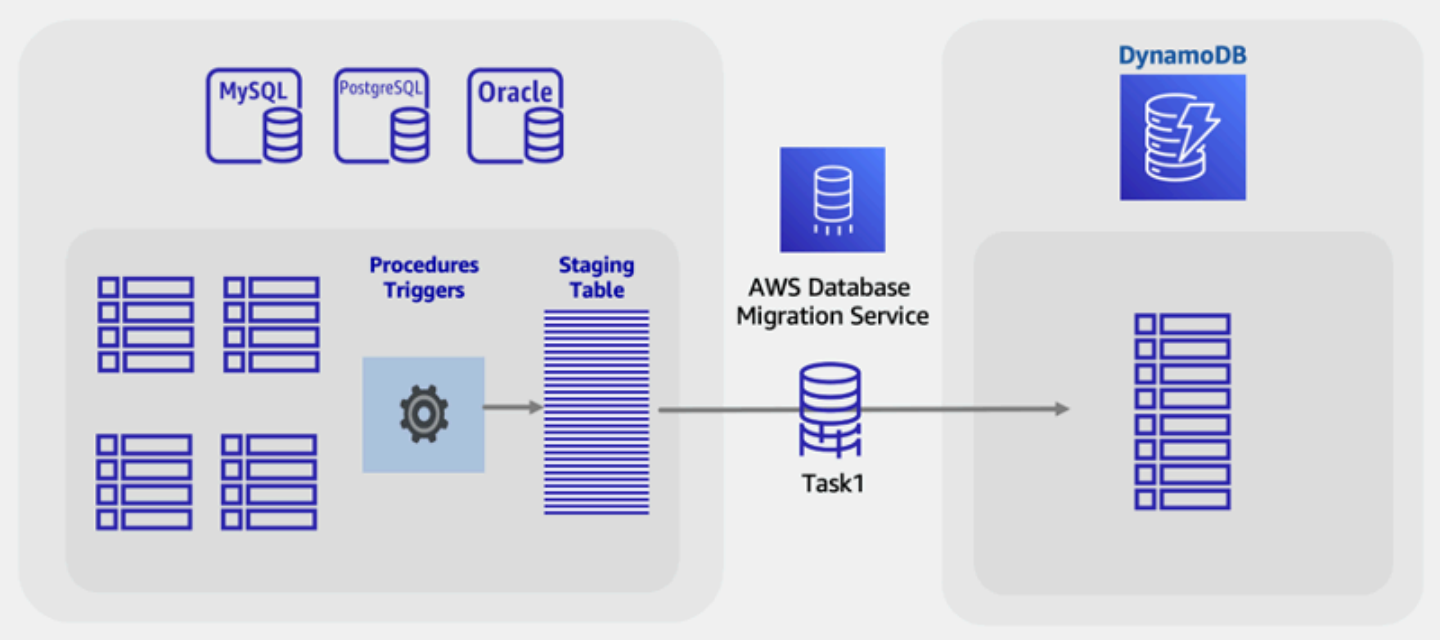
Esegui una migrazione online con una tabella intermedia SQL utilizzando AWS DMS

Strumenti

- Stored procedure o trigger SQL personalizzati
- AWS DMS

Fasi di migrazione online:

1. All'interno del motore di database relazionale di origine, assicurati che siano disponibili spazio su disco e capacità di elaborazione aggiuntivi.
2. Crea una nuova tabella di staging nel database SQL, con timestamp o funzionalità CDC abilitate.
3. Scrivi ed esegui una stored procedure per copiare i dati della tabella relazionale esistente nella tabella di staging.
4. Implementa i trigger o modifica le procedure esistenti per eseguire la doppia scrittura nella nuova tabella di staging mentre esegui le normali scritture sulle tabelle esistenti.
5. Esegui AWS DMS per migrare e sincronizzare questa tabella di origine con una tabella DynamoDB di destinazione



Questa guida ha presentato diverse considerazioni e approcci per la migrazione dei dati di database relazionali a DynamoDB, con particolare attenzione alla riduzione al minimo dei tempi di inattività e all'utilizzo di strumenti e tecniche di database comuni. Per ulteriori informazioni, consulta gli argomenti seguenti:

- AWS DMS Guida per l'utente
- AWS Glue Guida per l'utente
- Best practice per la migrazione da RDBMS a DynamoDB

NoSQL Workbench per DynamoDB

NoSQL Workbench per Amazon DynamoDB è un'applicazione GUI lato client multipiattaforma per operazioni e sviluppo di database moderni. È disponibile per Windows, macOS e Linux. NoSQL Workbench è uno strumento visuale di sviluppo che offre funzionalità di modellazione, visualizzazione dei dati, nonché funzionalità di sviluppo di query che consentono di progettare, creare, eseguire query e gestire le tabelle DynamoDB. NoSQL Workbench ora include la versione locale di DynamoDB come parte opzionale del processo di installazione, il che semplifica la modellazione dei dati nella versione locale di DynamoDB. Per ulteriori informazioni sulla versione locale di DynamoDB e i relativi requisiti, consulta Configurazione di DynamoDB locale (versione scaricabile) .

Modellazione dei dati

Grazie a NoSQL Workbench per DynamoDB, è possibile creare nuovi modelli di dati o progettare modelli in base ai modelli di dati esistenti che soddisfino i pattern di accesso ai dati delle applicazioni. Puoi anche importare ed esportare il modello di dati progettato alla fine del processo. Per ulteriori informazioni, consulta Creazione di modelli di dati con NoSQL Workbench.

Visualizzazione dei dati

Il visualizzatore del modello di dati fornisce un canvas in cui è possibile mappare query e visualizzare i pattern di accesso (facet) dell'applicazione senza dover scrivere codice. Ogni facet corrisponde a un pattern di accesso differente in DynamoDB. Puoi generare automaticamente dati di esempio da utilizzare nel modello di dati. Per ulteriori informazioni, consulta Visualizzazione dei modelli di accesso ai dati.

Creazione di operazioni

NoSQL Workbench offre una ricca intergaffia utente grafica per query di sviluppo e test. È possibile utilizzare Operation builder per visualizzare, esplorare ed eseguire query sui set di dati in tempo reale. Operation builder strutturato supporta espressioni di proiezione, espressioni condizionali e genera codice di esempio in più lingue. È possibile clonare direttamente le tabelle da un account Amazon DynamoDB a un altro in Regioni diverse. È anche possibile clonare direttamente le tabelle tra DynamoDB locale e un account Amazon DynamoDB per copiare più rapidamente lo schema delle chiavi della tabella (e facoltativamente lo schema e gli elementi degli indici secondari globali (GSI)) tra gli ambienti di sviluppo. Per ulteriori informazioni, consulta Esplorazione di set di dati e creazione di operazioni con NoSQL Workbench.

Il video seguente descrive in dettaglio i concetti di modellazione dei dati con NoSQL Workbench.

Argomenti

- Download di NoSQL Workbench per DynamoDB
- Installazione di NoSQL Workbench per DynamoDB
- Creazione di modelli di dati con NoSQL Workbench
- Visualizzazione dei modelli di accesso ai dati
- Esplorazione di set di dati e creazione di operazioni con NoSQL Workbench
- Modelli di dati di esempio per NoSQL Workbench
- Cronologia delle versioni di NoSQL Workbench

Download di NoSQL Workbench per DynamoDB

Segui queste istruzioni per scaricare e installare NoSQL Workbench e la versione locale di DynamoDB* per Amazon DynamoDB.

Prerequisiti

Per le installazioni di Ubuntu sono necessari due software prerequisiti: libfuse2 e curl.

libfuse2

A partire da Ubuntu 22.04, libfuse2 non è più installato per impostazione predefinita. Per risolvere questo problema, esegui `sudo add-apt-repository universe && sudo apt install libfuse2` per installare la versione più recente di Ubuntu.

curl

Per aggiornare Ubuntu esegui `sudo apt update && sudo apt upgrade`

Quindi, per installare cURL, esegui: `sudo apt install curl`

Per scaricare NoSQL Workbench e la versione locale di DynamoDB

1. Scaricare la versione appropriata di NoSQL Workbench per il sistema operativo in uso.

Sistema operativo	Collegamento per il download
macOS (Intel)**	Scarica per macOS (Intel)

Sistema operativo	Collegamento per il download
macOS (Apple SIlicon)	Scarica per macOS (Apple SIlicon)
Windows	Download per Windows
Linux***	Scarica per Linux

* NoSQL Workbench include la versione locale di DynamoDB come parte opzionale del processo di installazione.

** Se tentando di aprire NoSQL Workbench compare un messaggio di avviso che indica che l'app non è registrata presso Apple da uno sviluppatore identificato, procedi come segue:

1. Individua l'app e poi aprila.
2. Fai clic sull'icona dell'app tenendo premuto il tasto Ctrl, quindi seleziona Apri dal menu di scelta rapida.

   In questo modo l'app viene salvata come eccezione alle impostazioni di sicurezza. Apri l'app facendo doppio clic su di essa come con l'apertura di qualsiasi app registrata.

*** NoSQL Workbench supporta Ubuntu 12.04, Fedora 21 e Debian 8 o qualsiasi versione più recente di queste distribuzioni Linux.

2. Avvia l'applicazione scaricata e segui i passaggi indicati in Installazione di NoSQL Workbench.

> **Note**
>
> Per eseguire DynamoDB locale, è necessario disporre della versione 11.x o successiva di Ambiente di runtime Java (JRE).

Installazione di NoSQL Workbench per DynamoDB

Segui questi passaggi per installare NoSQL Workbench e la versione locale di DynamoDB su una piattaforma supportata.

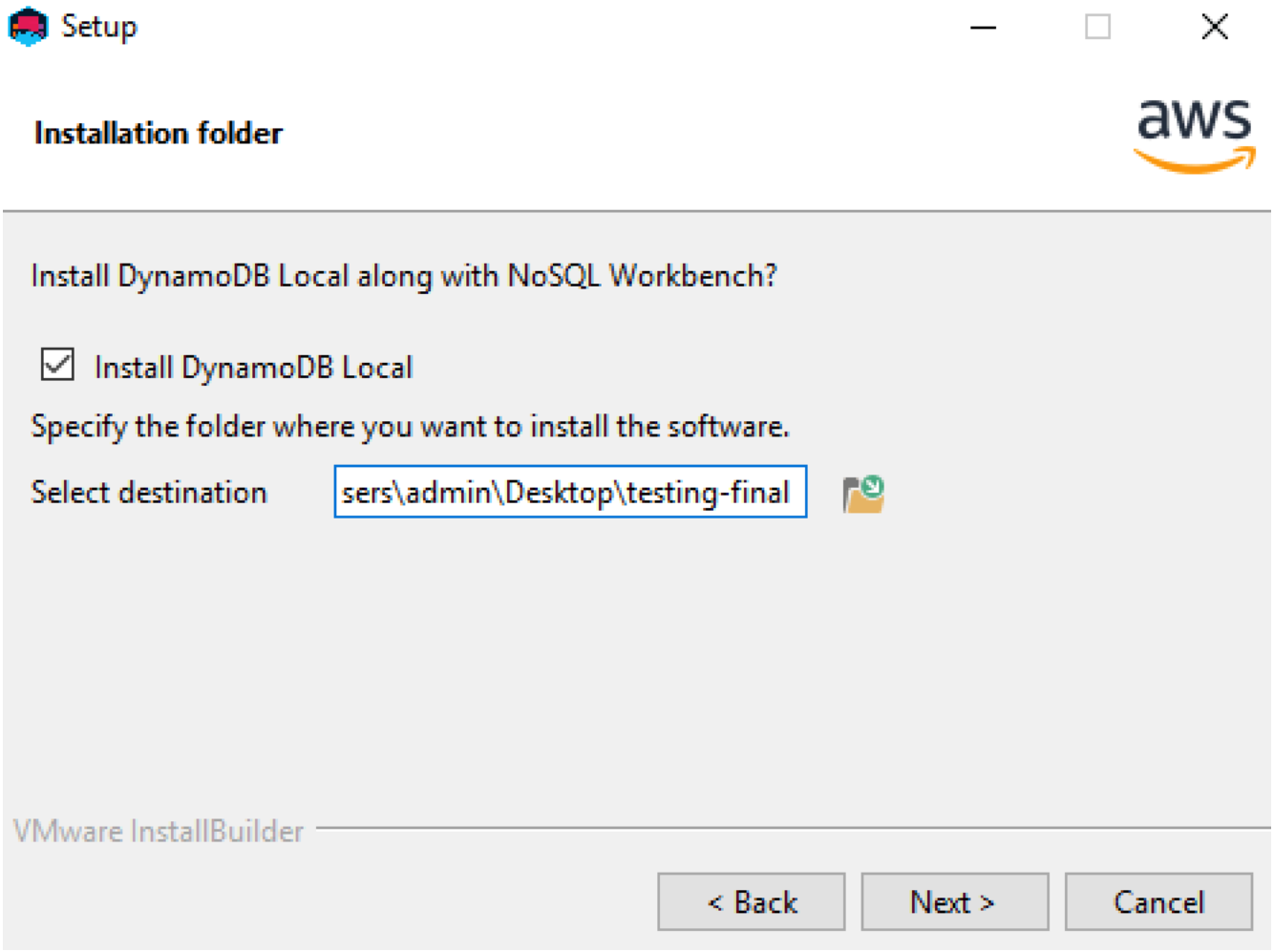
Windows

Per installare NoSQL Workbench su Windows

1. Esegui l'applicazione di installazione NoSQL Workbench e scegli la lingua di configurazione. Quindi scegli OK per iniziare la configurazione. Per ulteriori informazioni sul download di NoSQL Workbench, consulta Download di NoSQL Workbench per DynamoDB.
2. Scegli Next (Avanti) per continuare la configurazione, quindi scegli Next (Avanti) nella schermata successiva.
3. Per impostazione predefinita, la casella di controllo Installa versione locale di DynamoDB è selezionata per includere la versione locale di DynamoDB come parte dell'installazione. Se si mantiene questa opzione selezionata, la versione locale di DynamoDB verrà installata e il percorso di destinazione sarà identico a quello di NoSQL Workbench. Se si deseleziona la casella di controllo per questa opzione, l'installazione della versione locale di DynamoDB verrà ignorata e il percorso di installazione sarà solo per NoSQL Workbench.

   Scegli la destinazione in cui installare il software e scegli Next (Avanti).

   > **Note**
   >
   > Se hai scelto di non includere DynamoDB locale come parte della configurazione, deseleziona la casella di spunta Installa DynamoDB locale, seleziona Successivo e vai alla fase 6. Puoi scaricare la versione locale di DynamoDB separatamente come installazione autonoma in un secondo momento. Per ulteriori informazioni, consulta Configurazione di DynamoDB locale (versione scaricabile) .



4. Scegli il numero di porta da usare per la versione locale di DynamoDB. La porta predefinita è 8000. Dopo aver inserito il numero di porta, scegli Next (Avanti).
5. Scegli Next (Avanti) per iniziare la configurazione.
6. Una volta completata la configurazione, scegli Finish (Fine) per chiudere la schermata di configurazione.
7. Apri l'applicazione nel percorso di installazione, ad esempio /programs/DynamoDBWorkbench/.

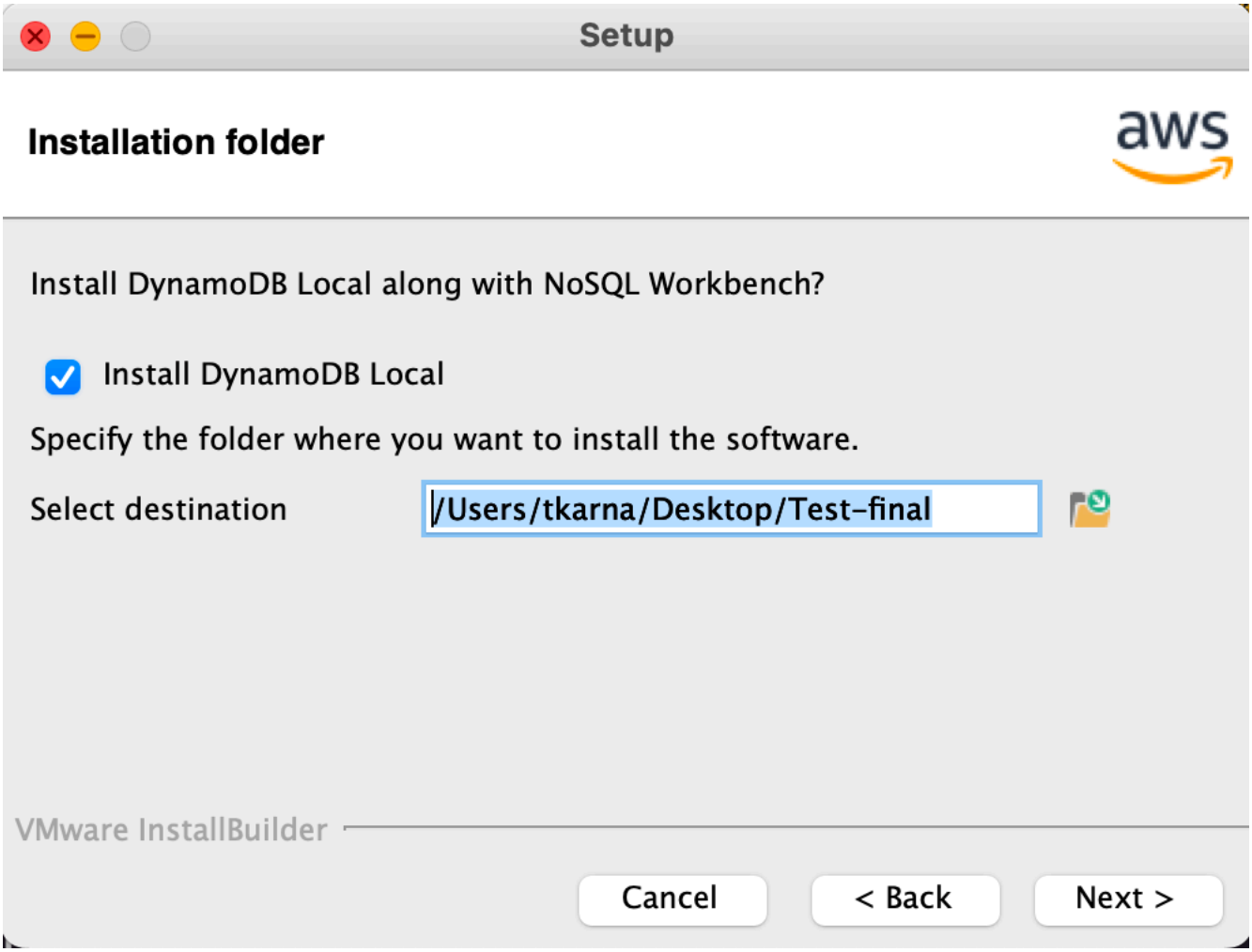
macOS

Per installare NoSQL Workbench su macOS

1. Esegui l'applicazione di installazione NoSQL Workbench e scegli la lingua di configurazione. Quindi scegli OK per iniziare la configurazione. Per ulteriori informazioni sul download di NoSQL Workbench, consulta Download di NoSQL Workbench per DynamoDB.
2. Scegli Next (Avanti) per continuare la configurazione, quindi scegli Next (Avanti) nella schermata successiva.
3. Per impostazione predefinita, la casella di controllo Installa versione locale di DynamoDB è selezionata per includere la versione locale di DynamoDB come parte dell'installazione. Se si mantiene questa opzione selezionata, la versione locale di DynamoDB verrà installata e il percorso di destinazione sarà identico a quello di NoSQL Workbench. Se si deseleziona questa opzione, l'installazione della versione locale di DynamoDB verrà ignorata e il percorso di installazione sarà solo per NoSQL Workbench.

   Scegli la destinazione in cui installare il software e scegli Next (Avanti).

   > **Note**
   >
   > Se hai scelto di non includere DynamoDB locale come parte della configurazione, deseleziona la casella di spunta Installa DynamoDB locale, seleziona Successivo e vai alla fase 6. Puoi scaricare la versione locale di DynamoDB separatamente come installazione autonoma in un secondo momento. Per ulteriori informazioni, consulta Configurazione di DynamoDB locale (versione scaricabile) .



4. Scegli il numero di porta da usare per la versione locale di DynamoDB. La porta predefinita è 8000. Dopo aver inserito il numero di porta, scegli Next (Avanti).
5. Scegli Next (Avanti) per iniziare la configurazione.
6. Una volta completata la configurazione, scegli Finish (Fine) per chiudere la schermata di configurazione.
7. Apri l'applicazione nel percorso di installazione, ad esempio /Applications/DynamoDBWorkbench/.

> **Note**
>
> NoSQL Workbench per macOS esegue aggiornamenti automatici. Per ricevere notifiche sugli aggiornamenti, abilita l'accesso alle notifiche a NoSQL Workbench in Preferenze di Sistema > Notifiche.

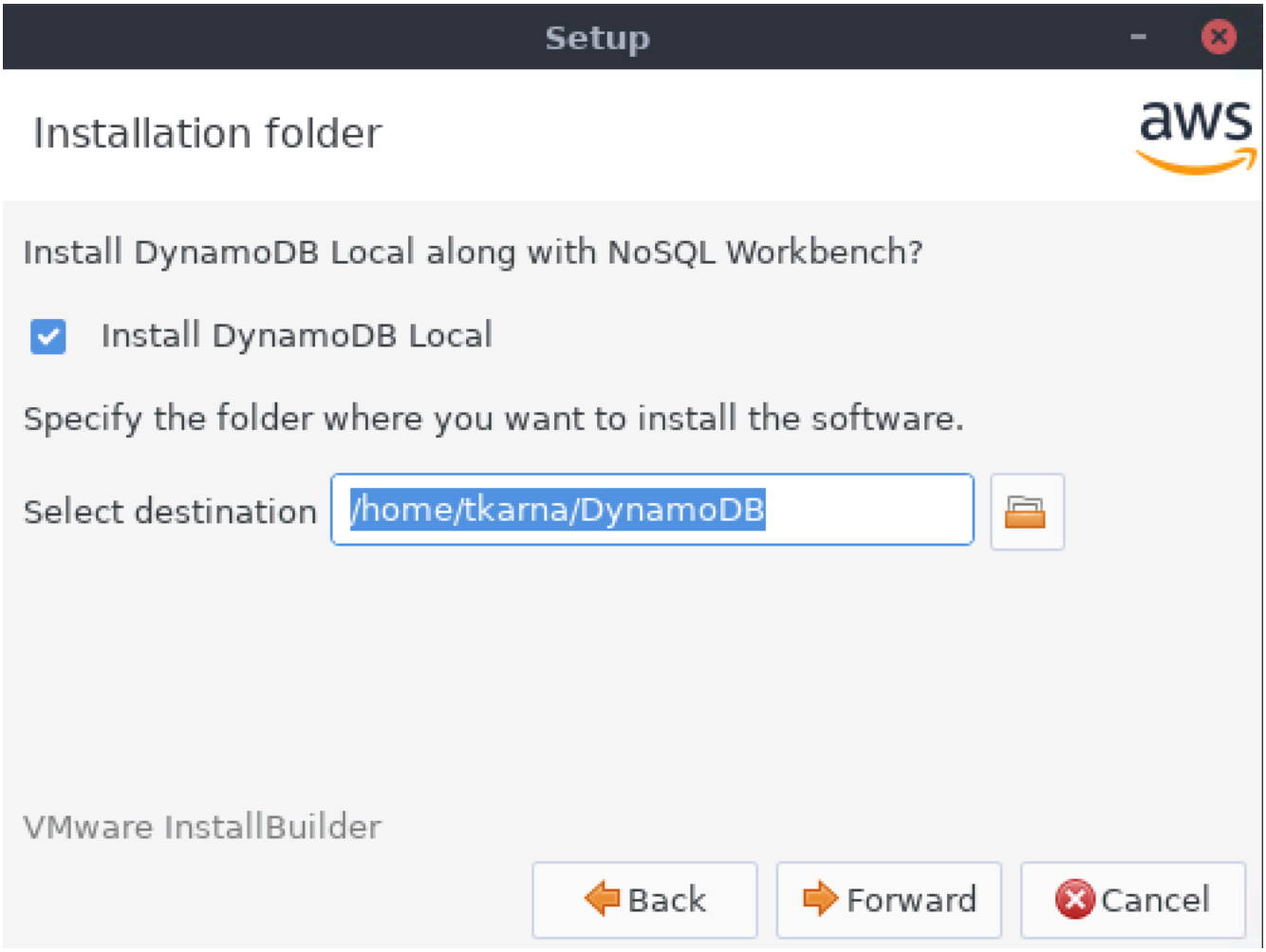
Linux

Per installare NoSQL Workbench su Linux

1. Esegui l'applicazione di installazione NoSQL Workbench e scegli la lingua di configurazione. Quindi scegli OK per iniziare la configurazione. Per ulteriori informazioni sul download di NoSQL Workbench, consulta Download di NoSQL Workbench per DynamoDB.
2. Scegli Forward (Avanti) per continuare la configurazione e scegli Forward (Avanti) nella schermata successiva.
3. Per impostazione predefinita, la casella di controllo Installa versione locale di DynamoDB è selezionata per includere la versione locale di DynamoDB come parte dell'installazione. Se si mantiene questa opzione selezionata, la versione locale di DynamoDB verrà installata e il percorso di destinazione sarà identico a quello di NoSQL Workbench. Se si deseleziona questa opzione, l'installazione della versione locale di DynamoDB verrà ignorata e il percorso di installazione sarà solo per NoSQL Workbench.

   Scegli la destinazione in cui desideri installare il software e scegli Forward (Avanti).

   > **Note**
   >
   > Se hai scelto di non includere DynamoDB locale come parte della configurazione, deseleziona la casella di spunta Installa DynamoDB locale, seleziona Successivo e vai alla fase 6. Puoi scaricare la versione locale di DynamoDB separatamente come installazione autonoma in un secondo momento. Per ulteriori informazioni, consulta Configurazione di DynamoDB locale (versione scaricabile) .



4. Scegli il numero di porta da usare per la versione locale di DynamoDB. La porta predefinita è 8000. Dopo aver inserito il numero di porta inserito, scegli Forward (Avanti).
5. Scegli Forward (Avanti) per iniziare la configurazione.
6. Una volta completata la configurazione, scegli Finish (Fine) per chiudere la schermata di configurazione.
7. Apri l'applicazione nel percorso di installazione, ad esempio /usr/local/programs/DynamoDBWorkbench/.

Come avviare un'AppImage su Linux

1. Rendi eseguibile il file AppImage:

   chmod +x noSQL-workbench-linux.AppImage

Sostituisci noSQL-workbench-linux.AppImage con il nome del file effettivo dell'AppImage scaricata.

2. Esegui l'AppImage:

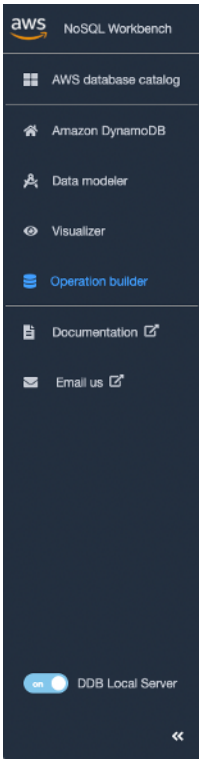
   ./noSQL-workbench-linux.AppImage

   Questo avvierà l'applicazione NoSQL Workbench.

> **Note**
>
> A seconda della distribuzione Linux in uso, potrebbe essere necessario installare dipendenze aggiuntive per il corretto funzionamento dell'AppImage. In caso di problemi, consulta la documentazione fornita dagli sviluppatori di AppImage o chiedi assistenza alla community.

> **Note**
>
> Se hai scelto di installare la versione locale di DynamoDB come parte dell'installazione di NoSQL Workbench, la versione locale di DynamoDB sarà preconfigurata con le opzioni predefinite. Per modificare le opzioni predefinite, modificare lo script DDBLocalStart che si trova nella directory /resources/DDBLocal_Scripts/. Puoi trovarlo nel percorso che hai fornito durante l'installazione. Per ulteriori informazioni sulle opzioni della versione locale di DynamoDB, consultare Note per l'utilizzo locale di DynamoDB .

Se hai scelto di installare la versione locale di DynamoDB come parte dell'installazione di NoSQL Workbench, avrai accesso a un interruttore per abilitare/disabilitare la versione locale di DynamoDB Local come mostrato nell'immagine seguente.



Creazione di modelli di dati con NoSQL Workbench

È possibile utilizzare lo strumento data modeler in NoSQL Workbench per Amazon DynamoDB per creare nuovi modelli di dati o per progettare modelli in base ai modelli di dati esistenti che soddisfino i pattern di accesso ai dati delle applicazioni. Il data modeler include alcuni esempi di modelli di dati che ti aiutano a iniziare.

Argomenti

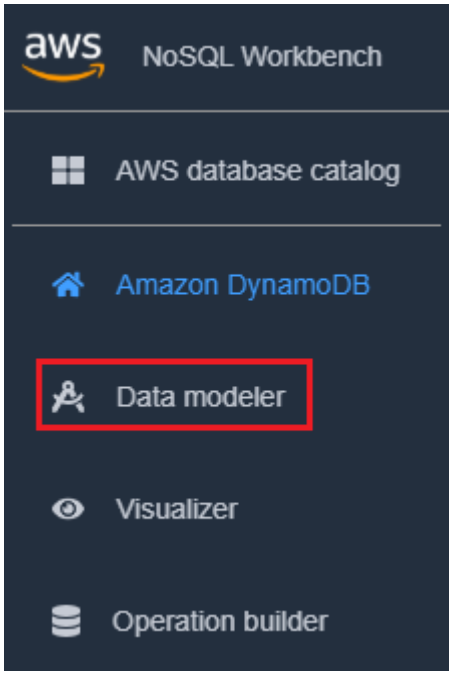
- Creazione di un nuovo modello di dati
- Importazione di un modello di dati esistente
- Esportazione di un modello di dati
- Modifica di un modello di dati esistente

Creazione di un nuovo modello di dati

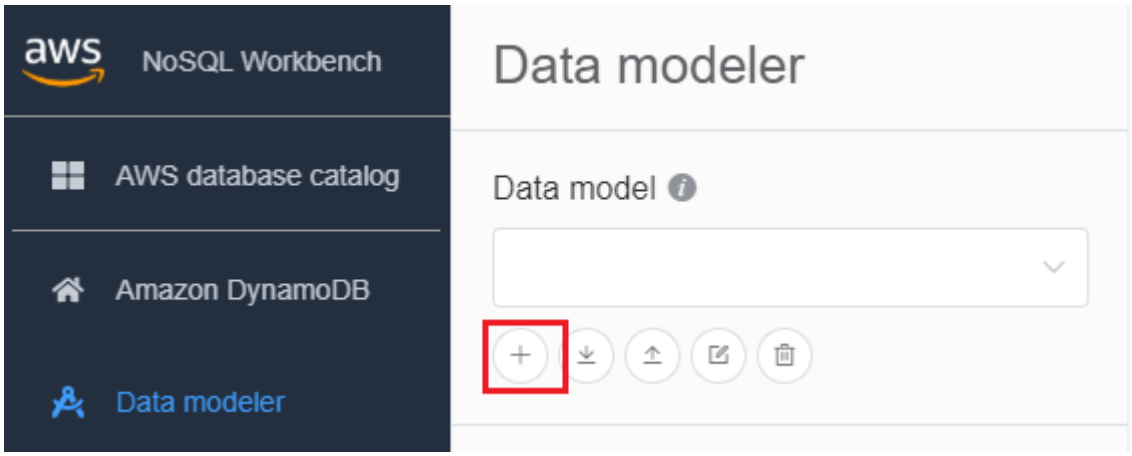
Per creare un nuovo modello di dati in Amazon DynamoDB utilizzando NoSQL Workbench, seguire queste istruzioni.

Per creare un nuovo modello di dati

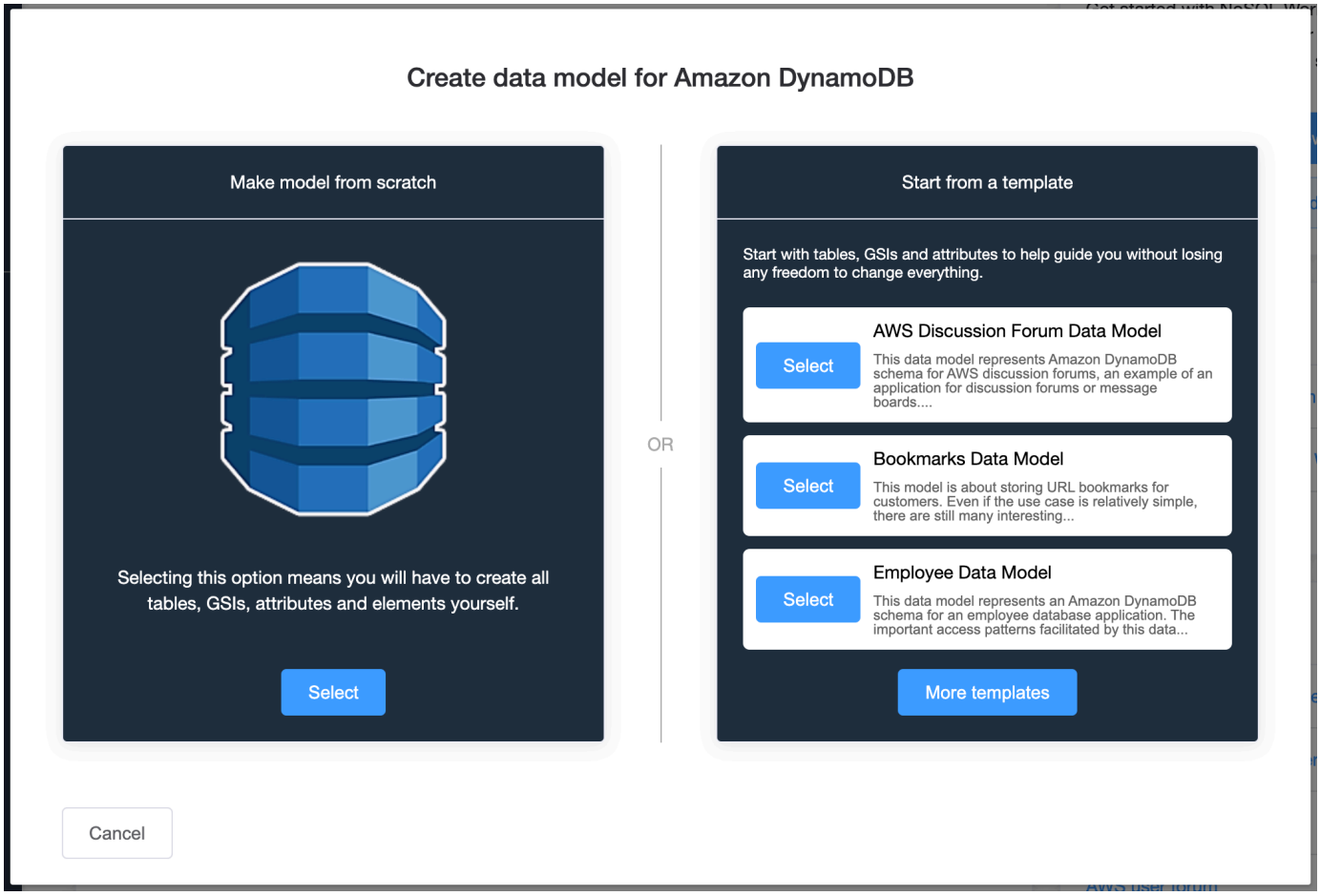
1. Aprire NoSQL Workbench e, nel pannello di navigazione a sinistra, seleziona l'icona Data modeler.



2. Seleziona Crea origine dati.

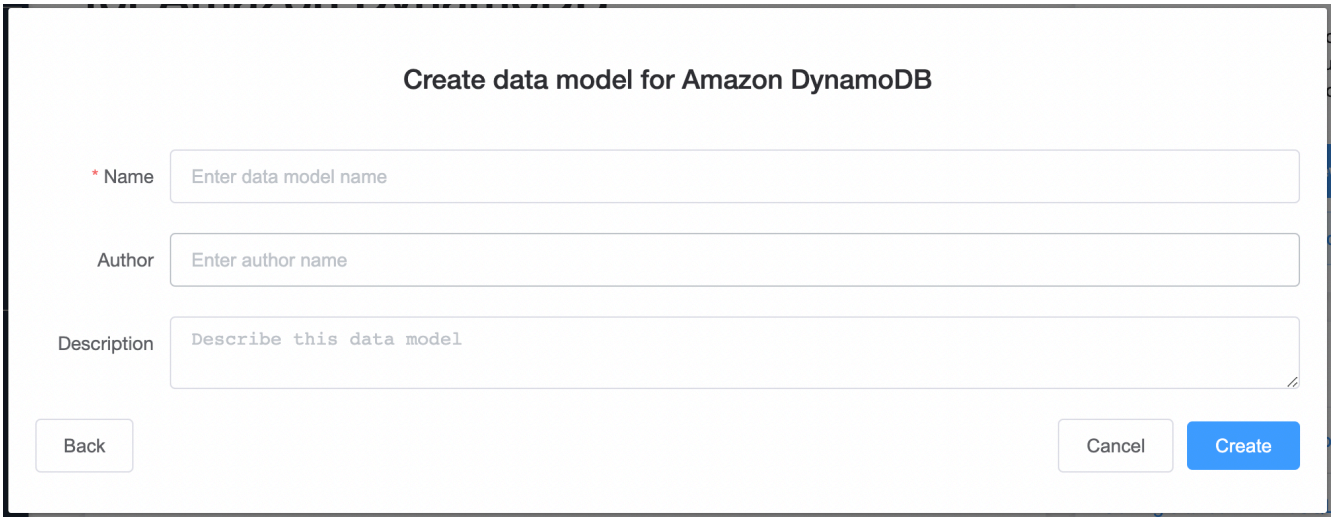


Per Create data model (Crea modello di dati) è possibile scegliere Make model from scratch (Crea un modello da zero) e Start from a template (Inizia da un modello).



Make model from scratch

Per creare un modello da zero, immettere un nome, un autore e una descrizione per il modello di dati. Al termine, scegli Create (Crea).



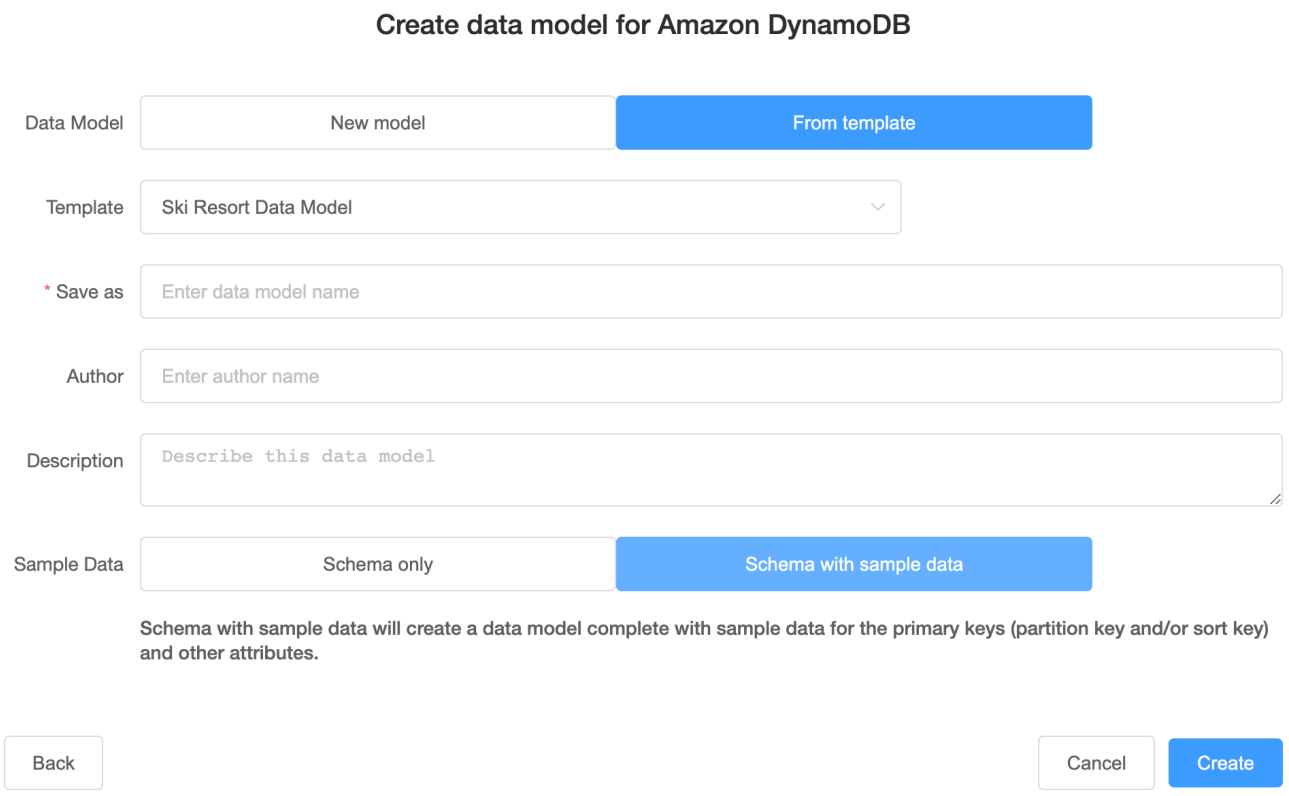
Start from a template

Partire da un modello consente di scegliere un modello di esempio per iniziare. Scegli More templates (Altri modelli) per visualizzare altre opzioni di modello. Scegli Select (Seleziona) per il modello da utilizzare.

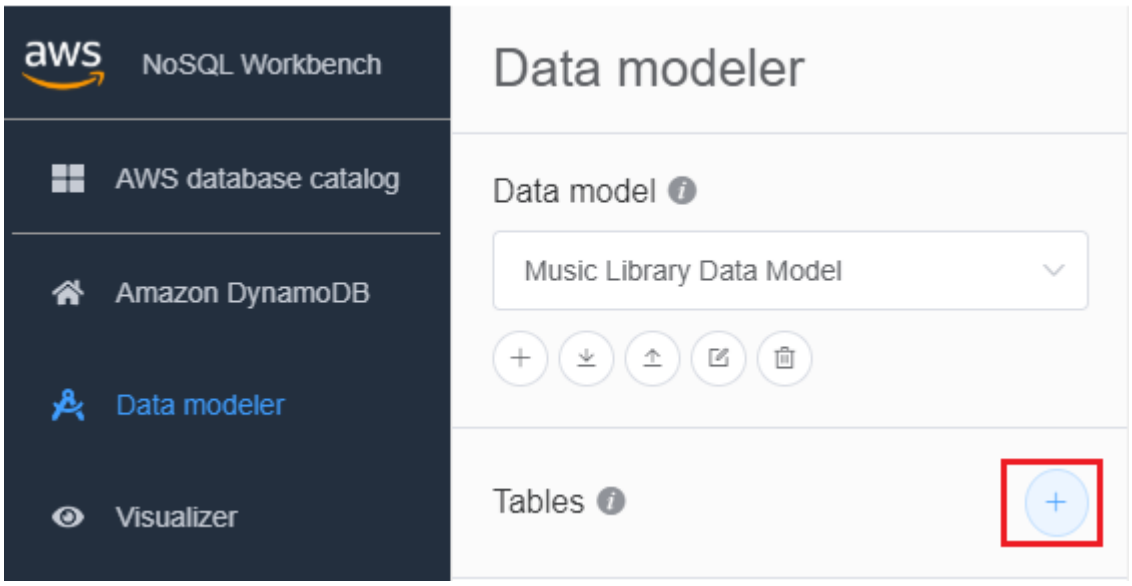
Inserire il nome, l'autore e la descrizione per il modello selezionato. È possibile scegliere tra Schema Only (Solo schema) e Schema with sample data (Schema con dati di esempio).

- Schema only (Solo schema) crea un modello di dati vuoto con la chiave primaria (chiave di partizione e chiave di ordinamento) e altri attributi.
- Schema with sample data (Schema con dati di esempio) creerà un modello di dati completo con dati di esempio per la chiave primaria (chiave partizione e chiave di ordinamento) e altri attributi.

Quando queste informazioni sono complete, scegli Create (Crea) per creare il modello.



3. Con il modello creato, scegli Add table (Aggiungi tabella).



Per ulteriori informazioni sulle tabelle, consulta Utilizzo delle tabelle in DynamoDB.

4. Specificare le impostazioni seguenti:

   - Table name (Nome tabella): inserire un nome univoco per la tabella.
   - Chiave di partizione: immetti il nome di una chiave di partizione e specificane il tipo. Facoltativamente, puoi anche selezionare un formato di tipo di dati più granulare per la generazione di dati di esempio.
   - Per aggiungere una chiave di ordinamento:
     1. Scegli Add sort key (Aggiungi chiave ordinamento).
     2. Specifica il nome della chiave di ordinamento e il suo tipo. Facoltativamente, puoi selezionare un formato di tipo di dati più granulare per la generazione di dati di esempio.

> **Note**
>
> Per ulteriori informazioni sulla progettazione della chiave primaria, sulla progettazione e sull'uso efficace delle chiavi di partizione e sull'uso delle chiavi di ordinamento, consulta quanto segue:
>
> - Chiave primaria
> - Best practice per la progettazione delle chiavi di partizione e relativo uso in DynamoDB
> - Best practice per l'uso delle chiavi di ordinamento per organizzare i dati in DynamoDB

5. Per aggiungere altri attributi, svolgere quanto segue per ogni attributo:

   1. Seleziona Aggiungi attributo.
   2. Specifica il nome e il tipo attributo. Facoltativamente, puoi selezionare un formato di tipo di dati più granulare per la generazione di dati di esempio.

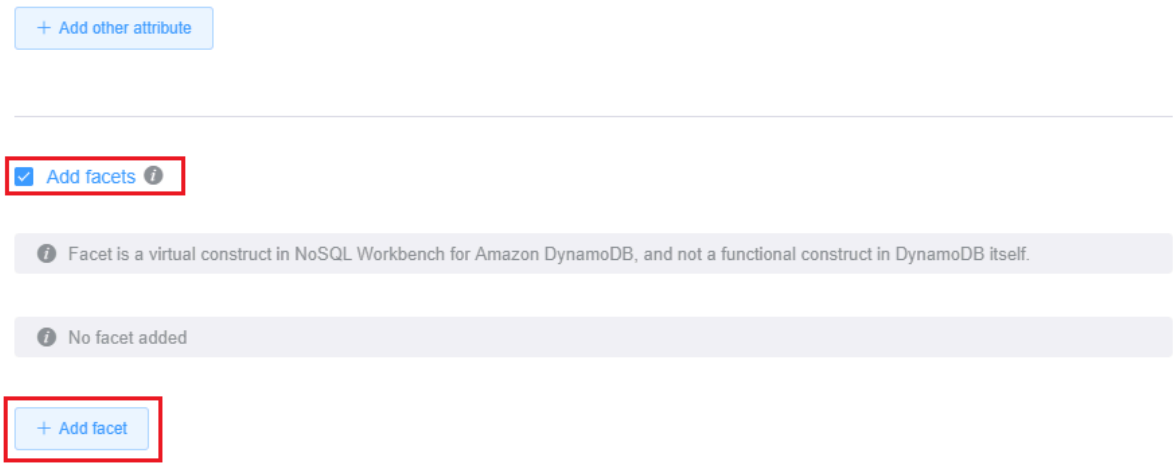
6. Aggiungi un facet:

   È inoltre possibile aggiungere un facet. Un facet è un costrutto virtuale in NoSQL Workbench. Non è un costrutto funzionale in DynamoDB stesso.

   > **Note**
   >
   > I facet in NoSQL Workbench ti aiutano a visualizzare modelli di accesso ai dati differenti di un'applicazione per Amazon DynamoDB con solo un sottoinsieme di dati in una tabella. Per ulteriori informazioni sui facet, consulta Visualizzazione dei pattern di accesso ai dati.

   Per aggiungere un facet,

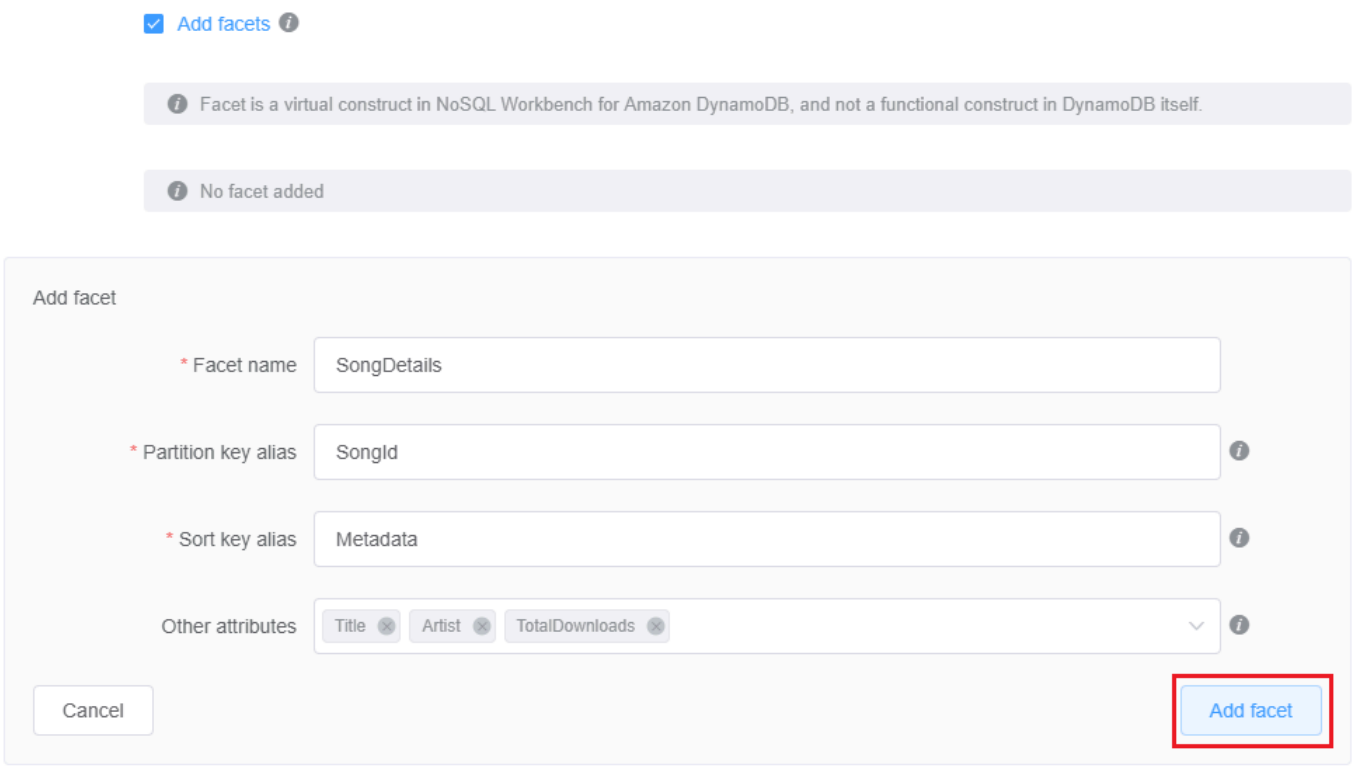
   - Seleziona Aggiungi facet.
   - Scegliere Add Facet (Aggiungi facet).



   - Specificare le impostazioni seguenti:
     - Il Facet name (Nome facet).
     - Un Alias della chiave di partizione per aiutare a distinguere questa vista del facet.

- Un Sort key alias (Alias della chiave di ordinamento).
- Scegli gli altri attributi che fanno parte di questo facet.

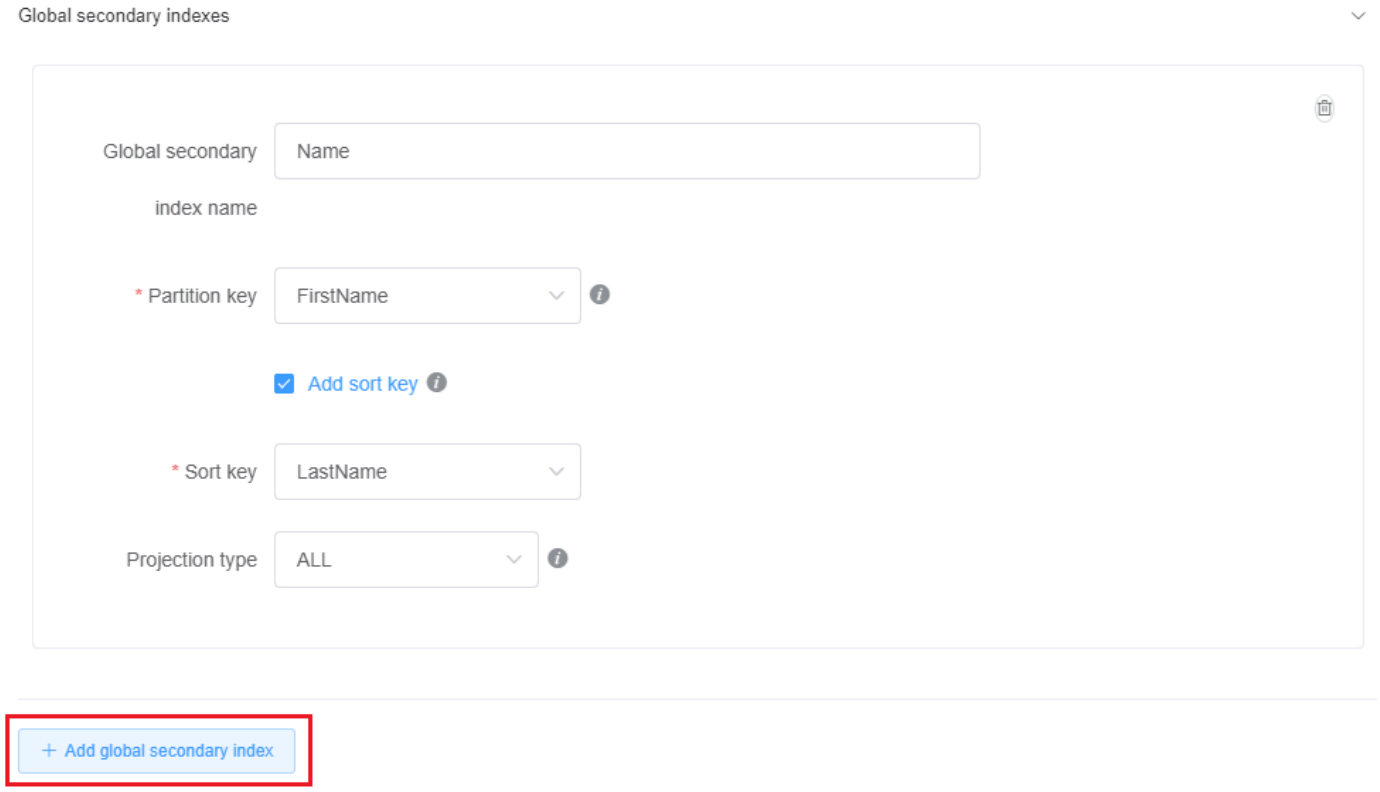
Scegliere Add Facet (Aggiungi facet).



Ripeti questo passaggio se desideri aggiungere altre sfaccettature.

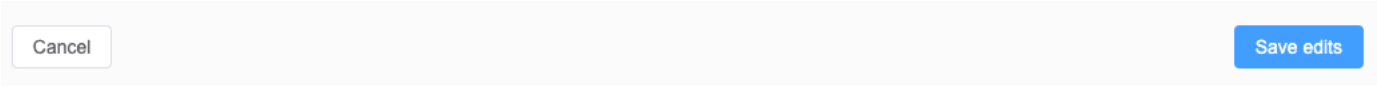
7. Per aggiungere un indice globale generale, seleziona Aggiungi indice secondario globale.

   Specifica il Nome indice secondario globale, l'attributo Chiave di partizione e Tipo di proiezione.



Per ulteriori informazioni sull'utilizzo di indici secondari globali in DynamoDB, consulta Indici secondari globali.

8. Salva le modifiche alle impostazioni della tabella.



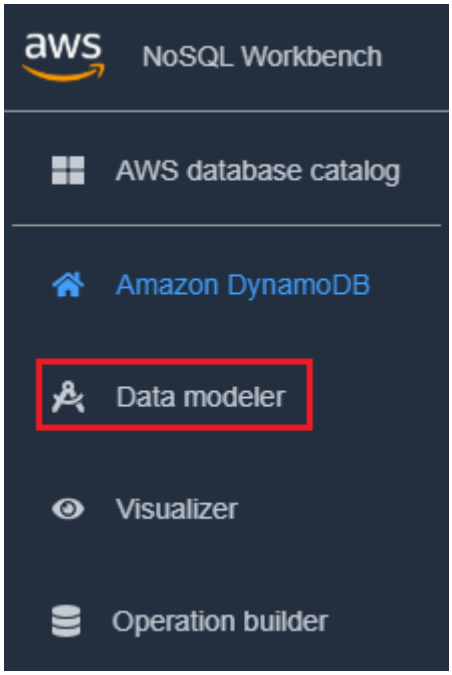
Per ulteriori informazioni sull'operazione API `CreateTable`, consulta CreateTable nella Documentazione di riferimento alle API di Amazon DynamoDB.

Importazione di un modello di dati esistente

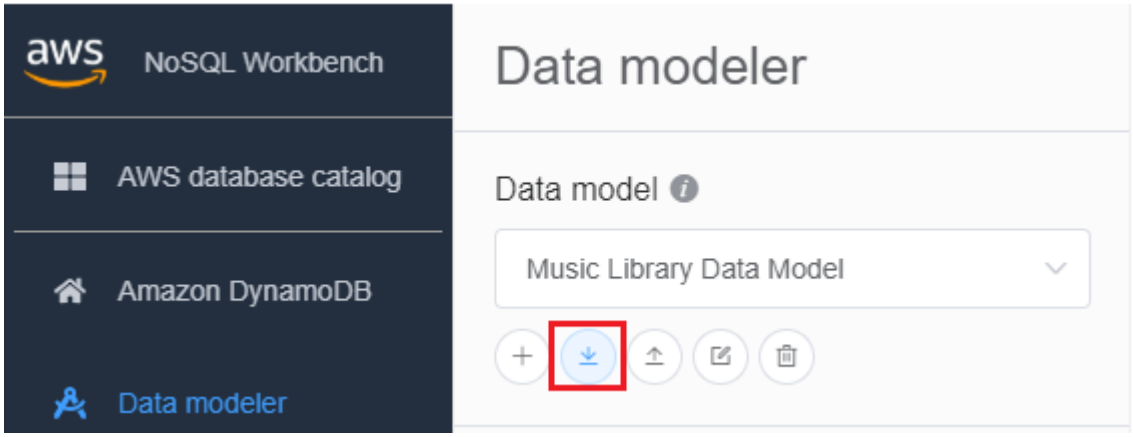
Ora è possibile utilizzare NoSQL Workbench per Amazon DynamoDB per creare un modello di dati importando e modificando un modello esistente. Adesso è possibile importare i modelli di dati in formato modello NoSQL Workbench o in Formato modello AWS CloudFormation JSON.

Per importare un modello di dati

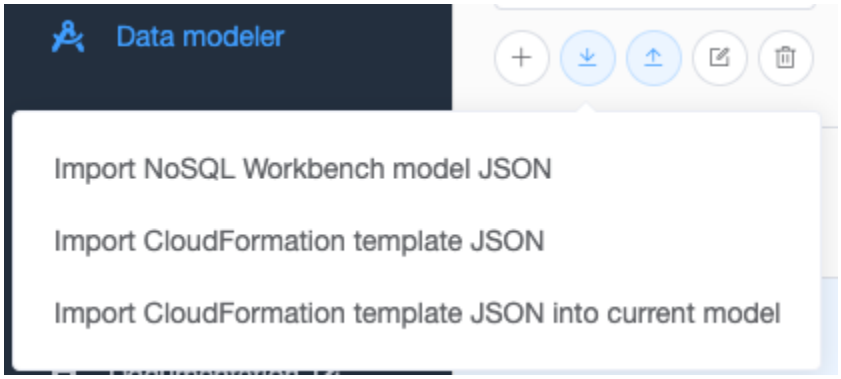
1. In NoSQL Workbench, nel riquadro di navigazione a sinistra, scegliere l'icona Data modeler.



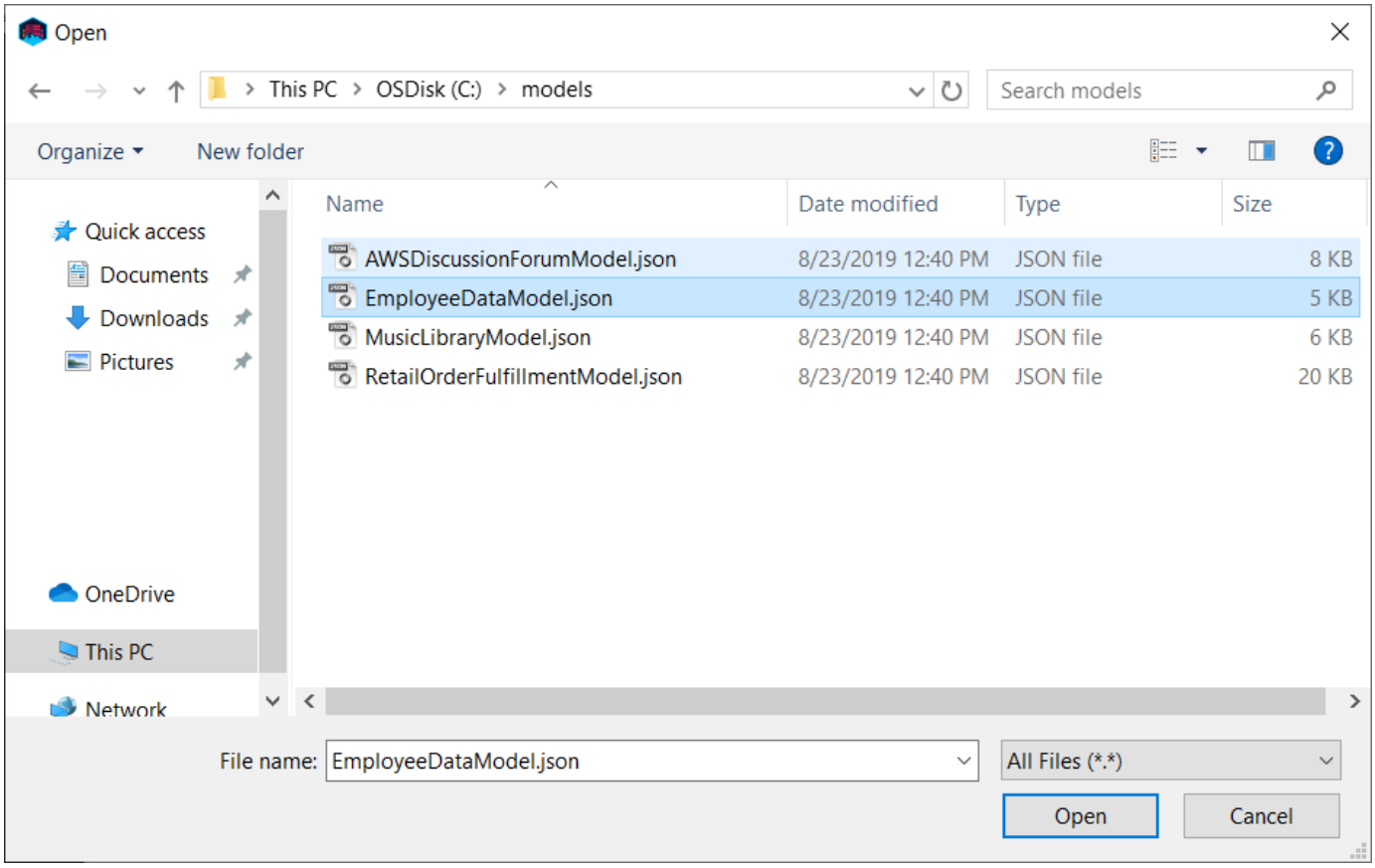
2. Passaggio con il puntatore su Import data model (Importa modello di dati).



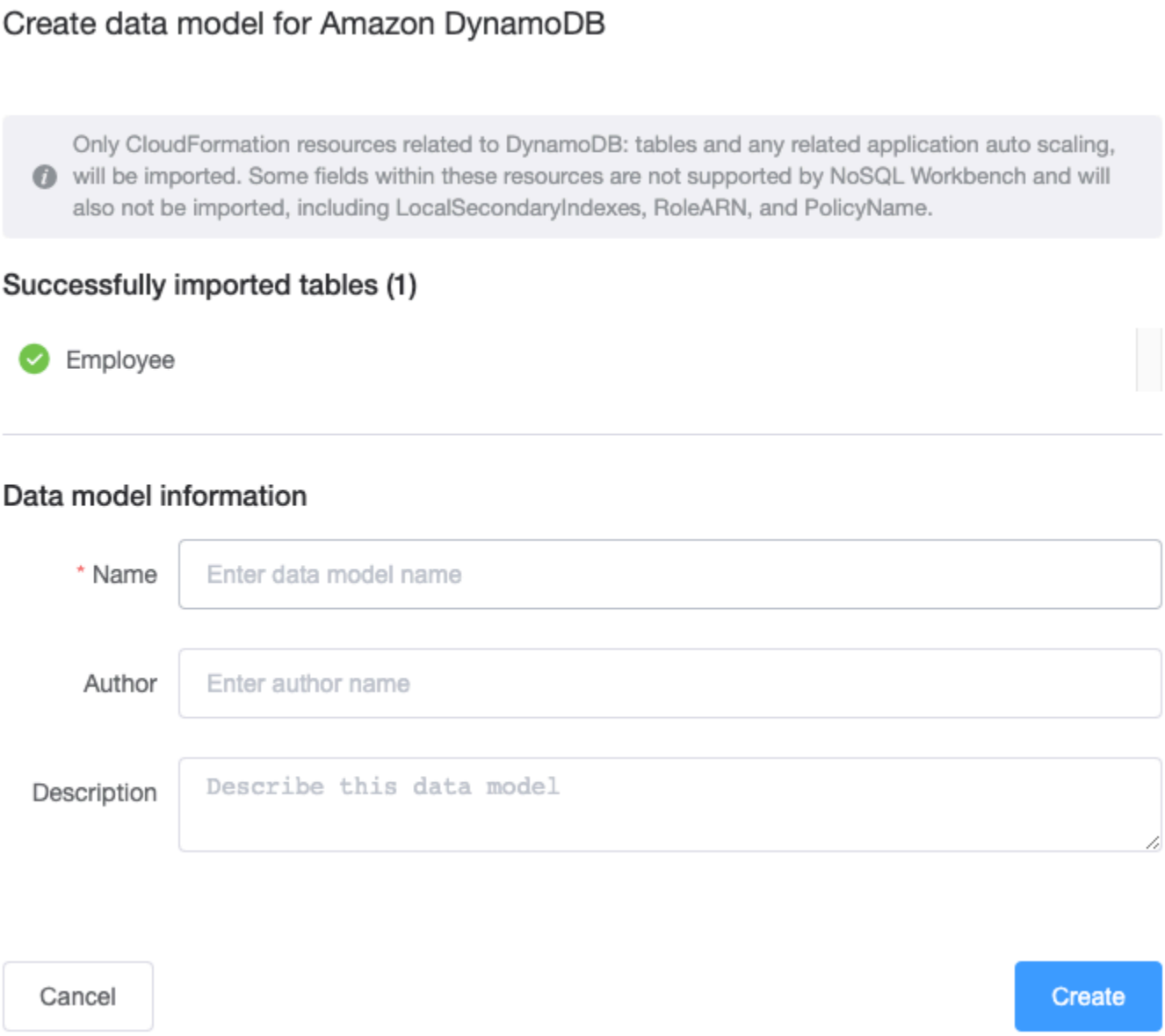
Nell'elenco a discesa, scegli se il modello che desideri importare è in formato modello NoSQL Workbench o in formato modello CloudFormation JSON. Se si dispone di un modello di dati esistente aperto in NoSQL Workbench, si avrà la possibilità di importare un modello CloudFormation nel modello corrente.



3. Seleziona un modello da importare.



4. Se il modello da importare è in formato modello CloudFormation, verrà visualizzato un elenco di tabelle da importare e sarà possibile specificare un nome, un autore e una descrizione del modello dati.

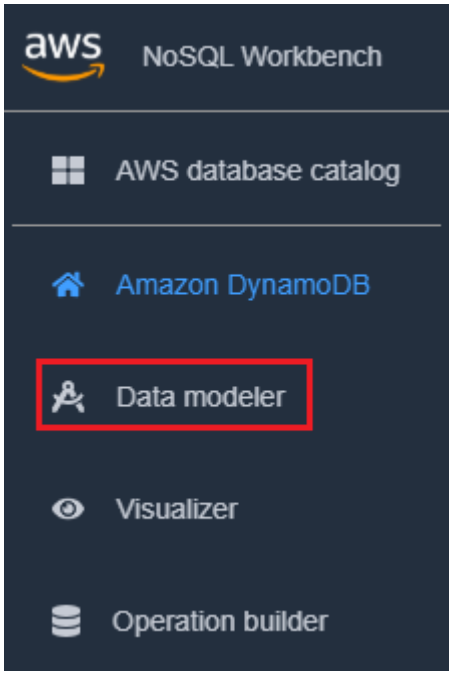


Esportazione di un modello di dati

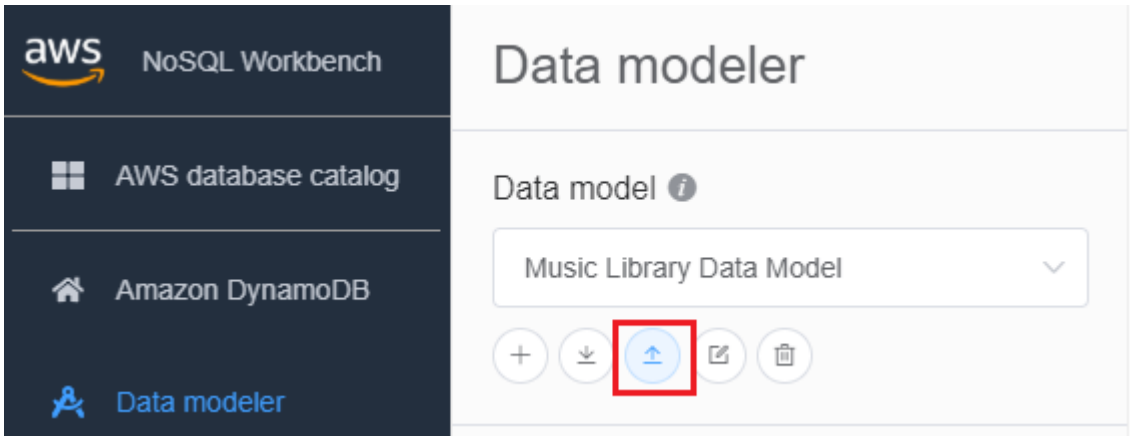
Dopo aver creato un modello utilizzando NoSQL Workbench per Amazon DynamoDB, è possibile salvare ed esportare il modello in formato NoSQL Workbench oppure in Formato modello AWS CloudFormation JSON.

Per esportare un modello di dati

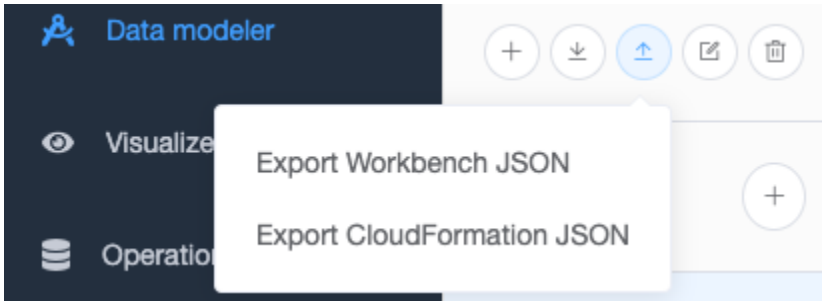
1. In NoSQL Workbench, nel riquadro di navigazione a sinistra, scegliere l'icona Data modeler.



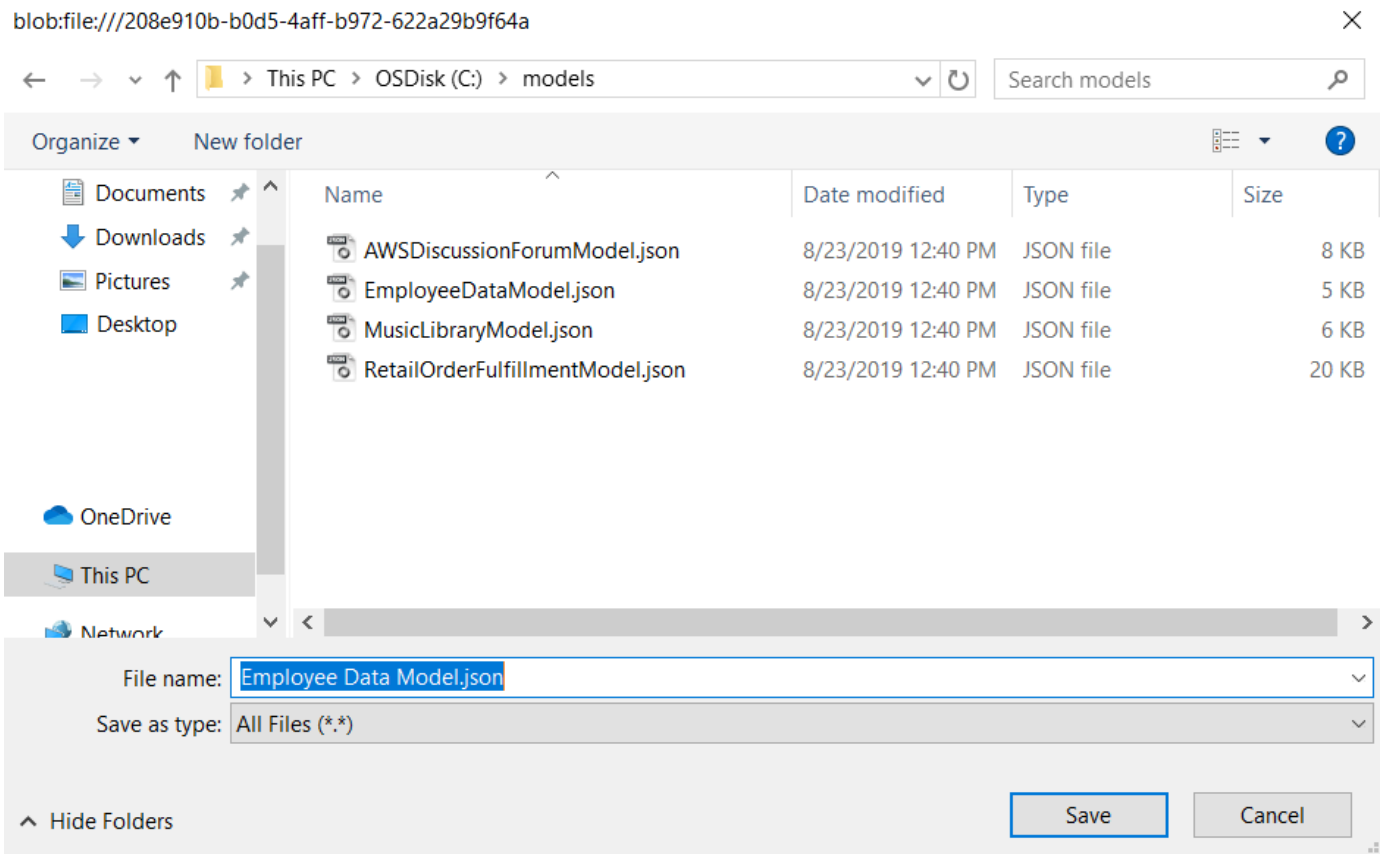
2. Passaggio con il puntatore su Export data model (Esporta modello di dati).



Nell'elenco a discesa, scegliere se esportare il modello di dati in formato modello NoSQL Workbench o in formato modello CloudFormation JSON.



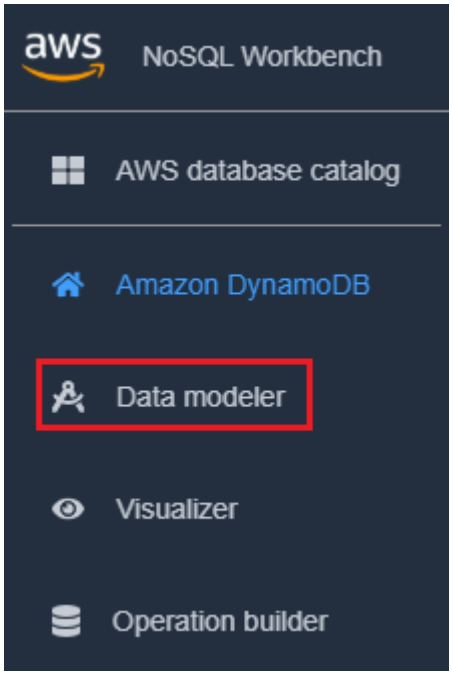
3. Seleziona una posizione in cui salvare il modello.



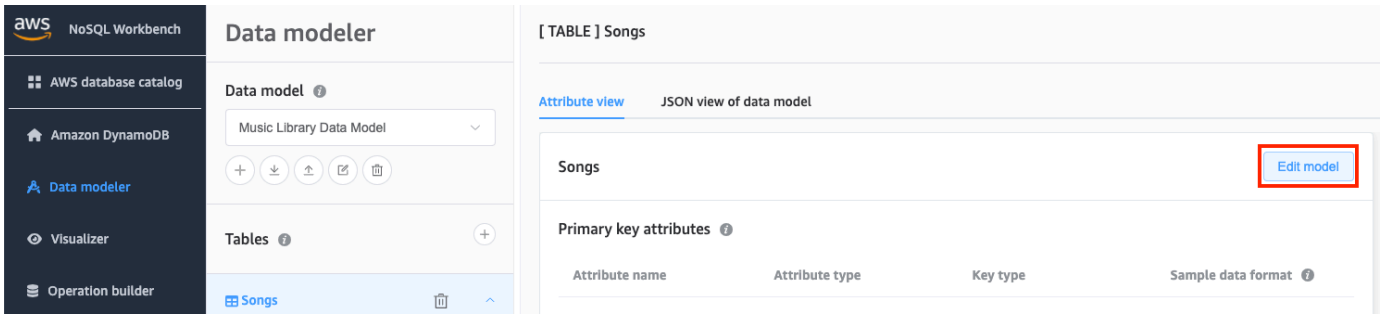
Modifica di un modello di dati esistente

Per modificare un modello esistente

1. In NoSQL Workbench, nel riquadro di navigazione a sinistra, seleziona il pulsante Data modeler.



2. Seleziona il modello di dati e scegli la tabella che desideri modificare. Scegli Modifica modello



3. Apporta le modifiche necessarie, quindi seleziona Salva modifiche.

Per modificare manualmente un modello esistente e aggiungere un facet

1. Esportare il modello Per ulteriori informazioni, consulta Esportazione di un modello di dati.
2. Aprire il file esportato in un editor.
3. Individuare l'oggetto `DataModel` per la tabella per cui si desidera creare un facet.

   Aggiungi una matrice `TableFacets` che rappresenta tutti i facet per la tabella.

   Per ogni facet aggiungere un oggetto alla matrice `TableFacets`. Ogni elemento della matrice ha le seguenti proprietà:

   - `FacetName`: un nome per il facet. Questo valore deve essere unico sul modello.

- `PartitionKeyAlias`: un nome intuitivo per la chiave di partizione della tabella. L'alias viene visualizzato quando viene visualizzato il facet in NoSQL Workbench.
- `SortKeyAlias`: un nome intuitivo per la chiave di ordinamento della tabella. L'alias viene visualizzato quando viene visualizzato il facet in NoSQL Workbench. Questa proprietà non è necessaria se la tabella non ha una chiave di ordinamento definita.
- `NonKeyAttributes`: una matrice di nomi di attributo necessari per il pattern di accesso. Questi nomi devono mappare ai nomi attributo definiti per la tabella.

```
{
  "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. Ora è possibile importare il modello modificato in NoSQL Workbench. Per ulteriori informazioni, consulta Importazione di un modello di dati esistente.

Visualizzazione dei modelli di accesso ai dati

È possibile utilizzare lo strumento visualizzatore in NoSQL Workbench per Amazon DynamoDB per mappare le query e visualizzare pattern di accesso differenti (noti come facet) di un'applicazione. Ogni facet corrisponde a un pattern di accesso differente in DynamoDB. È possibile aggiungere manualmente i dati al modello o importare i dati da MySQL.

Argomenti

- Aggiunta di dati di esempio a un modello di dati
- Importazione di dati di esempio da un file CSV
- Visualizzazione dei pattern di accesso ai dati
- Visualizzazione di tutte le tabelle in un modello di dati utilizzando la visualizzazione aggregata
- Commit di un modello di dati su DynamoDB

Aggiunta di dati di esempio a un modello di dati

L'aggiunta di dati di esempio al modello consente di visualizzare i dati durante la visualizzazione del modello e dei vari pattern di accesso ai dati o facet.

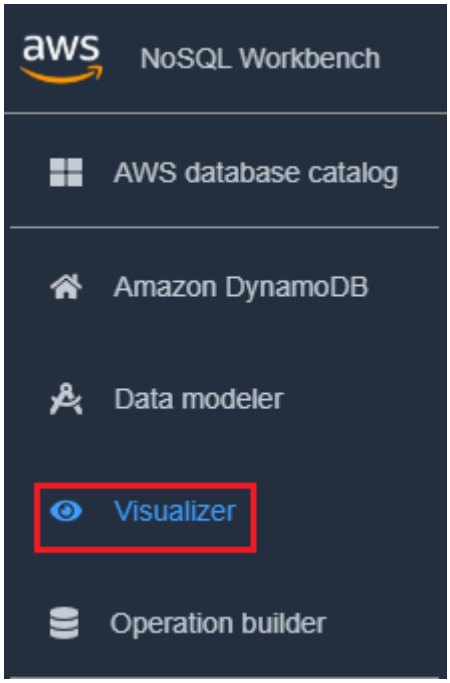
Esistono due modi per aggiungere dati di esempio. Uno è l'utilizzo del nostro strumento di generazione automatica di dati di esempio. L'altro è aggiungere i dati uno alla volta.

Seguire queste istruzioni per aggiungere dati di esempio a un modello di dati utilizzando NoSQL Workbench per Amazon DynamoDB.

Per generare automaticamente dati di esempio

La generazione automatica dei dati di esempio consente di generare da 1 a 5000 righe di dati per un utilizzo immediato. Puoi specificare un tipo di dati di esempio granulari per creare dati realistici in base alle esigenze di progettazione e test. Per utilizzare la capacità di generare dati realistici, devi specificare il formato del tipo di dati di esempio per gli attributi nel Data modeler. Consulta Creazione di un nuovo modello di dati per specificare i formati dei tipi di dati di esempio.

1. Nel riquadro di navigazione a sinistra, selezionare l'icona visualizer (visualizzatore).

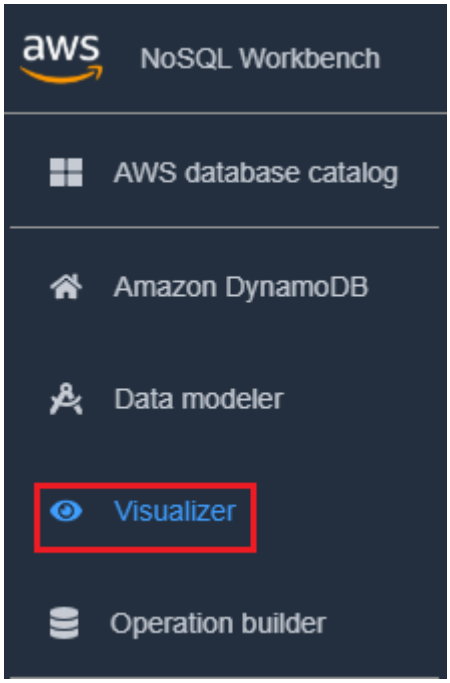


2. Nel visualizzatore, seleziona il modello di dati e scegli la tabella.
3. Scegli il menu a discesa Operazione e seleziona Aggiungi dati di esempio.

4. Inserisci il numero o gli elementi di dati di esempio che desideri generare, quindi seleziona Conferma.

Aggiunta di dati di esempio uno alla volta

1. Nel riquadro di navigazione a sinistra, selezionare l'icona visualizer (visualizzatore).

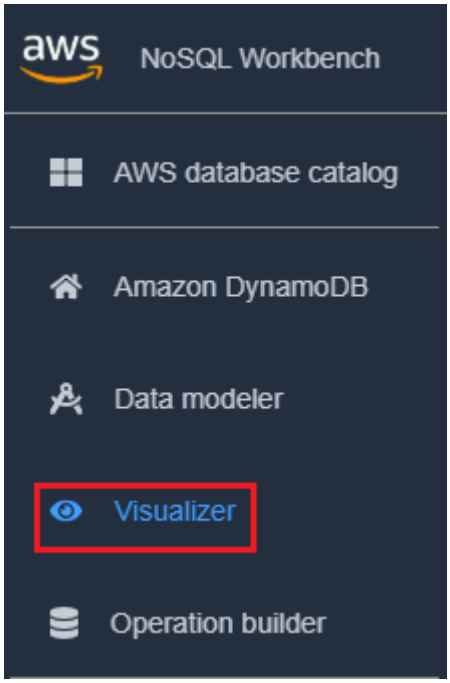


2. Nel visualizzatore, seleziona il modello di dati e scegli la tabella.
3. Scegli il menu a discesa Operazione e seleziona Modifica dati.

4. Seleziona Aggiungi nuova regola. Inserisci i dati di esempio nelle caselle di testo vuote, quindi scegli Aggiungi nuova riga per aggiungere ulteriori righe. Al termine, scegli Salva modifiche.

Eliminazione di dati di esempio

1. Nel riquadro di navigazione a sinistra, selezionare l'icona visualizer (visualizzatore).



2. Nel visualizzatore, seleziona il modello di dati e scegli la tabella.
3. Scegli il menu a discesa Operazione e seleziona Modifica dati.

4. Seleziona l'icona di eliminazione accanto a ogni riga di dati da eliminare.

Importazione di dati di esempio da un file CSV

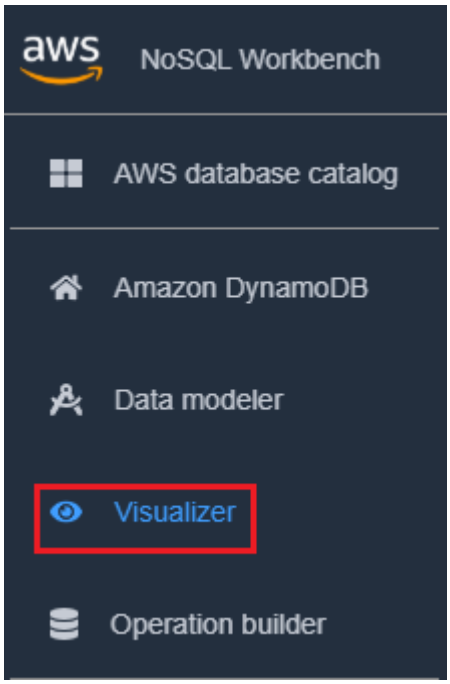
Se disponi di dati di esempio preesistenti in un file CSV, puoi importarli in NoSQL Workbench. Questo consente di popolare rapidamente il modello con dati di esempio senza doverli inserire riga per riga.

I nomi delle colonne nel file CSV devono corrispondere ai nomi degli attributi nel modello di dati, ma non devono necessariamente essere nello stesso ordine. Ad esempio, se il modello di dati contiene attributi chiamati `LoginAlias`, `FirstName` e `LastName`, le colonne del file CSV potrebbero essere `LastName`, `FirstName` e `LoginAlias`.

L'importazione di dati da un file CSV è limitata a 150 righe alla volta.

Per importare dati da un file CSV in NoSQL Workbench

1. Nel riquadro di navigazione a sinistra, selezionare l'icona visualizer (visualizzatore).



2. Nel visualizzatore, seleziona il modello di dati e scegli la tabella.
3. Scegli il menu a discesa Operazione e seleziona Modifica dati.
4. Scegli il menu a discesa Operazione e seleziona Importa file CSV.
5. Seleziona il file CSV e scegli Open (Apri). I dati nel file CSV vengono aggiunti alla tabella.

> **Note**
>
> Se il file CSV contiene una o più righe con le stesse chiavi degli elementi già presenti nella tabella, avrai la possibilità di sovrascrivere gli elementi esistenti o di aggiungerli alla fine della tabella. Se scegli di aggiungere gli elementi, il suffisso "-Copia" verrà aggiunto alla chiave di ogni elemento duplicato per differenziare gli elementi duplicati dagli elementi già presenti nella tabella.

Visualizzazione dei pattern di accesso ai dati

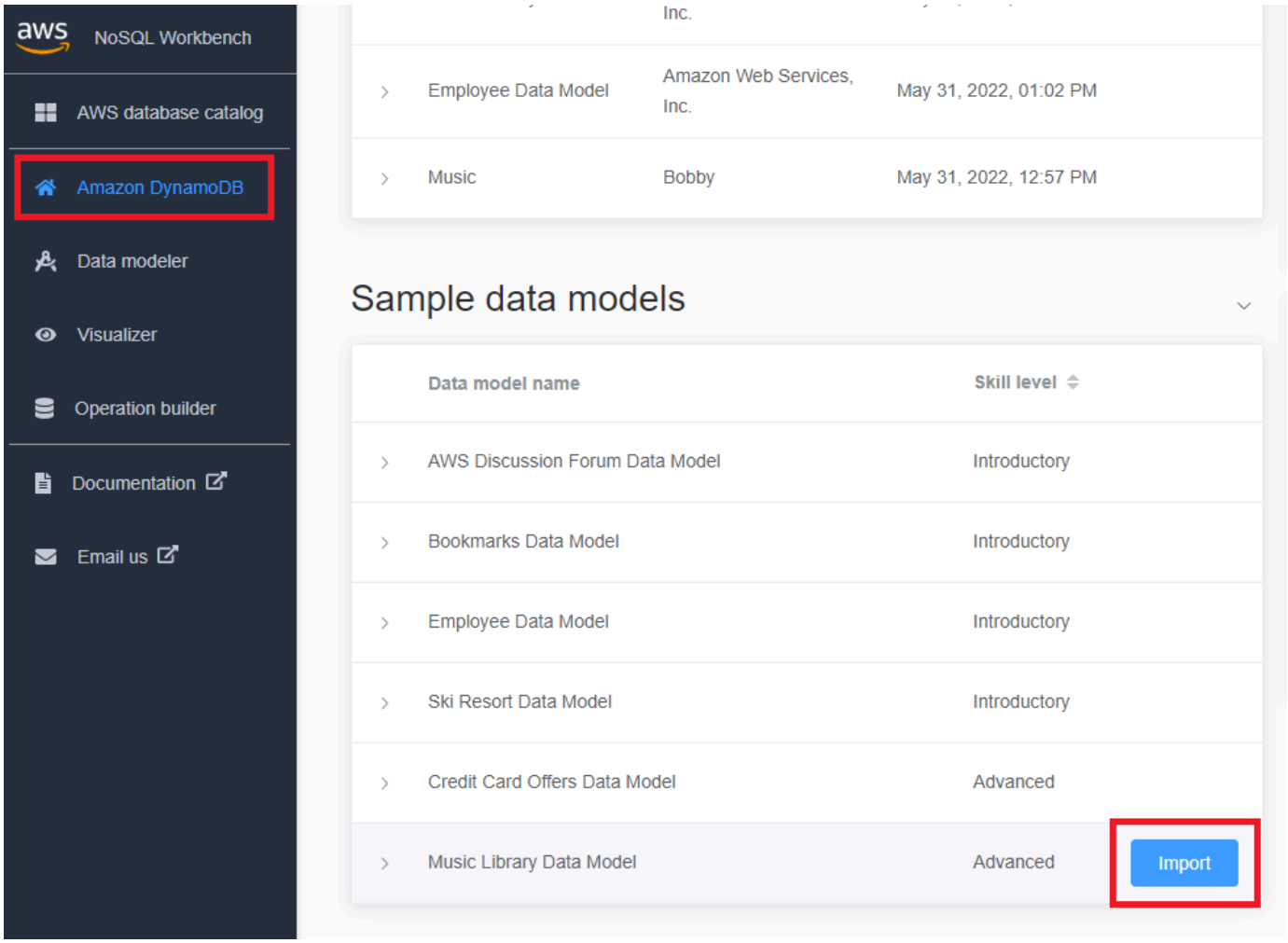
In NoSQL Workbench, i facet rappresentano pattern di accesso ai dati differenti di un'applicazione per Amazon DynamoDB. I facet possono aiutarti a visualizzare il modello di dati quando più tipi di dati sono rappresentati da una chiave di ordinamento. I facet consentono di visualizzare un

sottoinsieme di dati in una tabella, senza dover visualizzare record che non soddisfano i vincoli del facet. I facet sono considerati uno strumento di modellazione visiva dei dati e non esistono come costrutto utilizzabile in DynamoDB, in quanto sono puramente un aiuto alla modellazione dei modelli di accesso.

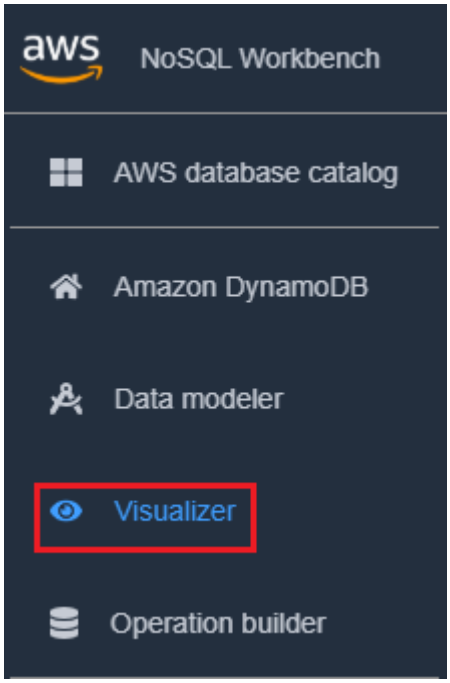
Per vedere un esempio di facet, puoi importare uno dei nostri modelli di dati di esempio con facet come parte del modello di dati.

Importazione di un modello di dati di esempio

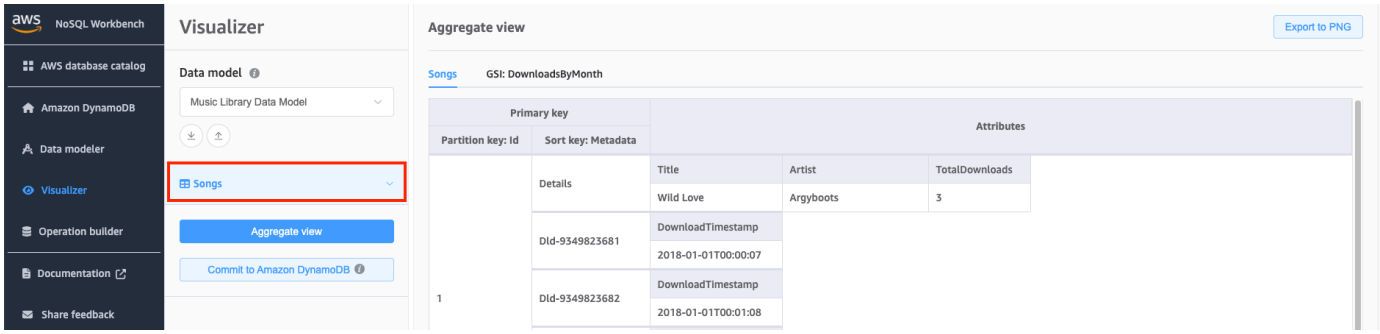
1. A sinistra, scegli Amazon DynamoDB.
2. Nella sezione dei modelli di dati di esempio, passa il puntatore su Music Library Data Model (Modello di dati della libreria musicale) e scegli Import (Importa).



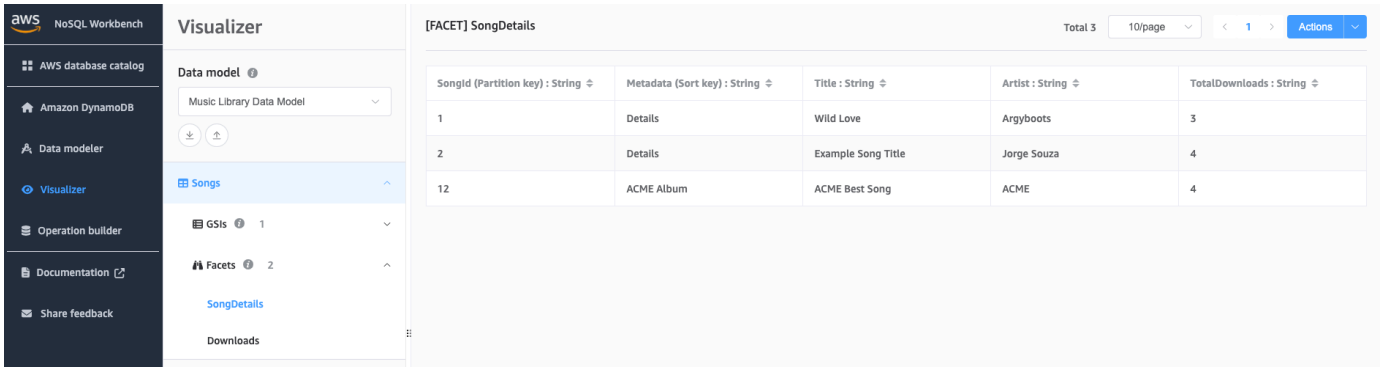
3. Nel riquadro di navigazione a sinistra, selezionare l'icona visualizer (visualizzatore).



4. Scegliere la tabella Brani per espanderla. Ti verrà mostrata una vista aggregata dei tuoi dati.



5. Scegli la freccia a discesa Facets (Facet) per espandere i facet disponibili.
6. Scegli il facet SongDetails per visualizzare i dati con applicato il facet SongDetails.



È inoltre possibile modificare le definizioni dei facet utilizzando Data Modeler. Per ulteriori informazioni, consulta Modifica di un modello di dati esistente.

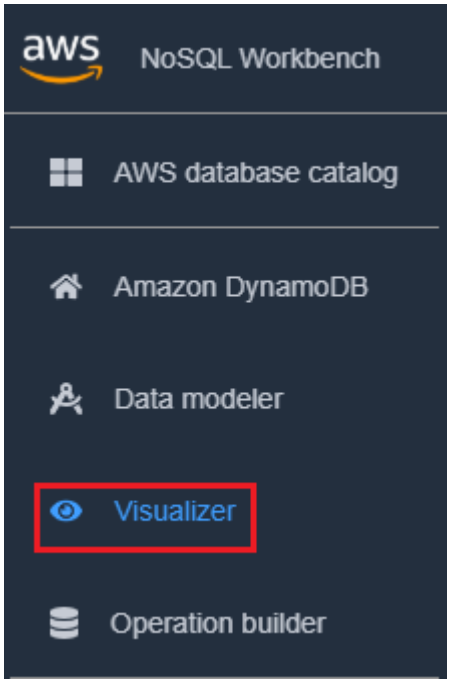
Visualizzazione di tutte le tabelle in un modello di dati utilizzando la visualizzazione aggregata

La visualizzazione aggregata in NoSQL Workbench per Amazon DynamoDB rappresenta tutte le tabelle in un modello di dati. Per ogni tabella, vengono visualizzate le seguenti informazioni:

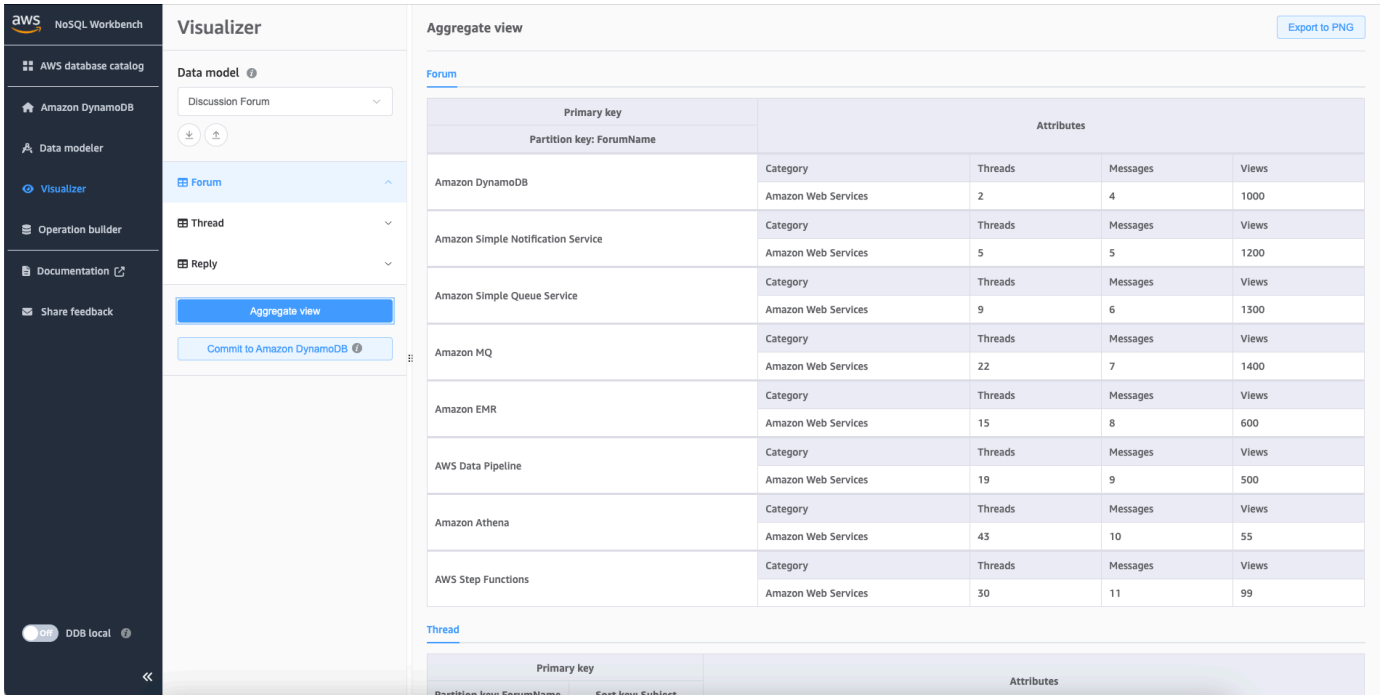
- Nomi di colonna delle tabelle
- Dati campione
- Tutti gli indici secondari globali associati alla tabella. Per ogni indice vengono visualizzate le seguenti informazioni:
  - Nomi colonna indice
  - Dati campione

Per visualizzare tutte le informazioni della tabella

1. Nel riquadro di navigazione a sinistra, selezionare l'icona visualizer (visualizzatore).



2. Nel visualizzatore, seleziona Visualizzazione aggregata.



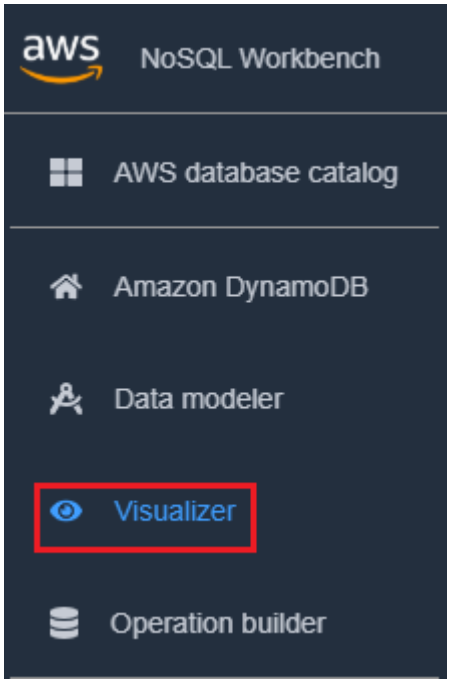
Commit di un modello di dati su DynamoDB

Una volta soddisfatti del proprio modello di dati, è possibile eseguire il commit del modello su Amazon DynamoDB.

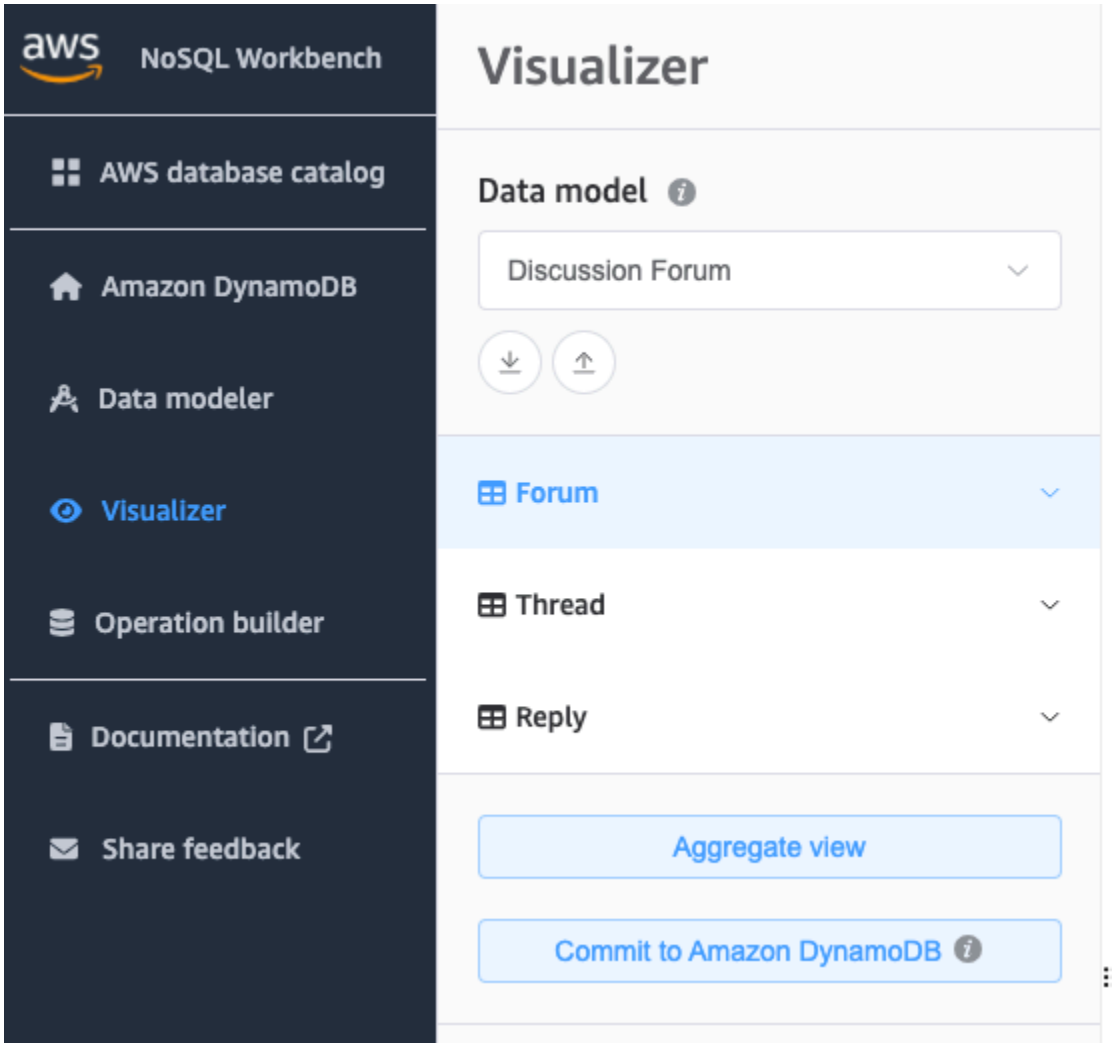
> **Note**
>
> - Questa azione comporta la creazione di risorse lato server in AWS per le tabelle e gli indici secondari globali rappresentati nel modello di dati.
> - Le tabelle vengono create con le seguenti caratteristiche:
>   - L'Auto Scaling è impostato sul 70% dell'utilizzo target.
>   - La capacità di provisioning è impostata su 5 unità di capacità di lettura e 5 unità di capacità di scrittura.
> - Gli indici secondari globali sono creati con capacità di provisioning di 10 unità di capacità di lettura e 5 unità di capacità di scrittura.

Come eseguire il commit del modello di dati su DynamoDB

1. Nel riquadro di navigazione a sinistra, selezionare l'icona visualizer (visualizzatore).



2. Seleziona Commit su DynamoDB.



3. Seleziona una connessione esistente o crearne una nuova selezionando la scheda Aggiungi nuova connessione remota.

   - Per aggiungere una nuova connessione, specificare le informazioni seguenti:
     - Account Alias (Alias account)
     - Regione AWS
     - ID chiave di accesso
     - Chiave di accesso segreta

     Per ulteriori informazioni su come ottenere le chiavi di accesso, consulta Come ottenere una chiave di accesso di AWS.
   - Facoltativamente, puoi specificare quanto segue:
     - Session token (Token di sessione)

- IAM role ARN (ARN del ruolo IAM)
- Se non vuoi sottoscrivere un account per un piano gratuito e preferisci usare DynamoDB Local (versione scaricabile):
  1. Scegli la scheda Add a new DynamoDB local connection (Aggiungi una nuova connessione DynamoDB locale).
  2. Specifica Nome della connessione e Porta.

4. Scegli Applica.

> **Note**
>
> Se hai installato la versione locale di DynamoDB come parte della configurazione di NoSQL Workbench, dovrai attivarla utilizzando l'interruttore Server locale DynamoDB in basso a sinistra nella schermata NoSQL Workbench. Per ulteriori informazioni su questo interruttore, consulta Installazione di NoSQL Workbench per DynamoDB.

Esplorazione di set di dati e creazione di operazioni con NoSQL Workbench

NoSQL Workbench per Amazon DynamoDB offre una ricca interfaccia utente grafica per lo sviluppo e il test di query. È possibile utilizzare l'Operation builder in NoSQL Workbench per visualizzare, esplorare ed eseguire query sui set di dati. Operation builder strutturato supporta espressioni di proiezione, espressioni condizionali e genera codice di esempio in più lingue. È possibile clonare direttamente le tabelle da un account Amazon DynamoDB a un altro in Regioni diverse. È anche possibile clonare direttamente le tabelle tra DynamoDB locale e un account Amazon DynamoDB per copiare più rapidamente lo schema delle chiavi della tabella (e facoltativamente lo schema e gli elementi degli indici secondari globali (GSI) tra gli ambienti di sviluppo. È possibile salvare fino a 50 operazioni sui dati DynamoDB in Operation builder.

Argomenti

- Connessione a set di dati in tempo reale
- Creazione di operazioni complesse
- Clonazione di tabelle con NoSQL Workbench
- Esportazione di dati in un file CSV

Connessione a set di dati in tempo reale

Per connetterti alle tabelle Amazon DynamoDB con NoSQL Workbench, devi prima connetterti al tuo account AWS.

Per aggiungere una connessione al database

1. In NoSQL Workbench, nel riquadro di navigazione a sinistra, seleziona l'icona Operation builder.
2. Scegli Aggiungi connessione.
3. Specificare le seguenti informazioni:

   - Nome della connessione
   - Regione AWS
   - ID chiave di accesso
   - Chiave di accesso segreta

   Per ulteriori informazioni su come ottenere le chiavi di accesso, consulta Come ottenere una chiave di accesso di AWS.

   Facoltativamente, puoi specificare quanto segue:

   - Session token (Token di sessione)
   - IAM role ARN (ARN del ruolo IAM)
4. Scegli Connetti.

   Se non vuoi sottoscrivere un account per un piano gratuito e preferisci usare DynamoDB Local (versione scaricabile):

   a. Scegli la scheda Local (Locale) nella schermata di connessione.
   b. Specificare le seguenti informazioni:

      - Nome della connessione
      - Porta

   c. Scegli il pulsante Connetti.

> **Note**
>
> Per connetterti a DynamoDB locale, avvia manualmente DynamoDB locale usando il terminale (consulta Distribuzione di DynamoDB in locale sul computer) o avvia DynamoDB locale direttamente utilizzando l'opzione DDB locale nel menu di navigazione di NoSQL Workbench. Assicurati che la porta di connessione sia la stessa della porta di DynamoDB locale.

5. Una volta creata la connessione, seleziona Apri.

Dopo aver effettuato la connessione al database DynamoDB, l'elenco di tabelle disponibili viene visualizzato nel riquadro a sinistra. Scegli una delle tabelle per ottenere un esempio di dati memorizzati nella tabella.

Ora è possibile eseguire query sulla tabella selezionata.

Per eseguire query su una tabella, consulta la sezione successiva sulla creazione di operazioni: consulta Creazione di operazioni complesse.

Creazione di operazioni complesse

L'operation builder in NoSQL Workbench per Amazon DynamoDB offre un'interfaccia visuale in cui è possibile effettuare operazioni complesse del piano dati. Include il supporto per le espressioni di proiezione e di condizione. Una volta creata un'operazione, sarà possibile salvarla per un utilizzo successivo (è possibile salvare fino a 50 operazioni). Sarà quindi possibile sfogliare un elenco delle operazioni del piano dati utilizzate di frequente nel menu Operazioni salvate e utilizzarle per popolare e creare automaticamente una nuova operazione. È possibile anche decidere di generare codici di esempio per queste operazioni, in più linguaggi.

NoSQL Workbench supporta la creazione di PartiQL per le istruzioni DynamoDB, che consente di interagire con DynamoDB utilizzando un linguaggio di query compatibile con SQL. NoSQL Workbench supporta anche la creazione di operazioni API CRUD di DynamoDB.

Per utilizzare NoSQL Workbench per creare operazioni, nel riquadro di navigazione a sinistra, seleziona l'icona Operation builder.

Argomenti

- Creazione di istruzioni PartiQL
- Creazione di operazioni API

Creazione di istruzioni PartiQL

Per utilizzare NoSQL Workbench per creare istruzioni PartiQL per DynamoDB, seleziona Editor PartiQL nella parte in alto dell'interfaccia utente di NoSQL Workbench.

È possibile creare i seguenti tipi di istruzione PartiQL con l'operation builder:

Argomenti

- Istruzioni singleton
- Transazioni
- Archiviazione

Istruzioni singleton

Per eseguire o generare codice per un'istruzione PartiQL, completare le seguenti operazioni.

1. Seleziona Editor PartiQL nella parte superiore della finestra.
2. Immettere una istruzione PartiQL valida.
3. Se l'istruzione utilizza parametri:
    a. Scegli Parametri di richiesta facoltativi.
    b. Scegli Aggiungi nuovi parametri.
    c. Inserire il tipo e il valore dell'attributo.
    d. Se desideri aggiungere altri parametri, ripeti le fasi b e c.
4. Per generare codice, selezionare Generate code (Genera codice).

    Selezionare la lingua desiderata dalle schede visualizzate. È possibile copiare questo codice e utilizzarlo nell'applicazione.
5. Per eseguire l'operazione immediatamente, seleziona Esegui.
6. Se desideri salvare questa operazione per un utilizzo successivo, scegli Save operation (Salva operazione). Quindi inserisci un nome per l'operazione e scegli Save (Salva).

Transazioni

Per eseguire o generare codice per una transazione PartiQL, completare le seguenti operazioni.

1. Seleziona PartiQLTransaction dal menu a discesa Altre operazioni.
2. Scegli Aggiungi una nuova istruzione.
3. Immettere una istruzione PartiQL valida.

   > **Note**
   >
   > Le operazioni di lettura e scrittura non sono supportate nella stessa richiesta di transazione PartiQL. Un'istruzione SELECT (SELEZIONA) non può trovarsi nella stessa richiesta con le Istruzioni INSERT (INSERISCI), UPDATE (AGGIORNA) e DELETE (ELIMINA). Consulta Esecuzione di transazioni con PartiQL per DynamoDB per maggiori dettagli.

4. Se l'istruzione utilizza parametri
   a. Scegli Parametri di richiesta facoltativi.
   b. Scegli Aggiungi nuovi parametri.
   c. Inserire il tipo e il valore dell'attributo.
   d. Se desideri aggiungere altri parametri, ripeti le fasi b e c.
5. Se desideri aggiungere altre istruzioni, ripeti le fasi da 2 a 4.
6. Per generare codice, selezionare Generate code (Genera codice).

   Selezionare la lingua desiderata dalle schede visualizzate. È possibile copiare questo codice e utilizzarlo nell'applicazione.
7. Per eseguire l'operazione immediatamente, seleziona Esegui.
8. Se desideri salvare questa operazione per un utilizzo successivo, scegli Save operation (Salva operazione). Quindi inserisci un nome per l'operazione e scegli Save (Salva).

Archiviazione

Per eseguire o generare codice per un batch PartiQL, completare le seguenti operazioni.

1. Seleziona PartiQLBatch dal menu a discesa Altre operazioni.
2. Scegli Aggiungi una nuova istruzione.

3. Immettere una istruzione PartiQL valida.

   > **Note**
   >
   > Le operazioni di lettura e scrittura non sono supportate nella stessa richiesta batch PartiQL, il che significa che un'istruzione SELECT (SELEZIONA) non può essere inclusa nella stessa richiesta con istruzioni INSERT (INSERISCI), UPDATE (AGGIORNA) e DELETE (ELIMINA). Le operazioni di scrittura sullo stesso elemento non sono consentite. Come per l'operazione BatchGetItem, sono supportate solo le operazioni di lettura singleton. Le operazioni di scansione e query non sono supportate. Consulta Esecuzione di operazioni in batch con PartiQL per DynamoDB per maggiori dettagli.

4. Se l'istruzione utilizza parametri:

   a. Scegli Parametri di richiesta facoltativi.

   b. Scegli Aggiungi nuovi parametri.

   c. Inserire il tipo e il valore dell'attributo.

   d. Se desideri aggiungere altri parametri, ripeti le fasi b e c.

5. Se desideri aggiungere altre istruzioni, ripeti le fasi da 2 a 4.

6. Per generare codice, selezionare Generate code (Genera codice).

   Selezionare la lingua desiderata dalle schede visualizzate. È possibile copiare questo codice e utilizzarlo nell'applicazione.

7. Per eseguire l'operazione immediatamente, seleziona Esegui.

8. Se desideri salvare questa operazione per un utilizzo successivo, scegli Save operation (Salva operazione). Quindi inserisci un nome per l'operazione e scegli Save (Salva).

Creazione di operazioni API

Per utilizzare NoSQL Workbench per creare API CRUD DynamoDB, seleziona Operation builder dalla parte a sinistra dell’interfaccia utente di NoSQL Workbench.

Quindi seleziona Apri e seleziona una connessione.

Puoi eseguire le seguenti operazioni con Operation builder:

- Delete Table

- Create Table
- Update Table

- Put Item
- Update Item
- Delete Item
- Query
- Scan
- Transact Get Items
- Transact Write Items

Delete Table (Elimina tabella)

Per eseguire un'operazione `Delete Table`, completa le seguenti operazioni.

1. Trova la tabella che desideri eliminare nella sezione Tabelle.
2. Seleziona Elimina tabella dal menu con puntini di sospensione orizzontali.
3. Conferma l'eliminazione della tabella inserendo il Nome della tabella.
4. Seleziona Elimina.

Per ulteriori informazioni su questa operazione, consulta Delete Table (Elimina tabella) nella Documentazione di riferimento delle API di Amazon DynamoDB.

Eliminazione di un GSI

Per eseguire un'operazione `Delete GSI`, completa le seguenti operazioni.

1. Trova il GSI di una tabella da eliminare nella sezione Tabelle.
2. Seleziona Elimina GSI dal menu con puntini di sospensione orizzontali.
3. Conferma l'eliminazione inserendo il Nome del GSI.
4. Seleziona Elimina.

Per ulteriori informazioni su questa operazione, consulta Delete Table (Elimina tabella) nella Documentazione di riferimento delle API di Amazon DynamoDB.

Create table (Crea tabella)

Per eseguire un'operazione `Create Table`, completa le seguenti operazioni.

1. Seleziona l'icona + accanto alla sezione Tabelle.
2. Immettere il nome della tabella desiderato.
3. Creare una chiave di partizione.
4. Facoltativo: crea una chiave di ordinamento.
5. Per personalizzare le impostazioni di capacità, deseleziona la casella accanto a Utilizza impostazioni di capacità predefinite.
   - Ora è possibile selezionare sia Provisioned (Con provisioning) che On-demand capacity (Capacità on-demand).

     Con Provisioned (Con provisioning) selezionato, è possibile impostare unità di capacità di lettura e scrittura minima e massima. È inoltre possibile abilitare o disabilitare il dimensionamento automatico.
   - Se la tabella è attualmente impostata su On demand, non sarà possibile specificare un throughput allocato.
   - Se si passa da Throughput on demand a Con provisioning, il dimensionamento automatico verrà automaticamente applicato a tutti i GSI con: min: 1, max: 10; destinazione: 70%.
6. Seleziona Ignora GSI e crea per creare questa tabella senza un GSI. Facoltativamente, è possibile selezionare Successivo per creare un GSI con questa nuova tabella.

Per ulteriori informazioni su questa operazione, consulta Create Table (Crea tabella) nella Documentazione di riferimento delle API di Amazon DynamoDB.

Creazione di un GSI

Per eseguire un'operazione `Create GSI`, completa le seguenti operazioni.

1. Trova una tabella a cui aggiungere un GSI.
2. Seleziona Crea GSI dal menu con puntini di sospensione orizzontali.
3. Assegna un nome al GSI in Nome indice.
4. Creare una chiave di partizione.
5. Facoltativo: crea una chiave di ordinamento.

6. Seleziona un'opzione di proiezione dal menu a discesa.
7. Seleziona Crea GSI.

Per ulteriori informazioni su questa operazione, consulta Create Table (Crea tabella) nella Documentazione di riferimento delle API di Amazon DynamoDB.

Update Table (Aggiorna tabella)

Per aggiornare le impostazioni di capacità per una tabella con un'operazione `Update Table`, completa le seguenti operazioni.

1. Trova la tabella per la quale aggiornare le impostazioni di capacità.
2. Dal menu con i puntini di sospensione orizzontali, seleziona Aggiorna impostazioni di capacità.
3. Seleziona Con provisioning o Capacità on demand.

   Con l'opzione Con provisioning selezionata è possibile impostare unità di capacità di lettura e scrittura minima e massima. È inoltre possibile abilitare o disabilitare il dimensionamento automatico.
4. Selezionare Update (Aggiorna).

Per ulteriori informazioni su questa operazione, consulta Update Table (Aggiorna tabella) nella Documentazione di riferimento delle API di Amazon DynamoDB.

Aggiornamento di un GSI

Per aggiornare le impostazioni di capacità per un indice secondario globale con un'operazione `Update Table`, completa le seguenti operazioni.

> **Note**
>
> Per impostazione predefinita, gli indici secondari globali ereditano le impostazioni di capacità della tabella di base. Gli indici secondari globali possono avere una modalità di capacità diversa solo quando la tabella di base è in modalità con capacità allocata. Quando si crea un indice secondario globale su una tabella con modalità assegnata, è necessario specificare le unità di capacità di lettura e scrittura per il carico di lavoro previsto sull'indice. Per ulteriori informazioni, consulta Considerazioni sulla velocità di trasmissione effettiva assegnata per indici secondari globali.

1. Trova il GSI per il quale aggiornare le impostazioni di capacità.
2. Dal menu con i puntini di sospensione orizzontali, seleziona Aggiorna impostazioni di capacità.
3. Ora è possibile selezionare sia Provisioned (Con provisioning) che On-demand capacity (Capacità on-demand).

   Con l'opzione Con provisioning selezionata è possibile impostare unità di capacità di lettura e scrittura minima e massima. È inoltre possibile abilitare o disabilitare il dimensionamento automatico.
4. Selezionare Update (Aggiorna).

Per ulteriori informazioni su questa operazione, consulta Update Table (Aggiorna tabella) nella Documentazione di riferimento delle API di Amazon DynamoDB.

Put Item

Per la creazione di un elemento viene utilizzata l'operazione `Put Item`. Per eseguire o generare codice per un'operazione `Put Item`, completare le seguenti operazioni.

1. Trova la tabella in cui desideri creare un elemento.
2. Dal menu a discesa Operazioni, seleziona Crea elemento.
3. Immettere il valore della chiave di partizione.
4. Immettere il valore della chiave di ordinamento, se esiste.
5. Se si desidera aggiungere attributi non di chiave, svolgere le seguenti operazioni:

   a. Seleziona + Aggiungi altri attributi.

   b. Specifica Nome attributo, Tipo e Valore.
6. Se un'espressione di condizione deve essere soddisfatta per la riuscita dell'operazione `Put Item`, svolgere le seguenti operazioni:

   a. Selezionare Condition (Condizioni).

   b. Specificare il nome attributo, l'operatore di confronto, il tipo di attributo e il valore dell'attributo.

   c. Se sono richieste altre condizioni, selezionare nuovamente Condition (Condizione).

   Per ulteriori informazioni, consulta Esempio di CLI di espressione condizionale in DynamoDB.
7. Per generare codice, selezionare Generate code (Genera codice).

Selezionare la lingua desiderata dalle schede visualizzate. È possibile copiare questo codice e utilizzarlo nell'applicazione.

8. Per eseguire l'operazione immediatamente, seleziona Esegui.
9. Se desideri salvare questa operazione per un utilizzo successivo, scegli Save operation (Salva operazione), quindi specifica un nome per l'operazione e scegli Save (Salva).

Per ulteriori informazioni su questa operazione, consulta PutItem nella Documentazione di riferimento delle API di Amazon DynamoDB.

Update Item

Per eseguire o generare codice per un'operazione `Update Item`, svolgere le seguenti operazioni:

1. Trova la tabella in cui desideri aggiornare un elemento.
2. Seleziona l'elemento.
3. Inserisci il nome attributo e il valore attributo per l'espressione selezionata.
4. Se desideri aggiungere altre espressioni, seleziona un'altra espressione nell'elenco a discesa Aggiorna espressione, quindi seleziona l'icona +.
5. Se un'espressione di condizione deve essere soddisfatta per la riuscita dell'operazione `Update Item`, svolgere le seguenti operazioni:

   a. Selezionare Condition (Condizioni).

   b. Specificare il nome attributo, l'operatore di confronto, il tipo di attributo e il valore dell'attributo.

   c. Se sono richieste altre condizioni, selezionare nuovamente Condition (Condizione).

   Per ulteriori informazioni, consulta Esempio di CLI di espressione condizionale in DynamoDB.
6. Per generare codice, selezionare Generate code (Genera codice).

   Seleziona la scheda per la lingua da utilizzare. È possibile copiare questo codice e utilizzarlo nell'applicazione.
7. Per eseguire l'operazione immediatamente, seleziona Esegui.
8. Se desideri salvare questa operazione per un utilizzo successivo, scegli Save operation (Salva operazione), quindi specifica un nome per l'operazione e scegli Save (Salva).

Per ulteriori informazioni su questa operazione, consulta UpdateItem nella Documentazione di riferimento delle API di Amazon DynamoDB.

Delete Item

Per eseguire un'operazione `Delete Item`, completa le seguenti operazioni.

1. Trova la tabella in cui desideri eliminare un elemento.
2. Seleziona l'elemento.
3. Dal menu a discesa Operazioni, seleziona Elimina voce.
4. Conferma l'eliminazione dell'elemento selezionando Elimina.

Per ulteriori informazioni su questa operazione, consulta DeleteItem nella Documentazione di riferimento delle API di Amazon DynamoDB.

Duplicazione di un elemento

È possibile duplicare un elemento creando un nuovo elemento con gli stessi attributi. Per duplicare un elemento, completa le seguenti operazioni.

1. Trova la tabella in cui desideri duplicare un elemento.
2. Seleziona l'elemento.
3. Dal menu a discesa Operazioni, seleziona Elemento duplicato.
4. Specifica una nuova chiave di partizione.
5. Specifica una nuova chiave di ordinamento (se necessario).
6. Seleziona Esegui.

Per ulteriori informazioni su questa operazione, consulta DeleteItem nella Documentazione di riferimento delle API di Amazon DynamoDB.

Query

Per eseguire o generare codice per un'operazione `Query`, completare le seguenti operazioni.

1. Seleziona Query nella parte superiore dell'interfaccia utente di NoSQL Workbench.
2. Specificare il valore della chiave di partizione.
3. Se è necessaria una chiave di ordinamento per l'operazione `Query`:

   a. Seleziona Chiavi di ordinamento.
   b. Specifica l'operatore di confronto e il valore dell'attributo.
4. Seleziona Query per eseguire l'operazione relativa. Se sono necessarie altre opzioni, seleziona la casella di spunta Altre opzioni e continua con la procedura seguente.
5. Se non tutti gli attributi devono essere restituiti con il risultato dell'operazione, seleziona Espressione di proiezione.
6. Scegli l'icona +.
7. Immettere l'attributo da restituire con i risultati della query.
8. Se sono necessari più attributi, seleziona +.
9. Se un'espressione di condizione deve essere soddisfatta per la riuscita dell'operazione `Query`, svolgere le seguenti operazioni:

   a. Selezionare Condition (Condizioni).
   b. Specificare il nome attributo, l'operatore di confronto, il tipo di attributo e il valore dell'attributo.
   c. Se sono richieste altre condizioni, selezionare nuovamente Condition (Condizione).

   Per ulteriori informazioni, consulta Esempio di CLI di espressione condizionale in DynamoDB.
10. Per generare codice, selezionare Generate code (Genera codice).

    Seleziona la scheda per la lingua da utilizzare. È possibile copiare questo codice e utilizzarlo nell'applicazione.
11. Per eseguire l'operazione immediatamente, seleziona Esegui.
12. Se desideri salvare questa operazione per un utilizzo successivo, scegli Save operation (Salva operazione), quindi specifica un nome per l'operazione e scegli Save (Salva).

Per ulteriori informazioni su questa operazione, consulta Query nella Documentazione di riferimento delle API di Amazon DynamoDB.

Scan

Per eseguire o generare codice per un'operazione `Scan`, completare le seguenti operazioni.

1. Seleziona Scansiona nella parte superiore dell'interfaccia utente di NoSQL Workbench.

2. Seleziona il pulsante Scansiona per eseguire questa operazione di scansione di base. Se sono necessarie altre opzioni, seleziona la casella di spunta Altre opzioni e continua con la procedura seguente.
3. Specifica il nome di un attributo per filtrare i risultati della scansione.
4. Se non tutti gli attributi devono essere restituiti con il risultato dell'operazione, seleziona Espressione di proiezione.
5. Se un'espressione di condizione deve essere soddisfatta per la riuscita dell'operazione di scansione, svolgere le seguenti operazioni:

   a. Selezionare Condition (Condizioni).
   b. Specificare il nome attributo, l'operatore di confronto, il tipo di attributo e il valore dell'attributo.
   c. Se sono richieste altre condizioni, selezionare nuovamente Condition (Condizione).

   Per ulteriori informazioni, consulta Esempio di CLI di espressione condizionale in DynamoDB.
6. Per generare codice, selezionare Generate code (Genera codice).

   Seleziona la scheda per la lingua da utilizzare. È possibile copiare questo codice e utilizzarlo nell'applicazione.
7. Per eseguire l'operazione immediatamente, seleziona Esegui.
8. Se desideri salvare questa operazione per un utilizzo successivo, scegli Save operation (Salva operazione), quindi specifica un nome per l'operazione e scegli Save (Salva).

TransactGetItems

Per eseguire o generare codice per un'operazione `TransactGetItems`, completare le seguenti operazioni.

1. Dal menu a discesa Altre operazioni nella parte superiore dell’interfaccia utente di NoSQL Workbench, seleziona TransactGetItems.
2. Seleziona l’icona + accanto a TransactGetItem.
3. Specifica una chiave di partizione.
4. Specifica una chiave di ordinamento (se necessario).
5. Seleziona Esegui per eseguire l’operazione, Salva operazione per salvarla o Genera codice per generare il codice corrispondente.

Per ulteriori informazioni sulle transazioni, consulta Transazioni Amazon DynamoDB.

TransactWriteItems

Per eseguire o generare codice per un'operazione `TransactWriteItems`, completare le seguenti operazioni.

1. Dal menu a discesa Altre operazioni nella parte superiore dell'interfaccia utente di NoSQL Workbench, seleziona TransactWriteItems.
2. Seleziona un'operazione dal menu a discesa Operazioni.
3. Seleziona l'icona + accanto a TransactWriteItem.
4. Dal menu a discesa Operazioni seleziona l'operazione desiderata.

   - Per `DeleteItem`, seguire le istruzioni per l'operazione Delete Item.
   - Per `PutItem`, seguire le istruzioni per l'operazione Put Item.
   - Per `UpdateItem`, seguire le istruzioni per l'operazione Update Item.

   Per modificare l'ordine delle operazioni, selezionare un'operazione nell'elenco a sinistra, quindi selezionare le frecce verso l'alto e verso il basso per spostarsi nell'elenco a discesa.

   Per eliminare un'operazione, selezionare l'operazione dall'elenco, quindi selezionare l'icona Delete (Elimina) (raffigurata dal cestino).
5. Seleziona Esegui per eseguire l'operazione, Salva operazione per salvarla o Genera codice per generare il codice corrispondente.

Per ulteriori informazioni sulle transazioni, consulta Transazioni Amazon DynamoDB.

Clonazione di tabelle con NoSQL Workbench

La clonazione delle tabelle copierà lo schema delle chiavi di una tabella (e facoltativamente lo schema e gli elementi degli indici secondari globali (GSI)) tra gli ambienti di sviluppo. È possibile clonare una tabella tra DynamoDB locale a un account Amazon DynamoDB e persino clonare una tabella da un account all'altro in diverse Regioni per una sperimentazione più rapida.

Come clonare una tabella

1. In Operation builder, seleziona la connessione e la Regione (la selezione della Regione non è disponibile per DynamoDB locale).

2. Una volta effettuata la connessione a DynamoDB, sfoglia le tabelle e seleziona la tabella da clonare.
3. Dal menu con i puntini di sospensione orizzontali, seleziona l'opzione Clona.
4. Inserisci i dettagli della destinazione del clone:

    a. Seleziona una connessione.

    b. Seleziona una Regione (la Regione non è disponibile per DynamoDB locale).

    c. Inserisci un nome per la nuova tabella.

    d. Seleziona un'opzione di clonazione:

        i. Schema chiave è selezionato per impostazione predefinita e non può essere deselezionato. Per impostazione predefinita, la clonazione di una tabella copierà la chiave primaria e la chiave di ordinamento, se disponibili.

        ii. Schema GSI è selezionato per impostazione predefinita se la tabella da clonare ha un GSI. La clonazione di una tabella copierà la chiave primaria e la chiave di ordinamento dei GSI, se disponibili. È possibile deselezionare lo schema dei GSI per saltarne la clonazione. La clonazione di una tabella copierà le impostazioni di capacità della tabella di base come impostazioni di capacità del GSI. È possibile utilizzare l'operazione `UpdateTable` in Operation builder per aggiornare l'impostazione della capacità dei GSI della tabella dopo il completamento della clonazione.
5. Inserisci il numero di elementi da clonare. Per clonare solo lo schema delle chiavi e facoltativamente lo schema dei GSI, è possibile mantenere il valore Elementi da clonare su 0. Il numero massimo di elementi che possono essere clonati è 5000.
6. Seleziona una modalità di capacità:

    a. Modalità on demand è selezionato per impostazione predefinita. DynamoDB on demand offre tariffe paga-su-richiesta per le richieste di lettura e scrittura, che consentono di pagare solo per quello che viene effettivamente utilizzato. Per ulteriori informazioni, consulta Modalità on demand di DynamoDB.

    b. Modalità con provisioning consente di specificare il numero di letture e scritture al secondo necessarie per l'applicazione. È possibile utilizzare il dimensionamento automatico per regolare automaticamente la capacità assegnata della tabella in risposta alle modifiche del traffico. Per ulteriori informazioni, consulta Modalità con provisioning di DynamoDB.
7. Seleziona Clona per iniziare la clonazione.

8. Il processo di clonazione verrà eseguito in background. La scheda Operation builder mostrerà una notifica in caso di modifica dello stato della tabella di clonazione. È possibile accedere a questo stato selezionando la scheda Operation builder e quindi selezionando il pulsante con la freccia. Il pulsante con la freccia si trova sul widget dello stato della clonazione della tabella situato nella parte inferiore della barra laterale del menu.

Esportazione di dati in un file CSV

Puoi esportare i risultati di una query da Operation Builder in un file CSV. Questa operazione consente di caricare i dati in un foglio di calcolo o di elaborarli utilizzando il linguaggio di programmazione preferito.

Esportazione in CSV

1. In Operation Builder, esegui un'operazione a tua scelta, ad esempio una scansione o una query.

   > **Note**
   >
   > - È possibile esportare solo i risultati delle operazioni API di lettura e delle istruzioni PartiQL in un file CSV. Non è possibile esportare i risultati delle istruzioni di lettura delle transazioni.
   > - Attualmente è possibile esportare i risultati una pagina alla volta in un file CSV. Se sono presenti più pagine di risultati, è necessario esportare ogni pagina singolarmente.

2. Seleziona gli elementi che desideri esportare dai risultati.
3. Dal menu a discesa Operazioni, seleziona Esporta come CSV.
4. Scegli un nome del file e una posizione per il file CSV e seleziona Save (Salva).

Modelli di dati di esempio per NoSQL Workbench

Nella home page del modellatore e del visualizzatore vengono visualizzati diversi modelli di esempio forniti con NoSQL Workbench. In questa sezione vengono descritti questi modelli e i loro potenziali utilizzi.

Argomenti

- Modello di dati del dipendente

- Modello di dati del forum di discussione
- Modello di dati della libreria musicale
- Modello di dati della stazione sciistica
- Modello di dati delle offerte per carta di credito
- Modello di dati dei segnalibri

Modello di dati del dipendente

Questo modello di dati è un modello introduttivo. Rappresenta i dettagli di base di un dipendente, ad esempio alias univoco, nome, cognome, designazione, manager e competenze.

Questo modello di dati descrive alcune tecniche come la gestione di attributi complessi per ad esempio più di una competenza. Questo modello è anche un esempio di relazione uno a molti per il manager e i dipendenti diretti ottenuta dall'indice secondario DirectReports.

I modelli di accesso agevolati da questo modello di dati sono:

- Recupero del record di un dipendente utilizzando l'alias di accesso del dipendente, agevolato dalla tabella chiamata `Employee`.
- Ricerca del dipendente per nome, facilitata dall'indice secondario globale della tabella Employee chiamato `Name`.
- Recupero di tutti i report diretti di un manager utilizzando l'alias di accesso del manager, agevolato dall'indice secondario globale della tabella Employee chiamato `DirectReports`.

Modello di dati del forum di discussione

Questo modello di dati rappresenta un forum di discussione. Utilizzando questo modello, i clienti possono interagire con la community degli sviluppatori, porre domande e rispondere ai post degli altri clienti. Ogni servizio AWS ha un forum dedicato. Chiunque può iniziare un nuovo thread di discussione postando un messaggio in un forum e ogni thread può ricevere un numero illimitato di risposte.

I modelli di accesso agevolati da questo modello di dati sono:

- Recupero di un record del forum utilizzando il nome del forum, agevolato dalla tabella chiamata `Forum`.

- Recupero di un thread specifico o di tutti i thread per un forum, agevolato dalla tabella chiamata `Thread`.
- Ricerca delle risposte utilizzando l'indirizzo e-mail dell'utente autore del post, agevolato dall'indice secondario globale della tabella Reply chiamato `PostedBy-Message-Index`.

Modello di dati della libreria musicale

Questo modello di dati rappresenta una libreria musicale che dispone di una vasta raccolta di canzoni e mostra le canzoni più scaricate in tempo reale.

I modelli di accesso agevolati da questo modello di dati sono:

- Recupero di un record di canzoni, agevolato dalla tabella denominata `Songs`.
- Recupero di un record di download specifico o di tutti i record di download per una canzone, agevolato dalla tabella chiamata `Songs`.
- Recupero di un record di conteggio download mensile specifico o di tutti i record di conteggio download mensili per una canzone, agevolato dalla tabella denominata `Song`.
- Recupero di tutti i record (inclusi record di canzoni, record di download e record di conteggio download mensili) per una canzone, agevolato dalla tabella chiamata `Songs`.
- Ricerca delle canzoni più scaricate, agevolata dall'indice secondario globale della tabella Canzoni chiamato `DownloadsByMonth`.

Modello di dati della stazione sciistica

Questo modello di dati rappresenta una stazione sciistica che dispone di una vasta raccolta di dati per ogni impianto di risalita acquisiti quotidianamente.

I modelli di accesso agevolati da questo modello di dati sono:

- Recupero di tutti i dati relativi a un determinato impianto di risalita o complesso turistico, dinamico e statico, facilitato da una tabella denominata `SkiLifts`.
- Recupero di tutti i dati dinamici (compresi i singoli rider di risalita, la copertura della neve, il pericolo di valanghe e lo stato dell'impianto di risalita) per uno skilift o l'intero resort in una data specifica, agevolato dalla tabella chiamata `SkiLifts`.

- Recupero di tutti i dati statici (incluso se l'impianto di risalita è solo per rider esperti, i metri verticali risaliti dall'impianto e il tempo di risalita) per uno specifico skilift, agevolato da un tavolo chiamato `SkiLifts`.
- Recupero della data della registrazione dei dati per uno specifico impianto di risalita o per l'intero resort ordinati in base al totale dei singoli rider, agevolato dall'indice secondario globale della tabella Skilifts chiamato `SkiLiftsByRiders`.

Modello di dati delle offerte per carta di credito

Questo modello di dati viene utilizzato da un'applicazione di offerte di carta di credito.

Un fornitore di carte di credito produce offerte nel tempo. Queste offerte includono trasferimenti di denaro senza commissioni, maggiori limiti di credito, tassi di interesse più bassi, cash back e miglia aeree. Dopo che un cliente accetta o rifiuta queste offerte, il rispettivo stato dell'offerta viene aggiornato di conseguenza.

I modelli di accesso agevolati da questo modello di dati sono:

- Recupero dei record di conto utilizzando `AccountId`, come agevolato dalla tabella principale.
- Recupero di tutti i conti con poche voci previste, come agevolato dall'indice secondario `AccountIndex`.
- Recupero dei conti e di tutti i record di offerta associati a tali conti utilizzando `AccountId`, come agevolato dalla tabella principale.
- Recupero di conti e record di offerta specifici associati a tali conti utilizzando `AccountId` e `OfferId`, come agevolato dalla tabella principale.
- Recupero di tutti i record di offerta `ACCEPTED/DECLINED` di `OfferType` specifico, associati agli account utilizzando `AccountId`, `OfferType` e `Status`, come agevolato dall'indice secondario `GSI1`.
- Recupero delle offerte e dei record degli articoli dell'offerta associati utilizzando `OfferId`, come agevolato dalla tabella principale.

Modello di dati dei segnalibri

Questo modello di dati viene utilizzato memorizzare i segnalibri per i clienti.

Un cliente può avere molti segnalibri e un segnalibro può appartenere a molti clienti. Questo modello di dati rappresenta una relazione molti-a-molti.

I modelli di accesso agevolati da questo modello di dati sono:

- Una singola query `customerId` può ora restituire i dati dei clienti e i segnalibri.
- Un indice `ByEmail` di query restituisce i dati dei clienti tramite indirizzo e-mail. Tenere presente che i segnalibri non vengono recuperati da questo indice.
- Un indice `ByUrl` di query ottiene i dati dei segnalibri in base all'URL. È stata utilizzata la chiave di ordinamento customerId per l'indice perché lo stesso URL può essere il segnalibro per più clienti.
- Un indice `ByCustomerFolder` di query ottiene segnalibri in base alla cartella per ogni cliente.

Cronologia delle versioni di NoSQL Workbench

La tabella seguente descrive le modifiche importanti apportate a ogni versione dello strumento client NoSQL Workbench.

Versione	Modifica	Descrizione	Data
3.13.5	La modalità di capacità per le impostazioni predefinite della tabella è ora on demand	Quando si crea una tabella con impostazioni predefinite, DynamoDB crea una tabella che utilizza la modalità con capacità on demand anziché la modalità con capacità allocata.	24 febbraio 2025
3.13.0	Miglioramenti a Operation builder di NoSQL Workbench	NoSQL Workbench ora include il supporto nativo per la modalità scura. Sono state migliorate le operazioni su tabelle ed elementi in Operation builder. I risultati degli elementi e le informazioni sulle	24 aprile 2024

Versione	Modifica	Descrizione	Data
		richieste di Operation Builder sono disponibili in formato JSON.	
3.12.0	Clonazione di tabelle con NoSQL Workbench e restituzione della capacità utilizzata	Ora è possibile clonare le tabelle tra un account del servizio web DynamoDB locale e un account del servizio web DynamoDB o tra account del servizio web DynamoDB per iterazioni di sviluppo più rapide. Ora è possibile visualizzare le RCU o le WCU utilizzate dopo l'esecuzione di un'operazione tramite Operation builder. È stato risolto il problema di sovrascrittura dei dati durante l'importazione da un file CSV.	26 febbraio 2024

Versione	Modifica	Descrizione	Data
3.11.0	Miglioramenti di DynamoDB locale	Ora è possibile specificare la porta all'avvio dell'istanza di DynamoDB locale integrata. NoSQL Workbench può ora essere installato su Windows senza diritti di amministratore. Sono stati aggiornati i modelli per i modelli di dati.	17 gennaio 2024
3.10.0	Supporto nativo per Apple SIlicon	NoSQL Workbench ora include il supporto nativo per Mac con Apple SIlicon. Ora è possibile configurare il formato di generazione dei dati di esempio per gli attributi di tipo `Number`.	5 dicembre 2023
3.9.0	Miglioramenti del Data modeler	Il visualizzatore ora supporta il commit di modelli di dati su DynamoDB locale con l'opzione di sovrascrivere le tabelle esistenti.	3 novembre 2023

Versione	Modifica	Descrizione	Data
3.8.0	Generazione di dati di esempio	NoSQL Workbench ora supporta la generazione di dati di esempio per i modelli di dati DynamoDB.	25 settembre 2023
3.6.0	Miglioramenti del generatore di operazioni	Miglioramenti della gestione delle connessioni nel generatore di operazioni. I nomi degli attributi in Data modeler ora possono essere modificati senza eliminare i dati. Altre correzioni di bug.	11 aprile 2023
3.5.0	Supporto per nuove regioni AWS	NoSQL Workbench ora supporta le regioni ap-south-2, ap-southeast-3, ap-southeast-4, eu-central-2, eu-south-2, me-central-1 e me-west-1.	23 febbraio 2023
3.4.0	Supporto per la versione locale di DynamoDB	NoSQL Workbench ora supporta l'installazione della versione locale di DynamoDB come parte del processo di installazione.	6 dicembre 2022

Versione	Modifica	Descrizione	Data
3.3.0	Supporto per operazioni del piano di controllo (control-plane)	Operation Builder ora supporta le operazioni del piano di controllo (control-plane).	1 giugno 2022
3.2.0	Importazione ed esportazione di CSV	Ora è possibile importare dati di esempio da un file CSV nello strumento Visualizer e anche esportare i risultati di una query di Operation Builder in un file CSV.	11 ottobre 2021
3.1.0	Salvataggio delle operazioni	Operation Builder in NoSQL Workbench ora supporta il salvataggio delle operazioni per un utilizzo successivo.	12 luglio 2021
3.0.0	Impostazioni di capacità e importazione/esportazione CloudFormation	NoSQL Workbench per Amazon DynamoDB ora supporta la specifica di una modalità di capacità di lettura/scrittura per le tabelle ed è in grado di importare ed esportare modelli di dati in formato CloudFormation.	21 aprile 2021

Versione	Modifica	Descrizione	Data
2.2.0	Supporto per PartiQL	NoSQL Workbench per Amazon DynamoDB aggiunge il supporto per la creazione di istruzioni PartiQL per DynamoDB.	4 dicembre 2020
1.1.0	Supporto per Linux.	NoSQL Workbench per Amazon DynamoDB è supportato in Linux-Ubuntu, Fedora e Debian.	4 maggio 2020
1.0.0	NoSQL Workbench per Amazon DynamoDB – GA.	NoSQL Workbench per Amazon DynamoDB è disponibile al pubblico.	2 marzo 2020
0.4.1	Supporto per i ruoli IAM e le credenziali di sicurezza temporane e.	NoSQL Workbench per Amazon DynamoDB aggiunge il supporto per i ruoli AWS Identity and Access Managemen t (IAM) e le credenzia li di sicurezza temporanee.	19 dicembre 2019

Versione	Modifica	Descrizione	Data
0.3.1	Supporto per la versione locale di DynamoDB (versione scaricabile).	NoSQL Workbench ora supporta la connessione alla versione locale di DynamoDB (versione scaricabile) per progettare, creare, eseguire query e gestire tabelle DynamoDB.	8 Novembre 2019
0.2.1	Non è stata rilasciata nessuna anteprima di NoSQL Workbench.	Questa è la versione iniziale di NoSQL Workbench per DynamoDB. Utilizza NoSQL Workbench per progettare, creare, interrogare e gestire tabelle DynamoDB.	16 settembre 2019

Backup e ripristino per DynamoDB

DynamoDB offre backup on demand e recupero point-in-time (PITR) per proteggere i dati DynamoDB da eventi di emergenza e offre l'archiviazione dei dati per la conservazione a lungo termine. È possibile eseguire il backup di tabelle di dimensioni variabili da pochi megabyte a centinaia di terabyte di dati senza nessun impatto sulle prestazioni e sulla disponibilità per le applicazioni di produzione. Tutti i backup sono automaticamente crittografati, catalogati e facilmente individuabili.

Con i backup on demand è possibile creare un backup istantaneo della propria tabella, che DynamoDB archivia e gestisce. I costi vengono addebitati in base alle dimensioni e alla durata dei backup. Utilizzando il backup on demand è possibile ripristinare l'intera tabella DynamoDB allo stato esatto in cui si trovava al momento della creazione del backup.

Sono disponibili due opzioni per creare e gestire i backup on demand di DynamoDB:

- DynamoDB
- AWS Backup

Puoi utilizzare la funzionalità di backup on demand di DynamoDB per creare backup completi delle tabelle per la conservazione e l'archiviazione a lungo termine per esigenze di conformità alle normative. È possibile eseguire il backup e ripristinare i dati della tabella in qualsiasi momento nella Console di gestione AWS o con una singola chiamata API.

I backup con recupero point-in-time (PITR) sono completamente gestiti da DynamoDB e forniscono fino a 35 giorni di punti di recupero con una granularità al secondo. Per utilizzare il recupero point-in-time, ossia backup continui, abilita il recupero point-in-time (PITR) sulla tabella DynamoDB. I costi vengono addebitati in base alle dimensioni della tabella DynamoDB per il periodo in cui il PITR è abilitato sulla tabella. L'abilitazione del recupero point-in-time (PITR) sulla tabella Amazon DynamoDB consente di eseguire backup continui dei dati. Ciò consente di ripristinare la tabella a un momento specifico all'interno del periodo di recupero PITR creando una nuova tabella DynamoDB con lo stato esatto della tabella originale in quel momento.

Il ripristino point-in-time (PITR) ti permette di proteggere le tabelle DynamoDB da operazioni di scrittura o eliminazione accidentali. Grazie al ripristino point-in-time, non devi preoccuparti di creare, gestire o programmare i backup on-demand. Ad esempio, supponi che uno script di prova scriva accidentalmente su una tabella DynamoDB di produzione.

Grazie al recupero point-in-time è possibile ripristinare la tabella a un momento qualsiasi negli ultimi 35 giorni. È possibile impostare il periodo di recupero su qualsiasi valore compreso tra 1 e 35 giorni. Dopo aver abilitato il recupero point-in-time il ripristino è possibile in qualsiasi momento compreso tra cinque minuti prima del momento attuale e l'intero periodo di recupero configurato. DynamoDB conserva i backup incrementali della tabella.

Inoltre, le operazioni point-in-time non influiscono sulle prestazioni o sulle latenze delle API.

È possibile eseguire il ripristino point-in-time di una tabella DynamoDB utilizzando la Console di gestione AWS, AWS Command Line Interface (AWS CLI) o l'API DynamoDB. Il processo di ripristino point-in-time (PITR) esegue sempre il ripristino in una nuova tabella.

Per informazioni sulla disponibilità e i prezzi delle regioni AWS, consulta Prezzi di Amazon DynamoDB.

> **Note**
>
> - L'assegnazione di tag e il controllo degli accessi basato su attributi (ABAC) non sono supportati per i backup DynamoDB. Per utilizzare l'ABAC con i backup, si consiglia di utilizzare AWS Backup.
> - I tag non vengono conservati nelle tabelle ripristinate. È necessario aggiungere tag alle tabelle ripristinate prima di poter utilizzare le condizioni basate sui tag nelle policy.

Il seguente video fornirà un'introduzione al concetto di backup e ripristino e illustrerà il ripristino point-in-time.

Backup e ripristino

Argomenti

- Backup point-in-time per DynamoDB
- Utilizzo del backup e del ripristino on demand di DynamoDB
- Informazioni sulla fatturazione di Amazon DynamoDB per i backup
- Ripristino di una tabella in DynamoDB
- Utilizzo di AWS Backup con Dynamo DB

Backup point-in-time per DynamoDB

Il recupero point-in-time (PITR) di Amazon DynamoDB offre backup continui automatici dei dati contenuti nelle tabelle DynamoDB. I backup con recupero point-in-time (PITR) sono completamente gestiti da DynamoDB e forniscono fino a 35 giorni di punti di recupero con una granularità al secondo. Grazie al ripristino point-in-time, non devi preoccuparti di creare, gestire o programmare i backup on-demand. In questa sezione viene fornita una panoramica di come funziona il processo in DynamoDB.

Prima di iniziare

Prima di abilitare il ripristino point-in-time (PITR) su una tabella Amazon DynamoDB, considerare i seguenti aspetti:

- L'impostazione di `RecoveryPeriodinDays` consente di abbreviare il periodo di tempo durante il quale vengono eseguiti backup continui. Per impostazione predefinita, il `RecoveryPeriodinDays` è 35. Tuttavia, è possibile impostarlo su qualsiasi valore compreso tra 1 e 35. L'abbreviazione di `RecoveryPeriodinDays` non ha alcun impatto sulla determinazione dei prezzi del recupero point-in-time (PITR) perché il prezzo si basa sulla dimensione della tabella e degli indici secondari locali.
- Se il ripristino point-in-time di una tabella viene disattivato e riattivato in un secondo tempo, viene reimpostata la data iniziale a cui può essere ripristinata. Di conseguenza, puoi ripristinare immediatamente la tabella solo usando il valore `LatestRestorableDateTime`.
- Puoi attivare il ripristino point-in-time su ciascuna replica locale di una tabella globale. Quando recuperi la tabella, il backup esegue il ripristino su una tabella indipendente che non fa parte della tabella globale. Se sono in uso le Tabelle globali versione 2019.11.21 (Corrente), è possibile creare una nuova tabella globale da quella ripristinata. Per ulteriori informazioni, consulta Come funzionano le tabelle globali DynamoDB.
- Inoltre è possibile ripristinare i dati della tabella DynamoDB nelle regioni AWS in modo che la tabella ripristinata venga creata in una regione diversa da quella in cui si trova la tabella di origine. È possibile eseguire ripristini tra regioni commerciali AWS, regioni AWS Cina e regioni AWS GovCloud (Stati Uniti). I prezzi sono calcolati solo in base ai dati trasferiti fuori dalla regione di origine e al ripristino in una nuova tabella nella regione di destinazione.
- AWS CloudTrail registra tutte le operazioni della console e delle API per il ripristino point-in-time per attivare la registrazione, il monitoraggio continuo e il controllo. Per ulteriori informazioni, consulta Registrazione delle operazioni di DynamoDB con AWS CloudTrail.

Argomenti

- Abilitazione del recupero point-in-time in DynamoDB

Abilitazione del recupero point-in-time in DynamoDB

Il ripristino point-in-time (PITR) di Amazon DynamoDB fornisce backup automatici dei dati contenuti nelle tabelle DynamoDB. In questa sezione viene fornita una panoramica di come funziona il processo in DynamoDB.

> **Note**
>
> DynamoDB addebita il recupero point-in-time (PITR) in base alla dimensione di ogni tabella DynamoDB, inclusi i dati della tabella e gli indici secondari locali. La modifica della finestra di recupero (ad esempio, da 35 giorni a 1 giorno) non riduce il prezzo. Il costo rimane lo stesso indipendentemente dal periodo di recupero scelto. Il periodo di recupero massimo configurato non influisce sul prezzo addebitato per l'attivazione del PITR. Per determinare i costi di backup, DynamoDB monitora continuamente le dimensioni delle tabelle con PITR attivato. L'utilizzo del PITR viene fatturato fino al momento della relativa disattivazione per ogni tabella.

Argomenti

- Abilitazione ripristino point-in-time (PITR)
- Abilitazione del PITR (console)
- Abilitazione del PITR (AWS CLI)
- Abilitazione del PITR (CloudFormation)
- Abilitazione del PITR (API)
- Periodo di recupero
- Modifica del PITR
- Eliminazione di una tabella con PITR abilitato

Abilitazione ripristino point-in-time (PITR)

È possibile abilitare il ripristino point-in-time (PITR) utilizzando la Console di gestione AWS, AWS Command Line Interface (AWS CLI) o l'API DynamoDB. Quando è attivato, il ripristino point-in-time fornisce backup continui finché non lo disattivi in modo esplicito.

Dopo aver abilitato il recupero point-in-time, è possibile eseguirlo in qualsiasi momento tra `EarliestRestorableDateTime` e `LatestRestorableDateTime`. `LatestRestorableDateTime` è in genere 5 minuti prima dell'ora corrente. Per ulteriori informazioni, consulta Ripristino point-in-time (PITR) di una tabella DynamoDB.

> **Note**
>
> Questo processo esegue sempre il ripristino in una nuova tabella.

Abilitazione del PITR (console)

Come abilitare il PITR tramite la console DynamoDB

1. Vai alla console DynamoDB.
2. Seleziona Tabelle dalla barra di navigazione a sinistra e seleziona la tabella DynamoDB.
3. Nella scheda Backup, per l'opzione Ripristino point-in-time (PITR), seleziona Modifica.
4. Seleziona Attiva ripristino point-in-time.
5. Seleziona un valore compreso tra 1 e 35 per il periodo di recupero. Indica il periodo di tempo massimo per il quale il backup continuo è recuperabile.

Abilitazione del PITR (AWS CLI)

> **Note**
>
> In caso di errori durante l'esecuzione dei comandi AWS CLI, consulta Troubleshoot AWS CLI errors. Assicurati di utilizzare la versione della AWS CLI più recente.

Esegui il comando update-continuous-backups con l'impostazione point-in-time-recovery-specifications attivata:

```
aws dynamodb update-continuous-backups \
--table-name Music \
--point-in-time-recovery-specification
 PointInTimeRecoveryEnabled=true,RecoveryPeriodInDays=35
```

Abilitazione del PITR (CloudFormation)

Utilizza la risorsa AWS::DynamoDB::Table con la proprietà `PointInTimeRecoverySpecification` attivata:

```
Resources:
  iotCatalog:
    Type: AWS::DynamoDB::Table
      Properties:
      ...
      PointInTimeRecoverySpecification:
        PointInTimeRecoveryEnabled: true
        RecoveryPeriodInDays: 35
```

Esempio di sintassi di richiesta:

```
{
   "PointInTimeRecoverySpecification": {
      "PointInTimeRecoveryEnabled": boolean,
      "RecoveryPeriodInDays: number
   },
   "TableName": "string"
}
```

Abilitazione del PITR (API)

Esegui l'operazione API UpdateContinuousBackups con il parametro `PointInTimeRecoverySpecification` attivato.

Esempio di sintassi di richiesta:

```
{
   "PointInTimeRecoverySpecification": {
      "PointInTimeRecoveryEnabled": boolean,
      "RecoveryPeriodInDays" : number
   },
   "TableName": "string"
```

```
}
```

Esempio di sintassi di risposta:

```
{
   "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 di recupero

È possibile impostare il periodo di recupero su un valore compreso tra 1 e 35 giorni. Questo `RecoveryPeriodInDays` determina il periodo di tempo per il quale vengono mantenuti i backup continui. Ad esempio, se si imposta questo valore su 30 giorni, si avrà la possibilità di ripristinare la tabella solo in un qualsiasi momento degli ultimi 30 giorni.

> **Note**
>
> DynamoDB addebita il recupero point-in-time (PITR) in base alla dimensione di ogni tabella DynamoDB, inclusi i dati della tabella e gli indici secondari locali. Il periodo di recupero

massimo configurato non influisce sul prezzo addebitato per l'attivazione del PITR. Per i dettagli sui prezzi, consulta Prezzi di DynamoDB.

Modifica del PITR

È possibile modificare l'impostazione del PITR sulla tabella e modificare il periodo di recupero. Se si modifica il periodo di recupero e lo si aumenta a un valore superiore a quello impostato in precedenza, `EarliestRestorePoint` non verrà modificato immediatamente. Poiché il periodo di recupero è una finestra variabile, DynamoDB continuerà a eseguire backup automatici fino al raggiungimento del nuovo periodo aumentato. Se si modifica il periodo di recupero e lo si riduce a un valore inferiore a quello impostato in precedenza, `EarliestRestorePoint` verrà immediatamente ridotto per allinearsi al periodo di recupero e qualsiasi backup continuo che non rientra nel nuovo valore impostato non sarà recuperabile.

Eliminazione di una tabella con PITR abilitato

Quando si elimina una tabella che dispone di un ripristino point-in-time abilitato, DynamoDB crea automaticamente uno snapshot di backup, denominato backup di sistema che viene conservato per 35 giorni (senza costi aggiuntivi). È possibile utilizzare i backup di sistema per ripristinare una tabella eliminata allo stato in cui era nel momento precedente l'eliminazione. Tutti i backup di sistema seguono una convenzione di denominazione standard per *`table-name`*`$DeletedTableBackup`.

> **Note**
>
> Una volta eliminata una tabella con il recupero point-in-time abilitato, è possibile utilizzare il backup del sistema per ripristinare quella tabella a un singolo punto temporale. Il backup del sistema verrà creato al momento dell'eliminazione della tabella ed è uno snapshot della tabella immediatamente prima della sua eliminazione.

Utilizzo del backup e del ripristino on demand di DynamoDB

Amazon DynamoDB supporta funzionalità autonome di backup e ripristino on demand. Queste funzionalità sono disponibili indipendentemente dall'utilizzo o meno di AWS Backup.

È possibile utilizzare la funzionalità di backup on demand di DynamoDB per creare backup completi delle tabelle per la conservazione e l'archiviazione a lungo termine per esigenze di conformità alle normative. È possibile eseguire il backup e il ripristino dei dati della tabella in qualsiasi momento con

un solo clic sulla console di AWS gestione o con una singola chiamata API. Le operazioni di backup e ripristino vengono eseguite con impatto zero sulle prestazioni o sulla disponibilità della tabella.

È possibile creare backup di tabelle utilizzando la console, l'interfaccia a riga di AWS comando (AWS CLI) o l'API DynamoDB. Per ulteriori informazioni, consulta Backup di una tabella DynamoDB.

Per ulteriori informazioni sul ripristino di una tabella da un backup, vedi Ripristino di una tabella DynamoDB da un backup.

Backup e ripristino di tabelle DynamoDB con DynamoDB: funzionamento

Puoi utilizzare la caratteristica di backup on demand DynamoDB per creare backup completi delle tabelle Amazon DynamoDB. Questa caratteristica è disponibile indipendentemente da AWS backup. In questa sezione viene fornita una panoramica di ciò che accade durante il processo di backup e ripristino di DynamoDB.

Backup

Quando crei un backup on demand con DynamoDB, viene catalogato un indicatore temporale della richiesta. Il backup viene creato in modo asincrono applicando tutte le modifiche all'ultima snapshot di tabella completa, fino al momento in cui viene effettuata la richiesta. Le richieste di backup di DynamoDB sono elaborate istantaneamente e diventano disponibili per il ripristino in pochi minuti.

> **Note**
>
> Ogni volta che crei un backup on-demand, viene eseguito il backup di tutti i dati della tabella. Non vi è un limite per il numero di backup on-demand che possono essere effettuati.

Tutti i backup in DynamoDB funzionano senza utilizzare alcuna velocità effettiva assegnata sulla tabella.

I backup DynamoDB non garantiscono la coerenza tra gli elementi. Tuttavia, il divario fra gli aggiornamenti in un backup è generalmente molto meno di un secondo.

Mentre è in corso un backup, non puoi effettuare le seguenti operazioni:

- Sospendere o annullare l'operazione di backup;
- Eliminare la tabella di origine del backup;
- Disabilitare i backup su una tabella se uno di questi è in corso.

Se non si desidera creare script di pianificazione e processi di pulizia, è possibile utilizzare AWS Backup per creare piani di backup con pianificazioni e policy di conservazione per le tabelle DynamoDB. AWS Backup esegue i backup e li elimina alla scadenza. Per ulteriori informazioni, consulta la Guida per gli sviluppatori di AWS Backup.

Oltre a AWS Backup, puoi pianificare backup periodici o futuri utilizzando le caratteristiche di AWS Lambda. Per ulteriori informazioni, consulta il post di blog Una soluzione serverless per programmare il tuo backup on demand di Amazon DynamoDB.

Se si utilizza la console, gli eventuali backup creati con AWS Backup verranno elencati nella scheda Backups (Backup) con Backup type (Tipo di backup) impostato su `AWS`.

> **Note**
>
> Non è possibile eliminare i backup contrassegnati con un tipo di backup di AWS utilizzando la console DynamoDB. Per gestire questi backup, utilizza la console AWS Backup.

Per scoprire come eseguire un backup, consulta Backup di una tabella DynamoDB.

Ripristini

Una tabella viene ripristinata senza utilizzare alcun throughput assegnato nella tabella. È possibile eseguire un ripristino completo della tabella dal backup DynamoDB oppure configurare le impostazioni della tabella di destinazione. Quando esegui un ripristino, puoi modificare le seguenti impostazioni della tabella:

- Indici secondari globali (GSI)
- Indici secondari locali (LSI)
- Modalità di fatturazione
- Capacità di lettura e scrittura di cui è stato effettuato il provisioning
- Impostazioni di crittografia

> **Important**
>
> Quando esegui un ripristino completo, la tabella di destinazione è impostata con le stesse unità di capacità di lettura e di scrittura di cui è stato effettuato il provisioning rispetto alla

tabella di origine, poiché sono registrate al momento della richiesta del backup. Il processo di ripristino ripristina anche gli indici secondari locali e gli indici secondari globali.

Inoltre è possibile ripristinare i dati della tabella DynamoDB nelle regioni AWS in modo che la tabella ripristinata venga creata in una regione diversa da quella in cui si trova il backup. È possibile eseguire ripristini tra regioni commerciali AWS, regioni AWS Cina e regioni AWS GovCloud (Stati Uniti). I prezzi sono calcolati solo in base ai dati trasferiti fuori dalla regione di origine e al ripristino in una nuova tabella nella regione di destinazione.

I ripristini possono essere più veloci ed economici se scegli di escludere alcuni o tutti gli indici secondari dalla creazione nella nuova tabella ripristinata.

È necessario configurare manualmente nella tabella ripristinata quanto segue:

- Policy di scalabilità automatica
- Policy AWS Identity and Access Management (IAM)
- Parametri e allarmi di Amazon CloudWatch
- Tag
- Impostazioni flusso
- Impostazioni Time to Live (TTL)
- Impostazioni di protezione dall'eliminazione
- Impostazioni Ripristino point-in-time (PITR)

Puoi ripristinare solo tutti i dati della tabella in una nuova tabella a partire da un backup. Puoi scrivere nella tabella ripristinata solo dopo che si attiva.

> **Note**
>
> Non puoi sovrascrivere una tabella esistente durante un'operazione di ripristino.

I parametri di servizio mostrano che il 95% dei ripristini delle tabelle dei clienti viene completato in meno di un'ora. Tuttavia, i tempi di ripristino sono direttamente correlati alla configurazione delle tabelle (ad esempio la dimensione delle tabelle e il numero di partizioni sottostanti) e ad altre variabili correlate. Una best practice quando si pianifica un ripristino di emergenza consiste nel documentare

regolarmente i tempi medi di completamento del ripristino e stabilire in che modo questi tempi influiscono sull'obiettivo del tempo di ripristino complessivo.

Per scoprire come eseguire un ripristino, consulta Ripristino di una tabella DynamoDB da un backup.

È possibile utilizzare le policy IAM per il controllo degli accessi. Per ulteriori informazioni, consulta Utilizzo di IAM con backup e ripristino di DynamoDB.

Tutte le console di backup e ripristino e le operazioni API vengono acquisite e registrate in AWS CloudTrail per la registrazione, il monitoraggio e l'audit.

Backup di una tabella DynamoDB

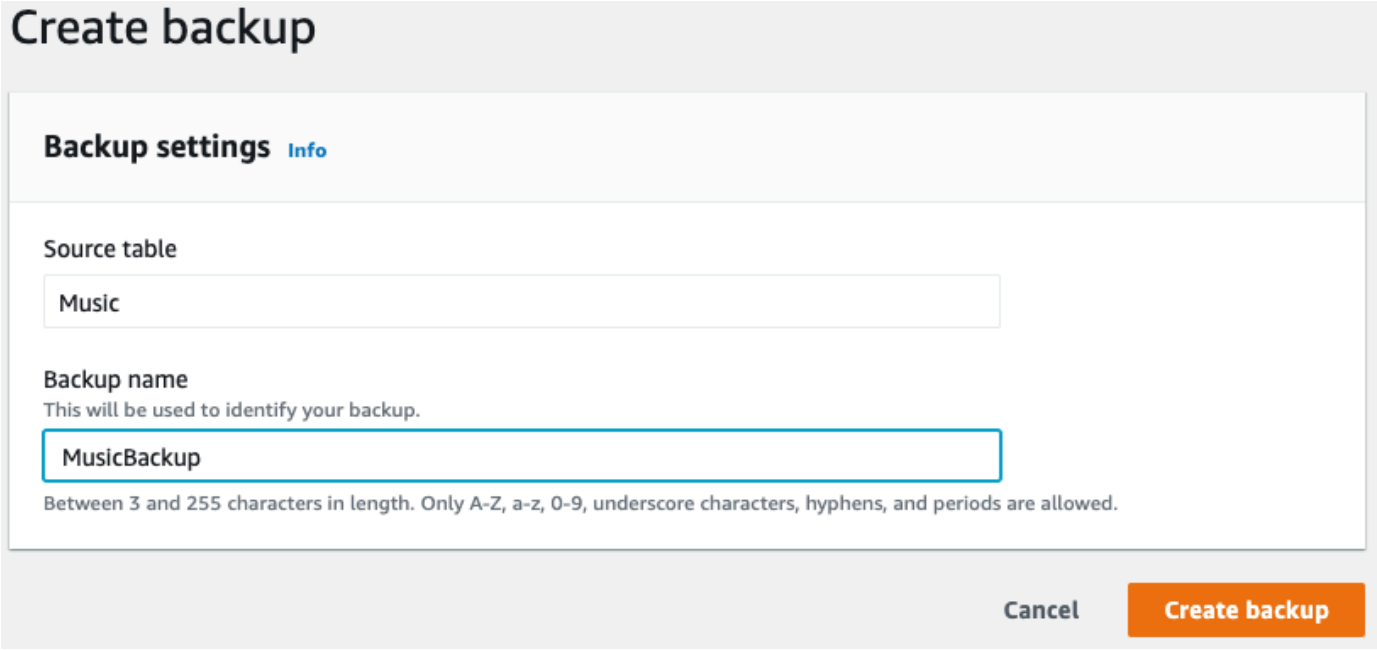
In questa sezione viene descritto come usare la console Amazon DynamoDB o AWS Command Line Interface per eseguire il backup di una tabella.

Creazione di un backup di tabella (console)

Segui queste fasi per creare un backup denominato `MusicBackup` per una tabella `Music` esistente usando la Console di gestione AWS.

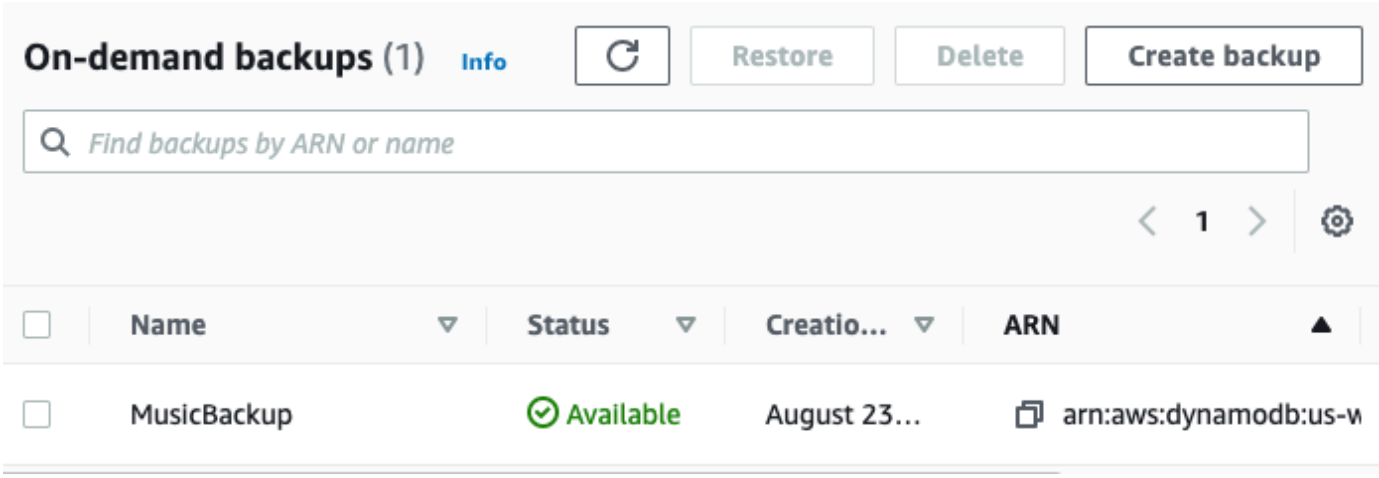
Per creare un backup della tabella

1. Accedi alla console Console di gestione AWS e apri la console DynamoDB all'indirizzo https://console.aws.amazon.com/elasticache/.
2. Puoi creare un backup tramite una delle seguenti operazioni:
   - Nella scheda Backup della tabella `Music`, scegli Crea backup.
   - Nel riquadro di navigazione sul lato sinistro della console scegliere Backups (Backup). Quindi scegli Crea backup.
3. Assicurati che `Music` sia il nome della tabella e specifica **`MusicBackup`** per il nome del backup. Seleziona Crea backup per creare il backup.



> **Note**
>
> Se i backup vengono creati utilizzando la sezione Backup del pannello di navigazione, la tabella non è preselezionata. Devi scegliere manualmente il nome della tabella di origine per il backup.

Durante la creazione del backup, lo stato del backup è impostato su Creating (Creazione in corso). Quando il backup è completato, lo stato del backup diventa Available (Disponibile).



Creazione di un backup di tabella (AWS CLI)

Seguire queste fasi per creare un backup per la tabella `Music` esistente usando l'AWS CLI.

Per creare un backup della tabella

- Crea un backup con il nome `MusicBackup` per la tabella `Music`:

```
aws dynamodb create-backup --table-name Music \
 --backup-name MusicBackup
```

  Durante la creazione del backup, lo stato del backup è impostato su `CREATING`:

```
{
    "BackupDetails": {
        "BackupName": "MusicBackup",
        "BackupArn": "arn:aws:dynamodb:us-east-1:123456789012:table/Music/
backup/01489602797149-73d8d5bc",
        "BackupStatus": "CREATING",
        "BackupCreationDateTime": 1489602797.149
    }
}
```

Al termine del backup, `BackupStatus` diventa `AVAILABLE`. Per verificare, usa il comando `describe-backup`. È possibile ottenere il valore di input di `backup-arn` dall'output della fase precedente o utilizzando il comando `list-backups`.

```
aws dynamodb describe-backup \
--backup-arn arn:aws:dynamodb:us-east-1:123456789012:table/Music/backup/01489173575360-
b308cd7d
```

Per tenere traccia dei tuoi backup, puoi utilizzare il comando `list-backups`. Elenca tutti i backup che sono nello stato `CREATING` o `AVAILABLE`.

```
aws dynamodb list-backups
```

Il comando `list-backups` e il comando `describe-backup` sono utili per controllare le informazioni sulla tabella di origine del backup.

Ripristino di una tabella DynamoDB da un backup

In questa sezione viene descritto come ripristinare una tabella da un backup usando la console Amazon DynamoDB o AWS Command Line Interface (AWS CLI).

> **Note**
>
> Se si desidera usare la AWS CLI, è necessario innanzitutto configurarla. Per ulteriori informazioni, consulta Accesso a DynamoDB.

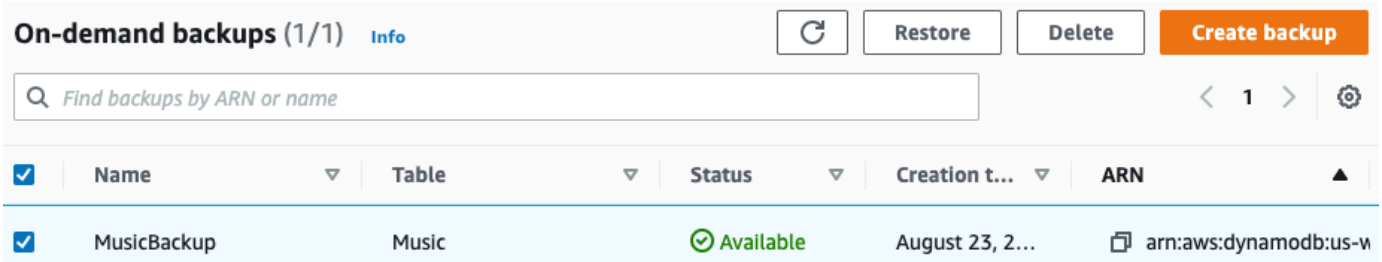
Ripristino di una tabella da un backup (console)

La procedura seguente illustra come ripristinare la tabella `Music` usando il file `MusicBackup` creato nel tutorial Backup di una tabella DynamoDB.

> **Note**
>
> In questa procedura si presuppone che la tabella `Music` non esista più prima del suo ripristino tramite il file `MusicBackup`.

Per ripristinare una tabella da un backup

1. Accedi alla console Console di gestione AWS e apri la console DynamoDB all'indirizzo https://console.aws.amazon.com/elasticache/.
2. Nel riquadro di navigazione sul lato sinistro della console scegliere Backups (Backup).
3. Nell'elenco di backup scegli `MusicBackup`.



4. Scegli Restore (Ripristina).
5. Come nuovo nome di tabella, immettere **`Music`**. Conferma il nome e altri dettagli del backup. Scegli quindi Restore table (Ripristina tabella) per avviare il processo di ripristino.

> **Note**
>
> È possibile ripristinare la tabella nella stessa regione AWS o in una regione diversa da quella dove si trova il backup. È possibile escludere la creazione di indici secondari nella nuova tabella ripristinata. Inoltre, puoi specificare una modalità di crittografia diversa. Le tabelle ripristinate dai backup vengono sempre create utilizzando la classe di tabella DynamoDB Standard.

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 tabella che viene ripristinata è visualizzata con lo stato Creating (Creazione). Al termine del processo di ripristino, lo stato della tabella `Music` cambia in Active (Attivo).

Ripristino di una tabella da un backup (AWS CLI)

Segui queste fasi per usare l'AWS CLI per ripristinare la tabella `Music` usando l'oggetto `MusicBackup` creato nel tutorial Backup di una tabella DynamoDB.

Per ripristinare una tabella da un backup

1. Conferma il backup da ripristinare usando il comando `list-backups`. Questo esempio usa `MusicBackup`.

   ```
   aws dynamodb list-backups
   ```

   Per ottenere dettagli aggiuntivi per il backup, usa il comando `describe-backup`. Puoi ottenere il valore di input di `backup-arn` dalla fase precedente:

   ```
   aws dynamodb describe-backup \
   --backup-arn arn:aws:dynamodb:us-east-1:123456789012:table/Music/
   backup/01489173575360-b308cd7d
   ```

2. Ripristina la tabella dal backup. In questo caso, `MusicBackup` ripristina la tabella `Music` nella stessa regione 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. Ripristina la tabella dal backup con le impostazioni personalizzate della tabella. In questo caso, `MusicBackup` ripristina la tabella `Music` e specifica una modalità di crittografia per la tabella ripristinata.

> **Note**
>
> Il parametro `sse-specification-override` accetta gli stessi valori del parametro `sse-specification-override` utilizzato nel comando `CreateTable`. Per ulteriori informazioni, consulta Gestione di tabelle crittografate in 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
```

È possibile ripristinare la tabella in una regione AWSdiversa da quella dove si trova il backup.

> **Note**
>
> - Il parametro `sse-specification-override` è obbligatorio per i ripristini tra regioni ma facoltativo per i ripristini nella stessa regione della tabella di origine.
> - Quando si esegue un ripristino tra regioni dalla riga di comando, è necessario impostare la regione AWS predefinita sulla regione di destinazione desiderata. Per ulteriori informazioni, consulta Opzioni della riga di comando nella Guida per l'utente di 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
```

Puoi ignorare la modalità di fatturazione e la velocità effettiva fornita per la tabella di ripristino.

```
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
```

È possibile escludere la creazione di alcuni o tutti gli indici secondari nella nuova tabella ripristinata.

> **Note**
>
> I ripristini possono essere più veloci ed economici se escludi alcuni o tutti gli indici secondari dalla creazione nella tabella ripristinata.

```
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**
>
> Gli indici secondari forniti devono corrispondere agli indici esistenti. Non è possibile creare nuovi indici al momento del ripristino.

È possibile utilizzare una combinazione di diverse sostituzioni. Ad esempio, puoi utilizzare un singolo indice secondario globale e modificare contemporaneamente il throughput assegnato, come segue.

```
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
```

Per verificare il ripristino, usa il comando `describe-table` per descrivere la tabella `Music`.

```
aws dynamodb describe-table --table-name Music
```

La tabella che viene ripristinata dal backup è visualizzata con lo stato Creating (Creazione). Al termine del processo di ripristino, lo stato della tabella `Music` cambia in Active (Attivo).

> **Important**
>
> Mentre è in corso il ripristino, non modificare o eliminare la policy del ruolo IAM, altrimenti potrebbe verificarsi un comportamento imprevisto. Supponi, ad esempio, di rimuovere le autorizzazioni di scrittura per una tabella mentre è in corso il ripristino. In questo caso, l'operazione `RestoreTableFromBackup` sottostante non sarebbe in grado di scrivere i dati ripristinati nella tabella.
> Al termine dell'operazione di ripristino, sarà possibile modificare o eliminare la policy del ruolo IAM.
> Le policy IAM che coinvolgono restrizioni sull'IP di origine per l'accesso alla tabella di ripristino di destinazione dovrebbero avere il set di chiavi `aws:ViaAWSService`impostato su `false` per garantire che le restrizioni si applichino solo alle richieste effettuate direttamente da un principal. In caso contrario, il ripristino verrà annullato.
> Se il backup è crittografato con una Chiave gestita da AWS o una chiave gestita dal cliente, non disabilitare o eliminare la chiave mentre è in corso un ripristino, altrimenti non riuscirà.
> Al termine dell'operazione di ripristino è possibile modificare la chiave di crittografia per la tabella ripristinata e disabilitare o eliminare la vecchia chiave.

Eliminazione di un backup di una tabella DynamoDB

In questa sezione viene descritto come utilizzare Console di gestione AWS o AWS Command Line Interface (AWS CLI) per eliminare un backup della tabella Amazon DynamoDB.

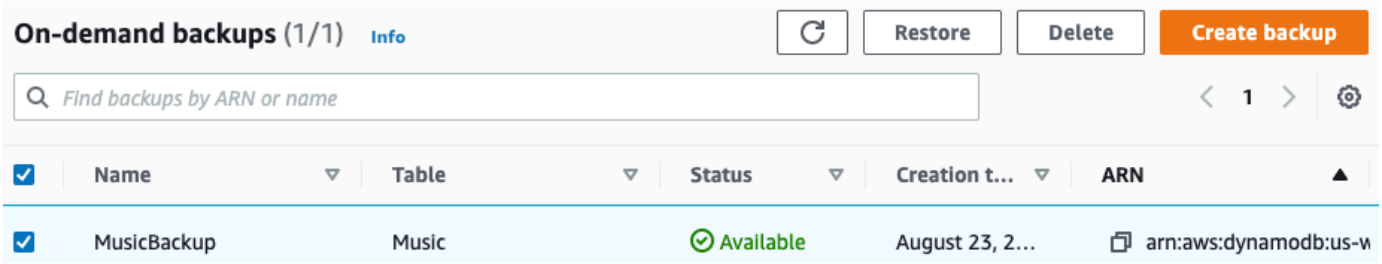
> **Note**
>
> Se desideri utilizzare la AWS CLI, dovrai prima configurarla. Per ulteriori informazioni, consulta Utilizzando il AWS CLI.

Eliminazione di un backup di tabella (console)

La procedura seguente illustra come eliminare il file `MusicBackup` creato nel corso del tutorial Backup di una tabella DynamoDB.

Per eliminare un backup

1. Accedi alla console Console di gestione AWS e apri la console DynamoDB all'indirizzo https://console.aws.amazon.com/elasticache/.
2. Nel riquadro di navigazione sul lato sinistro della console scegliere Backups (Backup).
3. Nell'elenco di backup scegli `MusicBackup`.



4. Scegliere Delete (Elimina). Conferma di voler eliminare il backup digitando **`delete`** e facendo clic su Detele (Elimina).

Eliminazione di un backup di tabella (AWS CLI)

L'esempio seguente elimina il backup di una tabella `Music` esistente utilizzando l'AWS CLI.

```
aws dynamodb delete-backup \
--backup-arn arn:aws:dynamodb:us-east-1:123456789012:table/Music/
backup/01489602797149-73d8d5bc
```

Utilizzo di IAM con backup e ripristino di DynamoDB

Puoi utilizzare AWS Identity and Access Management (IAM) per limitare le azioni di backup e ripristino di Amazon DynamoDB per alcune risorse. Le `RestoreTableFromBackup` APIs operazioni `CreateBackup` e vengono eseguite in base alle singole tabelle.

Per ulteriori informazioni sull'uso di policy IAM in DynamoDB, consulta Policy basate su identità per DynamoDB.

Di seguito sono riportati esempi di policy IAM che è possibile utilizzare per configurare funzionalità specifiche di backup e ripristino in DynamoDB.

Esempio 1: consentire le CreateBackup azioni e RestoreTableFromBackup

La seguente policy IAM concede le autorizzazioni per consentire le operazioni DynamoDB `CreateBackup` e `RestoreTableFromBackup` su tutte le tabelle.

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

Le autorizzazioni RestoreTableFromBackup DynamoDB sono necessarie sul backup di origine e le autorizzazioni di lettura e scrittura di DynamoDB sulla tabella di destinazione sono necessarie per ripristinare la funzionalità.
Le autorizzazioni RestoreTableToPointInTime DynamoDB sono necessarie nella tabella di origine e le autorizzazioni di lettura e scrittura di DynamoDB sulla tabella di destinazione sono necessarie per ripristinare la funzionalità.

Esempio 2: Consenti e nega CreateBackup RestoreTableFromBackup

La seguente policy IAM concede le autorizzazioni per l'operazione `CreateBackup` e rifiuta l'operazione `RestoreTableFromBackup`:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["dynamodb:CreateBackup"],
            "Resource": "*"
        },
        {
            "Effect": "Deny",
            "Action": ["dynamodb:RestoreTableFromBackup"],
            "Resource": "*"
        }

    ]
}
```

Esempio 3: consentire ListBackups e negare CreateBackup e RestoreTableFromBackup

La seguente policy IAM concede le autorizzazioni per l'operazione `ListBackups` e rifiuta le operazioni `CreateBackup` e `RestoreTableFromBackup`:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["dynamodb:ListBackups"],
            "Resource": "*"
```

```
        },
        {
            "Effect": "Deny",
            "Action": [
                "dynamodb:CreateBackup",
                "dynamodb:RestoreTableFromBackup"
            ],
            "Resource": "*"
        }

    ]
}
```

Esempio 4: consentire ListBackups e negare DeleteBackup

La seguente policy IAM concede le autorizzazioni per l'operazione `ListBackups` e rifiuta l'operazione `DeleteBackup`:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["dynamodb:ListBackups"],
            "Resource": "*"
        },
        {
            "Effect": "Deny",
            "Action": ["dynamodb:DeleteBackup"],
            "Resource": "*"
        }

    ]
}
```

Esempio 5: Consenti RestoreTableFromBackup e DescribeBackup per tutte le risorse e nega DeleteBackup per un backup specifico

La seguente policy IAM concede le autorizzazioni per le operazioni `RestoreTableFromBackup` e `DescribeBackup` e rifiuta l'operazione `DeleteBackup` per una risorsa di backup specifica.

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**
>
> Le autorizzazioni RestoreTableFromBackup DynamoDB sono necessarie sul backup di origine e le autorizzazioni di lettura e scrittura di DynamoDB sulla tabella di destinazione sono necessarie per ripristinare la funzionalità.
> Le autorizzazioni RestoreTableToPointInTime DynamoDB sono necessarie nella tabella di origine e le autorizzazioni di lettura e scrittura di DynamoDB sulla tabella di destinazione sono necessarie per ripristinare la funzionalità.

Esempio 6: Consenti una tabella specifica CreateBackup

La seguente policy IAM concede le autorizzazioni per l'operazione `CreateBackup` solo sulla tabella `Movies`:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["dynamodb:CreateBackup"],
            "Resource": [
                "arn:aws:dynamodb:us-east-1:123456789012:table/Movies"
            ]
        }
    ]
}
```

Esempio 7: Consenti ListBackups

La policy IAM seguente concede le autorizzazioni per l'operazione `ListBackups`:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["dynamodb:ListBackups"],
            "Resource": "*"
        }
    ]
}
```

> **Important**
>
> Non è possibile concedere le autorizzazioni per l'operazione `ListBackups` su una tabella specifica.

Esempio 8: consentire l'accesso alle AWS Backup funzionalità

Avrai bisogno delle autorizzazioni API per l'azione `StartAwsBackupJob` per un backup corretto con caratteristiche avanzate e per l'azione `dynamodb:RestoreTableFromAwsBackup` per ripristinare correttamente il backup.

La seguente policy IAM concede AWS Backup le autorizzazioni per attivare backup con funzionalità e ripristini avanzati. Si noti, inoltre, che se le tabelle sono crittografate, la policy dovrà poter accedere alla chiave KMS AWS.

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": "*"
        }
    ]
}
```

Esempio 9: nega per una tabella di origine RestoreTableToPointInTime specifica

La seguente policy IAM nega le autorizzazioni per l'azione `RestoreTableToPointInTime` per una tabella di origine specifica:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": [
                "dynamodb:RestoreTableToPointInTime"
            ],
            "Resource": "arn:aws:dynamodb:us-east-1:123456789012:table/Music"
        }
    ]
}
```

Esempio 10: Nega RestoreTableFromBackup per tutti i backup per una tabella di origine specifica

La seguente policy IAM nega le autorizzazioni per l'azione `RestoreTableToPointInTime` per tutti i backup per una tabella di origine specifica:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": [
                "dynamodb:RestoreTableFromBackup"
            ],
            "Resource": "arn:aws:dynamodb:us-east-1:123456789012:table/Music/
backup/*"
        }
    ]
}
```

Informazioni sulla fatturazione di Amazon DynamoDB per i backup

Questa guida fornisce dettagli su come funziona la fatturazione DynamoDB per i backup. Verranno analizzati i vari componenti che contribuiscono al costo complessivo, corredati da spiegazioni chiare ed esempi pratici.

DynamoDB offre backup on demand e recupero point-in-time (PITR) per proteggere i dati DynamoDB da eventi di emergenza e offre l’archiviazione dei dati per la conservazione a lungo termine.

Come funziona

I backup on demand di DynamoDB vengono fatturati mensilmente. Se un backup viene eseguito in un determinato giorno del mese, per quel backup sarà visibile un unico addebito, calcolato per i giorni rimanenti del mese (esempio: se viene creato un backup il 27 verranno addebitati solo i pochi giorni rimanenti di quel mese sotto forma di addebito singolo il giorno 27).

Se backup precedentemente eseguiti vengono conservati per i mesi successivi, verrà sempre visualizzato un addebito mensile completo per quel backup applicato il primo giorno. Se il backup viene rimosso prima della fine del mese, gli addebiti verranno adeguati in base all'utilizzo effettivo.

Ad esempio, se è stato creato un backup il 27 luglio e questo viene mantenuto per tutto il mese di agosto, verranno visualizzati i seguenti costi per quel backup:

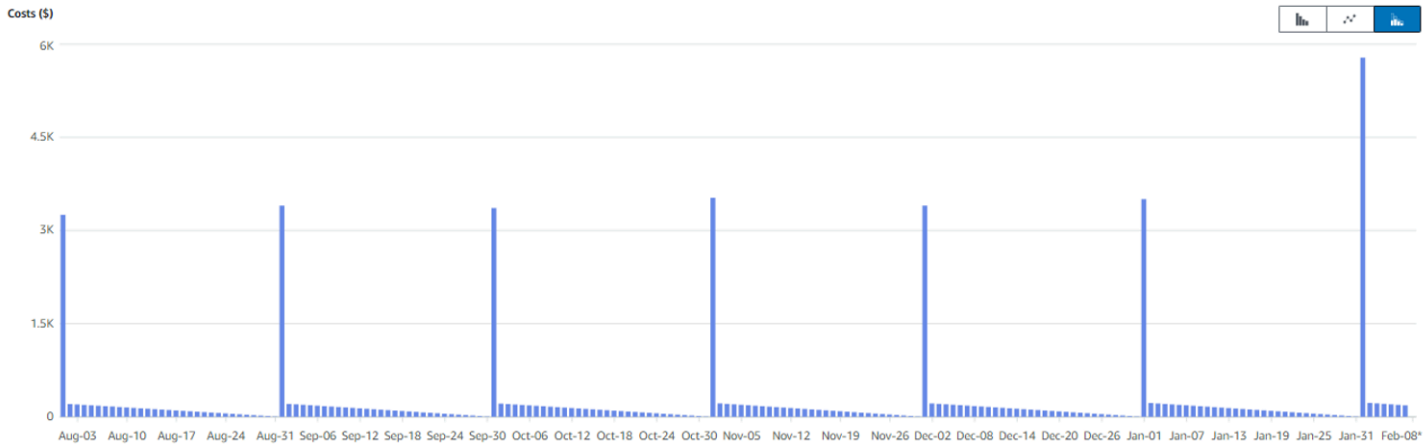
- Un addebito il 27 luglio per i restanti giorni di luglio
- Un addebito il 1° agosto per l'intero mese di agosto
- Un addebito il primo giorno di ogni mese successivo in cui esiste il backup

Quando vengono mantenuti i backup per le tabelle DynamoDB, è possibile osservare che la spesa per la metrica di utilizzo di `DynamoDB (Region)-TimedBackupStorage-ByteHrs` sembra elevata in modo anomalo il primo giorno del mese. Inoltre, controllando questa metrica all'inizio di un nuovo mese e confrontandola con i cicli di fatturazione precedenti, si potrebbe osservare quello che sembra essere un forte picco di utilizzo. Si tratta di un'impostazione predefinita. Il primo giorno di ogni mese a tutti i backup DynamoDB esistenti verranno applicati i costi di utilizzo per l'intero mese. I costi di utilizzo di tutti i backup DynamoDB rimossi nel corso del mese verranno ripartiti proporzionalmente in base all'utilizzo effettivo. Di conseguenza, è possibile che l'addebito (applicato il primo giorno del mese) diminuisca nel corso del mese. Questo perché le policy di conservazione applicano scadenze o eliminazioni manuali ai backup riportati. Questo aspetto verrà analizzato nello scenario seguente.

Analogamente, si noteranno picchi minori nel corso del mese man mano che vengono creati nuovi backup, i cui addebiti verranno applicati il giorno della creazione per il resto del mese.

Esempio di fatturazione dei backup DynamoDB

Di seguito è riportato un esempio di quanto visibile in Cost Explorer all'inizio del mese:



Si noti come il 1° febbraio sembri registrare un picco molto maggiore rispetto ai mesi precedenti. Di seguito viene analizzato il motivo per cui ciò si verifica.

Dalla pagina dei prezzi di DynamoDB:

“La dimensione totale dell’archiviazione di backup fatturata ogni mese è la somma di tutti i backup delle tabelle DynamoDB. DynamoDB monitora continuamente le dimensioni dei backup on demand per tutto il mese per determinare i costi di backup.”

Questo spiega perché la fattura mostra costantemente un forte picco di utilizzo il primo giorno di ogni mese. A tutti i backup esistenti che entrano in un nuovo mese vengono applicati i costi mensili completi per il primo mese. In altre parole, se si inserisce il mese con 300 backup DynamoDB, verranno visualizzati i costi di utilizzo del mese intero applicati il primo giorno del mese per tutti i 300 backup.

I nuovi backup eseguiti nel corso del mese comporteranno costi di utilizzo dal giorno della creazione fino alla fine del mese.

Cosa succede se il backup viene eliminato a metà mese?

Alcuni scenari da prendere in considerazione:

1. Se un backup del mese precedente viene eliminato il 15 del mese corrente, i costi di utilizzo per quel backup, comunque applicati il primo giorno del mese, verranno adeguati all’utilizzo effettivo anziché all’intero mese di utilizzo previsto applicato in precedenza. L’esempio seguente spiega questo aspetto in modo più dettagliato.
2. Quando si crea un backup durante il mese, i costi di utilizzo per il resto del mese vengono applicati il giorno della creazione. Tuttavia, se si elimina questo backup prima della fine del mese, i costi di

utilizzo verranno adeguati per includere solo i giorni in cui il backup era attivo e comunque applicati alla data di creazione.

Perché l'utilizzo del mese corrente sembra essere molto più elevato rispetto al primo mese precedente e cosa succede se rimuovo i backup?

Per rispondere a questa importante domanda in due parti, si procede a creare uno scenario di esempio utilizzando le seguenti informazioni:

- Durata del mese: 30 giorni
- Frequenza di backup DynamoDB: 10/giorno, 300/mese
- Policy di conservazione dei backup DynamoDB: 30 giorni
- Costo per backup DynamoDB: 2 USD al giorno, 60 USD al mese
- Totale al primo giorno del mese precedente (`TimedBackupStorage-ByteHrs`, verificato il primo giorno del mese corrente): 9.300 USD
- Totale del mese precedente (`TimedBackupStorage-ByteHrs`): 18.600 USD
- Totale al primo giorno del mese corrente (`TimedBackupStorage-ByteHrs`, verificato il 1° del mese): 18.000 USD
- Modifiche nell'utilizzo di DynamoDB mese per mese: nessuna

Utilizzando le informazioni di cui sopra, è possibile vedere che nel mese precedente sono stati creati 300 backup con una policy di conservazione per 30 giorni. Il primo giorno del nuovo mese tutti questi backup rimangono attivi perché non hanno ancora raggiunto la fine del periodo di recupero. Tuttavia, ogni giorno che passa, i set di backup più vecchi inizieranno a diminuire, come illustrato di seguito:

Tabella della rimozione progressiva dei backup DynamoDB

Nuovo mese	Giorno 1	Giorno 2	Giorno 3	Giorno 4	Giorno 5
Totale dei backup del mese precedente trasferiti	300	290	280	270	260

- Il primo giorno del mese è possibile vedere 300 backup a 60 USD/mese per backup, per un totale di 18.000 USD di `TimedBackupStorage-ByteHrs` applicati. Ciò è in contrasto con il mese precedente, in cui il totale dell'intero mese era di 18.600 USD.
- Il secondo giorno del mese, 10 di questi backup saranno scaduti e verranno eliminati. In tal caso, il costo applicato per tali backup verrà adeguato all'Utilizzo effettivo anziché all'Utilizzo previsto. Il risultato è che questi 10 backup, che in precedenza prevedevano un addebito il primo giorno del mese di 600 USD (10 backup x 30 giorni), sono stati ridotti a 20 USD (10 backup x 1 giorno).
- Il giorno successivo, il blocco successivo di 10 unità scadrà e verrà eliminato, con conseguente riduzione dell'utilizzo da 30 giorni a 2 giorni, con una riduzione del costo a 40 USD (10 backup x 2 giorni).

Ogni giorno che passerà, quel picco superiore a quello del mese precedente inizierà a ridursi. Estendendolo a tutto il mese si osserverà quanto segue:

Progressione dei costi dei backup DynamoDB (primo giorno del mese)

300 backup in blocchi da 10	Primo giorno	Decimo giorno	Ventesimo giorno	Trentesimo giorno
Blocco 1	600 USD	20 USD	20 USD	20 USD
Blocco 2	600 USD	40 USD	40 USD	40 USD
Blocco 3	600 USD	60 USD	60 USD	60 USD
Blocco 4	600 USD	80 USD	80 USD	80 USD
Blocco 5	600 USD	100 USD	100 USD	100 USD
Blocco 6	600 USD	120 USD	120 USD	120 USD
Blocco 7	600 USD	140 USD	140 USD	140 USD
Blocco 8	600 USD	160 USD	160 USD	160 USD
Blocco 9	600 USD	180 USD	180 USD	180 USD
Blocco 10	600 USD	600 USD	$200	$200
Blocco 11	600 USD	600 USD	220 USD	220 USD

300 backup in blocchi da 10	Primo giorno	Decimo giorno	Ventesimo giorno	Trentesimo giorno
Blocco 12	600 USD	600 USD	240 USD	240 USD
Blocco 13	600 USD	600 USD	260 USD	260 USD
Blocco 14	600 USD	600 USD	280 USD	280 USD
Blocco 15	600 USD	600 USD	$300	$300
Blocco 16	600 USD	600 USD	320 USD	320 USD
Blocco 17	600 USD	600 USD	340 USD	340 USD
Blocco 18	600 USD	600 USD	360 USD	360 USD
Blocco 19	600 USD	600 USD	380 USD	380 USD
Blocco 20	600 USD	600 USD	600 USD	400 USD
Blocco 21	600 USD	600 USD	600 USD	420 USD
Blocco 22	600 USD	600 USD	600 USD	440 USD
Blocco 23	600 USD	600 USD	600 USD	460 USD
Blocco 24	600 USD	600 USD	600 USD	480 USD
Blocco 25	600 USD	600 USD	600 USD	500 USD
Blocco 26	600 USD	600 USD	600 USD	520 USD
Blocco 27	600 USD	600 USD	600 USD	540 USD
Blocco 28	600 USD	600 USD	600 USD	560 USD
Blocco 29	600 USD	600 USD	600 USD	580 USD
Blocco 30	600 USD	600 USD	600 USD	600 USD

300 backup in blocchi da 10	Primo giorno	Decimo giorno	Ventesimo giorno	Trentesimo giorno
Totale al primo giorno del mese (USD)	18.000 USD	13.500 USD	10.400 USD	9.300 USD

Quando un nuovo blocco viene eliminato ogni giorno, il suo utilizzo viene adeguato in base al numero di giorni in cui è esistito, anziché all'intero importo mensile. Di conseguenza, entro la fine del mese gli addebiti osservati il primo giorno saranno scesi dai 18.000 USD iniziali ai 9.300 USD previsti. Questo numero, combinato con i nuovi backup creati nel corso del mese (che avranno una tabella di fatturazione simile a quella precedente, ma invertita), comporterà una spesa mensile in linea con i 18.600 USD del mese scorso.

Ripristino di una tabella in DynamoDB

È possibile eseguire il ripristino di una tabella DynamoDB da un backup PITR o dai backup on demand utilizzando la Console di gestione AWS, l'interfaccia a riga di comando AWS (AWS CLI) o l'API DynamoDB. Il processo di recupero esegue il ripristino in una nuova tabella DynamoDB.

Ripristino di una tabella tramite ripristino point-in-time (PITR)

È possibile eseguire il recupero point-in-time (PITR) della tabella fino a `EarliestRestoreableDateTime`.

> **Important**
>
> Se il recupero point-in-time di una tabella viene disabilitato e abilitato in un secondo momento, viene reimpostata la data iniziale a cui può la tabella essere ripristinata. Di conseguenza, puoi ripristinare immediatamente la tabella solo usando il valore `LatestRestorableDateTime`.

Quando si utilizza il recupero point-in-time, DynamoDB ripristina i dati della tabella allo stato in base alla data e all'ora selezionate (giorno:ora:minuto:secondo) in una nuova tabella. Una tabella viene ripristinata senza utilizzare alcun throughput assegnato nella tabella. Puoi eseguire un ripristino completo della tabella utilizzando il ripristino point-in-time oppure configurare le impostazioni della tabella di destinazione. Puoi modificare le seguenti impostazioni nella tabella ripristinata:

- Indici secondari globali (GSI)
- Indici secondari locali (LSI)
- Modalità di fatturazione
- Capacità di lettura e scrittura di cui è stato effettuato il provisioning
- Impostazioni di crittografia

> **Important**
>
> Quando esegui un ripristino completo della tabella, la tabella di destinazione è impostata con le stesse unità di capacità di lettura e di scrittura di cui è stato effettuato il provisioning che la tabella di origine aveva al momento della richiesta del backup. Ad esempio, supponi che il throughput assegnato di una tabella sia stato recentemente abbassato a 50 unità di capacità in lettura e 50 unità di capacità in scrittura. Lo stato della tabella viene quindi ripristinato a quello di tre settimane fa e il suo throughput assegnato in quel momento era impostato su 100 unità di capacità in lettura e su 100 unità di capacità in scrittura. In questo caso, DynamoDB ripristina i dati della tabella a quel momento, ma utilizza la velocità di trasmissione effettiva assegnata corrente (100 unità di capacità di lettura e 100 unità di capacità di scrittura).

Inoltre è possibile ripristinare i dati della tabella DynamoDB nelle Regioni AWS in modo che la tabella ripristinata venga creata in una Regione diversa da quella in cui si trova la tabella di origine. È possibile eseguire ripristini tra Regioni AWS commerciali, Regioni AWS Cina e AWS GovCloud (US). I prezzi sono calcolati solo in base ai dati trasferiti fuori dalla regione di origine e al ripristino in una nuova tabella nella regione di destinazione.

> **Note**
>
> Il ripristino tra Regioni non è supportato se la Regione di origine o di destinazione è Asia Pacifico (Hong Kong) o Medio Oriente (Bahrein).

I ripristini possono essere più veloci ed economici se escludi alcuni o tutti gli indici dalla creazione nella tabella ripristinata. È necessario configurare manualmente nella tabella ripristinata quanto segue:

- Policy di scalabilità automatica
- Policy AWS Identity and Access Management
- Metriche e allarmi per Eventi Amazon CloudWatch
- Tag
- Impostazioni flusso
- Impostazioni Time to Live (TTL)
- Impostazioni di ripristino point-in-time

Il tempo necessario per ripristinare una tabella varia in base a molteplici fattori e non è sempre correlato alle dimensioni della tabella.

Ripristino point-in-time (PITR) di una tabella DynamoDB

Il ripristino point-in-time (PITR) di Amazon DynamoDB fornisce backup continui dei dati contenuti nelle tabelle DynamoDB. È possibile eseguire il ripristino point-in-time (PITR) di una tabella utilizzando la console DynamoDB o AWS Command Line Interface (AWS CLI). Il processo di ripristino point-in-time (PITR) esegue sempre il ripristino in una nuova tabella.

Se si desidera usare la AWS CLI, è necessario innanzitutto configurarla. Per ulteriori informazioni, consulta Accesso a DynamoDB.

Argomenti

- Ripristino point-in-time (PITR) di una tabella DynamoDB (console)
- Ripristino point-in-time di una tabella (AWS CLI)

Ripristino point-in-time (PITR) di una tabella DynamoDB (console)

L'esempio seguente mostra come utilizzare la console DynamoDB per eseguire il ripristino point-in-time (PITR) di una tabella esistente denominata `Music`.

> **Note**
>
> La procedura presuppone che sia stato attivato il ripristino point-in-time (PITR). Per abilitarlo per la tabella `Music`, nella scheda Backups (Backup), nella sezione Point-in-time recovery (PITR) (Ripristino point-in-time (PITR)) scegli Edit (Modifica) e seleziona la casella accanto a Enable point-in-time-recovery (Abilita ripristino point-in-time (PITR)).

Per eseguire il ripristino point-in-time di una tabella

1. Accedi alla console Console di gestione AWS e apri la console DynamoDB all'indirizzo https://console.aws.amazon.com/elasticache/.
2. Nel riquadro di navigazione sul lato sinistro della console scegli Tables (Tabelle).
3. Nell'elenco delle tabelle, seleziona `Music`.
4. Nella scheda Backups (Backup) della tabella `Music`, nella sezione Point-in-time recovery (PITR) (Ripristino point-in-time (PITR)) scegli Restore (Ripristina).
5. Per il nome tabella, immetti **`MusicMinutesAgo`**.

   > **Note**
   >
   > È possibile ripristinare la tabella nella stessa regione AWS o in una regione diversa da quella dove si trova la tabella di origine. È anche possibile escludere la creazione di indici secondari nella nuova tabella ripristinata. Inoltre, puoi specificare una modalità di crittografia diversa.

6. Per confermare il momento del ripristino, imposta la data e l'ora del ripristino su Earliest (Meno recente). Quindi scegli Restore table (Ripristina tabella) per avviare il processo di ripristino.

   La tabella in fase di ripristino è visualizzata con lo stato Restoring (Ripristino). Al termine del processo di ripristino, lo stato della tabella `MusicMinutesAgo` cambia in Active (Attivo).

Ripristino point-in-time di una tabella (AWS CLI)

La procedura mostra come utilizzare l'AWS CLI per eseguire il ripristino point-in-time di una tabella esistente denominata `Music`.

> **Note**
>
> La procedura presuppone che sia stato attivato il ripristino point-in-time (PITR). Per abilitarlo per la tabella `Music`, emettere il comando seguente.
>
> ```
> aws dynamodb update-continuous-backups \
>     --table-name Music \
>     --point-in-time-recovery-specification PointInTimeRecoveryEnabled=True
> ```

Per eseguire il ripristino point-in-time di una tabella

1. Verifica che il ripristino point-in-time sia attivato per la tabella `Music` utilizzando il comando `describe-continuous-backups`.

   ```
   aws dynamodb describe-continuous-backups \
       --table-name Music
   ```

   I backup continui (opzione attivata automaticamente al momento della creazione della tabella) e il ripristino point-in-time sono attivati.

   ```
   {
       "ContinuousBackupsDescription": {
           "PointInTimeRecoveryDescription": {
               "PointInTimeRecoveryStatus": "ENABLED",
               "EarliestRestorableDateTime": 1519257118.0,
               "LatestRestorableDateTime": 1520018653.01
           },
           "ContinuousBackupsStatus": "ENABLED"
       }
   }
   ```

2. Esegui il ripristino point-in-time della tabella. In questo caso, la tabella `Music` viene ripristinata a `LatestRestorableDateTime` (~5 minuti fa) nella stessa regione AWS.

   ```
   aws dynamodb restore-table-to-point-in-time \
       --source-table-name Music \
       --target-table-name MusicMinutesAgo \
       --use-latest-restorable-time
   ```

   > **Note**
   >
   > Puoi anche eseguire il ripristino a un momento specifico nel tempo. A tale scopo, esegui il comando utilizzando l'argomento `--restore-date-time` e specifica un time stamp. È possibile specificare qualsiasi momento all'interno del periodo di recupero configurato, che può essere impostato su qualsiasi valore compreso tra 1 e 35 giorni. Ad esempio, il comando seguente ripristina la tabella a `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
```

Se esegui il ripristino a un momento specifico nel tempo, la definizione dell'argomento `--no-use-latest-restorable-time` è facoltativa.

3. Ripristina la tabella fino a un certo momento con le impostazioni personalizzate della tabella. In questo caso, la tabella `Music` viene ripristinata a `LatestRestorableDateTime` (~5 minuti fa).

   Puoi specificare una modalità di crittografia diversa per la tabella ripristinata, come segue.

   > **Note**
   >
   > Il parametro `sse-specification-override` accetta gli stessi valori del parametro `sse-specification-override` utilizzato nel comando `CreateTable`. Per ulteriori informazioni, consulta Gestione di tabelle crittografate in 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
   ```

   È possibile ripristinare la tabella in una regione AWSdiversa da quella in cui si trova la tabella di origine.

   > **Note**
   >
   > - Il parametro `sse-specification-override` è obbligatorio per i ripristini tra regioni ma facoltativo per i ripristini nella stessa regione della tabella di origine.
   > - Il parametro `source-table-arn` deve essere fornito per i ripristini tra regioni diverse.
   > - Quando si esegue un ripristino tra regioni dalla riga di comando, è necessario impostare la regione AWS predefinita sulla regione di destinazione desiderata. Per

ulteriori informazioni, consulta Opzioni della riga di comando nella Guida per l'utente di 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
```

Puoi ignorare la modalità di fatturazione e la velocità effettiva fornita per la tabella di ripristino.

```
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
```

È possibile escludere la creazione di alcuni o tutti gli indici secondari nella nuova tabella ripristinata.

> **Note**
>
> I ripristini possono essere più veloci ed economici se escludi alcuni o tutti gli indici secondari dalla creazione nella nuova tabella ripristinata.

```
aws dynamodb restore-table-to-point-in-time \
    --source-table-name Music \
    --target-table-name MusicMinutesAgo \
    --use-latest-restorable-time \
    --global-secondary-index-override '[]'
```

È possibile utilizzare una combinazione di diverse sostituzioni. Ad esempio, puoi utilizzare un singolo indice secondario globale e modificare contemporaneamente il throughput assegnato, come segue.

```
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
```

Per verificare il ripristino, usa il comando `describe-table` per descrivere la tabella `MusicEarliestRestorableDateTime`.

```
aws dynamodb describe-table --table-name MusicEarliestRestorableDateTime
```

La tabella in fase di ripristino è visualizzata con lo stato Creating (Creazione) e il ripristino in corso come true. Al termine del processo di ripristino, lo stato della tabella `MusicEarliestRestorableDateTime` cambia in Active (Attivo).

> **Important**
>
> Durante il ripristino, non modificare o eliminare le policy AWS Identity and Access Management (IAM) che concedono all'entità IAM (ad esempio, utente, gruppo o ruolo) l'autorizzazione a eseguire il ripristino. Diversamente, si potrebbero verificare comportamenti imprevisti. Supponi, ad esempio, di rimuovere le autorizzazioni di scrittura per una tabella mentre è in corso il ripristino. In questo caso, l'operazione `RestoreTableToPointInTime` sottostante non è in grado di scrivere i dati ripristinati nella tabella. Analogamente, le policy IAM che comportano restrizioni dell'IP di origine per l'accesso alla tabella di ripristino di destinazione possono causare dei problemi.
>
> Puoi modificare o eliminare le autorizzazioni solo dopo il completamento dell'operazione di ripristino.

Utilizzo di AWS Backup con Dynamo DB

Amazon DynamoDB può aiutarti a soddisfare i requisiti di conformità alle normative e di continuità aziendale attraverso le caratteristiche di backup avanzate in AWS Backup. AWS Backup è un servizio di protezione dei dati completamente gestito che semplifica la centralizzazione e l'automazione dei backup nei servizi AWS, nel cloud e on-premise. Utilizzando questo servizio, puoi configurare le policy di backup e monitorare l'attività delle tue risorse AWS in un'unica posizione. Per utilizzare AWS Backup, devi aderire affermativamente. Le scelte di adesione si applicano all'account e alla regione AWS specifici, quindi potresti dover aderire a più regioni utilizzando lo stesso account. Per ulteriori informazioni, consulta la AWS Backup Developer Guide.

Amazon DynamoDB è integrato in modo nativo con AWS Backup. Puoi utilizzare AWS Backup per pianificare, copiare, taggare e programmare il ciclo di vita dei backup on demand di DynamoDB automaticamente. Puoi continuare a visualizzare e ripristinare questi backup dalla console DynamoDB. Puoi utilizzare la console DynamoDB, l'API e l'interfaccia a riga di comando di AWS (AWS CLI) per abilitare backup automatici per le tabelle DynamoDB.

> **Note**
>
> Eventuali backup creati tramite DynamoDB rimarranno invariati. Potrai continuare a creare backup attraverso il flusso di lavoro DynamoDB corrente.

Le caratteristiche di backup avanzate disponibili tramite AWS Backup includono:

Scheduled backups (Backup pianificati): puoi configurare regolarmente backup pianificati delle tabelle DynamoDB utilizzando i piani di backup.

Cross-account and cross-Region copying (Copia tra account e tra regioni): puoi copiare automaticamente i backup in un altro vault di backup in una regione o account AWS diverso, che consente di supportare i requisiti di protezione dei dati.

Cold storage tiering (Tiering archiviazione a freddo): puoi configurare i backup per implementare regole del ciclo di vita al fine di eliminare i backup o trasferirli a un'archiviazione più a freddo. Questo può aiutarti a ottimizzare i costi di backup.

Tag (Tag): puoi taggare automaticamente i backup a scopo di fatturazione e allocazione dei costi.

Encryption (Crittografia): i backup on demand di DynamoDB gestiti tramite AWS Backup ora sono memorizzati nel vault AWS Backup. Questo consente di criptare e proteggere i backup utilizzando una AWS KMS keyindipendente dalla chiave di crittografia della tabella Dynamo DB.

Audit backups (Verifica backup): puoi utilizzare AWS Backup Audit Manager per verificare la conformità delle tue policy AWS Backup e per trovare attività di backup e risorse che non sono ancora conformi ai controlli definiti. Puoi inoltre utilizzarlo per generare automaticamente un percorso di audit di report giornalieri e on demand per i tuoi scopi di governance dei backup.

Secure backups using the WORM model (Backup sicuri utilizzando il modello WORM): puoi utilizzare AWS Backup Vault Lock per abilitare un'impostazione write-once-read-many (WORM) per i backup. Con AWS Backup Vault Lock è possibile aggiungere un ulteriore livello di difesa che protegge i backup da operazioni di eliminazione involontarie o malevole, modifiche ai periodi di recupero dei backup e aggiornamenti alle impostazioni del ciclo di vita. Per ulteriori informazioni, consulta AWS Backup Vault Lock.

Queste caratteristiche di backup avanzate sono disponibili in tutte le regioni AWS. Per ulteriori informazioni su queste caratteristiche, consulta la AWS Backup Developer Guide.

Argomenti

- Backup e ripristino delle tabelle DynamoDB con: Come funziona AWS Backup
- Creazione di backup delle tabelle DynamoDB con AWS Backup
- Copia di un backup di una tabella DynamoDB con AWS Backup
- Ripristino di un backup di una tabella DynamoDB da AWS Backup
- Eliminazione di un backup di una tabella DynamoDB con AWS Backup
- Differenze nelle note di utilizzo tra i backup on demand gestiti da AWS Backup e da DynamoDB

Backup e ripristino delle tabelle DynamoDB con: Come funziona AWS Backup

È possibile utilizzare la funzionalità di backup on demand per creare backup completi delle tabelle Amazon DynamoDB. In questa sezione viene fornita una panoramica di ciò che accade durante il processo di backup e ripristino.

Backup

Quando si crea un backup su richiesta con AWS Backup, viene catalogato un indicatore temporale della richiesta. Il backup viene creato in modo asincrono applicando tutte le modifiche all'ultima snapshot di tabella completa, fino al momento in cui viene effettuata la richiesta.

Ogni volta che crei un backup on-demand, viene eseguito il backup di tutti i dati della tabella. Non vi è un limite per il numero di backup on-demand che possono essere effettuati.

> **Note**
>
> A differenza dei backup DynamoDB, i backup eseguiti con non sono istantanei. AWS Backup

Mentre è in corso un backup, non puoi effettuare le seguenti operazioni:

- Sospendere o annullare l'operazione di backup;
- Eliminare la tabella di origine del backup;
- Disabilitare i backup su una tabella se uno di questi è in corso.

AWS Backup fornisce pianificazioni di backup automatizzate, gestione della conservazione e gestione del ciclo di vita. Ciò elimina la necessità di script personalizzati e processi manuali. AWS Backup esegue i backup e li elimina quando scadono. Per ulteriori informazioni, consulta la Guida per gli sviluppatori di AWS Backup.

Se utilizzi la console, tutti i backup creati utilizzando AWS Backup sono elencati nella scheda Backup con il Tipo di backup impostato su. `AWS_BACKUP`

> **Note**
>
> Non è possibile eliminare i backup contrassegnati con un tipo di backup di `AWS_BACKUP` utilizzando la console DynamoDB. Per gestire questi backup, usa la console. AWS Backup

Per scoprire come eseguire un backup, consulta Backup di una tabella DynamoDB.

Ripristini

Una tabella viene ripristinata senza utilizzare alcun throughput assegnato nella tabella. È possibile eseguire un ripristino completo della tabella dal backup DynamoDB oppure configurare le impostazioni della tabella di destinazione. Quando esegui un ripristino, puoi modificare le seguenti impostazioni della tabella:

- Impostazioni di crittografia

> **Important**
>
> Quando esegui un ripristino completo della tabella, la tabella di destinazione è impostata con le stesse unità di capacità di lettura e di scrittura di cui è stato effettuato il provisioning che la tabella di origine aveva al momento della richiesta del backup. Il processo di ripristino ripristina anche gli indici secondari locali e gli indici secondari globali.

È possibile copiare un backup dei dati della tabella DynamoDB in una regione AWS diversa e quindi ripristinarli in quella nuova regione. Puoi copiare e quindi ripristinare i backup tra regioni AWS commerciali, regioni della AWS Cina e regioni AWS GovCloud (Stati Uniti). I prezzi sono calcolati solo in base ai dati che copi dalla regione di origine e dai dati che ripristini in una nuova tabella nella regione di destinazione.

AWS Backup ripristinerà le tabelle con tutti gli indici originali.

È necessario configurare manualmente nella tabella ripristinata quanto segue:

- Policy di scalabilità automatica
- AWS Identity and Access Management politiche (IAM)
- Parametri e CloudWatch allarmi di Amazon
- Tag
- Impostazioni flusso
- Impostazioni Time to Live (TTL)
- Impostazioni di protezione dall'eliminazione
- Impostazioni Ripristino point-in-time (PITR)

Puoi ripristinare i dati dell'intera tabella solo su una nuova tabella da un backup. È possibile scrivere sulla tabella ripristinata solo dopo che è diventata attiva.

> **Note**
>
> AWS Backup i ripristini non sono distruttivi. Non puoi sovrascrivere una tabella esistente durante un'operazione di ripristino.

I parametri di servizio mostrano che il 95% dei ripristini delle tabelle dei clienti viene completato in meno di un'ora. Tuttavia, i tempi di ripristino sono direttamente correlati alla configurazione delle tabelle (ad esempio la dimensione delle tabelle e il numero di partizioni sottostanti) e ad altre variabili correlate. Una best practice quando si pianifica un ripristino di emergenza consiste nel documentare regolarmente i tempi medi di completamento del ripristino e stabilire in che modo questi tempi influiscono sull'obiettivo del tempo di ripristino complessivo.

Per scoprire come eseguire un ripristino, consulta Ripristino di una tabella DynamoDB da un backup.

È possibile utilizzare le policy IAM per il controllo degli accessi. Per ulteriori informazioni, consulta Utilizzo di IAM con backup e ripristino di DynamoDB.

Tutte le console di backup e ripristino e le operazioni API vengono acquisite e registrate in AWS CloudTrail per la registrazione, il monitoraggio e l'audit.

Creazione di backup delle tabelle DynamoDB con AWS Backup

In questa sezione viene descritto come attivare AWS Backup per creare i backup on demand e pianificati dalle tabelle DynamoDB.

Argomenti

- Attivazione delle funzionalità AWS Backup
- Backup on-demand
- Backup pianificati

Attivazione delle funzionalità AWS Backup

Per utilizzare AWS Backup con DynamoDB è necessario attivarlo.

Per attivare AWS Backup, esegui i passaggi seguenti:

1. Accedi ad AWS Management Console e apri la console DynamoDB all'indirizzo https://console.aws.amazon.com/dynamodb/.
2. Nel riquadro di navigazione sul lato sinistro della console scegliere Backups (Backup).
3. Nella finestra Impostazioni di backup scegli Abilita.
4. Verrà visualizzata una schermata di conferma. Scegli Attiva le funzionalità.

Le caratteristiche di AWS Backup ora sono disponibili per le tabelle Dynamo DB.

Se vuoi disattivare le funzionalità di AWS Backup dopo che le hai attivate, attieniti alla procedura seguente:

1. Accedi ad AWS Management Console e apri la console DynamoDB all'indirizzo https://console.aws.amazon.com/dynamodb/.
2. Nel riquadro di navigazione sul lato sinistro della console scegliere Backups (Backup).
3. Nella finestra Impostazioni di backup scegli Disattiva.
4. Verrà visualizzata una schermata di conferma. Scegli Disattiva le funzionalità.

Se non riesci ad attivare o disattivare le funzionalità di AWS Backup, l'amministratore AWS potrebbe dover eseguire queste azioni.

Backup on-demand

Per creare un backup on demand di una tabella Dynamo DB, attieniti alla procedura seguente:

1. Accedi ad AWS Management Console e apri la console DynamoDB all'indirizzo https://console.aws.amazon.com/dynamodb/.
2. Nel riquadro di navigazione sul lato sinistro della console scegliere Backups (Backup).
3. Scegliere Create backup (Crea backup).
4. Dal menu a discesa visualizzato, scegli Create an on-demand backup (Crea un backup on demand).
5. Per creare un backup gestito da AWS Backup con archiviazione a caldo e altre caratteristiche di base, scegli Impostazioni predefinite. Per creare un backup che può essere trasferito all'archiviazione a freddo o per creare un backup con le caratteristiche di DynamoDB anziché di AWS Backup, scegli Personalizza impostazioni.

Se invece vuoi creare questo backup con le caratteristiche DynamoDB precedenti, scegli Customize settings (Personalizza impostazioni) e quindi scegli Backup with DynamoDB (Backup con DynamoDB).

6. Dopo aver completato le impostazioni, scegli Create backup (Crea backup).

Backup pianificati

Per pianificare un backup, attieniti alla procedura seguente:

1. Accedi ad AWS Management Console e apri la console DynamoDB all'indirizzo https://console.aws.amazon.com/dynamodb/.
2. Nel riquadro di navigazione sul lato sinistro della console scegliere Backups (Backup).
3. Dal menu a discesa visualizzato, scegli Pianifica i backup con AWS Backup.
4. Verrà visualizzato AWS Backup per creare un piano di backup.

Copia di un backup di una tabella DynamoDB con AWS Backup

Puoi creare una copia di un backup corrente. Puoi copiare i backup in più account AWS o regioni AWS on demand o automaticamente come parte di un piano di backup pianificato. Puoi anche automatizzare una sequenza di copie tra account e tra regioni per Amazon DynamoDB Encryption Client.

La replica tra regioni è particolarmente utile se hai requisiti di continuità aziendale o di conformità per archiviare i backup a una distanza minima dai dati di produzione.

I backup tra account sono utili per copiare in modo sicuro i backup su uno o più account AWS della tua organizzazione per motivi operativi o di sicurezza. Se il backup originale viene eliminato inavvertitamente, puoi copiare il backup dall'account di destinazione all'account di origine e quindi avviare il ripristino. Prima di farlo, devi disporre di due account appartenenti alla stessa organizzazione nel servizio Organizations.

Le copie ereditano la configurazione del backup di origine, a meno che non specifichi diversamente, con un'eccezione: se specifichi che la nuova copia non scadrà "mai". Con questa impostazione, la nuova copia eredita ancora la data di scadenza dell'origine. Se vuoi che la nuova copia di backup sia permanente, imposta i backup di origine in modo che non scadano mai o specifica che la nuova copia scadrà 100 anni dopo la sua creazione.

> **Note**
>
> Se stai copiando in un altro account, devi prima avere l'autorizzazione da tale account.

Per copiare un backup, procedi come segue:

1. Accedi ad AWS Management Console e apri la console DynamoDB all'indirizzo https://console.aws.amazon.com/dynamodb/.
2. Nel riquadro di navigazione sul lato sinistro della console scegliere Backups (Backup).
3. Seleziona la casella di controllo accanto al backup che vuoi copiare.
   - Se il backup che vuoi copiare è disattivato, devi abilitare caratteristiche avanzate con AWS Backup. Quindi crea un nuovo backup. Ora puoi copiare questo nuovo backup in altre Regioni e account e copiare qualsiasi altro nuovo backup in futuro.
4. Scegli Copia.
5. Se vuoi copiare il backup in un altro account o in un'altra regione, seleziona la casella di controllo accanto a Copy the recovery point to another destination (Copia il punto di ripristino in un'altra destinazione). Quindi scegli se copiare in un'altra regione del tuo account o in un account diverso in un'altra regione.

   > **Note**
   >
   > Per ripristinare un backup in un'altra regione o in un altro account, devi prima copiare il backup in tale regione o account.

6. Seleziona il vault desiderato in cui verrà copiato il file. Se lo desideri, puoi anche creare un nuovo vault di backup.
7. Scegli Copy backup (Copia backup).

Ripristino di un backup di una tabella DynamoDB da AWS Backup

In questa sezione viene descritto come ripristinare un backup di una tabella DynamoDB da AWS Backup.

Argomenti

- Ripristino di una tabella DynamoDB da AWS Backup

- Ripristino di una tabella DynamoDB in un'altra regione o in un altro account

Ripristino di una tabella DynamoDB da AWS Backup

Per ripristinare le tabelle DynamoDB da AWS Backup, attieniti alla procedura seguente:

1. Accedi ad AWS Management Console e apri la console DynamoDB all'indirizzo https://console.aws.amazon.com/dynamodb/.
2. Nel riquadro di navigazione sul lato sinistro della console scegli Tables (Tabelle).
3. Scegli la scheda Backups (Backup).
4. Seleziona la casella di controllo accanto al backup precedente da cui eseguire il ripristino.
5. Scegli Restore (Ripristina). Verrà visualizzata la schermata Restore table from backup (Ripristina tabella dal backup).
6. Inserisci il nome della tabella appena ripristinata, la crittografia di questa nuova tabella, la chiave con cui vuoi criptare il ripristino e altre opzioni.
7. Al termine, scegli Restore (Ripristina).

Ripristino di una tabella DynamoDB in un'altra regione o in un altro account

Per ripristinare una tabella DynamoDB in un'altra regione o in un altro account, devi prima copiare il backup in quella nuova regione o account. Per copiare in un altro account, tale account deve prima concederti l'autorizzazione. Dopo aver copiato il backup DynamoDB nella nuova regione o nel nuovo account, puoi ripristinarlo con la procedura nella sezione precedente.

Eliminazione di un backup di una tabella DynamoDB con AWS Backup

In questa sezione viene descritto come eliminare un backup di una tabella DynamoDB con AWS Backup.

Un backup DynamoDB creato tramite le caratteristiche di AWS Backup viene archiviato in un vault di AWS Backup.

Per eliminare questo tipo di backup, attieniti alla procedura seguente:

1. Accedi alla console Console di gestione AWS e apri la console DynamoDB all'indirizzo https://console.aws.amazon.com/elasticache/.

2. Nel riquadro di navigazione sul lato sinistro della console scegliere Backups (Backup).
3. Nella schermata che segue, scegli Continua su AWS backup.

   Verrà visualizzato Console di backup AWS. Per ulteriori informazioni su come eliminare i backup su Console di backup AWS, consulta Eliminazione di backup.

   Per ulteriori informazioni su AWS Backup, consulta Backup and recovery using AWS Backup nel Prontuario AWS.

Differenze nelle note di utilizzo tra i backup on demand gestiti da AWS Backup e da DynamoDB

Questa sezione descrive le differenze tecniche tra i backup on demand gestiti da AWS Backup e DynamoDB.

AWS Backup ha flussi di lavoro e comportamenti diversi da DynamoDB. Ciò include:

Crittografia: i backup creati con il piano AWS Backup vengono archiviati in un vault crittografato con una chiave gestita dal servizio AWS Backup. Il vault dispone di policy di controllo degli accessi per una maggiore sicurezza.

ARN di Backup: i file di backup creati da AWS Backup ora avranno un ARN AWS Backup, che potrebbe influire sul modello di autorizzazione degli utenti. I nomi delle risorse di Backup (ARN) cambieranno da `arn:aws:dynamodb` a `arn:aws:backup`.

Eliminazione dei backup: i backup creati con AWS Backup possono essere eliminati solo dal vault AWS Backup. Non potrai eliminare i file AWS Backup dalla console DynamoDB.

Processo di backup: a differenza dei backup di DynamoDB, i backup realizzati con AWS Backup non sono istantanei.

Fatturazione: i backup delle tabelle DynamoDB con le caratteristiche di AWS Backup sono fatturati da AWS Backup.

Ruoli IAM: se gestisci gli accessi tramite i ruoli IAM, dovrai anche configurare un nuovo ruolo IAM con queste nuove autorizzazioni:

```
"dynamodb:StartAwsBackupJob",
"dynamodb:RestoreTableFromAwsBackup"
```

`dynamodb:StartAwsBackupJob` è necessario per eseguire correttamente un backup con le caratteristiche di AWS Backup e `dynamodb:RestoreTableFromAwsBackup` è necessario per il ripristino da un backup creato con le caratteristiche di AWS Backup.

Per visualizzare queste autorizzazioni in un criterio IAM completo, vedi l'Esempio 8 in Using IAM (Utilizzo di IAM).

Esempi di codice per l'utilizzo di DynamoDB AWS SDKs

I seguenti esempi di codice mostrano come utilizzare DynamoDB con AWS un kit di sviluppo software (SDK).

Le nozioni di base sono esempi di codice che mostrano come eseguire le operazioni essenziali all'interno di un servizio.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Gli scenari sono esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio o combinate con altri Servizi AWS.

AWS i contributi della community sono esempi che sono stati creati e gestiti da diversi team. AWS Per fornire un feedback, utilizza il meccanismo disponibile nei repository collegati.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUtilizzo di DynamoDB con un SDK AWS. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Nozioni di base

Hello DynamoDB

Gli esempi di codice seguenti mostrano come iniziare a utilizzare DynamoDB.

.NET

SDK per .NET (v4)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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}");
        }
    }
}
```

- Per i dettagli sull'API, consulta la ListTablessezione AWS SDK per .NET API Reference.

C++

SDK per C++

> **Note**
>
> C'è di più su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Codice per il CMake file CMake Lists.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})
```

Codice per il file origine 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;
}
```

- Per i dettagli sull'API, consulta la ListTablessezione AWS SDK per C++ API Reference.

Java

SDK per Java 2.x

> **Note**
>
> C'è di più su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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!");
```

```
    }
}
```

- Per i dettagli sull'API, consulta la ListTablessezione AWS SDK for Java 2.x API Reference.

JavaScript

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Per maggiori dettagli sull'utilizzo di DynamoDB AWS SDK per JavaScript in, consulta Programmazione di 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;
};
```

- Per i dettagli sulle API, consulta ListTablesla sezione API Reference.AWS SDK per JavaScript

Python

SDK per Python (Boto3)

> **Note**
>
> C'è di più su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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.")
```

- Per i dettagli sull'API, consulta ListTables AWSSDK for Python (Boto3) API Reference.

Ruby

SDK per Ruby

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
```

- Per i dettagli sull'API, consulta la ListTablessezione AWS SDK per Ruby API Reference.

Esempi di codice

- Esempi di base per l'utilizzo di DynamoDB AWS SDKs
  - Hello DynamoDB
  - Scopri le basi di DynamoDB con un SDK AWS
  - Azioni per l'utilizzo di DynamoDB AWS SDKs
    - Utilizzalo BatchExecuteStatement con un AWS SDK
    - Utilizzo BatchGetItem con un AWS SDK o una CLI
    - Utilizzo BatchWriteItem con un AWS SDK o una CLI
    - Utilizzo CreateTable con un AWS SDK o una CLI
    - Utilizzo DeleteItem con un AWS SDK o una CLI
    - Utilizzo DeleteTable con un AWS SDK o una CLI
    - Utilizzo DescribeTable con un AWS SDK o una CLI
    - Utilizzo DescribeTimeToLive con un AWS SDK o una CLI
    - Utilizzare ExecuteStatement con un SDK AWS
    - Utilizzo GetItem con un AWS SDK o una CLI
    - Utilizzo ListTables con un AWS SDK o una CLI
    - Utilizzo PutItem con un AWS SDK o una CLI
    - Utilizzo Query con un AWS SDK o una CLI
    - Utilizzo Scan con un AWS SDK o una CLI
    - Utilizzo UpdateItem con un AWS SDK o una CLI
    - Utilizzo UpdateTable con un AWS SDK o una CLI
    - Utilizzo UpdateTimeToLive con un AWS SDK o una CLI

- Scenari per l'utilizzo di DynamoDB AWS SDKs
  - Accelera le letture DynamoDB con DAX utilizzando un SDK AWS
  - Lavora con scenari avanzati di DynamoDB Global Secondary Index utilizzando v2 AWS Command Line Interface
  - Costruisci un'applicazione per inviare dati a una tabella DynamoDB
  - Confronta più valori con un singolo attributo in DynamoDB con un SDK AWS
  - Aggiornamento condizionale di un elemento DynamoDB con un TTL utilizzando un SDK AWS
  - Connect a un'istanza DynamoDB locale utilizzando un SDK AWS
  - Conta gli operatori di espressione in DynamoDB con un SDK AWS
  - Creazione di un'API REST di API Gateway per monitorare i dati COVID-19
  - Creazione di un'applicazione di messaggistica con Step Functions
  - Creazione di un'applicazione di gestione delle risorse fotografiche che consente agli utenti di gestire le foto utilizzando etichette
  - Crea una tabella DynamoDB con un indice secondario globale utilizzando l'SDK AWS
  - Crea una tabella DynamoDB con un'impostazione di throughput caldo utilizzando un SDK AWS
  - Creazione di un'applicazione Web per tracciare i dati DynamoDB
  - Creazione di un'applicazione di chat websocket con API Gateway
  - Crea un elemento DynamoDB con un TTL utilizzando un SDK AWS
  - Crea e gestisci tabelle globali DynamoDB con una forte coerenza multiregionale utilizzando un SDK AWS
  - Crea e gestisci tabelle globali DynamoDB dimostrando MREC utilizzando un SDK AWS
  - Eliminare i dati DynamoDB utilizzando le istruzioni PartiQL DELETE con un SDK AWS
  - Rileva PPE nelle immagini con Amazon Rekognition utilizzando un SDK AWS
  - Inserimento di dati DynamoDB utilizzando istruzioni PartiQL INSERT con un SDK AWS
  - Richiamo a una funzione Lambda da un browser
  - Gestisci gli indici secondari globali DynamoDB utilizzando v2 AWS Command Line Interface
  - Gestisci le policy basate sulle risorse di DynamoDB utilizzando la versione 2 AWS Command Line Interface
  - Monitora le prestazioni di Amazon DynamoDB utilizzando un SDK AWS
  - Esegui operazioni di interrogazione avanzate su DynamoDB utilizzando un SDK AWS
  - Esegui operazioni sugli elenchi in DynamoDB con un SDK AWS

- Esegui operazioni di mappatura in DynamoDB con un SDK AWS
- Esegui operazioni di set in DynamoDB con un SDK AWS
- Interroga una tabella DynamoDB utilizzando batch di istruzioni PartiQL e un SDK AWS
- Interroga una tabella DynamoDB utilizzando PartiQL e un SDK AWS
- Interroga una tabella DynamoDB utilizzando un indice secondario globale con un SDK AWS
- Interroga una tabella DynamoDB utilizzando una condizione begins_with con un SDK AWS
- Interroga una tabella DynamoDB utilizzando un intervallo di date nella chiave di ordinamento con un SDK AWS
- Interroga una tabella DynamoDB con un'espressione di filtro complessa con un SDK AWS
- Interroga una tabella DynamoDB con un'espressione di filtro dinamica con un SDK AWS
- Interroga una tabella DynamoDB con un'espressione di filtro e un limite con un SDK AWS
- Interroga una tabella DynamoDB con attributi annidati utilizzando un SDK AWS
- Interroga una tabella DynamoDB con impaginazione utilizzando un SDK AWS
- Interroga una tabella DynamoDB con letture fortemente coerenti utilizzando un SDK AWS
- Interroga i dati DynamoDB utilizzando istruzioni PartiQL SELECT con un SDK AWS
- Interroga una tabella DynamoDB per gli elementi TTL utilizzando un SDK AWS
- Interroga le tabelle DynamoDB utilizzando modelli di data e ora con un SDK AWS
- Salva EXIF e altre informazioni sull'immagine utilizzando un SDK AWS
- Configurare il controllo degli accessi basato sugli attributi per DynamoDB utilizzando v2 AWS Command Line Interface
- Comprendi l'ordine delle espressioni di aggiornamento in DynamoDB con un SDK AWS
- Aggiorna un'impostazione di tabella DynamoDB con un throughput caldo utilizzando un SDK AWS
- Aggiornare un elemento DynamoDB con un TTL utilizzando un SDK AWS
- Aggiorna i dati DynamoDB utilizzando le istruzioni PartiQL UPDATE con un SDK AWS
- Utilizzo di un'API Gateway per richiamare una funzione Lambda
- Utilizzo di Step Functions per richiamare le funzioni Lambda
- Usa un modello di documento per DynamoDB utilizzando un SDK AWS
- Utilizza un modello di persistenza degli oggetti di alto livello per DynamoDB utilizzando un SDK AWS

- Usa le operazioni di conteggio atomico in DynamoDB con un SDK AWS

- Usa operazioni condizionali in DynamoDB con un SDK AWS
- Usa i nomi degli attributi di espressione in DynamoDB con un SDK AWS
- Utilizzo degli eventi pianificati per richiamare una funzione Lambda
- Lavora con gli indici secondari locali di DynamoDB utilizzando v2 AWS Command Line Interface
- Lavora con DynamoDB Streams e usa v2 Time-to-Live AWS Command Line Interface
- Lavora con le tabelle globali di DynamoDB e la replica multiregione con eventuale coerenza (MREC) utilizzando v2 AWS Command Line Interface
- Lavora con il tagging delle risorse DynamoDB utilizzando la versione 2 AWS Command Line Interface
- Lavora con la crittografia delle tabelle DynamoDB utilizzando la versione 2 AWS Command Line Interface

- Esempi serverless per DynamoDB
  - Richiamare una funzione Lambda da un trigger DynamoDB
  - Segnalazione di errori di elementi batch per funzioni Lambda con un trigger DynamoDB
- AWS contributi della community per DynamoDB
  - Creare e testare un'applicazione serverless

Esempi di base per l'utilizzo di DynamoDB AWS SDKs

I seguenti esempi di codice mostrano come utilizzare le nozioni di base di Amazon DynamoDB con. AWS SDKs

Esempi

- Hello DynamoDB
- Scopri le basi di DynamoDB con un SDK AWS
- Azioni per l'utilizzo di DynamoDB AWS SDKs
  - Utilizzalo BatchExecuteStatement con un AWS SDK
  - Utilizzo BatchGetItem con un AWS SDK o una CLI
  - Utilizzo BatchWriteItem con un AWS SDK o una CLI
  - Utilizzo CreateTable con un AWS SDK o una CLI
  - Utilizzo DeleteItem con un AWS SDK o una CLI
  - Utilizzo DeleteTable con un AWS SDK o una CLI

- Utilizzo DescribeTable con un AWS SDK o una CLI
- Utilizzo DescribeTimeToLive con un AWS SDK o una CLI
- Utilizzare ExecuteStatement con un SDK AWS
- Utilizzo GetItem con un AWS SDK o una CLI
- Utilizzo ListTables con un AWS SDK o una CLI
- Utilizzo PutItem con un AWS SDK o una CLI
- Utilizzo Query con un AWS SDK o una CLI
- Utilizzo Scan con un AWS SDK o una CLI
- Utilizzo UpdateItem con un AWS SDK o una CLI
- Utilizzo UpdateTable con un AWS SDK o una CLI
- Utilizzo UpdateTimeToLive con un AWS SDK o una CLI

Hello DynamoDB

Gli esempi di codice seguenti mostrano come iniziare a utilizzare DynamoDB.

.NET

SDK per .NET (v4)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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}");
        }
    }
}
```

- Per i dettagli sull'API, consulta la ListTablessezione AWS SDK per .NET API Reference.

C++

SDK per C++

> **Note**
>
> C'è di più su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Codice per il CMake file CMake Lists.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})
```

Codice per il file origine 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;
}
```

- Per i dettagli sull'API, consulta la ListTablessezione AWS SDK per C++ API Reference.

Java

SDK per Java 2.x

> **Note**
>
> C'è di più su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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!");
    }
}
```

- Per i dettagli sull'API, consulta la ListTablessezione AWS SDK for Java 2.x API Reference.

JavaScript

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Per maggiori dettagli sull'utilizzo di DynamoDB AWS SDK per JavaScript in, consulta Programmazione di 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;
};
```

- Per i dettagli sulle API, consulta ListTablesla sezione API Reference.AWS SDK per JavaScript

Python

SDK per Python (Boto3)

> **Note**
>
> C'è di più su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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.")
```

- Per i dettagli sull'API, consulta ListTables AWSSDK for Python (Boto3) API Reference.

Ruby

SDK per Ruby

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
```

- Per i dettagli sull'API, consulta la ListTablessezione AWS SDK per Ruby API Reference.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUtilizzo di DynamoDB con un SDK AWS. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Scopri le basi di DynamoDB con un SDK AWS

Gli esempi di codice seguenti mostrano come:

- Crea una tabella in grado di contenere i dati del filmato.
- Inserire, ottenere e aggiornare un singolo filmato nella tabella.
- Scrivere i dati del filmato nella tabella da un file JSON di esempio.
- Eseguire una query sui filmati che sono stati rilasciati in un dato anno.
- Cerca i filmati che sono stati distribuiti in diversi anni.
- Elimina un filmato dalla tabella, quindi elimina la tabella.

.NET

SDK per .NET (v4)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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 tabella per contenere i dati del filmato.

```
    /// <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;
        }
    }
```

Aggiunge un singolo filmato alla tabella.

```
    /// <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;
        }
    }
```

Aggiorna un singolo elemento in una tabella.

```
    /// <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 singolo elemento dalla tabella del filmato.

```
    /// <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;
        }
    }
```

Scrive un batch di elementi nella tabella del filmato.

```
    /// <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 singolo elemento dalla tabella.

```
    /// <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;
        }
    }
```

Esegue una query sulla tabella per i filmati usciti in un determinato anno.

```
    /// <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;
        }
    }
```

Esegue la ricerca dei filmati che sono usciti in un intervallo di anni.

```
    /// <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 tabella del filmato.

```
    /// <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;
        }
    }
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento delle API AWS SDK per .NET .
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Query
  - Scan
  - UpdateItem

Bash

AWS CLI con lo script Bash

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Lo scenario introduttivo di 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
}
```

Le funzioni DynamoDB utilizzate in questo scenario.

```
###############################################################################
# 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
}
```

Le funzioni di utility utilizzate in questo scenario.

```
###############################################################################
# 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
}
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento dei comandi AWS CLI .
  - BatchWriteItem
  - CreateTable
  - DeleteItem

- DeleteTable
- DescribeTable
- GetItem
- PutItem
- Query
- Scan
- UpdateItem

C++

SDK per C++

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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;
}
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento delle API AWS SDK per C++ .
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Query
  - Scan
  - UpdateItem

Go

SDK per Go V2

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Esegui uno scenario interattivo per creare la tabella ed eseguire azioni su di essa.

```
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))
}
```

Per definire una struttura di filmato utilizzata in questo esempio.

```
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"])
}
```

Crea una struttura e metodi che chiamano azioni 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
}
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento delle API AWS SDK per Go .
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Query
  - Scan
  - UpdateItem

Java

SDK per Java 2.x

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Creazione di una tabella 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);
        }
    }
```

Crea una funzione helper per scaricare ed estrarre il file JSON di esempio.

```
    // 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++;
        }
    }
```

Ottieni un elemento da una tabella

```
    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);
```

```
        }
    }
```

Esempio 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);
        }
```

```
    }
}
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento delle API AWS SDK for Java 2.x .
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Query
  - Scan
  - UpdateItem

JavaScript

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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.");
};
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento delle API AWS SDK per JavaScript .
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Query
  - Scan
  - UpdateItem

Kotlin

SDK per Kotlin

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Creazione di una tabella 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}")
    }
}
```

Crea una funzione helper per scaricare ed estrarre il file JSON di esempio.

```
// 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.")
    }
}
```

Ottieni un elemento da una tabella

```
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)
        }
    }
}
```

Esempio 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]}")
            }
        }
    }
}
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento delle API SDK AWS per Kotlin.
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Query
  - Scan
  - UpdateItem

PHP

SDK per PHP

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Poiché questo esempio utilizza file di supporto, assicurati di leggere le indicazioni contenute nel file README.md degli esempi 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);
    }
}
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento delle API AWS SDK per PHP .
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable

- DescribeTable
- GetItem
- PutItem
- Query
- Scan
- UpdateItem

Python

SDK per Python (Boto3)

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Crea una classe che incapsula una tabella 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
```

Crea una funzione helper per scaricare ed estrarre il file JSON di esempio.

```
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]
```

Esegui uno scenario interattivo per creare la tabella ed eseguire azioni su di essa.

```
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}")
```

In questo scenario viene utilizzata la seguente classe helper per porre domande al prompt dei comandi.

```
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
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento delle API SDK AWS per Python (Boto3).
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Query
  - Scan
  - UpdateItem

Ruby

SDK per Ruby

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Crea una classe che incapsula una tabella 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
```

Crea una funzione helper per scaricare ed estrarre il file JSON di esempio.

```
  # 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
```

Esegui uno scenario interattivo per creare la tabella ed eseguire azioni su di essa.

```
  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
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento delle API AWS SDK per Ruby .
  - BatchWriteItem

- CreateTable
- DeleteItem
- DeleteTable
- DescribeTable
- GetItem
- PutItem
- Query
- Scan
- UpdateItem

SAP ABAP

SDK per SAP ABAP

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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.
```

- Per informazioni dettagliate sulle API, consulta i seguenti argomenti nella Documentazione di riferimento delle API SDK AWS per SAP ABAP.
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Query
  - Scan
  - UpdateItem

Swift

SDK per Swift

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Una classe Swift che gestisce le chiamate DynamoDB all'SDK per 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
        }
    }
}
```

Le strutture utilizzate dalla MovieTable classe per rappresentare i film.

```
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 programma che utilizza la MovieTable classe per accedere a un database 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)
        }
    }
}
```

- Per informazioni dettagliate sulle API, consulta i seguenti argomenti nella Documentazione di riferimento delle API SDK AWS per Swift.
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Query
  - Scan
  - UpdateItem

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Utilizzo di DynamoDB con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Azioni per l'utilizzo di DynamoDB AWS SDKs

I seguenti esempi di codice mostrano come eseguire singole azioni DynamoDB con. AWS SDKs Ogni esempio include un collegamento a GitHub, dove è possibile trovare le istruzioni per la configurazione e l'esecuzione del codice.

Questi estratti richiamano l'API DynamoDB e sono estratti di codice da programmi più grandi che devono essere eseguiti in modo contestuale. È possibile visualizzare le azioni nel contesto in Scenari per l'utilizzo di DynamoDB AWS SDKs .

Gli esempi seguenti includono solo le operazioni più comunemente utilizzate. Per un elenco completo, consulta la Documentazione di riferimento delle API Amazon DynamoDB.

Esempi

- Utilizzalo BatchExecuteStatement con un AWS SDK
- Utilizzo BatchGetItem con un AWS SDK o una CLI
- Utilizzo BatchWriteItem con un AWS SDK o una CLI
- Utilizzo CreateTable con un AWS SDK o una CLI
- Utilizzo DeleteItem con un AWS SDK o una CLI
- Utilizzo DeleteTable con un AWS SDK o una CLI
- Utilizzo DescribeTable con un AWS SDK o una CLI
- Utilizzo DescribeTimeToLive con un AWS SDK o una CLI
- Utilizzare ExecuteStatement con un SDK AWS
- Utilizzo GetItem con un AWS SDK o una CLI
- Utilizzo ListTables con un AWS SDK o una CLI
- Utilizzo PutItem con un AWS SDK o una CLI
- Utilizzo Query con un AWS SDK o una CLI
- Utilizzo Scan con un AWS SDK o una CLI
- Utilizzo UpdateItem con un AWS SDK o una CLI
- Utilizzo UpdateTable con un AWS SDK o una CLI
- Utilizzo UpdateTimeToLive con un AWS SDK o una CLI

Utilizzalo **`BatchExecuteStatement`** con un AWS SDK

Gli esempi di codice seguenti mostrano come utilizzare `BatchExecuteStatement`.

Gli esempi di operazioni sono estratti di codice da programmi più grandi e devono essere eseguiti nel contesto. È possibile visualizzare questa operazione nel contesto nei seguenti esempi di codice:

- Eliminare i dati utilizzando PartiQL DELETE

- Inserimento di dati con istruzioni PartiQL INSERT
- Esecuzione di una query su una tabella mediante batch di istruzioni PartiQL
- Esecuzione di query sui dati utilizzando istruzioni PartiQL SELECT
- Aggiornamento dei dati con PartiQL UPDATE

.NET

SDK per .NET

> **Note**
>
> C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Utilizzo di batch di istruzioni INSERT per aggiungere elementi.

```
        /// <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!;
            }
        }
```

Utilizzo di batch di istruzioni SELECT per ottenere elementi.

```
        /// <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;
            }

        }
```

Utilizzo di batch di istruzioni UPDATE per aggiornare elementi.

```
        /// <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;
        }
```

Utilizzo di batch di istruzioni DELETE per eliminare elementi.

```
        /// <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;
        }
```

- Per i dettagli sull'API, consulta la BatchExecuteStatementsezione AWS SDK per .NET API Reference.

C++

SDK per C++

> **Note**
>
> C'è di più su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Utilizzo di batch di istruzioni INSERT per aggiungere elementi.

```
    // 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;
        }
    }
```

Utilizzo di batch di istruzioni SELECT per ottenere elementi.

```
    // 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;
        }
    }
```

Utilizzo di batch di istruzioni UPDATE per aggiornare elementi.

```
    // 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;
        }
    }
```

Utilizzo di batch di istruzioni DELETE per eliminare elementi.

```
   // 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;
       }
   }
```

- Per i dettagli sull'API, consulta la BatchExecuteStatementsezione AWS SDK per C++ API Reference.

Go

SDK per Go V2

> **Note**
>
> C'è di più su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Per definire una struttura di ricevitore di funzioni per l'esempio.

```
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
}
```

Utilizzo di batch di istruzioni INSERT per aggiungere elementi.

```
// 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
}
```

Utilizzo di batch di istruzioni SELECT per ottenere elementi.

```
// 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
}
```

Utilizzo di batch di istruzioni UPDATE per aggiornare elementi.

```
// 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
}
```

Utilizzo di batch di istruzioni DELETE per eliminare elementi.

```
// 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
}
```

Per definire una struttura di filmato utilizzata in questo esempio.

```
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"])
}
```

- Per i dettagli sull'API, consulta la BatchExecuteStatementsezione AWS SDK per Go API Reference.

JavaScript

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Creazione di un batch di elementi mediante 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;
};
```

Ottenimento di un batch di elementi mediante 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;
};
```

Aggiornamento di un batch di elementi mediante 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;
};
```

Eliminazione di un batch di elementi mediante 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;
};
```

- Per i dettagli sull'API, consulta la BatchExecuteStatementsezione AWS SDK per JavaScript API Reference.

PHP

SDK per PHP

> **Note**
>
> C'è di più su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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,
                ],
            ],
        ]);
    }
```

- Per i dettagli sull'API, consulta la BatchExecuteStatementsezione AWS SDK per PHP API Reference.

Python

SDK per Python (Boto3)

> **Note**
>
> C'è di più su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
```

- Per i dettagli sull'API, consulta BatchExecuteStatement AWSSDK for Python (Boto3) API Reference.

Ruby

SDK per Ruby

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Lettura di un batch di elementi mediante 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
```

Eliminazione di un batch di elementi mediante 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
```

- Per i dettagli sull'API, consulta la BatchExecuteStatementsezione AWS SDK per Ruby API Reference.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUtilizzo di DynamoDB con un SDK AWS. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo **`BatchGetItem`** con un AWS SDK o una CLI

Gli esempi di codice seguenti mostrano come utilizzare `BatchGetItem`.

.NET

SDK per .NET

> **Note**
>
> C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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);
        }
    }
}
```

- Per i dettagli sull'API, consulta la BatchGetItemsezione AWS SDK per .NET API Reference.

Bash

AWS CLI con lo script Bash

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
}
```

Le funzioni di utilità utilizzate in questo esempio.

```
###############################################################################
# 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
}
```

- Per i dettagli sull'API, consulta BatchGetItem AWS CLICommand Reference.

C++

SDK per C++

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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;
}
```

- Per i dettagli sull'API, consulta la BatchGetItemsezione AWS SDK per C++ API Reference.

CLI

AWS CLI

Come recuperare più elementi da una tabella

Nell'esempio seguente `batch-get-items` legge più elementi dalla tabella `MusicCollection` utilizzando un batch di tre richieste `GetItem` e richiede il numero di unità di capacità di lettura utilizzate dall'operazione. Il comando restituisce solo l'attributo `AlbumTitle`.

```
aws dynamodb batch-get-item \
    --request-items file://request-items.json \
    --return-consumed-capacity TOTAL
```

Contenuto di `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"
    }
}
```

Output:

```
{
    "Responses": {
        "MusicCollection": [
            {
                "AlbumTitle": {
                    "S": "Somewhat Famous"
                }
            },
            {
                "AlbumTitle": {
                    "S": "Blue Sky Blues"
                }
            },
            {
                "AlbumTitle": {
                    "S": "Louder Than Ever"
                }
            }
        ]
    },
    "UnprocessedKeys": {},
    "ConsumedCapacity": [
        {
            "TableName": "MusicCollection",
            "CapacityUnits": 1.5
        }
    ]
}
```

Per ulteriori informazioni, consulta Operazioni batch nella Guida per gli sviluppatori di Amazon DynamoDB.

- Per i dettagli sull'API, consulta BatchGetItem AWS CLICommand Reference.

Java

SDK per Java 2.x

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Mostra come ottenere elementi batch utilizzando il client di servizio.

```
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.");
```

```
        }
    }
}
```

Mostra come ottenere elementi batch utilizzando il client di servizio e un impaginatore.

```
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());
            });
    }
}
```

- Per i dettagli sull'API, consulta la BatchGetItemsezione AWS SDK for Java 2.x API Reference.

JavaScript

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Questo esempio utilizza il client di documenti per semplificare il lavoro con gli elementi in DynamoDB. Per i dettagli sull'API, consulta 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;
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK per JavaScript.
- Per i dettagli sulle API, consulta la BatchGetItemsezione AWS SDK per JavaScript API Reference.

SDK per JavaScript (v2)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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);
    });
  }
});
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK per JavaScript.

- Per i dettagli sull'API, consulta la BatchGetItemsezione AWS SDK per JavaScript API Reference.

PowerShell

Strumenti per PowerShell V4

Esempio 1: ottiene l'elemento con SongTitle «Somewhere Down The Road» dalle tabelle «Music» e «Songs» di 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
```

Output:

```
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
```

- Per i dettagli sull'API, vedere BatchGetItemin AWS Strumenti per PowerShell Cmdlet Reference (V4).

Strumenti per V5 PowerShell

Esempio 1: ottiene l'elemento con SongTitle «Somewhere Down The Road» dalle tabelle «Music» e «Songs» di 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
```

Output:

```
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
```

- Per i dettagli sull'API, vedere BatchGetItemin AWS Strumenti per PowerShell Cmdlet Reference (V5).

Python

SDK per Python (Boto3)

> **Note**
>
> C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
```

- Per i dettagli sull'API, consulta BatchGetItem AWSSDK for Python (Boto3) API Reference.

Swift

SDK per Swift

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
        }
    }
```

- Per i dettagli sull'API, consulta la BatchGetItemguida di riferimento all'API AWS SDK for Swift.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Utilizzo di DynamoDB con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo **`BatchWriteItem`** con un AWS SDK o una CLI

Gli esempi di codice seguenti mostrano come utilizzare `BatchWriteItem`.

Gli esempi di operazioni sono estratti di codice da programmi più grandi e devono essere eseguiti nel contesto. È possibile visualizzare questa operazione nel contesto nel seguente esempio di codice:

- Informazioni di base

.NET

SDK per .NET (v4)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Scrive un batch di elementi nella tabella del filmato.

```
/// <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;
        }
    }
```

- Per i dettagli sull'API, consulta la BatchWriteItemsezione AWS SDK per .NET API Reference.

Bash

AWS CLI con lo script Bash

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
}
```

Le funzioni di utilità utilizzate in questo esempio.

```
###############################################################################
# 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
}
```

- Per i dettagli sull'API, consulta BatchWriteItem AWS CLICommand Reference.

C++

SDK per C++

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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;
}
```

- Per i dettagli sull'API, consulta la BatchWriteItemsezione AWS SDK per C++ API Reference.

CLI

AWS CLI

Come aggiungere più elementi a una tabella

Nell'esempio seguente `batch-write-item` aggiunge tre nuovi elementi alla tabella `MusicCollection` utilizzando un batch di tre richieste `PutItem`. Richiede inoltre informazioni sul numero di unità di capacità di scrittura utilizzate dall'operazione e su eventuali raccolte di elementi modificate dall'operazione.

```
aws dynamodb batch-write-item \
    --request-items file://request-items.json \
    --return-consumed-capacity INDEXES \
    --return-item-collection-metrics SIZE
```

Contenuto di `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"}
                }
            }
        }
    ]
}
```

Output:

```
{
    "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
                }
            }
        }
    ]
}
```

Per ulteriori informazioni, consulta Operazioni batch nella Guida per gli sviluppatori di Amazon DynamoDB.

- Per i dettagli sull'API, consulta BatchWriteItem AWS CLICommand Reference.

Go

SDK per Go V2

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
}
```

Per definire una struttura di filmato utilizzata in questo esempio.

```
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"])
}
```

- Per i dettagli sull'API, consulta la BatchWriteItemsezione AWS SDK per Go API Reference.

Java

SDK per Java 2.x

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Inserisce molti elementi in una tabella utilizzando il client di servizio.

```
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);
        }
    }
}
```

Inserisce molti elementi in una tabella utilizzando il client avanzato.

```
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);
                }
        }
}
```

- Per i dettagli sull'API, BatchWriteItemconsulta AWS SDK for Java 2.x API Reference.

JavaScript

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Questo esempio utilizza il client di documenti per semplificare il lavoro con gli elementi in DynamoDB. Per i dettagli sull'API, consulta 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);
  }
};
```

- Per i dettagli sulle API, consulta la BatchWriteItemsezione AWS SDK per JavaScript API Reference.

SDK per JavaScript (v2)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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);
  }
});
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK per JavaScript.
- Per i dettagli sull'API, BatchWriteItemconsulta AWS SDK per JavaScript API Reference.

PHP

SDK per PHP

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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;
            }
        }
    }
```

- Per i dettagli sull'API, BatchWriteItemconsulta AWS SDK per PHP API Reference.

PowerShell

Strumenti per PowerShell V4

Esempio 1: crea un nuovo elemento o sostituisce un elemento esistente con un nuovo elemento nelle tabelle DynamoDB Musica e Canzoni.

```
$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
```

Output:

```
$requestItem = @{
    'Music' = [Amazon.DynamoDBv2.Model.WriteRequest]($writeRequest)
    'Songs' = [Amazon.DynamoDBv2.Model.WriteRequest]($writeRequest)
}

Set-DDBBatchItem -RequestItem $requestItem
```

- Per i dettagli sull'API, vedere BatchWriteItemin AWS Strumenti per PowerShell Cmdlet Reference (V4).

Strumenti per V5 PowerShell

Esempio 1: crea un nuovo elemento o sostituisce un elemento esistente con un nuovo elemento nelle tabelle DynamoDB Musica e Canzoni.

```
$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
```

Output:

```
$requestItem = @{
    'Music' = [Amazon.DynamoDBv2.Model.WriteRequest]($writeRequest)
    'Songs' = [Amazon.DynamoDBv2.Model.WriteRequest]($writeRequest)
}

Set-DDBBatchItem -RequestItem $requestItem
```

- Per i dettagli sull'API, vedere BatchWriteItemin AWS Strumenti per PowerShell Cmdlet Reference (V5).

Python

SDK per Python (Boto3)

> **Note**
>
> C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
```

- Per i dettagli sull'API, consulta BatchWriteItem AWSSDK for Python (Boto3) API Reference.

Ruby

SDK per Ruby

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
```

- Per i dettagli sull'API, BatchWriteItemconsulta AWS SDK per Ruby API Reference.

Swift

SDK per Swift

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
        }
    }
```

- Per i dettagli sull'API, consulta la BatchWriteItemguida di riferimento all'API AWS SDK for Swift.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Utilizzo di DynamoDB con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo **`CreateTable`** con un AWS SDK o una CLI

Gli esempi di codice seguenti mostrano come utilizzare `CreateTable`.

Gli esempi di operazioni sono estratti di codice da programmi più grandi e devono essere eseguiti nel contesto. È possibile visualizzare questa operazione nel contesto nei seguenti esempi di codice:

- Informazioni di base
- Accelerazione delle letture con DAX
- Scenari avanzati basati sugli indici secondari globali
- Creazione di una tabella con un indice secondario globale
- Creazione di una tabella con throughput a caldo abilitato
- Creazione e gestione di tabelle globali che descrivono la MREC
- Creazione e gestione di tabelle globali MRSC
- Gestione degli indici secondari globali
- Gestione di policy basate su risorse
- Configurazione del controllo degli accessi basato su attributi
- Utilizzo di tabelle globali e replica multi-Regione con coerenza possibile (MREC)

- Utilizzo di indici secondari locali
- Utilizzo dell'assegnazione di tag per le risorse
- Lavora con Streams e Time-to-Live
- Utilizzo della crittografia delle tabelle

.NET

SDK per .NET (v4)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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;
        }
    }
```

- Per i dettagli sull'API, CreateTableconsulta AWS SDK per .NET API Reference.

Bash

AWS CLI con lo script Bash

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
}
```

Le funzioni di utilità utilizzate in questo esempio.

```
##############################################################################
# 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
}
```

- Per i dettagli sull'API, consulta CreateTable AWS CLICommand Reference.

C++

SDK per C++

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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);
}
```

Codice che attende che la tabella diventi attiva.

```
//! 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;
}
```

- Per i dettagli sull'API, CreateTableconsulta AWS SDK per C++ API Reference.

CLI

AWS CLI

Esempio 1: per creare una tabella con tag

Nell'esempio seguente `create-table` utilizza gli attributi e lo schema di chiavi specificati per creare una tabella denominata `MusicCollection`. Questa tabella utilizza la velocità effettiva assegnata ed è crittografata a riposo utilizzando la CMK di AWS proprietà predefinita. Il comando applica inoltre un tag alla tabella, con una chiave `Owner` e un valore `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
```

Output:

```
{
    "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"
    }
}
```

Per ulteriori informazioni, consulta Operazioni di base per le tabelle nella Guida per gli sviluppatori di Amazon DynamoDB.

Esempio 2: per creare una tabella in modalità on demand

Nell'esempio seguente viene creata una tabella denominata `MusicCollection` utilizzando la modalità on demand, anziché la modalità basata su throughput allocato. Ciò risulta utile per le tabelle con carichi di lavoro imprevedibili.

```
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
```

Output:

```
{
    "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"
        }
    }
}
```


Per ulteriori informazioni, consulta Operazioni di base per le tabelle nella Guida per gli sviluppatori di Amazon DynamoDB.

Esempio 3: per creare una tabella e crittografarla con una CMK gestita dal cliente

Nell'esempio seguente viene creata una tabella denominata `MusicCollection` e la crittografa utilizzando una gestita dal cliente (CMK).

```
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
```

Output:

```
{
    "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"
        }
    }
}
```

Per ulteriori informazioni, consulta Operazioni di base per le tabelle nella Guida per gli sviluppatori di Amazon DynamoDB.

Esempio 4: per creare una tabella con un indice secondario locale

Nell'esempio seguente vengono utilizzati gli attributi e lo schema di chiavi specificati per creare una tabella denominata `MusicCollection` con un indice secondario locale denominato `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\"]
                }
            }
        ]"
```

Output:

```
{
    "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"
            }
        ]
    }
}
```

Per ulteriori informazioni, consulta Operazioni di base per le tabelle nella Guida per gli sviluppatori di Amazon DynamoDB.

Esempio 5: per creare una tabella con un indice secondario globale

Nell'esempio seguente viene creata una tabella denominata `GameScores` con un indice secondario globale denominato `GameTitleIndex`. La tabella di base ha una chiave di partizione di `UserId` e una chiave di ordinamento di `GameTitle`, permettendo di trovare

il miglior punteggio di un singolo utente per un gioco specifico in modo efficiente, mentre il GSI ha una chiave di partizione di `GameTitle` e una chiave di ordinamento di `TopScore`, permettendo di trovare rapidamente il punteggio più alto complessivo per un determinato gioco.

```
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
                }
            }
        ]"
```

Output:

```
{
    "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"
            }
        ]
    }
}
```

Per ulteriori informazioni, consulta Operazioni di base per le tabelle nella Guida per gli sviluppatori di Amazon DynamoDB.

Esempio 6: per creare una tabella con più indici secondari globali contemporaneamente

Nell'esempio seguente viene creata una tabella denominata `GameScores` con due indici secondari globali. Gli schemi dei GSI vengono passati tramite un file, anziché nella riga di comando.

```
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
```

Contenuto di `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
        }
    }
]
```

Output:

```
{
    "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"
            }
        ]
    }
}
```

Per ulteriori informazioni, consulta Operazioni di base per le tabelle nella Guida per gli sviluppatori di Amazon DynamoDB.

Esempio 7: per creare una tabella con i flussi abilitati

Nell'esempio seguente viene creata una tabella denominata `GameScores` con i flussi DynamoDB abilitati. Le immagini vecchie e nuove di ciascun elemento verranno scritte nel flusso.

```
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
```

Output:

```
{
    "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"
    }
}
```

Per ulteriori informazioni, consulta Operazioni di base per le tabelle nella Guida per gli sviluppatori di Amazon DynamoDB.

Esempio 8: per creare una tabella con il flusso KEYS_ONLY abilitato

Nell'esempio seguente viene creata una tabella denominata `GameScores` con i flussi DynamoDB abilitati. Solo gli attributi chiave dell'elemento modificato vengono scritti nel flusso.

```
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
```

Output:

```
{
    "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
    }
}
```

Per ulteriori informazioni, consulta Acquisizione dei dati di modifica per DynamoDB Streams nella Guida per gli sviluppatori di Amazon DynamoDB.

Esempio 9: per creare una tabella con la classe Standard-Accesso infrequente

Nell'esempio seguente viene creata una tabella denominata `GameScores` e viene assegnata la classe di tabella Standard-Accesso Infrequente (DynamoDB Standard-IA). Questa classe di tabella è ottimizzata a livello di archiviazione in quanto l'archiviazione rappresenta il costo principale.

```
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
```

Output:

```
{
    "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
    }
}
```

Per ulteriori informazioni, consulta Classi di tabelle nella Guida per gli sviluppatori di Amazon DynamoDB.

Esempio 10: per creare una tabella con l'opzione Protezione da eliminazione abilitata

Nell'esempio seguente viene creata una tabella denominata `GameScores` e viene abilitata la protezione da eliminazione.

```
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
```

Output:

```
{
    "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
    }
}
```

Per ulteriori informazioni, consulta Uso della protezione da eliminazione nella Guida per gli sviluppatori di Amazon DynamoDB.

- Per i dettagli sull'API, consulta Command CreateTableReference AWS CLI .

Go

SDK per Go V2

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
}
```

- Per i dettagli sull'API, CreateTableconsulta AWS SDK per Go API Reference.

Java

SDK per Java 2.x

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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 "";
    }
}
```

- Per i dettagli sull'API, CreateTableconsulta AWS SDK for Java 2.x API Reference.

JavaScript

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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;
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK per JavaScript.
- Per i dettagli sull'API, CreateTableconsulta AWS SDK per JavaScript API Reference.

SDK per JavaScript (v2)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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);
  }
});
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK per JavaScript.
- Per i dettagli sull'API, CreateTableconsulta AWS SDK per JavaScript API Reference.

Kotlin

SDK per Kotlin

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
    }
}
```

- Per i dettagli sull'API, CreateTableconsulta AWS SDK for Kotlin API reference.

PHP

SDK per PHP

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Creare una tabella .

```
        $tableName = "ddb_demo_table_$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],
        ]);
    }
```

- Per i dettagli sull'API, CreateTableconsulta AWS SDK per PHP API Reference.

PowerShell

Strumenti per PowerShell V4

Esempio 1: Questo esempio crea una tabella denominata Thread con una chiave primaria composta da 'ForumName' (hash del tipo di chiave) e 'Subject' (intervallo dei tipi di chiave). Lo schema utilizzato per costruire la tabella può essere inserito in ogni cmdlet come mostrato o specificato mediante il parametro -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
```

Output:

```
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  : {}
```

Esempio 2: Questo esempio crea una tabella denominata Thread con una chiave primaria composta da 'ForumName' (hash del tipo di chiave) e 'Subject' (intervallo dei tipi di chiave). Viene inoltre definito un indice secondario locale. La chiave dell'indice secondario locale verrà impostata automaticamente dalla chiave hash primaria sulla tabella ()ForumName. Lo schema utilizzato per costruire la tabella può essere inserito in ogni cmdlet come mostrato o specificato mediante il parametro -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
```

Output:

```
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}
```

Esempio 3: Questo esempio mostra come utilizzare una singola pipeline per creare una tabella denominata Thread con una chiave primaria composta da " ForumName (hash del tipo di chiave) e 'Subject' (intervallo dei tipi di chiave) e un indice secondario locale. Add- DDBKey Schema e Add- DDBIndex Schema creano automaticamente un nuovo TableSchema oggetto se uno non viene fornito dalla pipeline o dal parametro -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
```

Output:

```
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}
```

- Per i dettagli sull'API, vedere CreateTablein AWS Strumenti per PowerShell Cmdlet Reference (V4).

Strumenti per V5 PowerShell

Esempio 1: Questo esempio crea una tabella denominata Thread con una chiave primaria composta da 'ForumName' (hash del tipo di chiave) e 'Subject' (intervallo dei tipi di chiave). Lo schema utilizzato per costruire la tabella può essere inserito in ogni cmdlet come mostrato o specificato mediante il parametro -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
```

Output:

```
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  : {}
```

Esempio 2: Questo esempio crea una tabella denominata Thread con una chiave primaria composta da 'ForumName' (hash del tipo di chiave) e 'Subject' (intervallo dei tipi di chiave). Viene inoltre definito un indice secondario locale. La chiave dell'indice secondario locale verrà impostata automaticamente dalla chiave hash primaria sulla tabella ()ForumName. Lo schema utilizzato per costruire la tabella può essere inserito in ogni cmdlet come mostrato o specificato mediante il parametro -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
```

Output:

```
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}
```

Esempio 3: Questo esempio mostra come utilizzare una singola pipeline per creare una tabella denominata Thread con una chiave primaria composta da " ForumName (hash del tipo di chiave) e 'Subject' (intervallo dei tipi di chiave) e un indice secondario locale. Add- DDBKey Schema e Add- DDBIndex Schema creano automaticamente un nuovo TableSchema oggetto se uno non viene fornito dalla pipeline o dal parametro -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
```

Output:

```
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}
```

- Per i dettagli sull'API, vedere CreateTablein AWS Strumenti per PowerShell Cmdlet Reference (V5).

Python

SDK per Python (Boto3)

> **Note**
>
> C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Crea una tabella per la memorizzazione dei dati del filmato.

```
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
```

- Per i dettagli sull'API, consulta CreateTable AWSSDK for Python (Boto3) API Reference.

Ruby

SDK per Ruby

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
```

- Per i dettagli sull'API, CreateTableconsulta AWS SDK per Ruby API Reference.

Rust

SDK per Rust

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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))
        }
    }
}
```

- Per i dettagli sulle API, consulta il riferimento CreateTableall'API AWS SDK for Rust.

SAP ABAP

SDK per SAP ABAP

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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.
```

- Per i dettagli sulle API, CreateTableconsulta AWS SDK for SAP ABAP API reference.

Swift

SDK per Swift

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
        }
    }
```

- Per i dettagli sull'API, consulta la CreateTableguida di riferimento all'API AWS SDK for Swift.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Utilizzo di DynamoDB con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo **`DeleteItem`** con un AWS SDK o una CLI

Gli esempi di codice seguenti mostrano come utilizzare `DeleteItem`.

Gli esempi di operazioni sono estratti di codice da programmi più grandi e devono essere eseguiti nel contesto. È possibile visualizzare questa operazione nel contesto nei seguenti esempi di codice:

- Informazioni di base
- Utilizzo di operazioni condizionali

.NET

SDK per .NET (v4)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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;
        }
    }
```

- Per i dettagli sull'API, DeleteItemconsulta AWS SDK per .NET API Reference.

Bash

AWS CLI con lo script Bash

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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

}
```

Le funzioni di utilità utilizzate in questo esempio.

```
###############################################################################
# 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
}
```

- Per i dettagli sull'API, consulta DeleteItem AWS CLICommand Reference.

C++

SDK per C++

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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);
}
```

Codice che attende che la tabella diventi attiva.

```
//! 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;
}
```

- Per i dettagli sull'API, DeleteItemconsulta AWS SDK per C++ API Reference.

CLI

AWS CLI

Esempio 1: eliminare un elemento

Nell'esempio seguente `delete-item` elimina un elemento dalla tabella `MusicCollection` e richiede i dettagli sull'elemento eliminato e sulla capacità utilizzata dalla richiesta.

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

Contenuto di `key.json`:

```
{
    "Artist": {"S": "No One You Know"},
    "SongTitle": {"S": "Scared of My Shadow"}
}
```

Output:

```
{
    "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
        ]
    }
}
```

Per ulteriori informazioni, consulta Scrittura di un elemento nella Guida per gli sviluppatori di Amazon DynamoDB.

Esempio 2: per eliminare un elemento in base a una condizione

Nell'esempio seguente viene eliminato un elemento dalla tabella `ProductCatalog` solo se `ProductCategory` è `Sporting Goods` o `Gardening Supplies` e se il relativo prezzo è compreso tra 500 e 600. Restituisce i dettagli sull'elemento eliminato.

```
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
```

Contenuto di `names.json`:

```
{
    "#P": "Price"
}
```

Contenuto di `values.json`:

```
{
    ":cat1": {"S": "Sporting Goods"},
    ":cat2": {"S": "Gardening Supplies"},
    ":lo": {"N": "500"},
    ":hi": {"N": "600"}
}
```

Output:

```
{
    "Attributes": {
        "Id": {
            "N": "456"
        },
        "Price": {
            "N": "550"
        },
        "ProductCategory": {
            "S": "Sporting Goods"
        }
    }
}
```

Per ulteriori informazioni, consulta Scrittura di un elemento nella Guida per gli sviluppatori di Amazon DynamoDB.

- Per i dettagli sull'API, consulta DeleteItem AWS CLICommand Reference.

Go

SDK per Go V2

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
}
```

Per definire una struttura di filmato utilizzata in questo esempio.

```
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"])
}
```

- Per i dettagli sull'API, DeleteItemconsulta AWS SDK per Go API Reference.

Java

SDK per Java 2.x

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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);
        }
    }
}
```

- Per i dettagli sull'API, DeleteItemconsulta AWS SDK for Java 2.x API Reference.

JavaScript

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Questo esempio utilizza il client di documenti per semplificare il lavoro con gli elementi in DynamoDB. Per i dettagli sull'API, consulta 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;
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK per JavaScript.
- Per i dettagli sulle API, consulta la DeleteItemsezione AWS SDK per JavaScript API Reference.

SDK per JavaScript (v2)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Elimina un item dalla tabella.

```
// 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);
  }
```

```
});
```

Elimina un elemento da una tabella utilizzando il client documento 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);
  }
});
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK per JavaScript.
- Per i dettagli sull'API, DeleteItemconsulta AWS SDK per JavaScript API Reference.

Kotlin

SDK per Kotlin

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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")
    }
}
```

- Per i dettagli sull'API, DeleteItemconsulta AWS SDK for Kotlin API reference.

PHP

SDK per PHP

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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,
        ]);
    }
```

- Per i dettagli sull'API, DeleteItemconsulta AWS SDK per PHP API Reference.

PowerShell

Strumenti per PowerShell V4

Esempio 1: rimuove l'elemento DynamoDB corrispondente alla chiave specificata.

```
$key = @{
    SongTitle = 'Somewhere Down The Road'
    Artist = 'No One You Know'
} | ConvertTo-DDBItem
Remove-DDBItem -TableName 'Music' -Key $key -Confirm:$false
```

- Per i dettagli sull'API, vedere DeleteItemin AWS Strumenti per PowerShell Cmdlet Reference (V4).

Strumenti per V5 PowerShell

Esempio 1: rimuove l'elemento DynamoDB corrispondente alla chiave specificata.

```
$key = @{
    SongTitle = 'Somewhere Down The Road'
    Artist = 'No One You Know'
} | ConvertTo-DDBItem
Remove-DDBItem -TableName 'Music' -Key $key -Confirm:$false
```

- Per i dettagli sull'API, vedere DeleteItemin AWS Strumenti per PowerShell Cmdlet Reference (V5).

Python

SDK per Python (Boto3)

> **Note**
>
> C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
```

È possibile specificare una condizione in modo che un elemento venga eliminato solo quando soddisfa determinati criteri.

```
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
```

- Per i dettagli sull'API, consulta DeleteItem AWSSDK for Python (Boto3) API Reference.

Ruby

SDK per Ruby

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
```

- Per i dettagli sull'API, DeleteItemconsulta AWS SDK per Ruby API Reference.

Rust

SDK per Rust

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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)),
    }
}
```

- Per i dettagli sulle API, consulta il riferimento DeleteItemall'API AWS SDK for Rust.

SAP ABAP

SDK per SAP ABAP

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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.
```

- Per i dettagli sulle API, DeleteItemconsulta AWS SDK for SAP ABAP API reference.

Swift

SDK per Swift

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
        }
    }
```

- Per i dettagli sull'API, consulta la DeleteItemguida di riferimento all'API AWS SDK for Swift.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Utilizzo di DynamoDB con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo **`DeleteTable`** con un AWS SDK o una CLI

Gli esempi di codice seguenti mostrano come utilizzare `DeleteTable`.

Gli esempi di operazioni sono estratti di codice da programmi più grandi e devono essere eseguiti nel contesto. È possibile visualizzare questa operazione nel contesto nei seguenti esempi di codice:

- Informazioni di base
- Accelerazione delle letture con DAX
- Creazione e gestione di tabelle globali MRSC
- Gestione degli indici secondari globali
- Utilizzo di tabelle globali e replica multi-Regione con coerenza possibile (MREC)

.NET

SDK per .NET (v4)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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;
        }
    }
```

- Per i dettagli sull'API, DeleteTableconsulta AWS SDK per .NET API Reference.

Bash

AWS CLI con lo script Bash

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
}
```

Le funzioni di utilità utilizzate in questo esempio.

```
###############################################################################
# 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
}
```

- Per i dettagli sull'API, consulta DeleteTable AWS CLICommand Reference.

C++

SDK per C++

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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();
}
```

- Per i dettagli sull'API, DeleteTableconsulta AWS SDK per C++ API Reference.

CLI

AWS CLI

Come eliminare una tabella

Nell'esempio seguente `delete-table` elimina la tabella `MusicCollection`.

```
aws dynamodb delete-table \
    --table-name MusicCollection
```

Output:

```
{
    "TableDescription": {
        "TableStatus": "DELETING",
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableName": "MusicCollection",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "WriteCapacityUnits": 5,
            "ReadCapacityUnits": 5
        }
    }
}
```

Per ulteriori informazioni, consulta Eliminazione di una tabella nella Guida per gli sviluppatori di Amazon DynamoDB.

- Per i dettagli sull'API, consulta DeleteTable AWS CLICommand Reference.

Go

SDK per Go V2

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
}
```

- Per i dettagli sull'API, DeleteTableconsulta AWS SDK per Go API Reference.

Java

SDK per Java 2.x

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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!");
    }
}
```

- Per i dettagli sull'API, DeleteTableconsulta AWS SDK for Java 2.x API Reference.

JavaScript

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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;
};
```

- Per i dettagli sull'API, DeleteTableconsulta AWS SDK per JavaScript API Reference.

SDK per JavaScript (v2)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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);
  }
});
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK per JavaScript.
- Per i dettagli sull'API, DeleteTableconsulta AWS SDK per JavaScript API Reference.

Kotlin

SDK per Kotlin

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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")
    }
```

```
}
```

- Per i dettagli sull'API, DeleteTableconsulta AWS SDK for Kotlin API reference.

PHP

SDK per PHP

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
    public function deleteTable(string $TableName)
    {
        $this->customWaiter(function () use ($TableName) {
            return $this->dynamoDbClient->deleteTable([
                'TableName' => $TableName,
            ]);
        });
    }
```

- Per i dettagli sull'API, DeleteTableconsulta AWS SDK per PHP API Reference.

PowerShell

Strumenti per PowerShell V4

Esempio 1: elimina la tabella specificata. Prima di procedere con l'operazione, viene richiesta una conferma.

```
Remove-DDBTable -TableName "myTable"
```

Esempio 2: elimina la tabella specificata. Prima di procedere con l'operazione, non viene richiesta una conferma.

```
Remove-DDBTable -TableName "myTable" -Force
```

- Per i dettagli sull'API, vedere DeleteTablein AWS Strumenti per PowerShell Cmdlet Reference (V4).

Strumenti per V5 PowerShell

Esempio 1: elimina la tabella specificata. Prima di procedere con l'operazione, viene richiesta una conferma.

```
Remove-DDBTable -TableName "myTable"
```

Esempio 2: elimina la tabella specificata. Prima di procedere con l'operazione, non viene richiesta una conferma.

```
Remove-DDBTable -TableName "myTable" -Force
```

- Per i dettagli sull'API, vedere DeleteTablein AWS Strumenti per PowerShell Cmdlet Reference (V5).

Python

SDK per Python (Boto3)

> **Note**
>
> C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
```

- Per i dettagli sull'API, consulta DeleteTable AWSSDK for Python (Boto3) API Reference.

Ruby

SDK per Ruby

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
```

- Per i dettagli sull'API, DeleteTableconsulta AWS SDK per Ruby API Reference.

Rust

SDK per Rust

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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())),
    }
}
```

- Per i dettagli sulle API, consulta il riferimento DeleteTableall'API AWS SDK for Rust.

SAP ABAP

SDK per SAP ABAP

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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.
```

- Per i dettagli sulle API, DeleteTableconsulta AWS SDK for SAP ABAP API reference.

Swift

SDK per Swift

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
        }
```

```
}
```

- Per i dettagli sull'API, consulta la DeleteTableguida di riferimento all'API AWS SDK for Swift.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Utilizzo di DynamoDB con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo **`DescribeTable`** con un AWS SDK o una CLI

Gli esempi di codice seguenti mostrano come utilizzare `DescribeTable`.

Gli esempi di operazioni sono estratti di codice da programmi più grandi e devono essere eseguiti nel contesto. È possibile visualizzare questa operazione nel contesto nei seguenti esempi di codice:

- Informazioni di base
- Creazione e gestione di tabelle globali che descrivono la MREC
- Creazione e gestione di tabelle globali MRSC
- Utilizzo di tabelle globali e replica multi-Regione con coerenza possibile (MREC)
- Lavora con Streams e Time-to-Live
- Utilizzo della crittografia delle tabelle

.NET

SDK per .NET

> **Note**
>
> C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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}");

    }
```

- Per i dettagli sull'API, DescribeTableconsulta AWS SDK per .NET API Reference.

Bash

AWS CLI con lo script Bash

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
}
```

Le funzioni di utilità utilizzate in questo esempio.

```
###############################################################################
# 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
}
```

- Per i dettagli sull'API, consulta DescribeTable AWS CLICommand Reference.

C++

SDK per C++

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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();
}
```

- Per i dettagli sull'API, DescribeTableconsulta AWS SDK per C++ API Reference.

CLI

AWS CLI

Come descrivere una tabella

Nell'esempio seguente `describe-table` descrive la tabella `MusicCollection`.

```
aws dynamodb describe-table \
    --table-name MusicCollection
```

Output:

```
{
    "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
    }
}
```

Per ulteriori informazioni, consulta Descrizione di una tabella nella Guida per gli sviluppatori di Amazon DynamoDB.

- Per i dettagli sull'API, consulta DescribeTable AWS CLICommand Reference.

Go

SDK per Go V2

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
}
```

- Per i dettagli sull'API, DescribeTableconsulta AWS SDK per Go API Reference.

Java

SDK per Java 2.x

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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!");
    }
}
```

- Per i dettagli sull'API, DescribeTableconsulta AWS SDK for Java 2.x API Reference.

JavaScript

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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;
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK per JavaScript.
- Per i dettagli sull'API, DescribeTableconsulta AWS SDK per JavaScript API Reference.

SDK per JavaScript (v2)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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);
  }
});
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK per JavaScript.
- Per i dettagli sull'API, DescribeTableconsulta AWS SDK per JavaScript API Reference.

PowerShell

Strumenti per PowerShell V4

Esempio 1: restituisce i dettagli della tabella specificata.

```
Get-DDBTable -TableName "myTable"
```

- Per i dettagli sull'API, vedere DescribeTablein AWS Strumenti per PowerShell Cmdlet Reference (V4).

Strumenti per V5 PowerShell

Esempio 1: restituisce i dettagli della tabella specificata.

```
Get-DDBTable -TableName "myTable"
```

- Per i dettagli sull'API, vedere DescribeTablein AWS Strumenti per PowerShell Cmdlet Reference (V5).

Python

SDK per Python (Boto3)

> **Note**
>
> C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
```

- Per i dettagli sull'API, consulta DescribeTable AWSSDK for Python (Boto3) API Reference.

Ruby

SDK per Ruby

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
```

- Per i dettagli sull'API, DescribeTableconsulta AWS SDK per Ruby API Reference.

SAP ABAP

SDK per SAP ABAP

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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.
```

- Per i dettagli sulle API, DescribeTableconsulta AWS SDK for SAP ABAP API reference.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Utilizzo di DynamoDB con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo **`DescribeTimeToLive`** con un AWS SDK o una CLI

Gli esempi di codice seguenti mostrano come utilizzare `DescribeTimeToLive`.

Gli esempi di operazioni sono estratti di codice da programmi più grandi e devono essere eseguiti nel contesto. È possibile visualizzare questa operazione nel contesto nel seguente esempio di codice:

- Lavora con Streams e Time-to-Live

CLI

AWS CLI

Per visualizzare le impostazioni del Time to Live per una tabella

Nell'esempio seguente `describe-time-to-live` mostra le impostazioni del Time to Live per la tabella `MusicCollection`.

```
aws dynamodb describe-time-to-live \
    --table-name MusicCollection
```

Output:

```
{
    "TimeToLiveDescription": {
        "TimeToLiveStatus": "ENABLED",
        "AttributeName": "ttl"
    }
}
```

Per ulteriori informazioni, consulta Time to Live nella Guida per gli sviluppatori di Amazon DynamoDB.

- Per i dettagli sull'API, consulta DescribeTimeToLive AWS CLICommand Reference.

Java

SDK per Java 2.x

Per descrivere la configurazione del TTL in una tabella DynamoDB esistente 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.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;
        }
    }
```

- Per i dettagli sull'API, DescribeTimeToLiveconsulta AWS SDK for Java 2.x API Reference.

JavaScript

SDK per JavaScript (v3)

Per descrivere la configurazione del TTL in una tabella DynamoDB esistente con AWS SDK per 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');
```

- Per i dettagli sull'API, consulta la sezione AWS SDK per JavaScript API DescribeTimeToLiveReference.

Python

SDK per Python (Boto3)

Per descrivere la configurazione del TTL in una tabella DynamoDB esistente con 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")
```

- Per i dettagli sull'API, consulta DescribeTimeToLive AWSSDK for Python (Boto3) API Reference.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Utilizzo di DynamoDB con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzare **`ExecuteStatement`** con un SDK AWS

Gli esempi di codice seguenti mostrano come utilizzare `ExecuteStatement`.

Gli esempi di operazioni sono estratti di codice da programmi più grandi e devono essere eseguiti nel contesto. È possibile visualizzare questa operazione nel contesto nei seguenti esempi di codice:

- Eliminare i dati utilizzando PartiQL DELETE
- Inserimento di dati con istruzioni PartiQL INSERT
- Esecuzione di una query mediante PartiQL
- Esecuzione di query sui dati utilizzando istruzioni PartiQL SELECT
- Aggiornamento dei dati con PartiQL UPDATE

.NET

SDK per .NET

> **Note**
>
> C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Utilizzo di un'istruzione INSERT per aggiungere 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;
        }
```

Utilizzo di un'istruzione SELECT per ottenere 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;
        }
```

Utilizzo di un'istruzione SELECT per ottenere un elenco di elementi.

```
        /// <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;
        }
```

Utilizzo di un'istruzione UPDATE per aggiornare 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;
        }
```

Utilizzo di un'istruzione DELETE per eliminare un singolo filmato.

```
        /// <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;
        }
```

- Per i dettagli sull'API, ExecuteStatementconsulta AWS SDK per .NET API Reference.

C++

SDK per C++

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Utilizzo di un'istruzione INSERT per aggiungere 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;
       }
   }
```

Utilizzo di un'istruzione SELECT per ottenere 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;
            }
        }
    }
```

Utilizzo di un'istruzione UPDATE per aggiornare 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;
        }
    }
```

Utilizzo di un'istruzione DELETE per eliminare 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;
        }
    }
```

- Per i dettagli sull'API, ExecuteStatementconsulta AWS SDK per C++ API Reference.

Go

SDK per Go V2

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Per definire una struttura di ricevitore di funzioni per l'esempio.

```
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
}
```

Utilizzo di un'istruzione INSERT per aggiungere 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
}
```

Utilizzo di un'istruzione SELECT per ottenere 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
}
```

Utilizzo di un'istruzione SELECT per ottenere un elenco di elementi e proiettare i risultati.

```
// 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
}
```

Utilizzo di un'istruzione UPDATE per aggiornare 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
}
```

Utilizzo di un'istruzione DELETE per eliminare 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
}
```

Per definire una struttura di filmato utilizzata in questo esempio.

```
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"])
}
```

- Per i dettagli sull'API, ExecuteStatementconsulta AWS SDK per Go API Reference.

JavaScript

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Creazione di un elemento mediante 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;
};
```

Ottenimento di un elemento mediante 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;
};
```

Aggiornamento di un elemento mediante 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;
};
```

Eliminazione di un elemento mediante 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;
};
```

- Per i dettagli sull'API, ExecuteStatementconsulta AWS SDK per JavaScript API Reference.

PHP

SDK per PHP

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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,
        ]);
    }
```

- Per i dettagli sull'API, ExecuteStatementconsulta AWS SDK per PHP API Reference.

Python

SDK per Python (Boto3)

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
```

- Per i dettagli sull'API, consulta ExecuteStatement AWSSDK for Python (Boto3) API Reference.

Ruby

SDK per Ruby

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Selezione di un elemento mediante 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
```

Aggiornamento di un elemento mediante 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
```

Aggiunta di un elemento mediante 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
```

Eliminazione di un elemento mediante 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
```

- Per i dettagli sull'API, ExecuteStatementconsulta AWS SDK per Ruby API Reference.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUtilizzo di DynamoDB con un SDK AWS. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo **`GetItem`** con un AWS SDK o una CLI

Gli esempi di codice seguenti mostrano come utilizzare `GetItem`.

Gli esempi di operazioni sono estratti di codice da programmi più grandi e devono essere eseguiti nel contesto. È possibile visualizzare questa operazione nel contesto nei seguenti esempi di codice:

- Informazioni di base
- Accelerazione delle letture con DAX
- Creazione e gestione di tabelle globali MRSC
- Utilizzo di tabelle globali e replica multi-Regione con coerenza possibile (MREC)

.NET

SDK per .NET (v4)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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;
        }
    }
```

- Per i dettagli sull'API, GetItemconsulta AWS SDK per .NET API Reference.

Bash

AWS CLI con lo script Bash

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
}
```

Le funzioni di utilità utilizzate in questo esempio.

```
###############################################################################
# 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
}
```

- Per i dettagli sull'API, consulta GetItem AWS CLICommand Reference.

C++

SDK per C++

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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();
```

```
}
```

- Per i dettagli sull'API, GetItemconsulta AWS SDK per C++ API Reference.

CLI

AWS CLI

Esempio 1: per leggere un elemento in una tabella

Nell'esempio seguente `get-item` recupera un elemento dalla tabella `MusicCollection`. La tabella ha una chiave hash-and-range primaria (`ArtisteSongTitle`), quindi è necessario specificare entrambi questi attributi. Il comando richiede anche informazioni sulla capacità di scrittura utilizzata dall'operazione.

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

Contenuto di `key.json`:

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

Output:

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

```
    },
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 0.5
    }
}
```

Per ulteriori informazioni, consulta Lettura di un elemento nella Guida per gli sviluppatori di Amazon DynamoDB.

Esempio 2: per leggere un elemento utilizzando una lettura consistente

L'esempio seguente recupera un elemento dalla tabella `MusicCollection` tramite operazioni a elevata consistenza di lettura.

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

Contenuto di `key.json`:

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

Output:

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

```
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 1.0
    }
}
```

Per ulteriori informazioni, consulta Lettura di un elemento nella Guida per gli sviluppatori di Amazon DynamoDB.

Esempio 3: per recuperare attributi specifici di un elemento

Nell'esempio seguente viene utilizzata un'espressione di proiezione per recuperare solo tre attributi dell'elemento desiderato.

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

Contenuto di `names.json`:

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

Output:

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

```
}
```

Per ulteriori informazioni, consulta Lettura di un elemento nella Guida per gli sviluppatori di Amazon DynamoDB.

- Per i dettagli sull'API, consulta GetItem AWS CLICommand Reference.

Go

SDK per Go V2

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
}
```

Per definire una struttura di filmato utilizzata in questo esempio.

```
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"])
}
```

- Per i dettagli sull'API, GetItemconsulta AWS SDK per Go API Reference.

Java

SDK per Java 2.x

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Ottiene un elemento da una tabella utilizzando 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);
```

```
        }
    }
}
```

- Per i dettagli sull'API, vedere GetItemin AWS SDK for Java 2.x API Reference.

JavaScript

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Questo esempio utilizza il client di documenti per semplificare il lavoro con gli elementi in DynamoDB. Per i dettagli sull'API, consulta 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;
};
```

- Per i dettagli sulle API, consulta la GetItemsezione AWS SDK per JavaScript API Reference.

SDK per JavaScript (v2)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Ottieni un elemento da una tabella.

```
// 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);
  }
});
```

Ottieni un elemento da una tabella utilizzando il client documento 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);
  }
});
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK per JavaScript.
- Per i dettagli sull'API, GetItemconsulta AWS SDK per JavaScript API Reference.

Kotlin

SDK per Kotlin

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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)
        }
    }
}
```

- Per i dettagli sull'API, GetItemconsulta AWS SDK for Kotlin API reference.

PHP

SDK per PHP

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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,
        ]);
    }
```

- Per i dettagli sull'API, GetItemconsulta AWS SDK per PHP API Reference.

PowerShell

Strumenti per PowerShell V4

Esempio 1: restituisce l'elemento DynamoDB con la chiave di partizione e la SongTitle chiave di ordinamento Artist.

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

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

Output:

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

- Per i dettagli sull'API, vedere GetItemin AWS Strumenti per PowerShell Cmdlet Reference (V4).

Strumenti per V5 PowerShell

Esempio 1: restituisce l'elemento DynamoDB con la chiave di partizione e la SongTitle chiave di ordinamento Artist.

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

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

Output:

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

- Per i dettagli sull'API, vedere GetItemin AWS Strumenti per PowerShell Cmdlet Reference (V5).

Python

SDK per Python (Boto3)

> **Note**
>
> C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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"]
```

- Per i dettagli sull'API, consulta GetItem AWSSDK for Python (Boto3) API Reference.

Ruby

SDK per Ruby

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
```

- Per i dettagli sull'API, GetItemconsulta AWS SDK per Ruby API Reference.

SAP ABAP

SDK per SAP ABAP

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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.
```

- Per i dettagli sulle API, GetItemconsulta AWS SDK for SAP ABAP API reference.

Swift

SDK per Swift

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
        }
    }
```

- Per i dettagli sull'API, consulta la GetItemguida di riferimento all'API AWS SDK for Swift.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Utilizzo di DynamoDB con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo **`ListTables`** con un AWS SDK o una CLI

Gli esempi di codice seguenti mostrano come utilizzare `ListTables`.

Gli esempi di operazioni sono estratti di codice da programmi più grandi e devono essere eseguiti nel contesto. È possibile visualizzare questa operazione nel contesto nel seguente esempio di codice:

- Configurazione del controllo degli accessi basato su attributi

.NET

SDK per .NET

> **Note**
>
> C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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);
    }
```

- Per i dettagli sull'API, ListTablesconsulta AWS SDK per .NET API Reference.

Bash

AWS CLI con lo script Bash

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
```

```
}
```

Le funzioni di utilità utilizzate in questo esempio.

```
###############################################################################
# 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
}
```

- Per i dettagli sull'API, consulta ListTables AWS CLICommand Reference.

C++

SDK per C++

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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;
}
```

- Per i dettagli sull'API, ListTablesconsulta AWS SDK per C++ API Reference.

CLI

AWS CLI

Esempio 1: elencare le tabelle

L'`list-tables`esempio seguente elenca tutte le tabelle associate all' AWS account corrente e alla regione.

```
aws dynamodb list-tables
```

Output:

```
{
    "TableNames": [
        "Forum",
        "ProductCatalog",
        "Reply",
        "Thread"
    ]
}
```

Per ulteriori informazioni, consulta Elencazione dei nomi delle tabelle nella Guida per gli sviluppatori di Amazon DynamoDB.

Esempio 2: per imitare le dimensioni della pagina

Nell'esempio seguente viene restituito un elenco di tutte le tabelle esistenti, ma viene recuperato solo un elemento in ogni chiamata, eseguendo più chiamate se necessario per ottenere l'intero elenco. La limitazione delle dimensioni della pagina è utile quando si eseguono comandi di tipo elenco su un numero elevato di risorse, il che può causare un errore di "timeout" quando si utilizza la dimensione di pagina predefinita di 1000.

```
aws dynamodb list-tables \
    --page-size 1
```

Output:

```
{
    "TableNames": [
        "Forum",
        "ProductCatalog",
        "Reply",
        "Thread"
    ]
}
```

Per ulteriori informazioni, consulta Elencazione dei nomi delle tabelle nella Guida per gli sviluppatori di Amazon DynamoDB.

Esempio 3: per limitare il numero di elementi restituiti

Nell'esempio seguente il numero di righe restituite è limitato a 2. La risposta include un valore `NextToken` con cui recuperare la pagina di risultati successiva.

```
aws dynamodb list-tables \
    --max-items 2
```

Output:

```
{
    "TableNames": [
        "Forum",
        "ProductCatalog"
    ],
    "NextToken":
 "abCDeFGhiJKlmnOPqrSTuvwxYZ1aBCdEFghijK7LM51nOpqRSTuv3WxY3ZabC5dEFGhI2Jk3LmnoPQ6RST9"
}
```

Per ulteriori informazioni, consulta Elencazione dei nomi delle tabelle nella Guida per gli sviluppatori di Amazon DynamoDB.

Esempio 4: per recuperare la pagina successiva dei risultati

Il seguente comando utilizza il valore `NextToken` di una precedente chiamata al comando `list-tables` per recuperare un'altra pagina di risultati. Poiché la risposta in questo caso non include un valore `NextToken`, ciò significa che è stata raggiunta la fine dei risultati.

```
aws dynamodb list-tables \
    --starting-
token abCDeFGhiJKlmnOPqrSTuvwxYZ1aBCdEFghijK7LM51nOpqRSTuv3WxY3ZabC5dEFGhI2Jk3LmnoPQ6RST9
```

Output:

```
{
    "TableNames": [
        "Reply",
        "Thread"
    ]
}
```

Per ulteriori informazioni, consulta Elencazione dei nomi delle tabelle nella Guida per gli sviluppatori di Amazon DynamoDB.

- Per i dettagli sull'API, consulta ListTables AWS CLICommand Reference.

Go

SDK per Go V2

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
}
```

- Per i dettagli sull'API, ListTablesconsulta AWS SDK per Go API Reference.

Java

SDK per Java 2.x

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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!");
    }
}
```

- Per i dettagli sull'API, ListTablesconsulta AWS SDK for Java 2.x API Reference.

JavaScript

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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;
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK per JavaScript.
- Per i dettagli sull'API, ListTablesconsulta AWS SDK per JavaScript API Reference.

SDK per JavaScript (v2)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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);
  }
});
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK per JavaScript.
- Per i dettagli sull'API, ListTablesconsulta AWS SDK per JavaScript API Reference.

Kotlin

SDK per Kotlin

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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")
        }
    }
}
```

- Per i dettagli sull'API, ListTablesconsulta AWS SDK for Kotlin API reference.

PHP

SDK per PHP

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
public function listTables($exclusiveStartTableName = "", $limit = 100)
{
    $this->dynamoDbClient->listTables([
        'ExclusiveStartTableName' => $exclusiveStartTableName,
        'Limit' => $limit,
    ]);
}
```

- Per i dettagli sull'API, ListTablesconsulta AWS SDK per PHP API Reference.

PowerShell

Strumenti per PowerShell V4

Esempio 1: restituisce i dettagli di tutte le tabelle, eseguendo un'iterazione automatica finché il servizio non indica che non esistono altre tabelle.

```
Get-DDBTableList
```

- Per i dettagli sull'API, vedere ListTablesin AWS Strumenti per PowerShell Cmdlet Reference (V4).

Strumenti per V5 PowerShell

Esempio 1: restituisce i dettagli di tutte le tabelle, eseguendo un'iterazione automatica finché il servizio non indica che non esistono altre tabelle.

```
Get-DDBTableList
```

- Per i dettagli sull'API, vedere ListTablesin AWS Strumenti per PowerShell Cmdlet Reference (V5).

Python

SDK per Python (Boto3)

> **Note**
>
> C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
```

- Per i dettagli sull'API, consulta ListTables AWSSDK for Python (Boto3) API Reference.

Ruby

SDK per Ruby

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Determina se esiste una tabella.

```
# 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
```

- Per i dettagli sull'API, ListTablesconsulta AWS SDK per Ruby API Reference.

Rust

SDK per Rust

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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)
}
```

Determina se esiste una tabella.

```
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()),
    }
}
```

- Per i dettagli sulle API, consulta il riferimento ListTablesall'API AWS SDK for Rust.

SAP ABAP

SDK per SAP ABAP

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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.
```

- Per i dettagli sulle API, ListTablesconsulta AWS SDK for SAP ABAP API reference.

Swift

SDK per Swift

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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)
    }
```

- Per i dettagli sull'API, consulta la ListTablesguida di riferimento all'API AWS SDK for Swift.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Utilizzo di DynamoDB con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo **`PutItem`** con un AWS SDK o una CLI

Gli esempi di codice seguenti mostrano come utilizzare `PutItem`.

Gli esempi di operazioni sono estratti di codice da programmi più grandi e devono essere eseguiti nel contesto. È possibile visualizzare questa operazione nel contesto nei seguenti esempi di codice:

- Informazioni di base
- Accelerazione delle letture con DAX
- Scenari avanzati basati sugli indici secondari globali
- Creazione di un elemento con un TTL
- Creazione e gestione di tabelle globali che descrivono la MREC
- Creazione e gestione di tabelle globali MRSC
- Utilizzo di operazioni condizionali
- Utilizzo di tabelle globali e replica multi-Regione con coerenza possibile (MREC)
- Lavora con Streams e Time-to-Live

.NET

SDK per .NET (v4)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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;
        }
    }
```

- Per i dettagli sull'API, PutItemconsulta AWS SDK per .NET API Reference.

Bash

AWS CLI con lo script Bash

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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

}
```

Le funzioni di utilità utilizzate in questo esempio.

```
########################################################################################
# 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
}
```

- Per i dettagli sull'API, consulta PutItem AWS CLICommand Reference.

C++

SDK per C++

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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);
}
```

Codice che attende che la tabella diventi attiva.

```
//! 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;
}
```

- Per i dettagli sull'API, PutItemconsulta AWS SDK per C++ API Reference.

CLI

AWS CLI

Esempio 1: per aggiungere un elemento a una tabella

L'`put-item`esempio seguente aggiunge un nuovo elemento alla MusicCollectiontabella.

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

Contenuto di `item.json`:

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

Output:

```
{
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 1.0
```

```
    },
    "ItemCollectionMetrics": {
        "ItemCollectionKey": {
            "Artist": {
                "S": "No One You Know"
            }
        },
        "SizeEstimateRangeGB": [
            0.0,
            1.0
        ]
    }
}
```

Per ulteriori informazioni, consulta Scrittura di un elemento nella Guida per gli sviluppatori di Amazon DynamoDB.

Esempio 2: per sovrascrivere in base a una condizione un elemento in una tabella

Nell'esempio seguente `put-item` sovrascrive un elemento esistente nella tabella `MusicCollection` solo se tale elemento include un attributo `AlbumTitle` con il valore `Greatest Hits`. Il comando restituisce il valore precedente dell'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
```

Contenuto di `item.json`:

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

Contenuto di `names.json`:

```
{
```

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

Contenuto di `values.json`:

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

Output:

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

Se la chiave esiste già, verrà visualizzato l'output seguente:

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

Per ulteriori informazioni, consulta Scrittura di un elemento nella Guida per gli sviluppatori di Amazon DynamoDB.

- Per i dettagli sull'API, vedere PutItemin AWS CLI Command Reference.

Go

SDK per Go V2

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
}
```

Per definire una struttura di filmato utilizzata in questo esempio.

```
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"])
}
```

- Per i dettagli sull'API, PutItemconsulta AWS SDK per Go API Reference.

Java

SDK per Java 2.x

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Mette un elemento in una tabella utilizzando 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);
        }
    }
}
```

- Per i dettagli sull'API, vedi PutItemin AWS SDK for Java 2.x API Reference.

JavaScript

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Questo esempio utilizza il client di documenti per semplificare il lavoro con gli elementi in DynamoDB. Per i dettagli sull'API, consulta 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;
};
```

- Per i dettagli sulle API, consulta la PutItemsezione AWS SDK per JavaScript API Reference.

SDK per JavaScript (v2)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Inserisci un elemento in una tabella.

```
// 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);
  }
});
```

Inserisci un elemento in una tabella utilizzando il client documento 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);
  }
});
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK per JavaScript.
- Per i dettagli sull'API, PutItemconsulta AWS SDK per JavaScript API Reference.

Kotlin

SDK per Kotlin

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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.")
    }
}
```

- Per i dettagli sull'API, PutItemconsulta AWS SDK for Kotlin API reference.

PHP

SDK per PHP

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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);
    }
```

- Per i dettagli sull'API, PutItemconsulta AWS SDK per PHP API Reference.

PowerShell

Strumenti per PowerShell V4

Esempio 1: crea un nuovo elemento o sostituisce un elemento esistente con uno nuovo.

```
$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
```

- Per i dettagli sull'API, vedere PutItemin AWS Strumenti per PowerShell Cmdlet Reference (V4).

Strumenti per V5 PowerShell

Esempio 1: crea un nuovo elemento o sostituisce un elemento esistente con uno nuovo.

```
$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
```

- Per i dettagli sull'API, vedere PutItemin AWS Strumenti per PowerShell Cmdlet Reference (V5).

Python

SDK per Python (Boto3)

> **Note**
>
> C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
```

- Per i dettagli sull'API, consulta PutItem AWSSDK for Python (Boto3) API Reference.

Ruby

SDK per Ruby

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
```

- Per i dettagli sull'API, PutItemconsulta AWS SDK per Ruby API Reference.

Rust

SDK per Rust

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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,
    })
}
```

- Per i dettagli sulle API, consulta la PutItemguida di riferimento all'API AWS SDK for Rust.

SAP ABAP

SDK per SAP ABAP

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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.
```

- Per i dettagli sulle API, PutItemconsulta AWS SDK for SAP ABAP API reference.

Swift

SDK per Swift

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
    }
```

- Per i dettagli sull'API, consulta la PutItemguida di riferimento all'API AWS SDK for Swift.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Utilizzo di DynamoDB con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo **`Query`** con un AWS SDK o una CLI

Gli esempi di codice seguenti mostrano come utilizzare `Query`.

Gli esempi di operazioni sono estratti di codice da programmi più grandi e devono essere eseguiti nel contesto. È possibile visualizzare questa operazione nel contesto nei seguenti esempi di codice:

- Informazioni di base
- Accelerazione delle letture con DAX
- Scenari avanzati basati sugli indici secondari globali
- Confronto di più valori con un unico attributo
- Gestione degli indici secondari globali
- Esecuzione di operazioni query avanzate
- Esecuzione di query su una tabella utilizzando una condizione begins_with

- Esecuzione di query su una tabella utilizzando un intervallo di date
- Esecuzione di query su una tabella utilizzando un indice secondario globale
- Esecuzione di query su una tabella con un'espressione di filtro complessa
- Esecuzione di query su una tabella con un'espressione di filtro dinamica
- Esecuzione di query su una tabella con un'espressione di filtro e un limite
- Esecuzione di query su una tabella con attributi annidati
- Esecuzione di query su una tabella con impaginazione
- Esecuzione di query su una tabella con operazioni a elevata consistenza di lettura
- Esecuzione di query per elementi TTL
- Esecuzione di query utilizzando modelli di data e ora
- Utilizzo dei nomi degli attributi di espressione
- Utilizzo di indici secondari locali

.NET

SDK per .NET (v4)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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;
        }
    }
```

- Per ulteriori informazioni sulle API, consulta Query nella Documentazione di riferimento delle API AWS SDK per .NET .

Bash

AWS CLI con lo script Bash

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
}
```

Le funzioni di utilità utilizzate in questo esempio.

```
###############################################################################
# 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
}
```

- Per informazioni dettagliate sulle API, consulta Query nella Documentazione di riferimento di AWS CLI .

C++

SDK per C++

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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;
}
```

- Per ulteriori informazioni sulle API, consulta Query nella Documentazione di riferimento delle API AWS SDK per C++ .

CLI

AWS CLI

Esempio 1: per eseguire una query su una tabella

Nell'esempio seguente `query` esegue query sulle voce della tabella `MusicCollection`. La tabella ha una chiave hash-and-range primaria (`ArtisteSongTitle`), ma questa query specifica solo il valore della chiave hash. Restituisce i titoli dei brani dell'artista "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
```

Contenuto di `expression-attributes.json`:

```
{
    ":v1": {"S": "No One You Know"}
}
```

Output:

```
{
    "Items": [
        {
            "SongTitle": {
                "S": "Call Me Today"
            },
            "SongTitle": {
                "S": "Scared of My Shadow"
            }
        }
    ],
    "Count": 2,
```

```
    "ScannedCount": 2,
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 0.5
    }
}
```

Per ulteriori informazioni, consulta Utilizzo delle query in DynamoDB nella Guida per gli sviluppatori di Amazon DynamoDB.

Esempio 2: per eseguire una query su una tabella utilizzando operazioni a elevata consistenza di lettura e attraversare l'indice in ordine decrescente

Nell'esempio seguente viene eseguita la stessa query del primo esempio, ma vengono restituiti i risultati in ordine inverso e vengono utilizzate operazioni a elevata consistenza di lettura.

```
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
```

Contenuto di `expression-attributes.json`:

```
{
    ":v1": {"S": "No One You Know"}
}
```

Output:

```
{
    "Items": [
        {
            "SongTitle": {
                "S": "Scared of My Shadow"
            }
        },
```

```
        {
            "SongTitle": {
                "S": "Call Me Today"
            }
        }
    ],
    "Count": 2,
    "ScannedCount": 2,
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 1.0
    }
}
```

Per ulteriori informazioni, consulta Utilizzo delle query in DynamoDB nella Guida per gli sviluppatori di Amazon DynamoDB.

Esempio 3: per filtrare risultati specifici

Nell'esempio seguente viene eseguita una query su `MusicCollection` ma vengono esclusi i risultati con valori specifici nell'attributo `AlbumTitle`. Si noti che ciò non influisce su `ScannedCount` o `ConsumedCapacity`, poiché il filtro viene applicato dopo la lettura degli elementi.

```
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
```

Contenuto di `values.json`:

```
{
    ":v1": {"S": "No One You Know"},
    ":v2": {"S": "Blue Sky Blues"},
    ":v3": {"S": "Greatest Hits"}
}
```

Contenuto di `names.json`:

```
{
    "#n1": "Artist",
    "#n2": "AlbumTitle"
}
```

Output:

```
{
    "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
    }
}
```

Per ulteriori informazioni, consulta Utilizzo delle query in DynamoDB nella Guida per gli sviluppatori di Amazon DynamoDB.

Esempio 4: per recuperare solo il conteggio degli elementi

Nell'esempio seguente viene recuperato il conteggio degli elementi corrispondenti alla query, ma non viene recuperato nessuno degli elementi stessi.

```
aws dynamodb query \
    --table-name MusicCollection \
    --select COUNT \
    --key-condition-expression "Artist = :v1" \
    --expression-attribute-values file://expression-attributes.json
```

Contenuto di `expression-attributes.json`:

```
{
    ":v1": {"S": "No One You Know"}
}
```

Output:

```
{
    "Count": 2,
    "ScannedCount": 2,
    "ConsumedCapacity": null
}
```

Per ulteriori informazioni, consulta Utilizzo delle query in DynamoDB nella Guida per gli sviluppatori di Amazon DynamoDB.

Esempio 5: per eseguire una query su un indice

Nell'esempio seguente viene eseguita una query su un indice locale secondario `AlbumTitleIndex`. La query restituisce tutti gli attributi della tabella di base proiettati nell'indice secondario locale. Si noti che durante l'esecuzione di query su un indice secondario locale o su un indice secondario globale è necessario fornire anche il nome della tabella di base mediante il parametro `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
```

Contenuto di `expression-attributes.json`:

```
{
    ":v1": {"S": "No One You Know"}
}
```

Output:

```
{
    "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
            }
        }
    }
}
```


Per ulteriori informazioni, consulta Utilizzo delle query in DynamoDB nella Guida per gli sviluppatori di Amazon DynamoDB.

- Per informazioni dettagliate sulle API, consulta Query nella Documentazione di riferimento di AWS CLI .

Go

SDK per Go V2

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
}
```

Per definire una struttura di filmato utilizzata in questo esempio.

```
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"])
}
```

- Per ulteriori informazioni sulle API, consulta Query nella Documentazione di riferimento delle API AWS SDK per Go .

Java

SDK per Java 2.x

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Interroga una tabella utilizzando 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;
    }
}
```

Esegue query su una tabella utilizzando `DynamoDbClient` e un indice secondario.

```
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);
        }
    }
}
```

- Per ulteriori informazioni sulle API, consulta Query nella Documentazione di riferimento delle API AWS SDK for Java 2.x .

JavaScript

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Questo esempio utilizza il client di documenti per semplificare il lavoro con gli elementi in DynamoDB. Per i dettagli sull'API, consulta 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;
};
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK per JavaScript.
- Per ulteriori informazioni sulle API, consulta Query nella Documentazione di riferimento delle API AWS SDK per JavaScript .

SDK per JavaScript (v2)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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);
  }
});
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK per JavaScript.
- Per ulteriori informazioni sulle API, consulta Query nella Documentazione di riferimento delle API AWS SDK per JavaScript .

Kotlin

SDK per Kotlin

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
    }
}
```

- Per informazioni dettagliate sulle API, consulta Query nella Documentazione di riferimento per le API di SDK AWS per Kotlin.

PHP

SDK per PHP

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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);
    }
```

- Per ulteriori informazioni sulle API, consulta Query nella Documentazione di riferimento delle API AWS SDK per PHP .

PowerShell

Strumenti per PowerShell V4

Esempio 1: richiama una query che restituisce elementi DynamoDB con l'elemento Artist specificato. SongTitle

```
$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
```

Output:

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

- Per informazioni dettagliate sull'API, consulta Query nella documentazione di riferimento dei cmdlet di AWS Strumenti per PowerShell (V4).

Strumenti per V5 PowerShell

Esempio 1: richiama una query che restituisce elementi DynamoDB con l'elemento Artist specificato. SongTitle

```
$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
```

Output:

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

- Per informazioni dettagliate sull'API, consulta Query nella documentazione di riferimento dei cmdlet di AWS Strumenti per PowerShell (V5).

Python

SDK per Python (Boto3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Esegui la query degli elementi utilizzando un'espressione di condizione chiave.

```
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"]
```

Esegui la query degli elementi e proiettali per restituire un sottoinsieme di dati.

```
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"]
```

- Per informazioni dettagliate sulle API, consulta Query nella Documentazione di riferimento per l'API SDK for Python (Boto3)AWS .

Ruby

SDK per Ruby

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
```

- Per ulteriori informazioni sulle API, consulta Query nella Documentazione di riferimento delle API AWS SDK per Ruby .

Rust

SDK per Rust

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Trova i filmati realizzati nell'anno specificato.

```
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![])
    }
}
```

- Per ulteriori informazioni sulle API, consulta Query nella Documentazione di riferimento delle API AWS .

SAP ABAP

SDK per SAP ABAP

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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.
```

- Per informazioni dettagliate sull'API, consulta Query nella Documentazione di riferimento dell'API dell'AWS SDK per SAP ABAP.

Swift

SDK per Swift

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
        }
    }
```

- Per informazioni dettagliate sulle API, consulta Query nella Documentazione di riferimento per le API di SDK AWS per Swift.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUtilizzo di DynamoDB con un SDK AWS. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo **`Scan`** con un AWS SDK o una CLI

Gli esempi di codice seguenti mostrano come utilizzare `Scan`.

Gli esempi di operazioni sono estratti di codice da programmi più grandi e devono essere eseguiti nel contesto. È possibile visualizzare questa operazione nel contesto nei seguenti esempi di codice:

- Informazioni di base
- Accelerazione delle letture con DAX
- Confronto di più valori con un unico attributo

.NET

SDK per .NET (v4)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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;
        }
    }
```

- Per informazioni dettagliate sulle API, consulta Scan nella Documentazione di riferimento per le API AWS SDK per .NET .

Bash

AWS CLI con lo script Bash

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
}
```

Le funzioni di utilità utilizzate in questo esempio.

```
###############################################################################
```

```
# 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
}
```

- Per informazioni dettagliate sulle API, consulta Scansione nella Documentazione di riferimento di AWS CLI .

C++

SDK per C++

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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;
}
```

- Per informazioni dettagliate sulle API, consulta Scan nella Documentazione di riferimento per le API AWS SDK per C++ .

CLI

AWS CLI

Come eseguire la scansione di una tabella

L'esempio seguente `scan` analizza l'intera tabella `MusicCollection`, quindi restringe i risultati alle canzoni dell'artista "No One You Know". Per ogni elemento, vengono restituiti solo il titolo dell'album e il titolo del brano.

```
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
```

Contenuto di `expression-attribute-names.json`:

```
{
    "#ST": "SongTitle",
    "#AT":"AlbumTitle"
}
```

Contenuto di `expression-attribute-values.json`:

```
{
    ":a": {"S": "No One You Know"}
}
```

Output:

```
{
    "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
}
```

Per ulteriori informazioni, consulta Utilizzo delle scansioni in DynamoDB nella Guida per gli sviluppatori di Amazon DynamoDB.

- Per informazioni dettagliate sulle API, consulta Scansione nella Documentazione di riferimento di AWS CLI .

Go

SDK per Go V2

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
}
```

Per definire una struttura di filmato utilizzata in questo esempio.

```
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"])
}
```

- Per informazioni dettagliate sulle API, consulta Scan nella Documentazione di riferimento per le API AWS SDK per Go .

Java

SDK per Java 2.x

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Esegue la scansione di una tabella Amazon DynamoDB utilizzando 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);
        }
    }
}
```

- Per informazioni dettagliate sulle API, consulta Scan nella Documentazione di riferimento per le API AWS SDK for Java 2.x .

JavaScript

SDK per (v3) JavaScript

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Questo esempio utilizza il client di documenti per semplificare il lavoro con gli elementi in DynamoDB. Per i dettagli sull'API, consulta 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;
};
```

- Per informazioni dettagliate sulle API, consulta Scan nella Documentazione di riferimento per le API AWS SDK per JavaScript .

SDK per JavaScript (v2)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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 + ")"
      );
    });
```

```
  }
});
```

- Per ulteriori informazioni, consulta la Guida per sviluppatori di AWS SDK per JavaScript.
- Per informazioni dettagliate sulle API, consulta Scan nella Documentazione di riferimento per le API AWS SDK per JavaScript .

Kotlin

SDK per Kotlin

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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]}")
            }
        }
    }
}
```

- Per informazioni dettagliate sulle API, consulta Scan nella Documentazione di riferimento per le API di SDK AWS per Kotlin.

PHP

SDK per PHP

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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);
}
```

- Per informazioni dettagliate sulle API, consulta Scan nella Documentazione di riferimento per le API AWS SDK per PHP .

PowerShell

Strumenti per PowerShell V4

Esempio 1: restituisce tutti gli elementi nella tabella Music.

```
Invoke-DDBScan -TableName 'Music' | ConvertFrom-DDBItem
```

Output:

```
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
```

Esempio 2: restituisce gli elementi nella tabella Music con un CriticRating valore maggiore o uguale a nove.

```
$scanFilter = @{
        CriticRating = [Amazon.DynamoDBv2.Model.Condition]@{
            AttributeValueList = @(@{N = '9'})
            ComparisonOperator = 'GE'
        }
    }
    Invoke-DDBScan -TableName 'Music' -ScanFilter $scanFilter | ConvertFrom-
DDBItem
```

Output:

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

- Per informazioni dettagliate sull'API, consulta Scan nella documentazione di riferimento dei cmdlet di AWS Strumenti per PowerShell (V4).

Strumenti per PowerShell V5

Esempio 1: restituisce tutti gli elementi nella tabella Music.

```
Invoke-DDBScan -TableName 'Music' | ConvertFrom-DDBItem
```

Output:

```
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
```

Esempio 2: restituisce gli elementi nella tabella Music con un CriticRating valore maggiore o uguale a nove.

```
$scanFilter = @{
        CriticRating = [Amazon.DynamoDBv2.Model.Condition]@{
            AttributeValueList = @(@{N = '9'})
            ComparisonOperator = 'GE'
```

```
        }
    }
    Invoke-DDBScan -TableName 'Music' -ScanFilter $scanFilter | ConvertFrom-
DDBItem
```

Output:

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

- Per informazioni dettagliate sull'API, consulta Scan nella documentazione di riferimento dei cmdlet di AWS Strumenti per PowerShell (V5).

Python

SDK per Python (Boto3)

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
```

- Per informazioni dettagliate sulle API, consulta Scan nella Documentazione di riferimento delle API SDK AWS per Python (Boto3).

Ruby

SDK per Ruby

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
```

- Per informazioni dettagliate sulle API, consulta Scan nella Documentazione di riferimento per le API AWS SDK per Ruby .

Rust

SDK per Rust

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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(())
}
```

- Per informazioni dettagliate sulle API, consulta Scan nella Documentazione di riferimento degli SDK AWS per l'API Rust.

SAP ABAP

SDK per SAP ABAP

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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.
```

- Per informazioni dettagliate sull'API, consulta Scan nella Documentazione di riferimento dell'API dell'AWS SDK per SAP ABAP.

Swift

SDK per Swift

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
        }
    }
```

- Per informazioni dettagliate sulle API, consulta Scan nella Documentazione di riferimento delle API SDK AWS per Swift.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUtilizzo di DynamoDB con un SDK AWS. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo **`UpdateItem`** con un AWS SDK o una CLI

Gli esempi di codice seguenti mostrano come utilizzare `UpdateItem`.

Gli esempi di operazioni sono estratti di codice da programmi più grandi e devono essere eseguiti nel contesto. È possibile visualizzare questa operazione nel contesto nei seguenti esempi di codice:

- Informazioni di base
- Aggiornamento condizionale del TTL di un elemento
- Conteggio degli operatori di espressione
- Creazione e gestione di tabelle globali MRSC
- Esecuzione di operazioni di tipo elenco
- Esecuzione di operazioni di mappatura

- Esecuzione di operazioni su set
- Funzionamento dell'ordine di aggiornamento delle espressioni
- Aggiornamento del TTL di un elemento
- Utilizzo di operazioni di contatori atomici
- Utilizzo di operazioni condizionali
- Utilizzo dei nomi degli attributi di espressione

.NET

SDK per .NET (v4)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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;
```

```
        }
    }
```

- Per i dettagli sull'API, UpdateItemconsulta AWS SDK per .NET API Reference.

Bash

AWS CLI con lo script Bash

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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

}
```

Le funzioni di utilità utilizzate in questo esempio.

```
###############################################################################
# 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
}
```

- Per i dettagli sull'API, consulta UpdateItem AWS CLICommand Reference.

C++

SDK per C++

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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);
}
```

Codice che attende che la tabella diventi attiva.

```
//! 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;
}
```

- Per i dettagli sull'API, UpdateItemconsulta AWS SDK per C++ API Reference.

CLI

AWS CLI

Esempio 1: per aggiornare un elemento in una tabella

L'esempio `update-item` seguente aggiorna un elemento nella tabella `MusicCollection`. Aggiunge un nuovo attributo (`Year`) e modifica l'attributo `AlbumTitle`. Tutti gli attributi dell'elemento come appaiono dopo l'aggiornamento vengono restituiti nella risposta.

```
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
```

Contenuto di `key.json`:

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

Contenuto di `expression-attribute-names.json`:

```
{
    "#Y":"Year", "#AT":"AlbumTitle"
```

```
}
```

Contenuto di `expression-attribute-values.json`:

```
{
    ":y":{"N": "2015"},
    ":t":{"S": "Louder Than Ever"}
}
```

Output:

```
{
    "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
        ]
```

```
    }
}
```

Per ulteriori informazioni, consulta Scrittura di un elemento nella Guida per gli sviluppatori di Amazon DynamoDB.

Esempio 2: per aggiornare un elemento in base a una condizione

Nell'esempio seguente viene aggiornato un elemento nella tabella `MusicCollection`, ma solo se l'elemento esistente non dispone già di un attributo `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)"
```

Contenuto di `key.json`:

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

Contenuto di `expression-attribute-names.json`:

```
{
    "#Y":"Year",
    "#AT":"AlbumTitle"
}
```

Contenuto di `expression-attribute-values.json`:

```
{
    ":y":{"N": "2015"},
    ":t":{"S": "Louder Than Ever"}
}
```

Se l'elemento include già un attributo `Year`, DynamoDB restituisce il seguente output.

```
An error occurred (ConditionalCheckFailedException) when calling the UpdateItem
 operation: The conditional request failed
```

Per ulteriori informazioni, consulta Scrittura di un elemento nella Guida per gli sviluppatori di Amazon DynamoDB.

- Per i dettagli sull'API, consulta UpdateItem AWS CLICommand Reference.

Go

SDK per Go V2

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
}
```

Per definire una struttura di filmato utilizzata in questo esempio.

```
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"])
}
```

- Per i dettagli sull'API, UpdateItemconsulta AWS SDK per Go API Reference.

Java

SDK per Java 2.x

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Aggiorna un elemento in una tabella utilizzando 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!");
    }
}
```

- Per i dettagli sull'API, vedi UpdateItemin AWS SDK for Java 2.x API Reference.

JavaScript

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Questo esempio utilizza il client di documenti per semplificare il lavoro con gli elementi in DynamoDB. Per i dettagli sull'API, consulta 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;
};
```

- Per i dettagli sulle API, consulta la UpdateItemsezione AWS SDK per JavaScript API Reference.

Kotlin

SDK per Kotlin

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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")
    }
}
```

- Per i dettagli sull'API, UpdateItemconsulta AWS SDK for Kotlin API reference.

PHP

SDK per PHP

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
                ]
            ],
        ]);
    }
```

- Per i dettagli sull'API, UpdateItemconsulta AWS SDK per PHP API Reference.

PowerShell

Strumenti per PowerShell V4

Esempio 1: imposta l'attributo genre su 'Rap' sull'elemento DynamoDB con la chiave di partizione e la SongTitle chiave di ordinamento 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
```

Output:

```
Name                           Value
----                           -----
Genre                          Rap
```

- Per i dettagli sull'API, vedere UpdateItemin AWS Strumenti per PowerShell Cmdlet Reference (V4).

Strumenti per V5 PowerShell

Esempio 1: imposta l'attributo genre su 'Rap' sull'elemento DynamoDB con la chiave di partizione e la SongTitle chiave di ordinamento 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
```

Output:

```
Name                           Value
----                           -----
Genre                          Rap
```

- Per i dettagli sull'API, vedere UpdateItemin AWS Strumenti per PowerShell Cmdlet Reference (V5).

Python

SDK per Python (Boto3)

> **Note**
>
> C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Aggiorna un elemento utilizzando un'espressione di aggiornamento.

```
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"]
```

Aggiorna un elemento utilizzando un'espressione di aggiornamento che include un'operazione aritmetica.

```
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"]
```

Aggiorna un elemento solo quando soddisfa determinate condizioni.

```
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"]
```

- Per i dettagli sull'API, consulta UpdateItem AWSSDK for Python (Boto3) API Reference.

Ruby

SDK per Ruby

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
```

- Per i dettagli sull'API, UpdateItemconsulta AWS SDK per Ruby API Reference.

SAP ABAP

SDK per SAP ABAP

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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.
```

- Per i dettagli sulle API, UpdateItemconsulta AWS SDK for SAP ABAP API reference.

Swift

SDK per Swift

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
        }
    }
```

- Per i dettagli sull'API, consulta la UpdateItemguida di riferimento all'API AWS SDK for Swift.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Utilizzo di DynamoDB con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo **`UpdateTable`** con un AWS SDK o una CLI

Gli esempi di codice seguenti mostrano come utilizzare `UpdateTable`.

Gli esempi di operazioni sono estratti di codice da programmi più grandi e devono essere eseguiti nel contesto. È possibile visualizzare questa operazione nel contesto nei seguenti esempi di codice:

- Creazione e gestione di tabelle globali che descrivono la MREC
- Creazione e gestione di tabelle globali MRSC
- Gestione degli indici secondari globali
- Aggiornamento dell'impostazione di throughput a caldo di una tabella
- Utilizzo di tabelle globali e replica multi-Regione con coerenza possibile (MREC)
- Utilizzo della crittografia delle tabelle

C++

SDK per C++

> **Note**
>
> C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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);
}
```

Codice che attende che la tabella diventi attiva.

```
//! 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;
}
```

- Per i dettagli sull'API, UpdateTableconsulta AWS SDK per C++ API Reference.

CLI

AWS CLI

Esempio 1: modificare la modalità di fatturazione di una tabella

Nell'esempio seguente `update-table` aumenta la capacità di lettura e scrittura allocata per la tabella `MusicCollection`.

```
aws dynamodb update-table \
    --table-name MusicCollection \
    --billing-mode PROVISIONED \
    --provisioned-throughput ReadCapacityUnits=15,WriteCapacityUnits=10
```

Output:

```
{
    "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"
        }
    }
}
```

Per ulteriori informazioni, consulta Aggiornamento di una tabella nella Guida per gli sviluppatori di Amazon DynamoDB.

Esempio 2: per creare un indice secondario globale

Nell'esempio seguente viene aggiunto un indice secondario globale alla tabella `MusicCollection`.

```
aws dynamodb update-table \
    --table-name MusicCollection \
    --attribute-definitions AttributeName=AlbumTitle,AttributeType=S \
    --global-secondary-index-updates file://gsi-updates.json
```

Contenuto di `gsi-updates.json`:

```
[
    {
        "Create": {
```

```
            "IndexName": "AlbumTitle-index",
            "KeySchema": [
                {
                    "AttributeName": "AlbumTitle",
                    "KeyType": "HASH"
                }
            ],
            "ProvisionedThroughput": {
                "ReadCapacityUnits": 10,
                "WriteCapacityUnits": 10
            },
            "Projection": {
                "ProjectionType": "ALL"
            }
        }
    }
]
```

Output:

```
{
    "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"
            }
        ]
    }
}
```

Per ulteriori informazioni, consulta Aggiornamento di una tabella nella Guida per gli sviluppatori di Amazon DynamoDB.

Esempio 3: per abilitare i flussi DynamoDB su una tabella

Il seguente comando abilita i flussi DynamoDB sulla tabella `MusicCollection`.

```
aws dynamodb update-table \
    --table-name MusicCollection \
    --stream-specification StreamEnabled=true,StreamViewType=NEW_IMAGE
```

Output:

```
{
    "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"
    }
}
```

Per ulteriori informazioni, consulta Aggiornamento di una tabella nella Guida per gli sviluppatori di Amazon DynamoDB.

Esempio 4: per abilitare la crittografia lato server

Nell'esempio seguente viene abilitata la crittografia lato server sulla tabella `MusicCollection`.

```
aws dynamodb update-table \
    --table-name MusicCollection \
    --sse-specification Enabled=true,SSEType=KMS
```

Output:

```
{
    "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"
        }
    }
}
```

Per ulteriori informazioni, consulta Aggiornamento di una tabella nella Guida per gli sviluppatori di Amazon DynamoDB.

- Per i dettagli sull'API, consulta UpdateTable AWS CLICommand Reference.

PowerShell

Strumenti per PowerShell V4

Esempio 1: aggiorna il throughput allocato per la tabella specificata.

```
Update-DDBTable -TableName "myTable" -ReadCapacity 10 -WriteCapacity 5
```

- Per i dettagli sull'API, vedere UpdateTablein AWS Strumenti per PowerShell Cmdlet Reference (V4).

Strumenti per V5 PowerShell

Esempio 1: aggiorna il throughput allocato per la tabella specificata.

```
Update-DDBTable -TableName "myTable" -ReadCapacity 10 -WriteCapacity 5
```

- Per i dettagli sull'API, vedere UpdateTablein AWS Strumenti per PowerShell Cmdlet Reference (V5).

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Utilizzo di DynamoDB con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo **`UpdateTimeToLive`** con un AWS SDK o una CLI

Gli esempi di codice seguenti mostrano come utilizzare `UpdateTimeToLive`.

Gli esempi di operazioni sono estratti di codice da programmi più grandi e devono essere eseguiti nel contesto. È possibile visualizzare questa operazione nel contesto nel seguente esempio di codice:

- Lavora con Streams e Time-to-Live

CLI

AWS CLI

Per aggiornare le impostazioni del Time to Live in una tabella

Nell'esempio seguente `update-time-to-live` abilita le impostazioni del Time to Live nella tabella specificata.

```
aws dynamodb update-time-to-live \
    --table-name MusicCollection \
    --time-to-live-specification Enabled=true,AttributeName=ttl
```

Output:

```
{
```

```
    "TimeToLiveSpecification": {
        "Enabled": true,
        "AttributeName": "ttl"
    }
}
```

Per ulteriori informazioni, consulta Time to Live nella Guida per gli sviluppatori di Amazon DynamoDB.

- Per i dettagli sull'API, consulta UpdateTimeToLive AWS CLICommand Reference.

Java

SDK per Java 2.x

Per abilitare il TTL in una tabella DynamoDB esistente 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;
        }
    }
```

Per disabilitare il TTL in una tabella DynamoDB esistente 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;
        }
    }
```

- Per i dettagli sull'API, UpdateTimeToLiveconsulta AWS SDK for Java 2.x API Reference.

JavaScript

SDK per JavaScript (v3)

Per abilitare il TTL in una tabella DynamoDB esistente.

```
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');
```

Per disabilitare il TTL in una tabella DynamoDB esistente.

```
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');
```

- Per i dettagli sull'API, consulta la sezione AWS SDK per JavaScript API [UpdateTimeToLive](#)Reference.

Python

SDK per Python (Boto3)

Per abilitare il TTL in una tabella DynamoDB esistente.

```
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")
```

Per disabilitare il TTL in una tabella DynamoDB esistente.

```
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")
```

- Per i dettagli sull'API, consulta UpdateTimeToLive AWSSDK for Python (Boto3) API Reference.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Utilizzo di DynamoDB con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Scenari per l'utilizzo di DynamoDB AWS SDKs

I seguenti esempi di codice mostrano come implementare scenari comuni in DynamoDB con. AWS SDKs Questi scenari illustrano come eseguire attività specifiche chiamando più funzioni all'interno di DynamoDB o in combinazione con altri Servizi AWS. Ogni scenario include un collegamento al codice sorgente completo, dove è possibile trovare le istruzioni su come configurare ed eseguire il codice.

Gli scenari sono relativi a un livello intermedio di esperienza per aiutarti a comprendere le azioni di servizio nel contesto.

Esempi

- Accelera le letture DynamoDB con DAX utilizzando un SDK AWS
- Lavora con scenari avanzati di DynamoDB Global Secondary Index utilizzando v2 AWS Command Line Interface
- Costruisci un'applicazione per inviare dati a una tabella DynamoDB

- Confronta più valori con un singolo attributo in DynamoDB con un SDK AWS
- Aggiornamento condizionale di un elemento DynamoDB con un TTL utilizzando un SDK AWS
- Connect a un'istanza DynamoDB locale utilizzando un SDK AWS
- Conta gli operatori di espressione in DynamoDB con un SDK AWS
- Creazione di un'API REST di API Gateway per monitorare i dati COVID-19
- Creazione di un'applicazione di messaggistica con Step Functions
- Creazione di un'applicazione di gestione delle risorse fotografiche che consente agli utenti di gestire le foto utilizzando etichette
- Crea una tabella DynamoDB con un indice secondario globale utilizzando l'SDK AWS
- Crea una tabella DynamoDB con un'impostazione di throughput caldo utilizzando un SDK AWS
- Creazione di un'applicazione Web per tracciare i dati DynamoDB
- Creazione di un'applicazione di chat websocket con API Gateway
- Crea un elemento DynamoDB con un TTL utilizzando un SDK AWS
- Crea e gestisci tabelle globali DynamoDB con una forte coerenza multiregionale utilizzando un SDK AWS
- Crea e gestisci tabelle globali DynamoDB dimostrando MREC utilizzando un SDK AWS
- Eliminare i dati DynamoDB utilizzando le istruzioni PartiQL DELETE con un SDK AWS
- Rileva PPE nelle immagini con Amazon Rekognition utilizzando un SDK AWS
- Inserimento di dati DynamoDB utilizzando istruzioni PartiQL INSERT con un SDK AWS
- Richiamo a una funzione Lambda da un browser
- Gestisci gli indici secondari globali DynamoDB utilizzando v2 AWS Command Line Interface
- Gestisci le policy basate sulle risorse di DynamoDB utilizzando la versione 2 AWS Command Line Interface
- Monitora le prestazioni di Amazon DynamoDB utilizzando un SDK AWS
- Esegui operazioni di interrogazione avanzate su DynamoDB utilizzando un SDK AWS
- Esegui operazioni sugli elenchi in DynamoDB con un SDK AWS
- Esegui operazioni di mappatura in DynamoDB con un SDK AWS
- Esegui operazioni di set in DynamoDB con un SDK AWS
- Interroga una tabella DynamoDB utilizzando batch di istruzioni PartiQL e un SDK AWS
- Interroga una tabella DynamoDB utilizzando PartiQL e un SDK AWS

- Interroga una tabella DynamoDB utilizzando un indice secondario globale con un SDK AWS
- Interroga una tabella DynamoDB utilizzando una condizione begins_with con un SDK AWS
- Interroga una tabella DynamoDB utilizzando un intervallo di date nella chiave di ordinamento con un SDK AWS
- Interroga una tabella DynamoDB con un'espressione di filtro complessa con un SDK AWS
- Interroga una tabella DynamoDB con un'espressione di filtro dinamica con un SDK AWS
- Interroga una tabella DynamoDB con un'espressione di filtro e un limite con un SDK AWS
- Interroga una tabella DynamoDB con attributi annidati utilizzando un SDK AWS
- Interroga una tabella DynamoDB con impaginazione utilizzando un SDK AWS
- Interroga una tabella DynamoDB con letture fortemente coerenti utilizzando un SDK AWS
- Interroga i dati DynamoDB utilizzando istruzioni PartiQL SELECT con un SDK AWS
- Interroga una tabella DynamoDB per gli elementi TTL utilizzando un SDK AWS
- Interroga le tabelle DynamoDB utilizzando modelli di data e ora con un SDK AWS
- Salva EXIF e altre informazioni sull'immagine utilizzando un SDK AWS
- Configurare il controllo degli accessi basato sugli attributi per DynamoDB utilizzando v2 AWS Command Line Interface
- Comprendi l'ordine delle espressioni di aggiornamento in DynamoDB con un SDK AWS
- Aggiorna un'impostazione di tabella DynamoDB con un throughput caldo utilizzando un SDK AWS
- Aggiornare un elemento DynamoDB con un TTL utilizzando un SDK AWS
- Aggiorna i dati DynamoDB utilizzando le istruzioni PartiQL UPDATE con un SDK AWS
- Utilizzo di un'API Gateway per richiamare una funzione Lambda
- Utilizzo di Step Functions per richiamare le funzioni Lambda
- Usa un modello di documento per DynamoDB utilizzando un SDK AWS
- Utilizza un modello di persistenza degli oggetti di alto livello per DynamoDB utilizzando un SDK AWS
- Usa le operazioni di conteggio atomico in DynamoDB con un SDK AWS
- Usa operazioni condizionali in DynamoDB con un SDK AWS
- Usa i nomi degli attributi di espressione in DynamoDB con un SDK AWS
- Utilizzo degli eventi pianificati per richiamare una funzione Lambda
- Lavora con gli indici secondari locali di DynamoDB utilizzando v2 AWS Command Line Interface
- Lavora con DynamoDB Streams e usa v2 Time-to-Live AWS Command Line Interface

- Lavora con le tabelle globali di DynamoDB e la replica multiregione con eventuale coerenza (MREC) utilizzando v2 AWS Command Line Interface
- Lavora con il tagging delle risorse DynamoDB utilizzando la versione 2 AWS Command Line Interface
- Lavora con la crittografia delle tabelle DynamoDB utilizzando la versione 2 AWS Command Line Interface

Accelera le letture DynamoDB con DAX utilizzando un SDK AWS

L'esempio di codice seguente mostra come:

- Creazione e scrittura di dati su una tabella con client DAX e SDK.
- Ricezione, query e analisi della tabella con i client e confronto delle prestazioni.

Per ulteriori informazioni, consulta Sviluppo con il client DynamoDB Accelerator.

Python

SDK per Python (Boto3)

> **Note**
>
> C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Crea una tabella con il client 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.")
```

Scrivi i dati di test nella tabella.

```
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)
```

Ottieni elementi per una serie di iterazioni sia per il client DAX che per il client Boto3 e segnala il tempo impiegato per ciascuno.

```
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}."
    )
```

Esegui una query sulla tabella per una serie di iterazioni sia per il client DAX che per il client Boto3 e segnala il tempo impiegato per ciascuno.

```
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}."
    )
```

Scansiona la tabella per una serie di iterazioni sia per il client DAX che per il client Boto3 e segnala il tempo impiegato per ciascuno.

```
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}."
    )
```

Eliminare la tabella .

```
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!")
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento delle API SDK AWS per Python (Boto3).
  - CreateTable
  - DeleteTable
  - GetItem
  - PutItem
  - Query
  - Scan

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUtilizzo di DynamoDB con un SDK AWS. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Lavora con scenari avanzati di DynamoDB Global Secondary Index utilizzando v2 AWS Command Line Interface

L'esempio di codice seguente mostra come utilizzare le configurazioni degli indici secondari globali.

- Crea una tabella con più GSIs.
- Per creare una tabella con capacità on demand e GSI.
- Metti gli elementi in una tabella con più elementi GSIs.

- Interrogazione GSIs multipla con condizioni diverse.

Bash

AWS CLI con lo script Bash

Crea una tabella con più GSIs di una tabella.

```
# 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\"]}
            }
        ]"
```

Per creare una tabella con capacità on demand e 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\"}
            }
        ]"
```

Metti gli elementi in una tabella con più elementi GSIs.

```
# 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"}
```

```
    }'
```

Interroga gli elementi da una tabella con più elementi GSIs.

```
# 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"}}'
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento dei comandi AWS CLI .
  - CreateTable
  - PutItem
  - Query

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUtilizzo di DynamoDB con un SDK AWS. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Costruisci un'applicazione per inviare dati a una tabella DynamoDB

Gli esempi di codice riportati di seguito mostrano come costruire un'applicazione che invia dati a una tabella Amazon DynamoDB e che ti avvisa quando un utente aggiorna la tabella.

Java

SDK per Java 2.x

Mostra come creare un'applicazione Web dinamica che invia dati utilizzando l'API Java di Amazon DynamoDB e invia un messaggio di testo utilizzando l'API Java di Amazon Simple Notification Service.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, consulta l'esempio completo su GitHub.

Servizi utilizzati in questo esempio

- DynamoDB
- Amazon SNS

JavaScript

SDK per JavaScript (v3)

Questo esempio mostra come creare un'app che consenta agli utenti di inviare dati a una tabella Amazon DynamoDB e un messaggio di testo all'amministratore utilizzando Amazon Simple Notification Service (Amazon SNS).

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su. GitHub

Questo esempio è anche disponibile nella Guida per lo sviluppatore di AWS SDK per JavaScript v3 .

Servizi utilizzati in questo esempio

- DynamoDB
- Amazon SNS

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUtilizzo di DynamoDB con un SDK AWS. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Confronta più valori con un singolo attributo in DynamoDB con un SDK AWS

Gli esempi di codice seguenti mostrano come confrontare più valori con un unico attributo in DynamoDB.

- Utilizzate l'operatore IN per confrontare più valori con un singolo attributo.
- Confronta l'operatore IN con più condizioni OR.
- Comprensione dei vantaggi in termini di prestazioni e complessità di espressione derivanti dall'utilizzo dell'operatore IN.

Java

SDK per Java 2.x

Confronta più valori con un singolo attributo in DynamoDB utilizzando. 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;
    }
```

Esempio di utilizzo del confronto di più valori 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();
        }
    }
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento delle API AWS SDK for Java 2.x .
  - Query
  - Scan

JavaScript

SDK per JavaScript (v3)

Confronta più valori con un singolo attributo utilizzando. AWS SDK per 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
    };
  }
}
```

Esempio di utilizzo del confronto di più valori con AWS SDK per 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);
  }
}
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento delle API AWS SDK per JavaScript .
  - Query
  - Scan

Python

SDK per Python (Boto3)

Confronta più valori con un singolo attributo utilizzando 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)}
```

Esempio di utilizzo del confronto di più valori 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")
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento delle API SDK AWS per Python (Boto3).
  - Query
  - Scan

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUtilizzo di DynamoDB con un SDK AWS. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Aggiornamento condizionale di un elemento DynamoDB con un TTL utilizzando un SDK AWS

Gli esempi di codice seguenti mostrano come aggiornare in modo condizionale il valore TTL di un elemento.

Java

SDK per Java 2.x

Per aggiornare il TTL di un elemento DynamoDB esistente in una tabella con una condizione.

```
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;
        }
    }
}
```

- Per i dettagli sull'API, consulta [UpdateItem AWS SDK for Java 2.x](#)API Reference.

JavaScript

SDK per JavaScript (v3)

Per aggiornare il TTL di un elemento DynamoDB esistente in una tabella con una condizione.

```
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');
```

- Per i dettagli sull'API, consulta la sezione AWS SDK per JavaScript API UpdateItemReference.

Python

SDK per Python (Boto3)

Per aggiornare il TTL di un elemento DynamoDB esistente in una tabella con una condizione.

```
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",
)
```

- Per i dettagli sull'API, consulta UpdateItem AWSSDK for Python (Boto3) API Reference.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Utilizzo di DynamoDB con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Connect a un'istanza DynamoDB locale utilizzando un SDK AWS

Il seguente esempio di codice mostra come sovrascrivere l'URL di un endpoint per connettersi a una distribuzione di sviluppo locale di DynamoDB e un SDK. AWS

Per ulteriori informazioni, consulta DynamoDB locale.

Rust

SDK per Rust

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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:?}"),
    }
}
```

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUtilizzo di DynamoDB con un SDK AWS. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Conta gli operatori di espressione in DynamoDB con un SDK AWS

Gli esempi di codice seguenti mostrano come conteggiare gli operatori delle espressioni in DynamoDB.

- Comprensione del limite dell’operatore 300 di DynamoDB.

- Conteggio degli operatori nelle espressioni complesse.
- Ottimizzazione delle espressioni affinché rientrino nei limiti.

Java

SDK per Java 2.x

Per descrivere il conteggio degli operatori di espressione 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;
    }
```

Esempio di utilizzo dell'operatore di espressione Counting with. 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();
        }
    }
```

- Per i dettagli sull'API, consulta la sezione UpdateItem AWS SDK for Java 2.xAPI Reference.

JavaScript

SDK per JavaScript (v3)

Per descrivere il conteggio degli operatori di espressione con AWS SDK per 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;
}
```

Esempio di utilizzo dell'operatore di espressione Counting with. AWS SDK per 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);
  }
}
```

- Per i dettagli sull'API, consulta la sezione UpdateItem AWS SDK per JavaScriptAPI Reference.

Python

SDK per Python (Boto3)

Per descrivere il conteggio degli operatori di espressione 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
```

Esempio di utilizzo dell'operatore di espressione Counting with 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")
```

- Per i dettagli sull'API, consulta UpdateItem AWSSDK for Python (Boto3) API Reference.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Utilizzo di DynamoDB con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Creazione di un'API REST di API Gateway per monitorare i dati COVID-19

Il seguente esempio di codice mostra come creare una REST API che simula un sistema per monitorare i casi quotidiani di COVID-19 negli Stati Uniti, utilizzando dati fittizi.

Python

SDK per Python (Boto3)

Mostra come usare AWS Chalice con per AWS SDK for Python (Boto3) creare un'API REST serverless che utilizzi Amazon API Gateway e Amazon DynamoDB. AWS Lambda La REST API simula un sistema che monitora i casi giornalieri di COVID-19 negli Stati Uniti, utilizzando dati fittizi. Scopri come:

- Usa AWS Chalice per definire i percorsi nelle funzioni Lambda che vengono chiamate per gestire le richieste REST che arrivano tramite API Gateway.
- Utilizza le funzioni Lambda per recuperare e archiviare i dati in una tabella DynamoDB per soddisfare le richieste REST.
- Definisci la struttura della tabella e le risorse dei ruoli di sicurezza in un AWS CloudFormation modello.
- Usa AWS Chalice e CloudFormation per impacchettare e distribuire tutte le risorse necessarie.
- Usa CloudFormation per ripulire tutte le risorse create.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, vedi l'esempio completo su GitHub.

Servizi utilizzati in questo esempio

- Gateway API
- CloudFormation
- DynamoDB
- Lambda

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUtilizzo di DynamoDB con un SDK AWS. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Creazione di un'applicazione di messaggistica con Step Functions

Il seguente esempio di codice mostra come creare un'applicazione di AWS Step Functions messaggistica che recupera i record dei messaggi da una tabella di database.

Python

SDK per Python (Boto3)

Mostra come usare AWS SDK for Python (Boto3) with per creare un'applicazione di messaggistica che AWS Step Functions recupera i record dei messaggi da una tabella Amazon DynamoDB e li invia con Amazon Simple Queue Service (Amazon SQS). La macchina a stati si integra con una AWS Lambda funzione per scansionare il database alla ricerca di messaggi non inviati.

- Crea una macchina a stati che recuperi e aggiorni i record di messaggi da una tabella Amazon DynamoDB.
- Aggiorna la definizione della macchina a stati per inviare messaggi anche ad Amazon Simple Queue Service (Amazon SQS).
- Avvia e arresta l'esecuzione della macchina a stati.
- Connettiti a Lambda, DynamoDB e Amazon SQS da una macchina a stati utilizzando le integrazioni di servizi.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, vedi l'esempio completo su. GitHub

Servizi utilizzati in questo esempio

- DynamoDB

- Lambda
- Amazon SQS
- Step Functions

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUtilizzo di DynamoDB con un SDK AWS. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Creazione di un'applicazione di gestione delle risorse fotografiche che consente agli utenti di gestire le foto utilizzando etichette

Nell'esempio di codice seguente viene illustrato come creare un'applicazione serverless che consente agli utenti di gestire le foto mediante etichette.

.NET

SDK per .NET

Mostra come sviluppare un'applicazione per la gestione delle risorse fotografiche che rileva le etichette nelle immagini utilizzando Amazon Rekognition e le archivia per recuperarle in seguito.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, consulta l'esempio completo su GitHub.

Per approfondire l'origine di questo esempio, consulta il post su AWS Community.

Servizi utilizzati in questo esempio

- Gateway API
- DynamoDB
- Lambda
- Amazon Rekognition
- Simple Storage Service (Amazon S3)
- Amazon SNS

C++

SDK per C++

Mostra come sviluppare un'applicazione per la gestione delle risorse fotografiche che rileva le etichette nelle immagini utilizzando Amazon Rekognition e le archivia per recuperarle in seguito.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su GitHub.

Per approfondire l'origine di questo esempio, consulta il post su AWS Community.

Servizi utilizzati in questo esempio

- Gateway API
- DynamoDB
- Lambda
- Amazon Rekognition
- Simple Storage Service (Amazon S3)
- Amazon SNS

Java

SDK per Java 2.x

Mostra come sviluppare un'applicazione per la gestione delle risorse fotografiche che rileva le etichette nelle immagini utilizzando Amazon Rekognition e le archivia per recuperarle in seguito.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su GitHub.

Per approfondire l'origine di questo esempio, consulta il post su AWS Community.

Servizi utilizzati in questo esempio

- Gateway API
- DynamoDB
- Lambda

- Amazon Rekognition
- Simple Storage Service (Amazon S3)
- Amazon SNS

JavaScript

SDK per JavaScript (v3)

Mostra come sviluppare un'applicazione per la gestione delle risorse fotografiche che rileva le etichette nelle immagini utilizzando Amazon Rekognition e le archivia per recuperarle in seguito.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su GitHub.

Per approfondire l'origine di questo esempio, consulta il post su AWS Community.

Servizi utilizzati in questo esempio

- Gateway API
- DynamoDB
- Lambda
- Amazon Rekognition
- Simple Storage Service (Amazon S3)
- Amazon SNS

Kotlin

SDK per Kotlin

Mostra come sviluppare un'applicazione per la gestione delle risorse fotografiche che rileva le etichette nelle immagini utilizzando Amazon Rekognition e le archivia per recuperarle in seguito.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su GitHub.

Per approfondire l'origine di questo esempio, consulta il post su AWS Community.

Servizi utilizzati in questo esempio

- Gateway API
- DynamoDB
- Lambda
- Amazon Rekognition
- Simple Storage Service (Amazon S3)
- Amazon SNS

PHP

SDK per PHP

Mostra come sviluppare un'applicazione per la gestione delle risorse fotografiche che rileva le etichette nelle immagini utilizzando Amazon Rekognition e le archivia per recuperarle in seguito.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su GitHub.

Per approfondire l'origine di questo esempio, consulta il post su AWS Community.

Servizi utilizzati in questo esempio

- Gateway API
- DynamoDB
- Lambda
- Amazon Rekognition
- Simple Storage Service (Amazon S3)
- Amazon SNS

Rust

SDK per Rust

Mostra come sviluppare un'applicazione per la gestione delle risorse fotografiche che rileva le etichette nelle immagini utilizzando Amazon Rekognition e le archivia per recuperarle in seguito.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su GitHub.

Per approfondire l'origine di questo esempio, consulta il post su AWS Community.

Servizi utilizzati in questo esempio

- Gateway API
- DynamoDB
- Lambda
- Amazon Rekognition
- Simple Storage Service (Amazon S3)
- Amazon SNS

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUtilizzo di DynamoDB con un SDK AWS. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Crea una tabella DynamoDB con un indice secondario globale utilizzando l'SDK AWS

L'esempio di codice seguente mostra come creare una tabella con un indice secondario globale.

Java

SDK per Java 2.x

Crea una tabella DynamoDB con Global Secondary Index utilizzando. 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();
        }
    }
```

- Per i dettagli sull'API, consulta la sezione AWS SDK for Java 2.x API [CreateTable](#)Reference.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta[Utilizzo di DynamoDB con un SDK AWS](#). Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Crea una tabella DynamoDB con un'impostazione di throughput caldo utilizzando un SDK AWS

Gli esempi di codice seguenti mostrano come creare una tabella con throughput a caldo abilitato.

Java

SDK per Java 2.x

Per creare una tabella DynamoDB con throughput a caldo con 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);
    }
```

- Per i dettagli sull'API, consulta CreateTable AWS SDK for Java 2.xAPI Reference.

JavaScript

SDK per JavaScript (v3)

Per creare una tabella DynamoDB con throughput a caldo con AWS SDK per 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
);
*/
```

- Per i dettagli sull'API, consulta la sezione AWS SDK per JavaScript API CreateTableReference.

Python

SDK per Python (Boto3)

Per creare una tabella DynamoDB con throughput a caldo con 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
```

- Per i dettagli sull'API, consulta CreateTable AWSSDK for Python (Boto3) API Reference.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Utilizzo di DynamoDB con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Creazione di un'applicazione Web per tracciare i dati DynamoDB

I seguenti esempi di codice mostrano come creare un'applicazione Web che traccia gli elementi di lavoro in una tabella Amazon DynamoDB e utilizza il Servizio di email semplice Amazon (Amazon SES) per inviare report.

.NET

SDK per .NET

Mostra come utilizzare l'API Amazon DynamoDB per creare un'applicazione Web dinamica che traccia i dati di lavoro DynamoDB.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, consulta l'esempio completo su GitHub.

Servizi utilizzati in questo esempio

- DynamoDB
- Amazon SES

Java

SDK per Java 2.x

Mostra come utilizzare l'API Amazon DynamoDB per creare un'applicazione Web dinamica che traccia i dati di lavoro DynamoDB.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su GitHub.

Servizi utilizzati in questo esempio

- DynamoDB
- Amazon SES

Kotlin

SDK per Kotlin

Mostra come utilizzare l'API Amazon DynamoDB per creare un'applicazione Web dinamica che traccia i dati di lavoro DynamoDB.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su GitHub.

Servizi utilizzati in questo esempio

- DynamoDB
- Amazon SES

Python

SDK per Python (Boto3)

Mostra come utilizzare per AWS SDK for Python (Boto3) creare un servizio REST che tenga traccia degli elementi di lavoro in Amazon DynamoDB e invii report tramite e-mail utilizzando

Amazon Simple Email Service (Amazon SES). Questo esempio utilizza il framework Web Flask per gestire il routing HTTP e si integra con una pagina Web React per presentare un'applicazione Web completamente funzionale.

- Crea un servizio Flask REST che si integri con. Servizi AWS
- Lettura, scrittura e aggiornamento di elementi di lavoro archiviati in una tabella DynamoDB.
- Utilizzo di Amazon SES per inviare report via e-mail sugli elementi di lavoro.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, consulta l'esempio completo nel AWS Code Examples Repository su. GitHub

Servizi utilizzati in questo esempio

- DynamoDB
- Amazon SES

Per un elenco completo delle guide e degli esempi di codice per sviluppatori AWS SDK, consulta. Utilizzo di DynamoDB con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Creazione di un'applicazione di chat websocket con API Gateway

L'esempio di codice seguente mostra come creare un'applicazione chat servita da un'API websocket creata su Gateway Amazon API.

Python

SDK per Python (Boto3)

Mostra come utilizzarlo AWS SDK for Python (Boto3) con Amazon API Gateway V2 per creare un'API websocket che si integri con Amazon AWS Lambda DynamoDB.

- Crea un'API WebSocket servita da API Gateway
- Definisci un gestore Lambda che memorizzi le connessioni in DynamoDB e invii messaggi ad altri partecipanti alla chat.
- Connettiti all'applicazione di chat websocket e invia messaggi con il pacchetto Websockets.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, consulta l'esempio completo su. GitHub

Servizi utilizzati in questo esempio

- Gateway API
- DynamoDB
- Lambda

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUtilizzo di DynamoDB con un SDK AWS. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Crea un elemento DynamoDB con un TTL utilizzando un SDK AWS

Gli esempi di codice seguenti mostrano come creare un elemento con un valore TTL.

Java

SDK per 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;
        }
    }
}
```

- Per i dettagli sull'API, consulta la sezione PutItem AWS SDK for Java 2.xAPI Reference.

JavaScript

SDK per 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');
```

- Per i dettagli sull'API, consulta la sezione AWS SDK per JavaScript API PutItemReference.

Python

SDK per 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"
)
```

- Per i dettagli sull'API, consulta PutItem AWSSDK for Python (Boto3) API Reference.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Utilizzo di DynamoDB con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Crea e gestisci tabelle globali DynamoDB con una forte coerenza multiregionale utilizzando un SDK AWS

Gli esempi di codice seguenti mostrano come creare e gestire tabelle globali DynamoDB con elevata consistenza multi-Regione (MRSC).

- Creazione di una tabella con elevata consistenza multi-Regione.
- Verifica della configurazione MRSC e dello stato della replica.
- Verifica dell'elevata consistenza tra Regioni con letture immediate.
- Esecuzione di scritture condizionali con garanzie MRSC.
- Esecuzione della pulizia delle risorse su una tabella globale MRSC.

Bash

AWS CLI con lo script Bash

Creazione di una tabella con elevata consistenza multi-Regione.

```
# 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
```

Verifica della configurazione MRSC e dello stato della replica.

```
# 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}}'
```

Verifica della coerenza forte multi-Regione con letture immediate.

```
# 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
```

Esecuzione di scritture condizionali con garanzie 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
```

Esecuzione della pulizia delle risorse su una tabella globale 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
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento dei comandi AWS CLI .
  - CreateTable
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - UpdateItem
  - UpdateTable

Java

SDK per Java 2.x

Crea una tabella regionale pronta per la conversione MRSC utilizzando. 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();
        }
    }
```

Converti una tabella regionale in MRSC con repliche e verifica utilizzando. 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();
        }
    }
```

Descrivi una configurazione di tabella globale MRSC utilizzando. 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();
        }
    }
```

Per aggiungere elementi di test per verificare l'elevata consistenza 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();
        }
    }
```

Leggi gli elementi con letture coerenti dalle repliche MRSC utilizzando. 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();
        }
    }
```

Esegui aggiornamenti condizionali con garanzie MRSC utilizzando. 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();
        }
    }
```

Attendi che le repliche e i testimoni MRSC diventino attivi utilizzando. 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;
        }
    }
```

Pulisci le repliche e i testimoni MRSC utilizzando. 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();
        }
    }
```

Dimostrazione completa del flusso di lavoro MRSC utilizzando. 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;
        }
    }
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento delle API AWS SDK for Java 2.x .
  - CreateTable
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - UpdateItem
  - UpdateTable

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Utilizzo di DynamoDB con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Crea e gestisci tabelle globali DynamoDB dimostrando MREC utilizzando un SDK AWS

L'esempio di codice seguente mostra come creare e gestire tabelle globali DynamoDB con repliche tra più Regioni.

- Crea una tabella con Global Secondary Index e DynamoDB Streams.
- Aggiunta di repliche in Regioni diverse per creare una tabella globale.
- Rimuovi le repliche da una tabella globale.
- Aggiungi elementi di test per verificare la replica tra le regioni.

- Descrivi la configurazione globale della tabella e lo stato della replica.

Java

SDK per Java 2.x

Crea una tabella con Global Secondary Index e DynamoDB Streams utilizzando. 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;
        }
    }
```

Attendi che una tabella diventi attiva utilizzando. 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;
        }
    }
```

Aggiungi una replica per creare o estendere una tabella globale utilizzando 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;
       }
   }
```

Rimuovi una replica da una tabella globale utilizzando. 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;
        }
    }
```

Aggiungi elementi di test per verificare la replica utilizzando. 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;
        }
    }
```

Descrivi la configurazione globale della tabella e le repliche utilizzando. 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;
        }
    }
```

Esempio completo di operazioni su tabelle globali utilizzando 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;
            }
        }
    }
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento delle API AWS SDK for Java 2.x .
  - CreateTable
  - DescribeTable
  - PutItem
  - UpdateTable

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUtilizzo di DynamoDB con un SDK AWS. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Eliminare i dati DynamoDB utilizzando le istruzioni PartiQL DELETE con un SDK AWS

L'esempio di codice seguente mostra come eliminare i dati mediante le istruzioni PartiQL DELETE.

JavaScript

SDK per JavaScript (v3)

Eliminare elementi da una tabella DynamoDB utilizzando le istruzioni PartiQL DELETE con. AWS SDK per 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");
};
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento delle API AWS SDK per JavaScript .
  - BatchExecuteStatement
  - ExecuteStatement

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Utilizzo di DynamoDB con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Rileva PPE nelle immagini con Amazon Rekognition utilizzando un SDK AWS

L'esempio di codice seguente mostra come creare un'applicazione che utilizza Amazon Rekognition per rilevare dispositivi di protezione individuale (DPI) nelle immagini.

Java

SDK per Java 2.x

Mostra come creare una AWS Lambda funzione che rileva le immagini con dispositivi di protezione individuale.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su GitHub.

Servizi utilizzati in questo esempio

- DynamoDB
- Amazon Rekognition
- Simple Storage Service (Amazon S3)
- Amazon SES

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUtilizzo di DynamoDB con un SDK AWS. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Inserimento di dati DynamoDB utilizzando istruzioni PartiQL INSERT con un SDK AWS

L'esempio di codice seguente mostra come inserire dati mediante le istruzioni PartiQL INSERT.

JavaScript

SDK per JavaScript (v3)

Inserisci elementi in una tabella DynamoDB utilizzando le istruzioni PartiQL INSERT con. AWS SDK per 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);
};
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento delle API AWS SDK per JavaScript .
  - BatchExecuteStatement
  - ExecuteStatement

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Utilizzo di DynamoDB con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Richiamo a una funzione Lambda da un browser

Il seguente esempio di codice mostra come richiamare una AWS Lambda funzione da un browser.

JavaScript

SDK per JavaScript (v2)

Puoi creare un'applicazione basata su browser che utilizza una AWS Lambda funzione per aggiornare una tabella Amazon DynamoDB con selezioni utente.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, consulta l'esempio completo su. GitHub

Servizi utilizzati in questo esempio

- DynamoDB
- Lambda

SDK per JavaScript (v3)

Puoi creare un'applicazione basata su browser che utilizza una AWS Lambda funzione per aggiornare una tabella Amazon DynamoDB con selezioni utente. Questa app utilizza la versione 3. AWS SDK per JavaScript

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su GitHub.

Servizi utilizzati in questo esempio

- DynamoDB
- Lambda

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUtilizzo di DynamoDB con un SDK AWS. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Gestisci gli indici secondari globali DynamoDB utilizzando v2 AWS Command Line Interface

Nell'esempio di codice seguente viene mostrato come gestire l'intero ciclo di vita degli indici secondari globali.

- Creazione di una tabella con un indice secondario globale.
- Aggiunta di un nuovo indice secondario globale a una tabella esistente.
- Aggiornamento (aumento) del throughput a caldo del GSI.
- Interroga i dati utilizzando GSIs.
- Eliminazione di un GSI.

Bash

AWS CLI con lo script Bash

Per creare una tabella con un indice secondario globale.

```
# 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}"
```

Per aggiungere un nuovo GSI (on demand) a una tabella esistente.

```
# 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\"}}}]"
```

Per aggiornare (aumentare) il throughput a caldo 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}}}]"
```

Interroga i dati usando GSIs.

```
# 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"}}'
```

Eliminazione di un GSI.

```
# Delete a GSI from a table
aws dynamodb update-table \
    --table-name MusicCollection \
    --global-secondary-index-updates \
        "[{\"Delete\":{\"IndexName\":\"GenreIndex\"}}]"
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento dei comandi AWS CLI .

- CreateTable
- DeleteTable
- Query
- UpdateTable

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUtilizzo di DynamoDB con un SDK AWS. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Gestisci le policy basate sulle risorse di DynamoDB utilizzando la versione 2 AWS Command Line Interface

Nell'esempio di codice seguente viene mostrato come gestire l'intero ciclo di vita delle policy basate sulle risorse per le tabelle DynamoDB.

- Creazione di una tabella con una policy sulle risorse.
- Recupero di una policy sulle risorse.
- Aggiornamento di una policy sulle risorse.
- Eliminazione di una policy sulle risorse.

Bash

AWS CLI con lo script Bash

Creazione di una tabella con una policy sulle risorse.

```
# 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
```

Recupero di una policy sulle risorse.

```
# 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
```

Aggiornamento di una policy sulle risorse.

```
# 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
```

Eliminazione di una policy sulle risorse.

```
# Delete the resource-based policy from a table
aws dynamodb delete-resource-policy \
    --resource-arn arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento dei comandi AWS CLI .
  - CreateTable
  - DeleteResourcePolicy
  - GetResourcePolicy
  - PutResourcePolicy

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Utilizzo di DynamoDB con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Monitora le prestazioni di Amazon DynamoDB utilizzando un SDK AWS

L'esempio di codice seguente mostra come configurare l'utilizzo di DynamoDB da parte di un'applicazione per monitorare le prestazioni.

Java

SDK per Java 2.x

Questo esempio mostra come configurare un'applicazione Java per monitorare le prestazioni di DynamoDB. L'applicazione invia i dati metrici a CloudWatch cui è possibile monitorare le prestazioni.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, consulta l'esempio completo su GitHub.

Servizi utilizzati in questo esempio

- CloudWatch
- DynamoDB

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUtilizzo di DynamoDB con un SDK AWS. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Esegui operazioni di interrogazione avanzate su DynamoDB utilizzando un SDK AWS

Gli esempi di codice seguenti mostrano come eseguire operazioni di query avanzate in DynamoDB.

- Tabelle di interrogazione che utilizzano varie tecniche di filtraggio e condizione.
- Implementa l'impaginazione per set di risultati di grandi dimensioni.
- Usa gli indici secondari globali per modelli di accesso alternativi.
- Applica controlli di coerenza in base ai requisiti dell'applicazione.

Java

SDK per Java 2.x

Interroga con letture fortemente coerenti utilizzando. 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;
        }
    }
```

Esegui una query utilizzando un indice secondario globale 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);
        }
```

```
    }
```

Interrogazione con impaginazione utilizzando 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;
        }
    }
```

Interrogazione con filtri complessi utilizzando 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);
    }
```

Interrogazione con un'espressione di filtro costruita dinamicamente utilizzando 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);
        }
    }
```

Interroga con un'espressione di filtro e limita l'utilizzo 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;
        }
    }
```

- Per ulteriori informazioni sulle API, consulta Query nella Documentazione di riferimento delle API AWS SDK for Java 2.x .

JavaScript

SDK per JavaScript (v3)

Interroga con letture fortemente coerenti utilizzando. AWS SDK per 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;
  }
}
```

Esegui una query utilizzando un indice secondario globale con AWS SDK per 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;
  }
}
```

Interrogazione con impaginazione utilizzando AWS SDK per 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 };
```

Interrogazione con filtri complessi utilizzando AWS SDK per 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;
  }
}
```

Interrogazione con un'espressione di filtro costruita dinamicamente utilizzando AWS SDK per 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;
  }
}
```

- Per ulteriori informazioni sulle API, consulta Query nella Documentazione di riferimento delle API AWS SDK per JavaScript .

Python

SDK per Python (Boto3)

Interrogazione con letture fortemente coerenti utilizzando. 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
```

Esegui una query utilizzando un indice secondario globale 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
```

Interrogazione con impaginazione utilizzando 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
```

Interrogazione con filtri complessi utilizzando 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
```

Interrogazione con un'espressione di filtro costruita dinamicamente utilizzando 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
```

Interroga con un'espressione di filtro e limita l'utilizzo 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
```

- Per informazioni dettagliate sulle API, consulta Query nella Documentazione di riferimento per l'API SDK for Python (Boto3)AWS .

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUtilizzo di DynamoDB con un SDK AWS. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Esegui operazioni sugli elenchi in DynamoDB con un SDK AWS

Gli esempi di codice seguenti mostrano come eseguire operazioni di tipo elenco in DynamoDB.

- Aggiungi elementi a un attributo di elenco.
- Rimuove elementi da un attributo di elenco.
- Aggiorna elementi specifici in un elenco per indice.
- Usa le funzioni di aggiunta alla lista e all'indice delle liste.

Java

SDK per Java 2.x

Dimostra le operazioni sugli elenchi utilizzando 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();
        }
    }
```

Esempio di utilizzo delle operazioni di elenco 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();
        }
    }
```

- Per i dettagli sull'API, consulta la UpdateItemsezione AWS SDK for Java 2.x API Reference.

JavaScript

SDK per JavaScript (v3)

Dimostra le operazioni sugli elenchi utilizzando. AWS SDK per 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;
}
```

Esempio di utilizzo delle operazioni di elenco con AWS SDK per 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;
}
```

- Per i dettagli sull'API, consulta la UpdateItemsezione AWS SDK per JavaScript API Reference.

Python

SDK per Python (Boto3)

Dimostra le operazioni sugli elenchi utilizzando 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
```

Esempio di utilizzo delle operazioni di elenco 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")
```

- Per i dettagli sull'API, consulta UpdateItem AWSSDK for Python (Boto3) API Reference.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Utilizzo di DynamoDB con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Esegui operazioni di mappatura in DynamoDB con un SDK AWS

Gli esempi di codice seguenti mostrano come eseguire operazioni di mappatura in DynamoDB.

- Aggiungi e aggiorna gli attributi annidati nelle strutture delle mappe.
- Rimuovi campi specifici dalle mappe.
- Lavora con attributi di mappa profondamente annidati.

Java

SDK per Java 2.x

Dimostra le operazioni sulla mappa utilizzando 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();
        }
    }
```

Esempio di utilizzo delle operazioni cartografiche 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();
        }
    }
```

- Per i dettagli sull'API, consulta la UpdateItemsezione AWS SDK for Java 2.x API Reference.

JavaScript

SDK per JavaScript (v3)

Dimostra le operazioni sulla mappa utilizzando. AWS SDK per 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();
}
```

- Per i dettagli sull'API, consulta la UpdateItemsezione AWS SDK per JavaScript API Reference.

Python

SDK per Python (Boto3)

Dimostra le operazioni sulla mappa utilizzando 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()
```

- Per i dettagli sull'API, consulta UpdateItem AWSSDK for Python (Boto3) API Reference.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Utilizzo di DynamoDB con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Esegui operazioni di set in DynamoDB con un SDK AWS

Gli esempi di codice seguenti mostrano come eseguire operazioni di impostazione in DynamoDB.

- Aggiunta di elementi a un attributo di set.
- Rimuove elementi da un attributo set.
- Utilizzo di operazioni ADD e DELETE con i set.

Java

SDK per Java 2.x

Dimostra le operazioni impostate utilizzando 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();
        }
    }
```

Esempio di utilizzo delle operazioni di set 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();
        }
    }
```

- Per i dettagli sull'API, vedere UpdateItemin AWS SDK for Java 2.x API Reference.

JavaScript

SDK per JavaScript (v3)

Dimostra le operazioni sul set utilizzando. AWS SDK per 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;
}
```

Esempio di utilizzo delle operazioni di set con AWS SDK per 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);
  }
}
```

- Per i dettagli sull'API, vedere UpdateItemin AWS SDK per JavaScript API Reference.

Python

SDK per Python (Boto3)

Dimostra le operazioni impostate utilizzando 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", {}),
    }
```

Esempio di utilizzo delle operazioni di set 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")
```

- Per i dettagli sull'API, consulta UpdateItem AWSSDK for Python (Boto3) API Reference.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Utilizzo di DynamoDB con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Interroga una tabella DynamoDB utilizzando batch di istruzioni PartiQL e un SDK AWS

Gli esempi di codice seguenti mostrano come:

- Ricezione di un batch di elementi mediante più istruzioni SELECT.

- Aggiunta di un batch di articoli eseguendo più istruzioni INSERT.
- Aggiornamento di un batch di elementi mediante più istruzioni UPDATE.
- Eliminazione di un batch di elementi mediante più istruzioni DELETE.

.NET

SDK per .NET

> **Note**
>
> C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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;
        }
```

- Per i dettagli sull'API, BatchExecuteStatementconsulta AWS SDK per .NET API Reference.

C++

SDK per C++

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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;
}
```

- Per i dettagli sull'API, BatchExecuteStatementconsulta AWS SDK per C++ API Reference.

Go

SDK per Go V2

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Esecuzione di uno scenario che crea una tabella ed esegue batch di query 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))
}
```

Per definire una struttura di filmato utilizzata in questo esempio.

```
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"])
}
```

Per creare una struttura e i metodi che eseguono istruzioni 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
}
```

- Per i dettagli sull'API, BatchExecuteStatementconsulta AWS SDK per Go API Reference.

Java

SDK per Java 2.x

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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);
    }
}
```

- Per i dettagli sull'API, BatchExecuteStatementconsulta AWS SDK for Java 2.x API Reference.

JavaScript

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Eseguire le istruzioni PartiQL in batch.

```
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.");
};
```

- Per i dettagli sull'API, BatchExecuteStatementconsulta AWS SDK per JavaScript API Reference.

Kotlin

SDK per Kotlin

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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")
    }
}
```

- Per i dettagli sull'API, BatchExecuteStatementconsulta AWS SDK for Kotlin API reference.

PHP

SDK per PHP

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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,
                ],
            ],
        ]);
    }
```

- Per i dettagli sull'API, BatchExecuteStatementconsulta AWS SDK per PHP API Reference.

Python

SDK per Python (Boto3)

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Creazione di una classe in grado di eseguire batch di istruzioni 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
```

Esecuzione di uno scenario che crea una tabella ed esegue query PartiQL in batch.

```
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}")
```

- Per i dettagli sull'API, consulta BatchExecuteStatement AWSSDK for Python (Boto3) API Reference.

Ruby

SDK per Ruby

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Esecuzione di uno scenario che crea una tabella ed esegue query PartiQL in batch.

```
 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
```

- Per i dettagli sull'API, BatchExecuteStatementconsulta AWS SDK per Ruby API Reference.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUtilizzo di DynamoDB con un SDK AWS. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Interroga una tabella DynamoDB utilizzando PartiQL e un SDK AWS

Gli esempi di codice seguenti mostrano come:

- Ricezione di un articolo eseguendo un'istruzione SELECT.
- Aggiunta di un elemento eseguendo un'istruzione INSERT.
- Aggiornamento di un elemento eseguendo un'istruzione UPDATE.
- Eliminazione di un elemento eseguendo un'istruzione DELETE.

.NET

SDK per .NET

> **Note**
>
> C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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;
        }
```

- Per i dettagli sull'API, ExecuteStatementconsulta AWS SDK per .NET API Reference.

C++

SDK per C++

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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;
}
```

- Per i dettagli sull'API, ExecuteStatementconsulta AWS SDK per C++ API Reference.

Go

SDK per Go V2

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Esecuzione di uno scenario che crea una tabella ed esegue query 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))
}
```

Per definire una struttura di filmato utilizzata in questo esempio.

```
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"])
}
```

Per creare una struttura e i metodi che eseguono istruzioni 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
}
```

- Per i dettagli sull'API, ExecuteStatementconsulta AWS SDK per Go API Reference.

Java

SDK per Java 2.x

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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());
    }
}
```

- Per i dettagli sull'API, ExecuteStatementconsulta AWS SDK for Java 2.x API Reference.

JavaScript

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Eseguire le singole istruzioni PartiQL.

```
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.");
};
```

- Per i dettagli sull'API, ExecuteStatementconsulta AWS SDK per JavaScript API Reference.

Kotlin

SDK per Kotlin

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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)
}
```

- Per i dettagli sull'API, ExecuteStatementconsulta AWS SDK for Kotlin API reference.

PHP

SDK per PHP

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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,
        ]);
    }
```

- Per i dettagli sull'API, ExecuteStatementconsulta AWS SDK per PHP API Reference.

Python

SDK per Python (Boto3)

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Creazione di una classe in grado di eseguire istruzioni 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
```

Esecuzione di uno scenario che crea una tabella ed esegue query 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}")
```

- Per i dettagli sull'API, consulta ExecuteStatement AWSSDK for Python (Boto3) API Reference.

Ruby

SDK per Ruby

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Esecuzione di uno scenario che crea una tabella ed esegue query 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
```

- Per i dettagli sull'API, ExecuteStatementconsulta AWS SDK per Ruby API Reference.

Rust

SDK per Rust

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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(())
}
```

- Per i dettagli sulle API, consulta la ExecuteStatementguida di riferimento all'API AWS SDK for Rust.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Utilizzo di DynamoDB con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Interroga una tabella DynamoDB utilizzando un indice secondario globale con un SDK AWS

L'esempio di codice seguente mostra come eseguire query su una tabella con un indice secondario globale.

- Interroga una tabella DynamoDB utilizzando la sua chiave primaria.
- Interroga un Global Secondary Index (GSI) per modelli di accesso alternativi.
- Confronto di query su tabelle e query su GSI.

Java

SDK per Java 2.x

Interroga una tabella DynamoDB utilizzando la sua chiave primaria e un Global Secondary Index (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);
       }
   }
```

Confronta l'interrogazione diretta di una tabella con l'interrogazione di 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);
        }
    }
```

- Per ulteriori informazioni sulle API, consulta Query nella Documentazione di riferimento delle API AWS SDK for Java 2.x .

JavaScript

SDK per (v3) JavaScript

Interroga una tabella DynamoDB utilizzando la chiave primaria con. AWS SDK per 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;
  }
}
```

Interroga un DynamoDB Global Secondary Index (GSI) con. AWS SDK per 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;
  }
}
```

- Per ulteriori informazioni sulle API, consulta Query nella Documentazione di riferimento delle API AWS SDK per JavaScript .

Python

SDK per Python (Boto3)

Interroga una tabella DynamoDB utilizzando la sua chiave primaria e un Global Secondary Index (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
```

- Per informazioni dettagliate sulle API, consulta Query nella Documentazione di riferimento per l'API SDK for Python (Boto3)AWS .

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Utilizzo di DynamoDB con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Interroga una tabella DynamoDB utilizzando una condizione begins_with con un SDK AWS

L'esempio di codice seguente mostra come eseguire query su una tabella utilizzando una condizione begins_with.

- Usa la funzione begins_with in un'espressione di condizione chiave.
- Filtra gli elementi in base a uno schema di prefisso nella chiave di ordinamento.

Java

SDK per Java 2.x

Per eseguire query su una tabella DynamoDB utilizzando una condizione begins_with nella chiave di ordinamento 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;
        }
    }
```

Per descrivere l'utilizzo della condizione begins_with con prefissi di diversa lunghezza 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);
        }
    }
```

- Per ulteriori informazioni sulle API, consulta Query nella Documentazione di riferimento delle API AWS SDK for Java 2.x .

JavaScript

SDK per JavaScript (v3)

Per eseguire query su una tabella DynamoDB utilizzando una condizione begins_with nella chiave di ordinamento con AWS SDK per 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;
  }
}
```

- Per ulteriori informazioni sulle API, consulta Query nella Documentazione di riferimento delle API AWS SDK per JavaScript .

Python

SDK per Python (Boto3)

Per eseguire query su una tabella DynamoDB utilizzando una condizione begins_with nella chiave di ordinamento 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
```

- Per informazioni dettagliate sulle API, consulta Query nella Documentazione di riferimento per l'API SDK for Python (Boto3)AWS .

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Utilizzo di DynamoDB con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Interroga una tabella DynamoDB utilizzando un intervallo di date nella chiave di ordinamento con un SDK AWS

Gli esempi di codice seguenti mostrano come eseguire query su una tabella utilizzando un intervallo di date nella chiave di ordinamento.

- Interroga gli elementi all'interno di un intervallo di date specifico.
- Utilizza gli operatori di confronto sulle chiavi di ordinamento in formato data.

Java

SDK per Java 2.x

Interroga una tabella DynamoDB per gli elementi compresi in un intervallo di date 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;
        }
    }
```

Per eseguire query su una tabella DynamoDB con filtri per intervalli di date.

```
    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);
        }
    }
```

- Per ulteriori informazioni sulle API, consulta Query nella Documentazione di riferimento delle API AWS SDK for Java 2.x .

JavaScript

SDK per (v3 JavaScript)

Interroga una tabella DynamoDB per gli elementi compresi in un intervallo di date con. AWS SDK per 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;
  }
}
```

- Per ulteriori informazioni sulle API, consulta Query nella Documentazione di riferimento delle API AWS SDK per JavaScript .

Python

SDK per Python (Boto3)

Interroga una tabella DynamoDB per gli elementi compresi in un intervallo di date 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
```

- Per informazioni dettagliate sulle API, consulta Query nella Documentazione di riferimento per l'API SDK for Python (Boto3)AWS .

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Utilizzo di DynamoDB con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Interroga una tabella DynamoDB con un'espressione di filtro complessa con un SDK AWS

Gli esempi di codice seguenti mostrano come eseguire query su una tabella con un'espressione di filtro complessa.

- Applica espressioni di filtro complesse ai risultati delle query.
- Combina più condizioni utilizzando operatori logici.
- Filtra gli elementi in base ad attributi non chiave.

Java

SDK per Java 2.x

Interroga una tabella DynamoDB con un'espressione di filtro complessa utilizzando. 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);
    }
```

- Per ulteriori informazioni sulle API, consulta Query nella Documentazione di riferimento delle API AWS SDK for Java 2.x .

JavaScript

SDK per (v3 JavaScript)

Interroga una tabella DynamoDB con un'espressione di filtro complessa utilizzando. AWS SDK per 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;
  }
}
```

- Per ulteriori informazioni sulle API, consulta Query nella Documentazione di riferimento delle API AWS SDK per JavaScript .

Python

SDK per Python (Boto3)

Interroga una tabella DynamoDB con un'espressione di filtro complessa utilizzando. 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
```

- Per informazioni dettagliate sulle API, consulta Query nella Documentazione di riferimento per l'API SDK for Python (Boto3)AWS .

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Utilizzo di DynamoDB con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Interroga una tabella DynamoDB con un'espressione di filtro dinamica con un SDK AWS

Gli esempi di codice seguenti mostrano come eseguire query su una tabella con un'espressione di filtro dinamica.

- Crea espressioni di filtro in modo dinamico in fase di esecuzione.
- Costruisci condizioni di filtro in base all'input dell'utente o allo stato dell'applicazione.
- Aggiungi o rimuovi i criteri di filtro in modo condizionale.

Java

SDK per Java 2.x

Interroga una tabella DynamoDB con un'espressione di filtro costruita dinamicamente utilizzando. 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();
            }
        }
    }
```

Dimostra come utilizzare le espressioni di filtro dinamiche 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);
        }
    }
```

- Per ulteriori informazioni sulle API, consulta Query nella Documentazione di riferimento delle API AWS SDK for Java 2.x .

JavaScript

SDK per JavaScript (v3)

Interroga una tabella DynamoDB con un'espressione di filtro costruita dinamicamente utilizzando. AWS SDK per 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;
  }
}
```

- Per ulteriori informazioni sulle API, consulta Query nella Documentazione di riferimento delle API AWS SDK per JavaScript .

Python

SDK per Python (Boto3)

Interroga una tabella DynamoDB con un'espressione di filtro costruita dinamicamente utilizzando. 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
```

Dimostra come utilizzare le espressioni di filtro dinamiche 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}")
```

- Per informazioni dettagliate sulle API, consulta Query nella Documentazione di riferimento per l'API SDK for Python (Boto3)AWS .

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Utilizzo di DynamoDB con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Interroga una tabella DynamoDB con un'espressione di filtro e un limite con un SDK AWS

Gli esempi di codice seguenti mostrano come eseguire query su una tabella con un'espressione di filtro e un limite.

- Applica le espressioni di filtro ai risultati delle query con un limite sugli elementi valutati.
- Scopri come il limite influisce sui risultati delle query filtrate.
- Controlla il numero massimo di elementi elaborati in una query.

Java

SDK per Java 2.x

Interroga una tabella DynamoDB con un'espressione di filtro e limita l'utilizzo. 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;
        }
    }
```

- Per ulteriori informazioni sulle API, consulta Query nella Documentazione di riferimento delle API AWS SDK for Java 2.x .

Python

SDK per Python (Boto3)

Interroga una tabella DynamoDB con un'espressione di filtro e limita l'utilizzo. 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
```

Dimostra come utilizzare le espressioni di filtro con limiti. 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.")
```

- Per informazioni dettagliate sulle API, consulta Query nella Documentazione di riferimento per l'API SDK for Python (Boto3)AWS .

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Utilizzo di DynamoDB con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Interroga una tabella DynamoDB con attributi annidati utilizzando un SDK AWS

Gli esempi di codice seguenti mostrano come eseguire una query su una tabella con attributi annidati.

- Accedi e filtra in base agli attributi annidati negli elementi DynamoDB.
- Utilizza le espressioni del percorso del documento per fare riferimento agli elementi nidificati.

Java

SDK per Java 2.x

Interroga una tabella DynamoDB con attributi annidati utilizzando. 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;
        }
    }
```

Per descrivere come eseguire query su una tabella DynamoDB con attributi annidati.

```
    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);
        }
    }
```

- Per ulteriori informazioni sulle API, consulta Query nella Documentazione di riferimento delle API AWS SDK for Java 2.x .

JavaScript

SDK per (v3) JavaScript

Interroga una tabella DynamoDB con attributi annidati utilizzando. AWS SDK per 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;
  }
}
```

- Per ulteriori informazioni sulle API, consulta Query nella Documentazione di riferimento delle API AWS SDK per JavaScript .

Python

SDK per Python (Boto3)

Interroga una tabella DynamoDB con attributi annidati utilizzando. 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
```

- Per informazioni dettagliate sulle API, consulta Query nella Documentazione di riferimento per l'API SDK for Python (Boto3)AWS .

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Utilizzo di DynamoDB con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Interroga una tabella DynamoDB con impaginazione utilizzando un SDK AWS

Gli esempi di codice seguenti mostrano come eseguire una query su una tabella con l'impaginazione.

- Implementa la paginazione per i risultati delle query DynamoDB.
- Utilizzate il LastEvaluatedKey per recuperare le pagine successive.
- Controlla il numero di elementi per pagina con il parametro Limit.

Java

SDK per Java 2.x

Interroga una tabella DynamoDB con impaginazione utilizzando. 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;
```

```
        }
    }
```

Per descrivere come eseguire query su una tabella DynamoDB con impaginazione.

```
    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);
        }
    }
```

- Per ulteriori informazioni sulle API, consulta Query nella Documentazione di riferimento delle API AWS SDK for Java 2.x .

JavaScript

SDK per (v3) JavaScript

Interroga una tabella DynamoDB con impaginazione utilizzando. AWS SDK per 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 };
```

- Per ulteriori informazioni sulle API, consulta Query nella Documentazione di riferimento delle API AWS SDK per JavaScript .

Python

SDK per Python (Boto3)

Interroga una tabella DynamoDB con impaginazione utilizzando. 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
```

- Per informazioni dettagliate sulle API, consulta Query nella Documentazione di riferimento per l'API SDK for Python (Boto3)AWS .

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Utilizzo di DynamoDB con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Interroga una tabella DynamoDB con letture fortemente coerenti utilizzando un SDK AWS

L'esempio di codice seguente mostra come eseguire query su una tabella con operazioni a elevata consistenza di lettura.

- Configura il livello di coerenza per le query DynamoDB.
- Utilizza letture fortemente coerenti per ottenere la maggior parte dei dati. up-to-date
- Comprendi i compromessi tra coerenza finale e forte coerenza.

Java

SDK per Java 2.x

Interroga una tabella DynamoDB con coerenza di lettura configurabile utilizzando. 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;
        }
    }
```

- Per ulteriori informazioni sulle API, consulta Query nella Documentazione di riferimento delle API AWS SDK for Java 2.x .

JavaScript

SDK per (v3) JavaScript

Interroga una tabella DynamoDB con coerenza di lettura configurabile utilizzando. AWS SDK per 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;
  }
}
```

- Per ulteriori informazioni sulle API, consulta Query nella Documentazione di riferimento delle API AWS SDK per JavaScript .

Python

SDK per Python (Boto3)

Interroga una tabella DynamoDB con l'opzione per letture fortemente coerenti utilizzando. 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
```

- Per informazioni dettagliate sulle API, consulta Query nella Documentazione di riferimento per l'API SDK for Python (Boto3)AWS .

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Utilizzo di DynamoDB con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Interroga i dati DynamoDB utilizzando istruzioni PartiQL SELECT con un SDK AWS

L'esempio seguente mostra come eseguire query sui dati mediante le istruzioni PartiQL SELECT.

JavaScript

SDK per JavaScript (v3)

Interroga gli elementi da una tabella DynamoDB utilizzando le istruzioni PartiQL SELECT con. AWS SDK per 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");
};
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento delle API AWS SDK per JavaScript .
  - BatchExecuteStatement
  - ExecuteStatement

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Utilizzo di DynamoDB con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Interroga una tabella DynamoDB per gli elementi TTL utilizzando un SDK AWS

Gli esempi di codice seguenti mostrano come eseguire query su elementi TTL.

Java

SDK per Java 2.x

Interroga l'espressione filtrata per raccogliere elementi TTL in una tabella DynamoDB utilizzando. 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;
        }
```

- Per ulteriori informazioni sulle API, consulta Query nella Documentazione di riferimento delle API AWS SDK for Java 2.x .

JavaScript

SDK per (v3) JavaScript

Interroga l'espressione filtrata per raccogliere elementi TTL in una tabella DynamoDB utilizzando. AWS SDK per 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');
```

- Per ulteriori informazioni sulle API, consulta Query nella Documentazione di riferimento delle API AWS SDK per JavaScript .

Python

SDK per Python (Boto3)

Interroga l'espressione filtrata per raccogliere elementi TTL in una tabella DynamoDB utilizzando. 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")
```

- Per informazioni dettagliate sulle API, consulta Query nella Documentazione di riferimento per l'API SDK for Python (Boto3)AWS .

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Utilizzo di DynamoDB con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Interroga le tabelle DynamoDB utilizzando modelli di data e ora con un SDK AWS

Gli esempi di codice seguenti mostrano come eseguire query su tabelle utilizzando modelli di data e ora.

- Archivia e interroga date/time i valori in DynamoDB.
- Implementa interrogazioni su intervalli di date utilizzando chiavi di ordinamento.
- Formatta le stringhe di data per interrogazioni efficaci.

Java

SDK per Java 2.x

Esegui una query utilizzando intervalli di date nelle chiavi di ordinamento 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;
        }
    }
```

Interroga utilizzando variabili data-ora 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;
        }
    }
```

Per eseguire query all'interno di intervalli di date in timestamp per epoch 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;
        }
    }
```

Esegui una query entro intervalli di date utilizzando LocalDateTime oggetti 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);
    }
```

- Per ulteriori informazioni sulle API, consulta Query nella Documentazione di riferimento delle API AWS SDK for Java 2.x .

JavaScript

SDK per JavaScript (v3)

Esegui una query utilizzando intervalli di date nelle chiavi di ordinamento con. AWS SDK per 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;
  }
}
```

Interroga utilizzando variabili data-ora con. AWS SDK per 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;
  }
}
```

- Per ulteriori informazioni sulle API, consulta Query nella Documentazione di riferimento delle API AWS SDK per JavaScript .

Python

SDK per Python (Boto3)

Esegui una query utilizzando intervalli di date nelle chiavi di ordinamento 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
```

Interroga utilizzando variabili data-ora 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}")
```

- Per informazioni dettagliate sulle API, consulta Query nella Documentazione di riferimento per l'API SDK for Python (Boto3)AWS .

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Utilizzo di DynamoDB con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Salva EXIF e altre informazioni sull'immagine utilizzando un SDK AWS

L'esempio di codice seguente mostra come:

- Recuperare informazioni EXIF da un file JPG, JPEG o PNG.
- Carica il file immagine in un bucket Amazon S3.
- Utilizza Amazon Rekognition per identificare i tre attributi principali (etichette) nel file.
- Aggiungi le informazioni su EXIF ed etichette a una tabella Amazon DynamoDB nella regione.

Rust

SDK per Rust

Recupera le informazioni EXIF da un file JPG, JPEG o PNG, carica il file di immagine in un bucket Amazon S3, utilizza Amazon Rekognition per identificare i tre attributi principali (etichette in Amazon Rekognition) nel file e aggiungi le informazioni su EXIF ed etichette a una tabella Amazon DynamoDB nella regione.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su GitHub.

Servizi utilizzati in questo esempio

- DynamoDB
- Amazon Rekognition
- Simple Storage Service (Amazon S3)

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUtilizzo di DynamoDB con un SDK AWS. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Configurare il controllo degli accessi basato sugli attributi per DynamoDB utilizzando v2 AWS Command Line Interface

Nell'esempio di codice seguente viene mostrato come implementare il controllo degli accessi basato su attributi (ABAC) per DynamoDB.

- Creazione di una policy IAM per ABAC.
- Per creare tabelle con tag per reparti diversi.
- Per elencare e filtrare tabelle in base ai tag.

Bash

AWS CLI con lo script Bash

Per creare una policy IAM per l'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
```

Per creare tabelle con tag per reparti diversi.

```
# 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
```

Per elencare e filtrare tabelle in base ai tag.

```
# 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
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento dei comandi AWS CLI .
  - CreatePolicy
  - CreateTable
  - ListTables

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Utilizzo di DynamoDB con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Comprendi l'ordine delle espressioni di aggiornamento in DynamoDB con un SDK AWS

Gli esempi di codice seguenti illustrano l'ordine delle espressioni di aggiornamento.

- Informazioni su come DynamoDB elabora le espressioni di aggiornamento.
- Comprendi l'ordine delle operazioni nelle espressioni di aggiornamento.
- Evita risultati imprevisti comprendendo la valutazione delle espressioni.

Java

SDK per Java 2.x

Dimostra l'ordine di aggiornamento delle espressioni utilizzando 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();
        }
    }
```

Esempio di utilizzo dell'ordine di aggiornamento delle espressioni 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();
        }
    }
```

- Per i dettagli sull'API, vedere UpdateItemin AWS SDK for Java 2.x API Reference.

JavaScript

SDK per JavaScript (v3)

Dimostra l'ordine di aggiornamento delle espressioni utilizzando. AWS SDK per 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
  };
}
```

Esempio di utilizzo dell'ordine di aggiornamento delle espressioni con AWS SDK per 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);
  }
}
```

- Per i dettagli sull'API, vedere UpdateItemin AWS SDK per JavaScript API Reference.

Python

SDK per Python (Boto3)

Dimostra l'ordine di aggiornamento delle espressioni utilizzando 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
```

Esempio di utilizzo dell'ordine di aggiornamento delle espressioni 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")
```

- Per i dettagli sull'API, consulta UpdateItem AWSSDK for Python (Boto3) API Reference.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Utilizzo di DynamoDB con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Aggiorna un'impostazione di tabella DynamoDB con un throughput caldo utilizzando un SDK AWS

Gli esempi di codice seguenti mostrano come aggiornare l’impostazione del throughput a caldo di una tabella.

Java

SDK per Java 2.x

Per aggiornare l’impostazione del throughput a caldo di una tabella DynamoDB esistente con un SDK 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);
    }
```

- Per i dettagli sull'API, consulta la sezione UpdateTable AWS SDK for Java 2.xAPI Reference.

JavaScript

SDK per JavaScript (v3)

Per aggiornare l'impostazione del throughput a caldo di una tabella DynamoDB esistente con un SDK AWS SDK per 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
);
*/
```

- Per i dettagli sull'API, consulta la sezione AWS SDK per JavaScript API UpdateTableReference.

Python

SDK per Python (Boto3)

Per aggiornare l'impostazione del throughput a caldo di una tabella DynamoDB esistente con un SDK 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
```

- Per i dettagli sull'API, consulta UpdateTable AWSSDK for Python (Boto3) API Reference.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Utilizzo di DynamoDB con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Aggiornare un elemento DynamoDB con un TTL utilizzando un SDK AWS

Gli esempi di codice seguenti mostrano come aggiornare il valore TTL di un elemento.

Java

SDK per Java 2.x

Per aggiornare il TTL su un elemento DynamoDB esistente in una tabella.

```
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;
        }
    }
```

- Per i dettagli sull'API, consulta la sezione UpdateItem AWS SDK for Java 2.xAPI Reference.

JavaScript

SDK per 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');
```

- Per i dettagli sull'API, consulta la sezione AWS SDK per JavaScript API UpdateItemReference.

Python

SDK per 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"
)
```

- Per i dettagli sull'API, consulta UpdateItem AWSSDK for Python (Boto3) API Reference.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Utilizzo di DynamoDB con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Aggiorna i dati DynamoDB utilizzando le istruzioni PartiQL UPDATE con un SDK AWS

L'esempio di codice seguente mostra come aggiornare i dati mediante le istruzioni PartiQL UPDATE.

JavaScript

SDK per JavaScript (v3)

Aggiorna gli elementi in una tabella DynamoDB utilizzando le istruzioni PartiQL UPDATE con. AWS SDK per 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(),
    },
  ]);
};
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento delle API AWS SDK per JavaScript .
  - BatchExecuteStatement
  - ExecuteStatement

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Utilizzo di DynamoDB con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo di un'API Gateway per richiamare una funzione Lambda

I seguenti esempi di codice mostrano come creare una AWS Lambda funzione richiamata da Amazon API Gateway.

Java

SDK per Java 2.x

Mostra come creare una AWS Lambda funzione utilizzando l'API runtime Lambda Java. Questo esempio richiama diversi AWS servizi per eseguire un caso d'uso specifico. In questo esempio viene illustrato come creare una funzione Lambda richiamata da Gateway Amazon API che analizza una tabella Amazon DynamoDB per le ricorrenze di lavoro e utilizza Amazon Simple Notification Service (Amazon SNS) per inviare un messaggio di testo ai dipendenti che si congratula con loro alla data dell'anniversario di un anno.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su. GitHub

Servizi utilizzati in questo esempio

- Gateway API
- DynamoDB
- Lambda
- Amazon SNS

JavaScript

SDK per JavaScript (v3)

Mostra come creare una AWS Lambda funzione utilizzando l'API di JavaScript runtime Lambda. Questo esempio richiama diversi AWS servizi per eseguire un caso d'uso specifico. In questo esempio viene illustrato come creare una funzione Lambda richiamata da Gateway Amazon API che analizza una tabella Amazon DynamoDB per le ricorrenze di lavoro e utilizza Amazon Simple Notification Service (Amazon SNS) per inviare un messaggio di testo ai dipendenti che si congratula con loro alla data dell'anniversario di un anno.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su. GitHub

Questo esempio è anche disponibile nella Guida per lo sviluppatore di AWS SDK per JavaScript v3 .

Servizi utilizzati in questo esempio

- Gateway API

- DynamoDB
- Lambda
- Amazon SNS

Python

SDK per Python (Boto3)

L'esempio mostra come creare e utilizzare una REST API di Gateway Amazon API destinata a una funzione AWS Lambda . Il gestore Lambda dimostra come definire percorsi in base ai metodi HTTP, come ottenere dati dalla stringa, dall'intestazione e dal corpo della query e come restituire una risposta JSON.

- Distribuire una funzione Lambda.
- Creare una REST API di API Gateway.
- Creare una risorsa REST destinata alla funzione Lambda.
- Concedere l'autorizzazione affinché l'API Gateway richiami la funzione Lambda.
- Utilizza il pacchetto Richieste per inviare le richieste alla REST API.
- Eliminare tutte le risorse create durante la demo.

Questo esempio è visualizzato al meglio su GitHub. Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, vedi l'esempio completo su GitHub.

Servizi utilizzati in questo esempio

- Gateway API
- DynamoDB
- Lambda
- Amazon SNS

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUtilizzo di DynamoDB con un SDK AWS. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo di Step Functions per richiamare le funzioni Lambda

Il seguente esempio di codice mostra come creare una macchina a AWS Step Functions stati che richiama AWS Lambda funzioni in sequenza.

Java

SDK per Java 2.x

Mostra come creare un flusso di lavoro AWS serverless utilizzando AWS Step Functions and. AWS SDK for Java 2.x Ogni fase del flusso di lavoro viene implementata utilizzando una AWS Lambda funzione.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, vedi l'esempio completo su GitHub.

Servizi utilizzati in questo esempio

- DynamoDB
- Lambda
- Amazon SES
- Step Functions

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUtilizzo di DynamoDB con un SDK AWS. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Usa un modello di documento per DynamoDB utilizzando un SDK AWS

Il seguente esempio di codice mostra come eseguire operazioni di creazione, lettura, aggiornamento ed eliminazione (CRUD) e batch utilizzando un modello di documento per DynamoDB e un SDK. AWS

Per ulteriori informazioni, consulta Modello di documento.

.NET

SDK per .NET

> **Note**
>
> C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Esecuzione di operazioni CRUD utilizzando un modello di 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);
            }
        }
    }
```

Esecuzione di operazioni di scrittura in batch utilizzando un modello di 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();
        }
    }
```

Scansione di una tabella utilizzando un modello di 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}");
            }
        }
    }
```

Esecuzione di query e scansione di una tabella utilizzando un modello di 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}");
            }
        }
    }
```

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUtilizzo di DynamoDB con un SDK AWS. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizza un modello di persistenza degli oggetti di alto livello per DynamoDB utilizzando un SDK AWS

Il seguente esempio di codice mostra come eseguire operazioni di creazione, lettura, aggiornamento ed eliminazione (CRUD) e in batch utilizzando un modello di persistenza a oggetti per DynamoDB e un SDK. AWS

Per ulteriori informazioni, consulta Modello di persistenza degli oggetti.

.NET

SDK per .NET

> **Note**
>
> C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Esecuzione di operazioni CRUD utilizzando un modello di persistenza degli oggetti di alto livello.

```
    /// <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");
            }
        }
    }
```

Esecuzione di operazioni di scrittura in batch utilizzando un modello di persistenza degli oggetti di alto livello.

```
    /// <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);
        }
    }
```

Mappatura dei dati arbitrari su una tabella utilizzando un modello di persistenza degli oggetti di alto livello.

```
    /// <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);
        }
    }
```

Esecuzione di query e scansione di una tabella utilizzando un modello di persistenza degli oggetti di alto livello.

```
    /// <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}");
            }
        }
    }
```

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUtilizzo di DynamoDB con un SDK AWS. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Usa le operazioni di conteggio atomico in DynamoDB con un SDK AWS

Gli esempi di codice seguenti mostrano come utilizzare le operazioni dei contatori atomici in DynamoDB.

- Incrementa i contatori in modo atomico utilizzando le operazioni ADD e SET.
- Incrementa in modo sicuro i contatori che potrebbero non esistere.
- Implementa un blocco ottimistico per le operazioni di sportello.

Java

SDK per Java 2.x

Dimostra le operazioni di conteggio atomico utilizzando. 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;
    }
```

Esempio di utilizzo delle operazioni di conteggio atomico 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();
        }
    }
```

- Per i dettagli sull'API, vedere UpdateItemin AWS SDK for Java 2.x API Reference.

JavaScript

SDK per JavaScript (v3)

Dimostra le controoperazioni atomiche utilizzando. AWS SDK per 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;
}
```

Esempio di utilizzo delle operazioni di conteggio atomico con. AWS SDK per 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);
  }
}
```

- Per i dettagli sull'API, vedere UpdateItemin AWS SDK per JavaScript API Reference.

Python

SDK per Python (Boto3)

Dimostra le controoperazioni atomiche utilizzando 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
```

Esempio di utilizzo delle operazioni di conteggio atomico 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")
```

- Per i dettagli sull'API, consulta UpdateItem AWSSDK for Python (Boto3) API Reference.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Utilizzo di DynamoDB con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Usa operazioni condizionali in DynamoDB con un SDK AWS

Gli esempi di codice seguenti mostrano come utilizzare le operazioni condizionali in DynamoDB.

- Implementa scritture condizionali per evitare la sovrascrittura dei dati.
- Utilizza le espressioni condizionali per applicare le regole aziendali.
- Gestisci con garbo gli errori dei controlli condizionali.

Java

SDK per Java 2.x

Dimostra le operazioni condizionali utilizzando. 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;
    }
```

Esempio di utilizzo di operazioni condizionali 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();
```

```
        }
    }
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento delle API AWS SDK for Java 2.x .
  - DeleteItem
  - PutItem
  - UpdateItem

JavaScript

SDK per JavaScript (v3)

Dimostra le operazioni condizionali utilizzando. AWS SDK per 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;
}
```

Esempio di utilizzo di operazioni condizionali con. AWS SDK per 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);
  }
}
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento delle API AWS SDK per JavaScript .
  - DeleteItem
  - PutItem
  - UpdateItem

Python

SDK per Python (Boto3)

Dimostra le operazioni condizionali utilizzando. 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
```

Esempio di utilizzo di operazioni condizionali 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"
    )
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento delle API SDK AWS per Python (Boto3).
  - DeleteItem
  - PutItem
  - UpdateItem

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Utilizzo di DynamoDB con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Usa i nomi degli attributi di espressione in DynamoDB con un SDK AWS

Gli esempi di codice seguenti mostrano come utilizzare i nomi degli attributi di espressione in DynamoDB.

- Lavora con parole riservate nelle espressioni DynamoDB.
- Usa i segnaposto per i nomi degli attributi di espressione.
- Gestisci i caratteri speciali nei nomi degli attributi.

Java

SDK per Java 2.x

Dimostra i nomi degli attributi di espressione utilizzando 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);
    }
```

Esempio di utilizzo dei nomi degli attributi di espressione 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();
        }
    }
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento delle API AWS SDK for Java 2.x .
  - Query
  - UpdateItem

JavaScript

SDK per JavaScript (v3)

Dimostra i nomi degli attributi di espressione utilizzando. AWS SDK per 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;
}
```

Esempio di utilizzo dei nomi degli attributi di espressione con AWS SDK per 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);
  }
}
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento delle API AWS SDK per JavaScript .
  - Query
  - UpdateItem

Python

SDK per Python (Boto3)

Dimostra i nomi degli attributi di espressione utilizzando 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
```

Esempio di utilizzo dei nomi degli attributi di espressione 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")
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento delle API SDK AWS per Python (Boto3).
  - Query
  - UpdateItem

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUtilizzo di DynamoDB con un SDK AWS. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo degli eventi pianificati per richiamare una funzione Lambda

I seguenti esempi di codice mostrano come creare una AWS Lambda funzione richiamata da un evento EventBridge pianificato di Amazon.

Java

SDK per Java 2.x

Mostra come creare un evento EventBridge pianificato da Amazon che richiami una AWS Lambda funzione. Configura EventBridge per utilizzare un'espressione cron per pianificare quando viene richiamata la funzione Lambda. In questo esempio, viene creata una funzione Lambda utilizzando l'API di runtime Lambda Java. Questo esempio richiama diversi AWS servizi per eseguire un caso d'uso specifico. Questo esempio dimostra come creare un'app che invia un messaggio di testo via mobile ai tuoi dipendenti che si congratula con loro alla data dell'anniversario di un anno.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su. GitHub

Servizi utilizzati in questo esempio

- CloudWatch Registri

- DynamoDB
- EventBridge
- Lambda
- Amazon SNS

JavaScript

SDK per JavaScript (v3)

Mostra come creare un evento EventBridge pianificato da Amazon che richiami una AWS Lambda funzione. Configura EventBridge per utilizzare un'espressione cron per pianificare quando viene richiamata la funzione Lambda. In questo esempio, crei una funzione Lambda utilizzando l'API JavaScript Lambda runtime. Questo esempio richiama diversi AWS servizi per eseguire un caso d'uso specifico. Questo esempio dimostra come creare un'app che invia un messaggio di testo via mobile ai tuoi dipendenti che si congratula con loro alla data dell'anniversario di un anno.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su. GitHub

Questo esempio è anche disponibile nella Guida per lo sviluppatore di AWS SDK per JavaScript v3 .

Servizi utilizzati in questo esempio

- CloudWatch Registri
- DynamoDB
- EventBridge
- Lambda
- Amazon SNS

Python

SDK per Python (Boto3)

Questo esempio mostra come registrare una AWS Lambda funzione come destinazione di un EventBridge evento Amazon pianificato. Il gestore Lambda scrive un messaggio intuitivo

e i dati completi dell'evento su Amazon CloudWatch Logs per recuperarli in un secondo momento.

- Distribuzione di una funzione Lambda.
- Crea un evento EventBridge pianificato e rende la funzione Lambda la destinazione.
- Concede il permesso di EventBridge invocare la funzione Lambda.
- Stampa i dati più recenti dai CloudWatch registri per mostrare il risultato delle chiamate pianificate.
- Elimina tutte le risorse create durante la demo.

Questo esempio è visualizzato al meglio su. GitHub Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, vedi l'esempio completo su GitHub.

Servizi utilizzati in questo esempio

- CloudWatch Registri
- DynamoDB
- EventBridge
- Lambda
- Amazon SNS

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Utilizzo di DynamoDB con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Lavora con gli indici secondari locali di DynamoDB utilizzando v2 AWS Command Line Interface

Nell'esempio di codice seguente viene mostrato come creare ed eseguire query su tabelle con indici secondari locali.

- Creazione di una tabella con un indice secondario locale (LSI).
- Crea una tabella multipla LSIs con diversi tipi di proiezione.
- Interroga i dati utilizzando LSIs.

Bash

AWS CLI con lo script Bash

Per creare una tabella con un LSI.

```
# 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
```

Crea una tabella con più LSIs di una tabella.

```
# 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
```

Interroga i dati utilizzando LSIs.

```
# 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"}
    }'
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento dei comandi AWS CLI .
  - CreateTable
  - Query

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUtilizzo di DynamoDB con un SDK AWS. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Lavora con DynamoDB Streams e usa v2 Time-to-Live AWS Command Line Interface

Il seguente esempio di codice mostra come gestire i flussi e le funzionalità di DynamoDB. Time-to-Live

- Per creare una tabella con i flussi abilitati.
- Descrizione dei flussi.
- Creazione di una funzione Lambda per l'elaborazione dei flussi.
- Abilitazione di TTL in una tabella.
- Per aggiungere elementi con attributi TTL.
- Per descrivere le impostazioni del TTL.

Bash

AWS CLI con lo script Bash

Per creare una tabella con i flussi abilitati.

```
# 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
```

Per descrivere i flussi.

```
# 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
```

Per creare una funzione Lambda per i flussi.

```
# 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"
```

Per abilitare il TTL in una tabella.

```
# 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"
```

Per aggiungere elementi con attributi 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'"}
    }'
```

Per descrivere le impostazioni del TTL.

```
# Describe TTL settings for a table
aws dynamodb describe-time-to-live \
    --table-name TTLDemo
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento dei comandi AWS CLI .
  - AttachRolePolicy
  - CreateRole
  - CreateTable
  - DescribeTable
  - DescribeTimeToLive
  - PutItem
  - UpdateTimeToLive

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Utilizzo di DynamoDB con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Lavora con le tabelle globali di DynamoDB e la replica multiregione con eventuale coerenza (MREC) utilizzando v2 AWS Command Line Interface

Nell'esempio di codice seguente viene mostrato come creare e gestire le tabelle globali DynamoDB con replica multi-Regione con coerenza possibile (MREC).

- Creazione di una tabella con replica multi-Regione (MREC).
- Inserimento e recupero di elementi dalle tabelle di replica.
- Rimuovi le repliche one-by-one.
- Per eseguire la pulizia mediante l'eliminazione delle tabelle.

Bash

AWS CLI con lo script Bash

Per creare una tabella con replica multi-Regione.

```
# 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
```

Per descrivere la tabella multi-Regione.

```
# 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}}'
```

Per inserire elementi in una tabella di replica.

```
# 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
```

Per ottenere elementi dalle tabelle di replica.

```
# 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
```

Per rimuovere le repliche.

```
# 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
```

Per eseguire la pulizia mediante l'eliminazione delle tabelle.

```
# Clean up: Delete the primary table
aws dynamodb delete-table --table-name MusicTable --region us-east-2

echo "Global table demonstration complete."
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento dei comandi AWS CLI .
  - CreateTable
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - UpdateTable

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Utilizzo di DynamoDB con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Lavora con il tagging delle risorse DynamoDB utilizzando la versione 2 AWS Command Line Interface

Nell'esempio di codice seguente viene mostrato come gestire i tag per le risorse DynamoDB.

- Creazione di una tabella con i tag.
- Elenca i tag associati a una risorsa.
- Aggiunta di tag a una risorsa.
- Rimozione di tag da una risorsa.
- Per filtrare le tabelle in base ai tag.

Bash

AWS CLI con lo script Bash

Creazione di una tabella con i tag.

```
# 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
```

Elenca i tag associati a una risorsa.

```
# 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
```

Aggiunta di tag a una risorsa.

```
# Add tags to an existing table
aws dynamodb tag-resource \
    --resource-arn $TABLE_ARN \
    --tags \
        Key=CostCenter,Value=12345 \
        Key=BackupSchedule,Value=Daily
```

Rimozione di tag da una risorsa.

```
# Remove tags from a table
aws dynamodb untag-resource \
    --resource-arn $TABLE_ARN \
    --tag-keys Owner BackupSchedule
```

Per filtrare le tabelle in base ai tag.

```
# 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
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento dei comandi AWS CLI .
  - CreateTable
  - ListTagsOfResource
  - TagResource
  - UntagResource

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Utilizzo di DynamoDB con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Lavora con la crittografia delle tabelle DynamoDB utilizzando la versione 2 AWS Command Line Interface

L'esempio di codice seguente mostra come gestire le opzioni di crittografia per le tabelle DynamoDB.

- Creazione di una tabella con la crittografia predefinita.
- Creazione di una tabella con una chiave gestita dal cliente (CMK).
- Aggiornamento delle impostazioni di crittografia delle tabelle.
- Descrizione della crittografia delle tabelle.

Bash

AWS CLI con lo script Bash

Per creare una tabella con la crittografia predefinita.

```
# 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
```

Per creare una tabella con una CMK.

```
# 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
```

Per aggiornare la crittografia delle tabelle.

```
# 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
```

Per descrivere la crittografia delle tabelle.

```
# Describe the table to see encryption settings
aws dynamodb describe-table \
    --table-name CustomerData \
    --query "Table.SSEDescription"
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento dei comandi AWS CLI .
  - CreateKey
  - CreateTable
  - DescribeTable
  - UpdateTable

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Utilizzo di DynamoDB con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Esempi serverless per DynamoDB

I seguenti esempi di codice mostrano come utilizzare DynamoDB con. AWS SDKs

Esempi

- Richiamare una funzione Lambda da un trigger DynamoDB
- Segnalazione di errori di elementi batch per funzioni Lambda con un trigger DynamoDB

Richiamare una funzione Lambda da un trigger DynamoDB

I seguenti esempi di codice spiegano come implementare una funzione Lambda che riceve un evento attivato dalla ricezione di record da un flusso DynamoDB. La funzione recupera il payload DocumentDB e registra il contenuto del record.

.NET

SDK per .NET

> **Note**
>
> C'è altro su. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento DynamoDB con Lambda tramite .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 per Go V2

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento DynamoDB con Lambda tramite 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 per Java 2.x

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento DynamoDB con Lambda tramite 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 per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Consumo di un evento DynamoDB con Lambda utilizzando. 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 di un evento DynamoDB con Lambda utilizzando. 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 per PHP

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento DynamoDB con Lambda tramite 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 per Python (Boto3)

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento DynamoDB con Lambda tramite 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 per Ruby

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento DynamoDB con Lambda tramite 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 per Rust

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento DynamoDB con Lambda tramite 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(())

}
```

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUtilizzo di DynamoDB con un SDK AWS. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Segnalazione di errori di elementi batch per funzioni Lambda con un trigger DynamoDB

I seguenti esempi di codice spiegano come implementare una risposta batch parziale per funzioni Lambda che ricevono eventi da un flusso DynamoDB. La funzione riporta gli errori degli elementi batch nella risposta, segnalando a Lambda di riprovare tali messaggi in un secondo momento.

.NET

SDK per .NET

> **Note**
>
> C'è altro su. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Segnalazione di errori di elementi batch di DynamoDB con Lambda tramite .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 per Go V2

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Segnalazione di errori di elementi batch di DynamoDB con Lambda tramite 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 per Java 2.x

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Segnalazione di errori di elementi batch di DynamoDB con Lambda tramite 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 per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Segnalazione degli errori degli elementi batch di DynamoDB utilizzando Lambda. 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: [] };
};
```

Segnalazione degli errori degli elementi batch di DynamoDB utilizzando Lambda. 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 per PHP

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Segnalazione di errori di elementi batch di DynamoDB con Lambda tramite 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 per Python (Boto3)

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Segnalazione di errori di elementi batch di DynamoDB con Lambda tramite 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 per Ruby

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Segnalazione di errori di elementi batch di DynamoDB con Lambda tramite 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 per Rust

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Segnalazione di errori di elementi batch di DynamoDB con Lambda tramite 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
}
```

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUtilizzo di DynamoDB con un SDK AWS. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

AWS contributi della community per DynamoDB

AWS i contributi della community sono esempi che sono stati creati e gestiti da più team all'interno AWS. Per fornire un feedback, utilizza il meccanismo disponibile nei repository collegati.

Esempi

- Creare e testare un’applicazione serverless

Creare e testare un’applicazione serverless

Gli esempi di codice seguenti mostrano come creare e testare un’applicazione serverless utilizzando Gateway API con Lambda e DynamoDB

.NET

SDK per .NET

Mostra come creare e testare un’applicazione serverless composta da Gateway API con Lambda e DynamoDB utilizzando l’SDK per .NET.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, consulta l'esempio completo su GitHub.

Servizi utilizzati in questo esempio

- Gateway API
- DynamoDB
- Lambda

Go

SDK per Go V2

Mostra come creare e testare un’applicazione serverless composta da Gateway API con Lambda e DynamoDB utilizzando l’SDK per Go.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su GitHub.

Servizi utilizzati in questo esempio

- Gateway API
- DynamoDB
- Lambda

Java

SDK per Java 2.x

Mostra come creare e testare un'applicazione serverless composta da Gateway API con Lambda e DynamoDB utilizzando l'SDK per Java.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su GitHub.

Servizi utilizzati in questo esempio

- Gateway API
- DynamoDB
- Lambda

Rust

SDK per Rust

Mostra come creare e testare un'applicazione serverless composta da Gateway API con Lambda e DynamoDB utilizzando l'SDK per Rust.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su GitHub.

Servizi utilizzati in questo esempio

- Gateway API
- DynamoDB
- Lambda

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUtilizzo di DynamoDB con un SDK AWS. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Sicurezza e conformità in Amazon DynamoDB

La sicurezza del cloud AWS è la massima priorità. In qualità di AWS cliente, puoi beneficiare di un data center e di un'architettura di rete progettati per soddisfare i requisiti delle organizzazioni più sensibili alla sicurezza.

La sicurezza è una responsabilità condivisa tra AWS te e te. Il modello di responsabilità condivisa descrive questo come sicurezza del cloud e sicurezza nel cloud:

- Sicurezza del cloud: AWS è responsabile della protezione dell'infrastruttura che gestisce AWS i servizi nel AWS cloud. AWS ti fornisce anche servizi che puoi utilizzare in modo sicuro. L'efficacia della nostra sicurezza è regolarmente testata e verificata da revisori di terze parti come parte dei programmi di conformitàAWS. Per ulteriori informazioni sui programmi di conformità che si applicano a DynamoDB, consulta Servizi AWS coperti dal programma di compliance.
- Sicurezza nel cloud: la tua responsabilità è determinata dal AWS servizio che utilizzi. L'utente è anche responsabile per altri fattori, tra cui la riservatezza dei dati, i requisiti dell'azienda e leggi e normative applicabili.

Questa documentazione consente di comprendere come applicare il modello di responsabilità condivisa quando si usa DynamoDB. Nei seguenti argomenti viene descritto come configurare DynamoDB per soddisfare gli obiettivi di sicurezza e conformità. Imparerai anche come utilizzare altri AWS servizi che possono aiutarti a monitorare e proteggere le tue risorse DynamoDB.

Argomenti

- AWS politiche gestite per Amazon DynamoDB
- Utilizzo di policy basate su risorse per DynamoDB
- Utilizzo del controllo degli accessi basato su attributi con DynamoDB
- Protezione dei dati in DynamoDB
- AWS Identity and Access Management (IAM) e DynamoDB
- Convalida della conformità per settore per DynamoDB
- Resilienza e ripristino di emergenza in Amazon DynamoDB
- Sicurezza dell'infrastruttura in Amazon DynamoDB
- AWS PrivateLink per DynamoDB
- Analisi della configurazione e delle vulnerabilità in Amazon DynamoDB

- Best practice di sicurezza per Amazon DynamoDB

AWS politiche gestite per Amazon DynamoDB

DynamoDB AWS utilizza policy gestite per definire una serie di autorizzazioni necessarie al servizio per eseguire azioni specifiche. DynamoDB mantiene e aggiorna AWS le sue policy gestite. Non è possibile modificare le autorizzazioni nelle AWS politiche gestite. Per ulteriori informazioni sulle policy AWS gestite, consulta le policy AWS gestite nella IAM User Guide.

DynamoDB può occasionalmente aggiungere autorizzazioni aggiuntive a AWS una policy gestita per supportare nuove funzionalità. Questo tipo di aggiornamento interessa tutte le identità (utenti, gruppi e ruoli) a cui è collegata la policy. È molto probabile che una policy AWS gestita venga aggiornata quando viene lanciata una nuova funzionalità o quando diventano disponibili nuove operazioni. DynamoDB non rimuoverà le autorizzazioni da AWS una policy gestita, quindi gli aggiornamenti delle policy non comprometteranno le autorizzazioni esistenti. Per un elenco completo delle politiche gestite, consulta le politiche AWS gestite.AWS

AWS politica gestita: Dynamo DBReplication ServiceRolePolicy

Non è possibile allegare la policy `DynamoDBReplicationServiceRolePolicy` alle entità IAM. Questa policy è associata a un ruolo collegato ai servizi che consente a DynamoDB di eseguire operazioni per conto dell'utente. Per ulteriori informazioni, consulta Uso di IAM con le tabelle globali.

Questa policy concede autorizzazioni che consentono al ruolo collegato ai servizi di eseguire la replica dei dati tra repliche di tabelle globali. Concede inoltre le autorizzazioni amministrative per gestire le repliche delle tabelle globali per conto dell'utente.

Dettagli delle autorizzazioni

Questa policy concede le seguenti autorizzazioni:

- `dynamodb`: consente di eseguire la replica dei dati e gestire le repliche delle tabelle.
- `application-autoscaling`: consente di recuperare e gestire le impostazioni di Auto Scaling delle tabelle
- `account`: consente di recuperare lo stato della regione per valutare l'accessibilità della replica.
- `iam`: consente di creare il ruolo collegato al servizio per Application Auto Scaling se il ruolo collegato al servizio non esiste già.

La definizione di questa policy gestita è disponibile qui.

AWS policy gestita: Access_v2 AmazonDynamo DBFull

La policy `AmazonDynamoDBFullAccess_v2` ad ambito ridotto concede privilegi di accesso specifici agli utenti. È possibile allegare la policy `AmazonDynamoDBFullAccess_v2` alle identità IAM. Questa policy concede l'accesso amministrativo alle risorse di Amazon DynamoDB e concede a un'identità IAM (ad esempio un utente, un gruppo o un ruolo) l'accesso alle risorse con cui DynamoDB è Servizi AWS integrato per utilizzare tutte le funzionalità di DynamoDB. L'utilizzo di questa policy consente di accedere a tutte le funzionalità di DynamoDB disponibili nella Console di gestione AWS.

Dettagli dell'autorizzazione

Questa policy include le seguenti autorizzazioni:

- `Amazon DynamoDB`
- `DynamoDB Accelerator`
- `AWS KMS`
- `AWS Resource Groups Tagging`
- `Lambda`
- `Application Auto Scaling`
- `CloudWatch`
- `Amazon Kinesis`
- `Amazon EC2`
- `IAM`

Per rivedere il formato della policy, consulta Access_v2. `JSON` AmazonDynamo DBFull

AWS politica gestita: AmazonDynamo DBRead OnlyAccess

È possibile allegare la policy `AmazonDynamoDBReadOnlyAccess` alle identità IAM.

Questa policy concede l'accesso in lettura ad Amazon DynamoDB.

Dettagli dell'autorizzazione

Questa policy include le seguenti autorizzazioni:

- `Amazon DynamoDB`: fornisce l'accesso in lettura ad Amazon DynamoDB.
- `Amazon DynamoDB Accelerator (DAX)`: fornisce l'accesso in lettura ad Amazon DynamoDB Accelerator (DAX).
- `Application Auto Scaling`: consente ai principali di visualizzare le configurazioni da Application Auto Scaling. Ciò è necessario per consentire agli utenti di visualizzare le policy di dimensionamento automatico collegate a una tabella.
- `CloudWatch`— Consente ai responsabili di visualizzare i dati metrici e gli allarmi configurati in. CloudWatch Ciò è necessario per consentire agli utenti di visualizzare le dimensioni fatturabili della tabella e gli CloudWatch allarmi configurati per una tabella.
- `AWS Data Pipeline`— Consente ai mandanti di visualizzare AWS Data Pipeline e gli oggetti associati.
- `Amazon EC2`— Consente ai mandanti di visualizzare Amazon EC2 VPCs, sottoreti e gruppi di sicurezza.
- `IAM`: consente ai principali di visualizzare i ruoli IAM.
- `AWS KMS`— Consente ai presidi di visualizzare le chiavi configurate in. AWS KMS Ciò è necessario per consentire agli utenti di visualizzare AWS KMS keys ciò che creano e gestiscono nel proprio account.
- `Amazon SNS`: consente ai principali di elencare gli argomenti relativi ad Amazon SNS e agli abbonamenti per argomento.
- `AWS Resource Groups`: consente ai principali di visualizzare i gruppi di risorse e le relative query.
- `AWS Resource Groups Tagging`: consente ai principali di elencare tutte le risorse con tag o con tag apposti in precedenza in una Regione.
- `Kinesis`: consente ai principali di visualizzare le descrizioni del flusso di dati Kinesis.
- `Amazon CloudWatch Contributor Insights`: consente ai principali di visualizzare i dati di serie temporali raccolti tramite le regole di Contributor Insights.

Per rivedere il `JSON` formato della politica, consulta AmazonDynamoDBReadOnlyAccess.

DynamoDB aggiorna le policy gestite AWS

Questa tabella mostra gli aggiornamenti alle politiche di gestione degli AWS accessi per DynamoDB.

Modifica	Descrizione	Data della modifica
AmazonDynamoDBFullAccess : obsoleta	Questa policy è stata sostituita da una policy ad ambito ridotto denominata AmazonDynamoDBFullAccess_v2 . Dopo aprile 2025 non sarà possibile associare la policy AmazonDynamoDBFullAccess a nuovi utenti, gruppi o ruoli. Per ulteriori informazioni, consulta AWS policy gestita: Access_v2 AmazonDynamo DBFull.	28 aprile 2025
AmazonDynamoDBReadOnlyAccess aggiornamento a una policy esistente	AmazonDynamoDBReadOnlyAccess ha aggiunto le autorizzazioni: dynamodb:GetAbacStatus e dynamodb:UpdateAbacStatus . Queste autorizzazioni ti consentono di visualizzare lo stato ABAC e abilitare ABAC per la tua Account AWS regione corrente.	18 novembre 2024
AmazonDynamoDBReadOnlyAccess aggiornamento a una policy esistente	AmazonDynamoDBReadOnlyAccess ha aggiunto l'autorizzazione dynamodb:GetResourcePolicy . Questa autorizzazione fornisce l'accesso alle policy in lettura basate su risorse collegate alle risorse DynamoDB.	20 marzo 2024
DynamoDBReplicationServiceRolePolicy aggiornam	DynamoDBReplicationServiceRolePolicy ha aggiunto l'autorizzazione dynamodb:GetResourcePolicy . Questa autorizzazione consente al ruolo	15 dicembre 2023

Modifica	Descrizione	Data della modifica
ento a una policy esistente	collegato al servizio di leggere le policy basate su risorse collegate alle risorse DynamoDB.	
`DynamoDBReplicationServiceRolePolicy` aggiornamento a una policy esistente	`DynamoDBReplicationServiceRolePolicy` ha aggiunto l'autorizzazione `account:ListRegions`. Questa autorizzazioni consente al ruolo collegato ai servizi di valutare l'accessibilità della replica.	10 maggio 2023
`DynamoDBReplicationServiceRolePolicy` aggiunta all'elenco delle policy gestite	Sono state aggiunte informazioni sulla policy gestita `DynamoDBReplicationServiceRolePolicy`, utilizzata dal ruolo collegato ai servizi delle tabelle globali DynamoDB.	10 maggio 2023
Le tabelle globali DynamoDB hanno iniziato a monitorare le modifiche	Le tabelle globali di DynamoDB hanno iniziato a tenere traccia delle modifiche per AWS le policy gestite.	10 maggio 2023

Utilizzo di policy basate su risorse per DynamoDB

DynamoDB supporta le policy basate su risorse per tabelle, indici e flussi. Le policy basate su risorse consentono di definire le autorizzazioni di accesso specificando chi ha accesso a ciascuna risorsa e le azioni eseguibili su ciascuna risorsa.

È possibile collegare una policy basata su risorse alle risorse DynamoDB, come una tabella o un flusso. In questa policy si specificano le autorizzazioni per i principali di Identity and Access Management (IAM) che possono eseguire azioni specifiche su queste risorse DynamoDB. Ad

esempio, la policy collegata a una tabella conterrà le autorizzazioni per l'accesso alla tabella e ai relativi indici. Di conseguenza, le policy basate su risorse possono aiutare a semplificare il controllo degli accessi per le tabelle, gli indici e i flussi di DynamoDB definendo le autorizzazioni a livello di risorsa. La dimensione massima di una policy che è possibile collegare a una risorsa DynamoDB è di 20 KB.

Un vantaggio significativo dell'utilizzo di policy basate su risorse consiste nel semplificare il controllo degli accessi multi-account per fornire l'accesso multi-account ai principali IAM in diversi Account AWS. Per ulteriori informazioni, consulta Policy basata su risorse per accesso multi-account.

Le policy basate su risorse supportano anche le integrazioni con lo strumento di analisi degli accessi esterni IAM Access Analyzer e le funzionalità di blocco dell'accesso pubblico (BPA, Block Public Access). Lo strumento di analisi degli accessi IAM segnala l'accesso multi-account a entità esterne specificate nelle policy basate su risorse. Fornisce inoltre visibilità per aiutare a perfezionare le autorizzazioni e conformarsi al principio del privilegio minimo. Il BPA aiuta a impedire l'accesso pubblico alle tabelle, agli indici e ai flussi di DynamoDB e viene abilitato automaticamente nei flussi di lavoro di creazione e modifica delle policy basate su risorse.

Argomenti

- Creazione di una tabella con una policy basata su risorse
- Collegamento di una policy a una tabella DynamoDB esistente
- Collegamento di una policy basata su risorse a un flusso DynamoDB
- Rimozione di una policy basata su risorse da una tabella DynamoDB
- Accesso multi-account con policy basate su risorse in DynamoDB
- Blocco dell'accesso pubblico con policy basate su risorse in DynamoDB
- Operazioni API DynamoDB supportate da policy basate su risorse
- Autorizzazione con policy basate sull'identità IAM e policy basate su risorse DynamoDB
- Esempi di policy basate su risorse per DynamoDB
- Considerazioni sulle policy basate su risorse per DynamoDB
- Best practice per policy basate su risorse di DynamoDB

Creazione di una tabella con una policy basata su risorse

Puoi aggiungere una policy basata sulle risorse mentre crei una tabella utilizzando la console DynamoDB, l'CreateTableAPI, AWS CLI l'SDK o un modello.AWS CloudFormation

AWS CLI

L'esempio seguente crea una tabella denominata utilizzando il comando. *MusicCollection* `create-table` AWS CLI Questo comando include anche il parametro `resource-policy` che aggiunge una policy basata su risorse alla tabella. Questo criterio consente *John* all'utente di eseguire le azioni RestoreTableToPointInTimeGetItem, e PutItemAPI sulla tabella.

Ricordati di sostituire il *italicized* testo con le informazioni specifiche della risorsa.

```
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\"
                }
            ]
        }"
```

Console di gestione AWS

1. Accedi Console di gestione AWS e apri la console DynamoDB all'indirizzo. https://console.aws.amazon.com/dynamodb/
2. Nella dashboard, seleziona Crea tabella.

3. In Dettagli tabella, inserisci il nome della tabella, la chiave di partizione e i dettagli della chiave di ordinamento.
4. In Impostazioni tabella, seleziona Personalizza impostazioni.
5. (Facoltativo) Specifica le opzioni per Classe tabella, Calcolatore di capacità, Impostazioni di capacità in scrittura/lettura, Indici secondari, Crittografia dei dati inattivi e Protezione da eliminazione.
6. In Policy basata sulle risorse, aggiungi una policy per definire le autorizzazioni di accesso per la tabella e i relativi indici. In questa policy si specificano gli utenti che hanno accesso a queste risorse e le azioni che è consentito eseguire su ciascuna risorsa. Per aggiungere una policy, procedi in uno dei seguenti modi:

   - Digitare o incollare un documento di policy JSON. Per i dettagli sul linguaggio della policy IAM, consulta Creating policies using the JSON editor nella Guida per l'utente di IAM.

     > **Tip**
     >
     > Per vedere esempi di policy basate su risorse nella Guida per gli sviluppatori di Amazon DynamoDB, seleziona Esempi di policy.

   - Seleziona Aggiungi nuova istruzione per aggiungere una nuova istruzione e inserisci le informazioni nei campi forniti. Ripeti questo passaggio per tutte le istruzioni che desideri aggiungere.

   > **Important**
   >
   > Assicurati di risolvere gli avvisi di sicurezza, gli errori o i suggerimenti prima di salvare la policy.

   Il seguente esempio di policy IAM consente *John* all'utente di eseguire le azioni RestoreTableToPointInTimeGetItem, e PutItemAPI sulla tabella. *MusicCollection*

   Ricordati di sostituire il *italicized* testo con le informazioni specifiche della risorsa.

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. (Facoltativo) Scegli Preview external access (Anteprima accesso esterno) nell'angolo in alto a destra per visualizzare in anteprima in che modo la nuova policy influisce sull'accesso pubblico e multi-account alla risorsa. Prima di salvare la policy, puoi verificare se introduce nuovi risultati di IAM Access Analyzer o risolve i risultati esistenti. Se non è presente uno strumento di analisi attivo, scegli Go to Access Analyzer (Passa a strumento analisi accessi) per creare uno strumento di analisi degli account in IAM Access Analyzer. Per ulteriori informazioni, consulta Preview access.
8. Scegliere Create table (Crea tabella).

AWS CloudFormation modello

Using the AWS::DynamoDB::Table resource

Il CloudFormation modello seguente crea una tabella con un flusso utilizzando la risorsa AWS: :DynamoDB: :Table. Questo modello include anche policy basate su risorse e collegate sia alla tabella che al flusso.

```
{
```

```
    "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

Il CloudFormation modello seguente crea una tabella con la GlobalTable risorsa AWS: :DynamoDB:: e allega una policy basata sulle risorse alla tabella e al relativo flusso.

```
{
    "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": "*"
                                    }]
                                }
                            }
                        }
                    }
```

```
                ]
            }
        }
    }
}
```

Collegamento di una policy a una tabella DynamoDB esistente

È possibile allegare una policy basata sulle risorse a una tabella esistente o modificare una policy esistente utilizzando la console DynamoDB, l'PutResourcePolicyAPI, l' AWS CLI SDK o un modello. AWSCloudFormation

AWS CLI esempio per allegare una nuova policy

Il seguente esempio di policy IAM utilizza il `put-resource-policy` AWS CLI comando per allegare una policy basata sulle risorse a una tabella esistente. Questo esempio consente all'utente di *John* eseguire le azioni GetItem, PutItemUpdateItem, e UpdateTableAPI su una tabella esistente denominata. *MusicCollection*

Ricordati di sostituire il *italicized* testo con le informazioni specifiche della risorsa.

```
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\"
                }
```

```
            ]
        }"
```

AWS CLI esempio di aggiornamento condizionale di una politica esistente

Per aggiornare in modo condizionale una policy basata su risorse esistente di una tabella, è possibile utilizzare il parametro opzionale `expected-revision-id`. L'esempio seguente aggiornerà la policy solo se esiste in DynamoDB e il suo ID di revisione corrente corrisponde al parametro `expected-revision-id` fornito.

```
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\"
                }
            ]
        }"
```

Console di gestione AWS

1. Accedi Console di gestione AWS e apri la console DynamoDB all'indirizzo. https://console.aws.amazon.com/dynamodb/
2. Dalla dashboard seleziona una tabella esistente.
3. Vai alla scheda Autorizzazioni e seleziona Crea una policy per le tabelle.
4. Nell'editor della policy basata su risorse, aggiungi la policy che desideri collegare e seleziona Crea policy.

Il seguente esempio di policy IAM consente *John* all'utente di eseguire le azioni GetItemPutItem, UpdateItem, e UpdateTableAPI su una tabella esistente denominata. *MusicCollection*

Ricordati di sostituire il *italicized* testo con le informazioni specifiche della risorsa.

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

Il seguente esempio di policy IAM utilizza il metodo `putResourcePolicy` per collegare una policy basata su risorse a una tabella esistente. Questa politica consente a un utente di eseguire l'azione GetItemAPI su una tabella esistente.

```
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.
                    allowed AWS Principal - 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" +
                "}";
    }
}
```

Collegamento di una policy basata su risorse a un flusso DynamoDB

È possibile allegare una policy basata su risorse allo stream di una tabella esistente o modificare una policy esistente utilizzando la console DynamoDB, l'PutResourcePolicyAPI, l' AWS CLI SDK o un modello. AWSCloudFormation

> **Note**
>
> Non è possibile allegare una policy a uno stream mentre lo si crea utilizzando o. CreateTableUpdateTable APIs Tuttavia, è possibile modificare o eliminare una policy dopo l'eliminazione di una tabella. È possibile anche modificare o eliminare la policy di un flusso disabilitato.

AWS CLI

Il seguente esempio di policy IAM utilizza il `put-resource-policy` AWS CLI comando per allegare una policy basata sulle risorse al flusso di una tabella denominata. *`MusicCollection`* Questo esempio consente all'utente *`John`* di eseguire le azioni GetRecordsGetShardIterator, e DescribeStreamAPI sullo stream.

Ricordati di sostituire il *`italicized`* testo con le informazioni specifiche della risorsa.

```
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\"
                }
            ]
        }"
```

Console di gestione AWS

1. Accedi Console di gestione AWS e apri la console DynamoDB all'indirizzo. https://console.aws.amazon.com/dynamodb/
2. Nella dashboard della console DynamoDB, seleziona Tabelle e quindi seleziona una tabella esistente.

   Assicurati che nella tabella selezionata i flussi siano attivati. Per informazioni su come attivare i flussi per una tabella, consulta Abilitazione di un flusso.
3. Scegli la scheda Autorizzazioni.
4. In Policy basata sulle risorse per il flusso attivo seleziona Crea una policy di flusso.
5. Nell'editor Policy basata sulle risorse, aggiungi una policy per definire le autorizzazioni di accesso per il flusso. In questa policy si specificano gli utenti che hanno accesso al flusso e le azioni che questi possono eseguirvi. Per aggiungere una policy, procedi in uno dei seguenti modi:

   - Digitare o incollare un documento di policy JSON. Per i dettagli sul linguaggio della policy IAM, consulta Creating policies using the JSON editor nella Guida per l'utente di IAM.

     > **Tip**
     >
     > Per vedere esempi di policy basate su risorse nella Guida per gli sviluppatori di Amazon DynamoDB, seleziona Esempi di policy.

   - Seleziona Aggiungi nuova istruzione per aggiungere una nuova istruzione e inserisci le informazioni nei campi forniti. Ripeti questo passaggio per tutte le istruzioni che desideri aggiungere.

   > **Important**
   >
   > Assicurati di risolvere gli avvisi di sicurezza, gli errori o i suggerimenti prima di salvare la policy.
6. (Facoltativo) Scegli Preview external access (Anteprima accesso esterno) nell'angolo in alto a destra per visualizzare in anteprima in che modo la nuova policy influisce sull'accesso pubblico e multi-account alla risorsa. Prima di salvare la policy, puoi verificare se introduce nuovi risultati di IAM Access Analyzer o risolve i risultati esistenti. Se non è presente uno strumento di

analisi attivo, scegli Go to Access Analyzer (Passa a strumento analisi accessi) per creare uno strumento di analisi degli account in IAM Access Analyzer. Per ulteriori informazioni, consulta Preview access.

7. Scegli Crea policy.

Il seguente esempio di policy IAM collega una policy basata sulle risorse al flusso di una tabella denominata. *MusicCollection* Questo esempio consente all'utente *John* di eseguire le azioni GetRecordsGetShardIterator, e DescribeStreamAPI sullo stream.

Ricordati di sostituire il *italicized* testo con le informazioni specifiche della risorsa.

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"
      ]
    }
  ]
}
```

Rimozione di una policy basata su risorse da una tabella DynamoDB

È possibile eliminare una policy basata sulle risorse da una tabella esistente utilizzando la console DynamoDB, l'DeleteResourcePolicyAPI, l' AWS CLI SDK o un modello. AWS CloudFormation

AWS CLI

L'esempio seguente utilizza il `delete-resource-policy` AWS CLI comando per rimuovere una politica basata sulle risorse da una tabella denominata. *`MusicCollection`*

Ricordati di sostituire il *`italicized`* testo con le informazioni specifiche della risorsa.

```
aws dynamodb delete-resource-policy \
    --resource-arn arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection
```

Console di gestione AWS

1. Accedi Console di gestione AWS e apri la console DynamoDB all'indirizzo. https://console.aws.amazon.com/dynamodb/
2. Nella dashboard della console DynamoDB, seleziona Tabelle e quindi seleziona una tabella esistente.
3. Seleziona Autorizzazioni.
4. Dal menu a discesa Gestisci policy, seleziona Elimina policy.
5. Nella finestra di dialogo Elimina la policy basata sulle risorse per la tabella, digita **`confirm`** per confermare l'azione di eliminazione.
6. Scegli Elimina.

Accesso multi-account con policy basate su risorse in DynamoDB

Utilizzando una policy basata su risorse, è possibile fornire l'accesso multi-account a risorse disponibili in diversi Account AWS. Tutti gli accessi tra account consentiti dalle politiche basate sulle risorse verranno segnalati tramite i risultati degli accessi esterni di IAM Access Analyzer se disponi di un analizzatore nella stessa risorsa. Regione AWS IAM Access Analyzer esegue controlli sulle policy per convalidare le policy rispetto alla grammatica delle policy IAM e alle best practice. Questi controlli generano risultati e forniscono raccomandazioni attuabili per aiutarti a creare policy funzionali e conformi alle best practice di sicurezza. È possibile visualizzare i risultati attivi dello strumento di analisi degli accessi IAM nella scheda Autorizzazioni della console DynamoDB.

Per ulteriori informazioni sulla convalida delle policy tramite lo strumento di analisi degli accessi IAM, consulta Convalida delle policy di Sistema di analisi degli accessi AWS IAM nella Guida per l'utente di IAM. Per visualizzare un elenco delle avvertenze, degli errori e dei suggerimenti restituiti da IAM Access Analyzer, consulta il Riferimento al controllo delle policy di IAM Access Analyzer.

Per concedere l'GetItemautorizzazione a un utente A nell'account A per accedere a una tabella B nell'account B, procedi nel seguente modo:

1. Collega alla tabella B una policy basata su risorse che conceda l'autorizzazione all'utente A per eseguire l'azione `GetItem`.
2. Collega una policy basata sull'identità all'utente A che gli conceda l'autorizzazione per eseguire l'azione `GetItem` sulla tabella B.

Utilizzando l'opzione Anteprima dell'accesso esterno disponibile nella console DynamoDB, è possibile visualizzare in anteprima in che modo la nuova policy influisce sull'accesso pubblico e multi-account alla risorsa. Prima di salvare la policy, puoi verificare se introduce nuovi risultati di IAM Access Analyzer o risolve i risultati esistenti. Se non è presente uno strumento di analisi attivo, scegli Go to Access Analyzer (Passa a strumento analisi accessi) per creare uno strumento di analisi degli account in IAM Access Analyzer. Per ulteriori informazioni, consulta Preview access.

Il parametro table name nel piano dati e nel piano di controllo di DynamoDB APIs accetta l'Amazon Resource Name (ARN) completo della tabella per supportare le operazioni tra account. Se si fornisce solo il parametro relativo al nome della tabella anziché un ARN completo, l'operazione API verrà eseguita sulla tabella dell'account a cui appartiene il richiedente. Per un esempio di una policy che utilizza l'accesso multi-account, consulta Policy basata su risorse per accesso multi-account.

L'account del proprietario della risorsa subirà un addebito anche quando un principale di un altro account sta leggendo o scrivendo sulla tabella DynamoDB nell'account del proprietario. Se la tabella prevede la velocità effettiva, la somma di tutte le richieste provenienti dagli account proprietari e dai richiedenti degli altri account determinerà se la richiesta verrà limitata (se la scalabilità automatica è disabilitata) o ridimensionata se la scalabilità automatica è abilitata. up/down

Le richieste verranno registrate nei CloudTrail registri degli account proprietario e richiedente in modo che ciascuno dei due account possa tenere traccia dell'account a cui ha avuto accesso a quali dati.

> **Note**
>
> L'accesso a più account al piano di controllo APIs prevede un limite inferiore di transazioni al secondo (TPS) di 500 richieste.

Blocco dell’accesso pubblico con policy basate su risorse in DynamoDB

Il blocco dell’accesso pubblico (BPA) è una funzionalità che identifica e impedisce il collegamento di policy basate su risorse che garantiscono l’accesso pubblico alle tabelle, agli indici o ai flussi di DynamoDB tra gli account Amazon Web Services (AWS). Con il BPA è possibile impedire l’accesso pubblico alle proprie risorse DynamoDB. Il BPA esegue controlli durante la creazione o la modifica di una policy basata su risorse e aiuta a migliorare il livello di sicurezza con DynamoDB.

Il BPA utilizza il ragionamento automatico per analizzare l’accesso concesso dalla policy basata su risorse e avvisa l’utente se tali autorizzazioni vengono rilevate al momento della gestione di una policy basata su risorse. L’analisi verifica l’accesso a tutte le istruzioni della policy basata su risorse, alle azioni e al set di chiavi di condizione utilizzate nelle policy.

> **Important**
>
> Il BPA aiuta a proteggere le risorse impedendo che venga concesso l’accesso pubblico attraverso le policy basate su risorse direttamente collegate alle risorse DynamoDB, come tabelle, indici e flussi. Oltre ad attivare il BPA, controlla attentamente le seguenti policy per verificare che non concedano l’accesso pubblico:
>
> - Politiche basate sull'identità collegate ai AWS principali associati (ad esempio, ruoli IAM)
> - Politiche basate sulle risorse collegate alle AWS risorse associate (ad esempio, chiavi (KMS)) AWS Key Management Service

È necessario assicurarsi che il principale non includa una voce * o che una delle chiavi di condizione specificate limiti l’accesso dei principali alla risorsa. Se la policy basata sulle risorse concede l'accesso pubblico alla tabella, agli indici o allo stream, Account AWS DynamoDB ti impedirà di creare o modificare la policy finché la specifica all'interno della policy non sarà corretta e considerata non pubblica.

È possibile rendere una policy non pubblica specificando uno o più principi all’interno del blocco del `Principal`. Il seguente esempio di policy basata su risorse blocca l’accesso pubblico specificando due principali.

```
{
  "Effect": "Allow",
  "Principal": {
```

```
    "AWS": [
      "123456789012",
      "111122223333"
    ]
  },
  "Action": "dynamodb:*",
  "Resource": "*"
}
```

Inoltre, le policy che limitano l'accesso specificando determinate chiavi di condizione non sono considerate pubbliche. Oltre alla valutazione del principale specificato nella policy basata su risorse, vengono utilizzate le seguenti chiavi di condizione attendibili per completare la valutazione di una policy basata su risorse per l'accesso non pubblico:

- `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`

Inoltre, affinché una policy basata su risorse non sia pubblica, i valori del nome della risorsa Amazon (ARN) e le chiavi di stringa non devono contenere caratteri jolly o variabili. Se la propria policy basata su risorse utilizza la chiave `aws:PrincipalIsAWSService`, è necessario assicurarsi di aver impostato il valore della chiave su true.

La policy seguente limita l'accesso all'utente `John` nell'account specificato. La condizione rende il `Principal` vincolato e non lo considera pubblico.

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": "*"
  },
  "Action": "dynamodb:*",
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "aws:PrincipalArn": "arn:aws:iam::123456789012:user/John"
    }
  }
}
```

L'esempio seguente di una policy basata su risorse non pubblica limita `sourceVPC` a utilizzare l'operatore `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"
          ]
        }
      }
    }
  ]
```

```
}
```

Operazioni API DynamoDB supportate da policy basate su risorse

Questo argomento elenca le operazioni API supportate dalle policy basate su risorse. Tuttavia, per l'accesso a più account, è possibile utilizzare solo un determinato set di APIs DynamoDB tramite policy basate sulle risorse. Non è possibile collegare policy basate su risorse a tipi di risorse come backup e importazioni. Le azioni IAM, che corrispondono al APIs funzionamento su questi tipi di risorse, sono escluse dalle azioni IAM supportate nelle politiche basate sulle risorse. Poiché gli amministratori delle tabelle configurano le impostazioni interne delle tabelle all'interno dello stesso account, ad esempio UpdateTimeToLivee APIs DisableKinesisStreamingDestination, non supportano l'accesso tra account tramite politiche basate sulle risorse.

Il piano dati e il piano di controllo APIs di DynamoDB che supportano l'accesso tra account supportano anche l'overload dei nomi delle tabelle, che consente di specificare l'ARN della tabella anziché il nome della tabella. È possibile specificare l'ARN della tabella nel `TableName` parametro di questi. APIs Tuttavia, non tutti APIs supportano l'accesso tra account.

Argomenti

- Operazioni API del piano dati
- Operazioni API PartiQL
- Operazioni API del piano di controllo (control-plane)
- Operazioni API delle tabelle globali versione 2019.11.21 (Corrente)
- Operazioni API delle tabelle globali versione 2017.11.29 (Legacy)
- Operazione API dei tag
- Operazioni API di backup e ripristino
- Operazioni API continue Backup/Restore (PITR)
- Operazioni API di Contributor Insights
- Operazioni API di esportazione
- Operazioni API di importazione
- Operazioni API del flusso di dati Amazon Kinesis
- Operazioni API di policy basate su risorse
- Time-to-Live Operazioni API
- Altre operazioni API

- Operazioni API per i flussi DynamoDB

Operazioni API del piano dati

La tabella seguente elenca il supporto a livello di API fornito dalle operazioni API del piano dati per le policy basate su risorse e l'accesso multi-account.

Piano dati - Tables/indexes APIs	Supporto per policy basate su risorse	Supporto multi-account
DeleteItem	Sì	Sì
GetItem	Sì	Sì
PutItem	Sì	Sì
Query	Sì	Sì
Scan	Sì	Sì
UpdateItem	Sì	Sì
TransactGetItems	Sì	Sì
TransactWriteItems	Sì	Sì
BatchGetItem	Sì	Sì
BatchWriteItem	Sì	Sì

Operazioni API PartiQL

La tabella seguente elenca il supporto a livello di API fornito dalle operazioni API PartiQL per le policy basate su risorse e l'accesso multi-account.

PartiQL APIs	Supporto per policy basate su risorse	Supporto multi-account
BatchExecuteStatement	Sì	No

PartiQL APIs	Supporto per policy basate su risorse	Supporto multi-account
ExecuteStatement	Sì	No
ExecuteTransaction	Sì	No

Operazioni API del piano di controllo (control-plane)

La tabella seguente elenca il supporto a livello di API fornito dalle operazioni API del piano di controllo (control-plane) per le policy basate su risorse e l'accesso multi-account.

Control Plane - Tabelle APIs	Supporto per policy basate su risorse	Supporto multi-account
CreateTable	No	No
DeleteTable	Sì	Sì
DescribeTable	Sì	Sì
UpdateTable	Sì	Sì

Operazioni API delle tabelle globali versione 2019.11.21 (Corrente)

La tabella seguente elenca il supporto a livello di API fornito dalle operazioni API delle Tabelle globali versione 2019.11.21 (Corrente) per le policy basate su risorse e l'accesso multi-account.

Tabelle globali versione 2019.11.21 (attuale) APIs	Supporto per policy basate su risorse	Supporto multi-account
DescribeTableReplicaAutoScaling	Sì	No
UpdateTableReplicaAutoScaling	Sì	No

Operazioni API delle tabelle globali versione 2017.11.29 (Legacy)

La tabella seguente elenca il supporto a livello di API fornito dalle operazioni API delle tabelle globali versione 2017.11.29 (Legacy) per le policy basate su risorse e l'accesso multi-account.

Tabelle globali versione 2017.11.29 (Legacy) APIs	Supporto per policy basate su risorse	Supporto multi-account
CreateGlobalTable	No	No
DescribeGlobalTable	No	No
DescribeGlobalTableSettings	No	No
ListGlobalTables	No	No
UpdateGlobalTable	No	No
UpdateGlobalTableSettings	No	No

Operazione API dei tag

La tabella seguente elenca il supporto a livello di API fornito dalle operazioni API relative ai tag per le policy basate su risorse e l'accesso multi-account.

Tag APIs	Supporto per policy basate su risorse	Supporto multi-account
ListTagsOfResource	Sì	Sì
TagResource	Sì	Sì
UntagResource	Sì	Sì

Operazioni API di backup e ripristino

La tabella seguente elenca il supporto a livello di API fornito dalle operazioni API relative al backup e al ripristino per le policy basate su risorse e l'accesso multi-account.

Backup e ripristino APIs	Supporto per policy basate su risorse	Supporto multi-account
CreateBackup	Sì	No
DescribeBackup	No	No
DeleteBackup	No	No
RestoreTableFromBackup	No	No

Operazioni API continue Backup/Restore (PITR)

La tabella seguente elenca il supporto a livello di API fornito dalle operazioni API relative a Continuous Backup/Restore (PITR) per le politiche basate sulle risorse e l'accesso tra account.

Continuo (PITR) Backup/Restore APIs	Supporto per policy basate su risorse	Supporto multi-account
DescribeContinuousBackups	Sì	No
RestoreTableToPointInTime	Sì	No
UpdateContinuousBackups	Sì	No

Operazioni API di Contributor Insights

La tabella seguente elenca il supporto a livello di API fornito dalle operazioni API relative a Continuous Backup/Restore (PITR) per le politiche basate sulle risorse e l'accesso tra account.

Approfondimenti per i contributori APIs	Supporto per policy basate su risorse	Supporto multi-account
DescribeContributorInsights	Sì	No
ListContributorInsights	No	No
UpdateContributorInsights	Sì	No

Operazioni API di esportazione

La tabella seguente elenca il supporto a livello di API fornito dalle operazioni API di esportazione per le policy basate su risorse e l'accesso multi-account.

Esportazione APIs	Supporto per policy basate su risorse	Supporto multi-account
DescribeExport	No	No
ExportTableToPointInTime	Sì	No
ListExports	No	No

Operazioni API di importazione

La tabella seguente elenca il supporto a livello di API fornito dalle operazioni API di esportazione per le policy basate su risorse e l'accesso multi-account.

Importazione APIs	Supporto per policy basate su risorse	Supporto multi-account
DescribeImport	No	No
ImportTable	No	No
ListImports	No	No

Operazioni API del flusso di dati Amazon Kinesis

La tabella seguente elenca il supporto a livello di API fornito dalle operazioni API del flusso di dati Kinesis per le policy basate su risorse e l'accesso multi-account.

Kinesis APIs	Supporto per policy basate su risorse	Supporto multi-account
DescribeKinesisStreamingDestination	Sì	No
DisableKinesisStreamingDestination	Sì	No
EnableKinesisStreamingDestination	Sì	No
UpdateKinesisStreamingDestination	Sì	No

Operazioni API di policy basate su risorse

La tabella seguente elenca il supporto a livello di API fornito dalle operazioni API di policy basate su risorse per le policy basate su risorse e l'accesso multi-account.

Politica basata sulle risorse APIs	Supporto per policy basate su risorse	Supporto multi-account
GetResourcePolicy	Sì	No
PutResourcePolicy	Sì	No
DeleteResourcePolicy	Sì	No

Time-to-Live Operazioni API

La tabella seguente elenca il supporto a livello di API fornito dalle operazioni API Time to Live (TTL) per le policy basate su risorse e l'accesso multi-account.

TTL APIs	Supporto per policy basate su risorse	Supporto multi-account
DescribeTimeToLive	Sì	No
UpdateTimeToLive	Sì	No

Altre operazioni API

La tabella seguente elenca il supporto a livello di API fornito da altre operazioni API varie per le policy basate su risorse e l'accesso multi-account.

Altro APIs	Supporto per policy basate su risorse	Supporto multi-account
DescribeLimits	No	No
DescribeEndpoints	No	No
ListBackups	No	No
ListTables	No	No

Operazioni API per i flussi DynamoDB

La tabella seguente elenca il supporto a livello di API di APIs DynamoDB Streams per le policy basate sulle risorse e l'accesso tra account.

Streams DynamoDB APIs	Supporto per policy basate su risorse	Supporto multi-account
DescribeStream	Sì	Sì
GetRecords	Sì	Sì
GetShardIterator	Sì	Sì
ListStreams	No	No

Autorizzazione con policy basate sull'identità IAM e policy basate su risorse DynamoDB

Le policy basate sull'identità sono collegate a un'identità, come utenti, gruppi di utenti e ruoli IAM. Si tratta di documenti di policy IAM che controllano le azioni autorizzate per l'identità, nonché le risorse e le condizioni di esecuzione. Le policy basate sull'identità possono essere policy gestite o in linea.

Le policy basate su risorse sono documenti di policy IAM che vengono collegate a una risorsa, come una tabella DynamoDB. Queste politiche concedono l'autorizzazione principale specificata per eseguire azioni specifiche su quella risorsa e definiscono a quali condizioni ciò si applica. Ad esempio, la policy basate su risorse per una tabella DynamoDB include anche l'indice associato alla tabella. Le policy basate su risorse sono policy inline. Non esistono policy basate su risorse gestite.

Per ulteriori informazioni su queste policy, consulta Identity-based policies and resource-based policies nella Guida per l'utente di IAM.

Se il principale IAM proviene dallo stesso account del proprietario della risorsa, una policy basate su risorse è sufficiente per specificare le autorizzazioni di accesso alla risorsa. È comunque possibile scegliere di disporre di una policy basata sull'identità IAM insieme a una policy basata su risorse. Per l'accesso multi-account, è necessario consentire esplicitamente l'accesso sia nelle policy basate sull'identità che in quelle basate su risorse, come specificato in Accesso multi-account con policy basate su risorse in DynamoDB. Quando si utilizzano entrambi i tipi di policy, una policy viene valutata come descritto in Determinazione se una richiesta è consentita o rifiutata in un account.

Esempi di policy basate su risorse per DynamoDB

Quando si specifica un ARN nel campo `Resource` di una policy basata su risorse, la policy ha effetto solo se l'ARN specificato corrisponde all'ARN della risorsa DynamoDB a cui è collegato.

> **Note**
>
> Ricordati di sostituire il *`italicized`* testo con le informazioni specifiche della risorsa.

Policy basata su risorse per una tabella

La seguente policy basata sulle risorse allegata a una tabella DynamoDB denominata*`MusicCollection`*, fornisce agli utenti IAM *`John`* e il *`Jane`* permesso di eseguire azioni sulla risorsa. GetItemBatchGetItem*`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"
        ]
    }
  ]
}
```

Policy basata su risorse per un flusso

La seguente policy basata sulle risorse allegata a un flusso DynamoDB denominato `2024-02-12T18:57:26.492` fornisce agli utenti IAM GetRecordsGetShardIterator, il *Jane* permesso di eseguire *John* e le azioni API sulla risorsa. DescribeStream`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"
      ]
    }
  ]
}
```

Politica basata su risorse per l'accesso all'esecuzione di tutte le azioni su risorse specifiche

Per consentire a un utente di eseguire tutte le azioni su una tabella e tutti gli indici associati a una tabella, è possibile utilizzare un carattere jolly (*) per rappresentare le azioni e le risorse associate alla tabella. L'utilizzo di un carattere jolly per le risorse consentirà all'utente di accedere alla tabella DynamoDB e a tutti gli indici associati, inclusi quelli che non sono ancora stati creati. Ad esempio, la seguente politica consentirà all'utente *John* di eseguire qualsiasi azione sulla *MusicCollection* tabella e su tutti i relativi indici, inclusi gli indici che verranno creati in 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"
      ]
    }
  ]
}
```

Policy basata su risorse per accesso multi-account

È possibile specificare le autorizzazioni per un'identità IAM multi-account per accedere alle risorse DynamoDB. Ad esempio, potrebbe essere necessario un utente di un account attendibile per accedere alla lettura del contenuto della tabella, a condizione che acceda solo a elementi e attributi specifici di tali elementi. La seguente politica consente l'accesso all'utente *John* da un Account AWS ID affidabile *111111111111* per accedere ai dati da una tabella nell'account *123456789012* utilizzando l'API. GetItem La politica garantisce che l'utente possa accedere solo agli elementi con una chiave primaria *Jane* e che l'utente possa recuperare solo gli attributi `Artist` e `SongTitle` nessun altro attributo.

> **Important**
>
> Se non si specifica la condizione `SPECIFIC_ATTRIBUTES`, si vedranno tutti gli attributi degli elementi restituiti.

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"
                }
            }
        }
    ]
}
```

Oltre alla precedente politica basata sulle risorse, *John* anche la politica basata sull'identità allegata all'utente deve consentire l'azione dell'`GetItem`API per il funzionamento dell'accesso tra account. Di seguito è riportato un esempio di policy basata sull'identità che è necessario allegare all'utente. *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"
                }
            }
        }
    ]
}
```

L'utente John può effettuare una `GetItem` richiesta specificando la tabella ARN nel parametro per `table-name` l'accesso alla *`MusicCollection`* tabella nell'account. *`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
```

Policy basata su risorse con condizioni di indirizzo IP

È possibile applicare una condizione per limitare gli indirizzi IP di origine, i cloud privati virtuali (VPCs) e gli endpoint VPC (VPCE). È possibile specificare le autorizzazioni in base agli indirizzi di origine della richiesta di origine. Ad esempio, si potrebbe voler consentire a un utente di accedere alle risorse DynamoDB solo se vi accede da un'origine IP specifica, come un endpoint VPN aziendale. Specifica questi indirizzi IP nell'istruzione `Condition`.

L'esempio seguente consente all'utente di *`John`* accedere a qualsiasi risorsa DynamoDB quando le IPs sorgenti sono e. `54.240.143.0/24` `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"
          ]
        }
      }
    }
  ]
}
```

Puoi anche negare tutti gli accessi alle risorse DynamoDB tranne quando l'origine è un endpoint VPC specifico, ad esempio. *vpce-1a2b3c4d*

> **Important**
>
> Quando si utilizza DAX con tabelle DynamoDB con policy di risorse basate su IP IPv6 in ambienti -only, è necessario configurare regole di accesso aggiuntive. Se la politica delle risorse limita l'accesso allo spazio degli IPv4 indirizzi `0.0.0.0/0` sulle tabelle, è necessario consentire l'accesso al ruolo IAM associato al cluster DAX. Aggiungi una `ArnNotEquals` condizione alla tua policy per garantire che DAX mantenga l'accesso alle tue tabelle DynamoDB. Per ulteriori informazioni, vedere 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"
        }
      }
    }
  ]
}
```

Policy basata su risorse che utilizza un ruolo IAM

È inoltre possibile specificare un ruolo di servizio IAM nella policy basata su risorse. Le entità IAM che assumono questo ruolo sono limitate dalle azioni consentite specificate per il ruolo e allo specifico set di risorse all'interno della policy basata su risorse.

L'esempio seguente consente a un'entità IAM di eseguire tutte le azioni DynamoDB sulle risorse *MusicCollection* e *MusicCollection* 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/*"
      ]
    }
  ]
}
```

Considerazioni sulle policy basate su risorse per DynamoDB

Quando si definiscono policy basate su risorse per risorse DynamoDB, si applicano le seguenti considerazioni:

Considerazioni generali

- La dimensione massima supportata per un documento di policy basata su risorse è di 20 KB. DynamoDB non calcola gli spazi vuoti nella determinazione delle dimensioni di una policy rispetto a tale limite.
- Gli aggiornamenti successivi a una policy per una determinata risorsa vengono bloccati per 15 secondi dopo un aggiornamento riuscito della policy per la stessa risorsa.
- Attualmente è possibile collegare una policy basata su risorse solo ai flussi esistenti, non a un flusso durante la sua creazione.

Considerazioni sulle tabelle globali

- Le policy basate su risorse non sono supportate per le repliche delle Tabelle globali versione 2017.11.29 (Legacy).
- All'interno di una policy basata su risorse, se l'azione per un ruolo collegato al servizio (SLR, Service-Linked Role) DynamoDB per replicare i dati per una tabella globale viene negata, l'aggiunta o l'eliminazione di una replica avrà esito negativo e genererà un errore.
- La GlobalTable risorsa AWS: :DynamoDB:: non supporta la creazione di una replica e l'aggiunta di una policy basata sulle risorse a tale replica nello stesso aggiornamento dello stack in regioni diverse dalla regione in cui viene distribuito l'aggiornamento dello stack.

Considerazioni relative a più account

- L'accesso tra account tramite politiche basate sulle risorse non supporta tabelle crittografate con chiavi gestite perché non è possibile concedere l'accesso tra account diversi alla politica KMS gestita AWS . AWS

CloudFormation considerazioni

- Le policy basate su risorse non supportano il rilevamento delle deviazioni. Se aggiorni una policy basata sulle risorse al di fuori del modello di AWS CloudFormation stack, dovrai aggiornare lo stack con le CloudFormation modifiche.
- Le policy basate su risorse non supportano modifiche fuori banda. Se aggiungi, aggiorni o elimini una politica all'esterno del CloudFormation modello, la modifica non verrà sovrascritta se non ci sono modifiche alla politica all'interno del modello.

  Ad esempio, si supponga che il modello contenga una policy basata su risorse che successivamente verrà aggiornata al di fuori del modello. Se non si apportano modifiche alla policy nel modello, la policy aggiornata in DynamoDB non verrà sincronizzata con la policy nel modello.

  Al contrario, si supponga che il modello non contenga una policy basata su risorse, ma che si aggiunga una policy esterna al modello. Questa policy non verrà rimossa da DynamoDB finché non la si aggiunge al modello. Durante l'aggiunta di una policy al modello e l'aggiornamento dello stack, la policy esistente in DynamoDB verrà aggiornata in modo che corrisponda a quella definita nel modello.

Best practice per policy basate su risorse di DynamoDB

Questo argomento descrive le best practice per definire le autorizzazioni di accesso per le risorse DynamoDB e le azioni consentite su tali risorse.

Semplificazione del controllo degli accessi alle risorse DynamoDB

Se AWS Identity and Access Management i principali che devono accedere a una risorsa DynamoDB fanno parte Account AWS dello stesso proprietario della risorsa, non è richiesta una policy basata sull'identità IAM per ogni principale. Sarà sufficiente una policy basata su risorse collegata alle risorse fornite. Questo tipo di configurazione semplifica il controllo degli accessi.

Protezione delle risorse DynamoDB con policy basate su risorse

Per tutte le tabelle e i flussi DynamoDB è possibile creare policy basate su risorse per imporre il controllo degli accessi a queste risorse. Le policy basate su risorse consentono di centralizzare le autorizzazioni a livello di risorsa, semplificare il controllo degli accessi a tabelle, indici e flussi di DynamoDB e ridurre il sovraccarico di gestione. Se non viene specificata alcuna policy basata su risorse per una tabella o un flusso, l'accesso alla tabella o al flusso verrà implicitamente negato, a meno che le policy basate sull'identità associate ai principali IAM non consentano l'accesso.

Assegna le autorizzazioni con privilegi minimi

Durante l'impostazione delle autorizzazioni con policy basate su risorse per risorse DynamoDB, concedi solo le autorizzazioni necessarie per eseguire un'operazione. Puoi farlo definendo le azioni che possono essere intraprese su risorse specifiche in condizioni specifiche, note anche come autorizzazioni con privilegi minimi. Potresti iniziare con autorizzazioni generiche mentre esplori le autorizzazioni necessarie per il tuo carico di lavoro o il caso d'uso. Man mano che il tuo caso d'uso matura, puoi lavorare per ridurre le autorizzazioni concesse per lavorare con il privilegio minimo.

Analisi dell'attività di accesso multi-account per generare policy con privilegi minimi

Lo strumento di analisi degli accessi IAM segnala l'accesso multi-account a entità esterne specificate nelle policy basate su risorse e fornisce visibilità per aiutare a perfezionare le autorizzazioni e conformarsi al privilegio minimo. Per ulteriori informazioni sulla generazione delle policy, consulta IAM Access Analyzer policy generation.

Utilizzo dello strumento di analisi degli accessi IAM per generare policy con privilegi minimi

Per concedere solo le autorizzazioni richieste per eseguire un'attività, puoi generare policy in funzione dell'attività di accesso che hai effettuato l'accesso in AWS CloudTrail. Lo strumento di analisi degli accessi IAM analizza i servizi e le azioni utilizzate dalle policy.

Utilizzo del controllo degli accessi basato su attributi con DynamoDB

Il controllo degli accessi basato sugli attributi (ABAC) è una strategia di autorizzazione che definisce le autorizzazioni di accesso in base alle condizioni dei tag nelle politiche basate sull'identità o in altre politiche, come le politiche basate sulle risorse e le politiche IAM dell'organizzazione. AWS È

possibile associare tag alle tabelle DynamoDB, che vengono poi valutate in base alle condizioni basate sui tag. Gli indici associati a una tabella ereditano i tag aggiunti alla tabella. È possibile aggiungere fino a 50 tag per ogni tabella DynamoDB. La dimensione massima supportata per tutti i tag in una tabella è 10 KB. Per ulteriori informazioni sull'assegnazione di tag alle risorse DynamoDB e sulle restrizioni relative ai tag, consulta Assegnazione di tag alle risorse in DynamoDB e the section called "Limitazioni dell'assegnazione di tag in DynamoDB".

Per ulteriori informazioni sull'utilizzo dei tag per controllare l'accesso alle AWS risorse, consulta i seguenti argomenti nella Guida per l'utente IAM:

- A cosa serve ABAC AWS
- Controllo dell'accesso alle AWS risorse tramite tag

Utilizzando l'ABAC è possibile applicare diversi livelli di accesso per i propri team e le proprie applicazioni per eseguire azioni sulle tabelle DynamoDB utilizzando un minor numero di policy. Per controllare gli accessi alle tabelle o agli indici DynamoDB è invece possibile specificare un tag nell'elemento Condition di una policy IAM. Queste condizioni determinano il livello di accesso di un principale, un utente o un ruolo IAM alle tabelle e agli indici DynamoDB. Quando un principale IAM effettua una richiesta di accesso a DynamoDB, i tag della risorsa e dell'identità vengono valutati in base alle condizioni dei tag nella policy IAM. Successivamente, la policy diventa effettiva solo se vengono soddisfatte le condizioni dei tag. Ciò consente di creare una policy IAM che risponda efficacemente a una delle seguenti condizioni:

- Consentire all'utente di gestire solo le risorse che hanno un tag con una chiave *X* e un valore *Y*.
- Negare l'accesso a tutti gli utenti alle risorse contrassegnate con una chiave *X*.

Ad esempio, è possibile creare una policy che consenta agli utenti di aggiornare una tabella solo se presenta la coppia chiave-valore del tag: `"environment": "staging"`. Puoi usare la chiave aws:ResourceTag condition per consentire o negare l'accesso a una tabella in base ai tag allegati a quella tabella.

Puoi includere condizioni basate sugli attributi durante la creazione della policy o in un secondo momento utilizzando Console di gestione AWS, AWS API, AWS Command Line Interface (AWS CLI), AWS SDK o. AWS CloudFormation

L'esempio seguente consente l'UpdateItemazione su una tabella denominata `MusicTable` se include una chiave di tag con il nome e il valore. `environment` `production`

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "dynamodb:UpdateItem"
      ],
      "Resource": "arn:aws:dynamodb:*:*:table/MusicTable",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/environment": "production"
        }
      }
    }
  ]
}
```

Argomenti

- Qual è il vantaggio di utilizzare l'ABAC?
- Chiavi di condizione per implementare l'ABAC con DynamoDB
- Considerazioni sull'utilizzo dell'ABAC con DynamoDB
- Abilitazione dell'ABAC in DynamoDB
- Utilizzo dell'ABAC con tabelle e indici DynamoDB
- Esempi di utilizzo dell'ABAC con tabelle e indici DynamoDB
- Risoluzione dei problemi più comuni dell'ABAC per le tabelle e gli indici DynamoDB

Qual è il vantaggio di utilizzare l'ABAC?

- Gestione delle policy più semplice: si utilizza un minor numero di policy perché non è necessario creare policy diverse per definire il livello di accesso per ogni principale IAM.
- Controllo scalabile degli accessi: scalare il controllo degli accessi è più facile con l'ABAC perché non è necessario aggiornare le policy IAM quando vengono create nuove risorse. È possibile utilizzare i tag per autorizzare l'accesso ai principali IAM che contengono tag corrispondenti ai

tag della risorsa. È possibile integrare nuovi principali IAM o risorse DynamoDB e applicare i tag appropriati per concedere automaticamente le autorizzazioni necessarie senza dover apportare modifiche alle policy.

- Gestione granulare delle autorizzazioni: è una best practice concedere il privilegio minimo quando si creano le policy. Utilizzando l'ABAC è possibile creare tag per il principale IAM e utilizzarli per concedere l'accesso ad azioni e risorse specifiche che corrispondono ai tag sul principale IAM.
- Allineamento con la directory aziendale: è possibile mappare i tag con gli attributi dei dipendenti esistenti presenti nella directory aziendale per allineare le policy di controllo degli accessi alla struttura organizzativa.

Chiavi di condizione per implementare l'ABAC con DynamoDB

È possibile utilizzare le seguenti chiavi di condizione nelle AWS policy per controllare il livello di accesso alle tabelle e agli indici DynamoDB:

- aws: ResourceTag /tag-key: Controlla l'accesso in base al fatto che la coppia chiave-valore del tag su una tabella o un indice DynamoDB corrisponda o meno alla chiave e al valore del tag in una policy. Questa chiave condizionale è rilevante per tutto ciò APIs che opera su una tabella o un indice esistente.

  Le condizioni `dynamodb:ResourceTag` vengono valutate come se non fosse stato assegnato alcun tag a una risorsa.
- aws: RequestTag /tag-key: consente di confrontare la coppia chiave-valore del tag passata nella richiesta con la coppia di tag specificata nella politica. Questa chiave condizionale è pertinente a APIs quella che contiene tag come parte del payload della richiesta. Questi APIs includono CreateTablee. TagResource
- aws: TagKeys: Confronta le chiavi dei tag in una richiesta con le chiavi specificate nella policy. Questa chiave condizionale è rilevante per APIs quella che contiene tag come parte del payload della richiesta. Questi APIs includono `CreateTableTagResource`, e`UntagResource`.

Considerazioni sull'utilizzo dell'ABAC con DynamoDB

Quando si utilizza l'ABAC con tabelle o indici DynamoDB, valgono le seguenti considerazioni:

- L'assegnazione di tag e l'ABAC non sono supportati per i flussi DynamoDB.

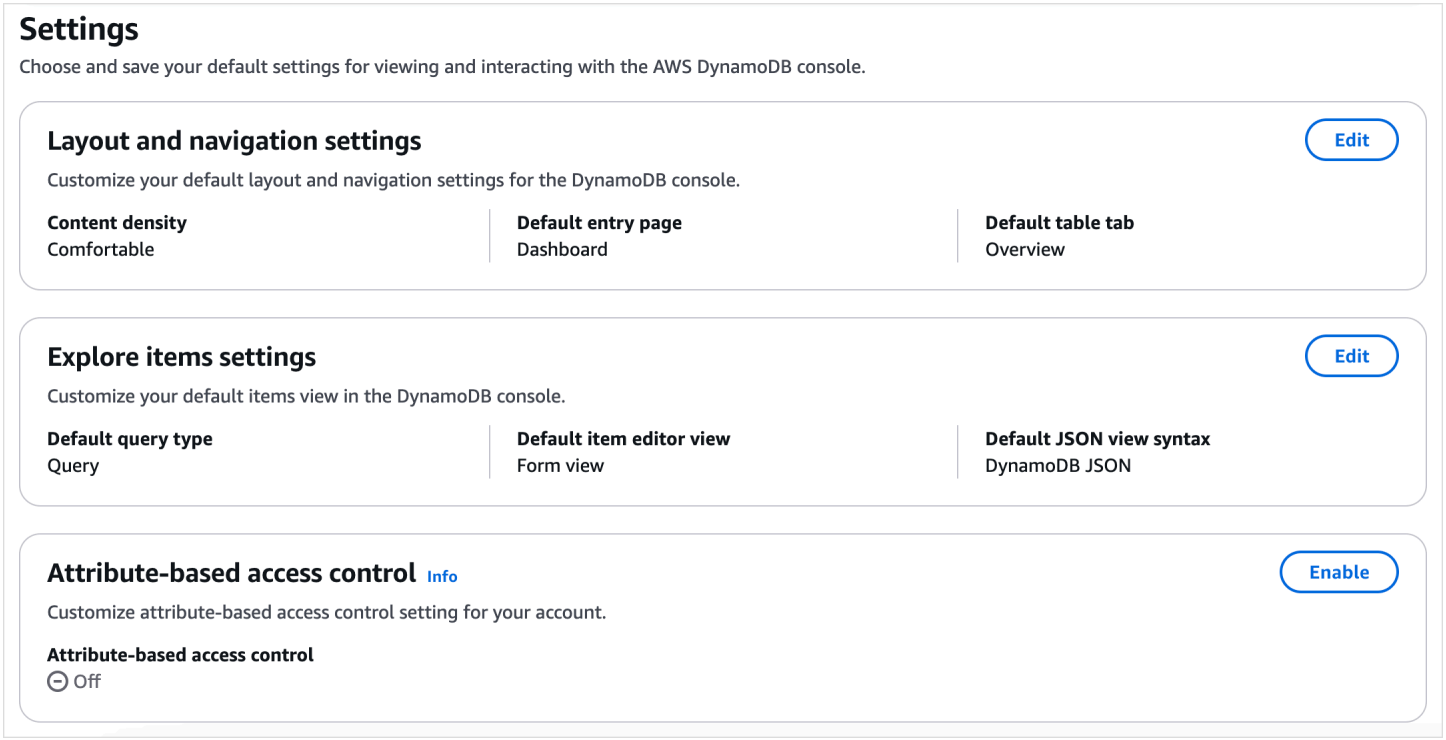
- L'assegnazione di tag e l'ABAC non sono supportati per i backup DynamoDB. Per utilizzare l'ABAC con i backup, si consiglia di utilizzare AWS Backup.
- I tag non vengono conservati nelle tabelle ripristinate. È necessario aggiungere tag alle tabelle ripristinate prima di poter utilizzare le condizioni basate sui tag nelle policy.

Abilitazione dell'ABAC in DynamoDB

Per la maggior parte Account AWS, ABAC è abilitato per impostazione predefinita. Utilizzando la console DynamoDB è possibile confermare se l'ABAC è abilitato per il proprio account. Per fare ciò, assicurati di aprire la console DynamoDB con un ruolo con autorizzazione dynamodb:. GetAbacStatus Quindi, apri la pagina Impostazioni della console DynamoDB.

Se non vedi la scheda Controllo degli accessi basato sugli attributi o se la scheda mostra lo stato Attivo, significa che l'ABAC è abilitato per l'account. Tuttavia, se vedi la scheda Controllo degli accessi basato sugli attributi con lo stato Non attivo, come mostrato nell'immagine seguente, l'ABAC non è abilitato per l'account.

Controllo degli accessi basato su attributi – non abilitato



ABAC non è abilitato Account AWS per cui le condizioni basate sui tag specificate nelle loro politiche basate sull'identità o altre politiche devono ancora essere verificate. Se l'ABAC non è abilitato per il proprio account, le condizioni basate sui tag nelle policy destinate ad agire sulle tabelle o sugli indici

DynamoDB vengono valutate come se non fossero presenti tag per le proprie risorse o richieste API. Quando l'ABAC è abilitato per il proprio account, le condizioni basate sui tag nelle policy dell'account vengono valutate considerando i tag allegati alle proprie tabelle o richieste API.

Per abilitare l'ABAC per il proprio account, si consiglia innanzitutto di sottoporre ad audit le policy come descritto nella sezione Audit delle policy. Quindi, si consiglia di includere le autorizzazioni richieste per l'ABAC nella policy IAM. Infine, si consiglia di eseguire le operazioni descritte in Abilitazione dell'ABAC nella console per abilitare l'ABAC per l'account nella Regione corrente. Dopo aver abilitato l'ABAC, è possibile disattivarlo entro i successivi sette giorni di calendario dall'attivazione.

Argomenti

- Esecuzione di audit sulle policy prima dell'abilitazione dell'ABAC
- Autorizzazioni IAM necessarie per abilitare l'ABAC
- Abilitazione dell'ABAC nella console

Esecuzione di audit sulle policy prima dell'abilitazione dell'ABAC

Prima di abilitare l'ABAC per l'account, sottoponi ad audit le policy per verificare che le condizioni basate sui tag che potrebbero esistere nelle policy dell'account siano configurate come previsto. L'audit delle policy aiuterà a evitare imprevisti dovuti alle modifiche delle autorizzazioni con i flussi di lavoro DynamoDB dopo l'abilitazione dell'ABAC. Per visualizzare esempi di utilizzo di condizioni basate su attributi con i tag e il comportamento prima e dopo l'implementazione dell'ABAC, consulta Esempi di utilizzo dell'ABAC con tabelle e indici DynamoDB.

Autorizzazioni IAM necessarie per abilitare l'ABAC

È necessaria l'autorizzazione `dynamodb:UpdateAbacStatus` per abilitare l'ABAC per l'account nella Regione corrente. Per verificare se l'ABAC è abilitato per l'account, è necessario anche disporre dell'autorizzazione `dynamodb:GetAbacStatus`. Con questa autorizzazione è possibile visualizzare lo stato dell'ABAC di un account in qualsiasi Regione. Queste autorizzazioni si aggiungono a quelle necessarie per accedere alla console DynamoDB e visualizzare le tabelle.

La seguente policy IAM concede l'autorizzazione per abilitare l'ABAC e visualizzarne lo stato per un account nella Regione corrente.

```
{
"version": "2012-10-17",&TCX5-2025-waiver;
```

```
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:UpdateAbacStatus",
                "dynamodb:GetAbacStatus"
             ],
            "Resource": "*"
        }
    ]
}
```

Abilitazione dell'ABAC nella console

1. Accedi Console di gestione AWS e apri la console DynamoDB all'indirizzo. https://console.aws.amazon.com/dynamodb/
2. Dal riquadro di navigazione in alto, seleziona la Regione per la quale desideri abilitare l'ABAC.
3. Nel riquadro di navigazione a sinistra, scegli Impostazioni.
4. Nella pagina Settings (Impostazioni), eseguire le operazioni descritte di seguito.

    a. Nella scheda Controllo degli accessi basato sugli attributi, seleziona Abilita.

    b. Nella casella Conferma l'impostazione del controllo degli accesso basato su attributi, seleziona Abilita per confermare la scelta.

    Questa operazione abilita l'ABAC per la Regione corrente e la scheda Controllo degli accessi basato sugli attributi mostra lo stato Attivo.

    Dopo aver abilitato l'ABAC, è possibile disattivarlo entro i successivi sette giorni di calendario dall'attivazione. Per disattivarlo, seleziona Disabilita nella scheda Controllo degli accessi basato sugli attributi nella pagina Impostazioni.

    > **Note**
    >
    > L'aggiornamento dello stato dell'ABAC è un'operazione asincrona. Se i tag delle policy non vengono valutati immediatamente, potrebbe essere necessario attendere qualche istante perché l'applicazione delle modifiche presenti coerenza finale.

Utilizzo dell'ABAC con tabelle e indici DynamoDB

La seguente procedura mostra come impostare le autorizzazioni attraverso l'ABAC. In questo scenario di esempio verranno aggiunti tag a una tabella DynamoDB e verrà creato un ruolo IAM con una policy che include condizioni basate su tag. Quindi, verranno testate le autorizzazioni consentite sulla tabella DynamoDB soddisfacendo le condizioni dei tag.

Argomenti

- Fase 1: aggiungere tag a una tabella DynamoDB
- Fase 2: creare un ruolo IAM con una policy che include condizioni basate su tag
- Fase 3: testare le autorizzazioni concesse

Fase 1: aggiungere tag a una tabella DynamoDB

Puoi aggiungere tag a tabelle DynamoDB nuove o esistenti utilizzando Console di gestione AWS AWS , API,AWS CLI() AWS Command Line Interface AWS , SDK o. AWS CloudFormation Ad esempio, il seguente comando della CLI tag-resource aggiunge un tag a una tabella denominata `MusicTable`.

```
aws dynamodb tag-resource –resource-arn arn:aws:dynamodb:us-east-1:123456789012:table/
MusicTable –tags Key=environment,Value=staging
```

Fase 2: creare un ruolo IAM con una policy che include condizioni basate su tag

Crea una policy IAM utilizzando la chiave di condizione aws: ResourceTag /tag-key per confrontare la coppia chiave-valore del tag specificata nella policy IAM con la coppia chiave-valore allegata alla tabella. La seguente policy di esempio consente agli utenti di inserire o aggiornare elementi nelle tabelle se queste tabelle contengono la coppia chiave-valore del tag: `"environment": "staging"`. Se una tabella non ha la coppia chiave-valore del tag specificata, queste azioni vengono negate.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
```

```
            "Effect": "Allow",
            "Action": [
                "dynamodb:PutItem",
                "dynamodb:UpdateItem"
            ],
            "Resource": "arn:aws:dynamodb:*:*:table/*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/environment": "staging"
                }
            }
        }
    ]
}
```

Fase 3: testare le autorizzazioni concesse

1. Collega la policy IAM a un utente o a un ruolo di prova nell' Account AWS. Assicurati che il principale IAM che utilizzi non abbia già accesso alla tabella DynamoDB tramite una policy diversa.
2. Assicurati che la tabella DynamoDB contenga la chiave del tag `"environment"` con un valore di `"staging"`.
3. Esegui le azioni `dynamodb:PutItem` e `dynamodb:UpdateItem` sulla tabella con tag. Queste azioni dovrebbero andare a buon fine se è presente la coppia chiave-valore del tag `"environment": "staging"`.

   Se si eseguono queste azioni su una tabella che non ha la coppia chiave-valore del tag `"environment": "staging"`, la richiesta avrà esito negativo restituendo una `AccessDeniedException`.

È anche possibile esaminare gli altri casi d'uso di esempio descritti nella sezione seguente per implementare l'ABAC ed eseguire altri test.

Esempi di utilizzo dell'ABAC con tabelle e indici DynamoDB

Gli esempi seguenti illustrano alcuni casi d'uso per implementare condizioni basate su attributi utilizzando i tag.

Argomenti

- Esempio 1: consenti un'azione utilizzando aws: ResourceTag
- Esempio 2: consenti un'azione utilizzando aws: RequestTag
- Esempio 3: Negare un'azione utilizzando aws: TagKeys

Esempio 1: consenti un'azione utilizzando aws: ResourceTag

Utilizzando la chiave di condizione `aws:ResourceTag/tag-key` è possibile confrontare la coppia chiave-valore del tag specificata in una policy IAM con la coppia chiave-valore collegata in una tabella DynamoDB. Ad esempio, puoi consentire un'azione specifica, ad esempio se le condizioni del tag corrispondono in una politica IAM e in una tabella. PutItem A tal fine, segui questa procedura:

Using the AWS CLI

1. Creare una tabella . L'esempio seguente utilizza il AWS CLI comando create-table per creare una tabella denominata. `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. Aggiungi un tag a questa tabella. Il seguente esempio di AWS CLI comando tag-resource aggiunge la coppia chiave-valore del tag a. `Title: ProductManager` `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. Create una policy in linea e aggiungetela a un ruolo a cui è associata la policy AmazonDynamoDBReadOnlyAccess AWS gestita, come illustrato nell'esempio seguente.

   JSON

   ```
   {
     "Version":"2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
   ```

```
      "Action": "dynamodb:PutItem",
      "Resource": "arn:aws:dynamodb:*:*:table/*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/Title": "ProductManager"
        }
      }
    }
  ]
}
```

Questa policy consente l'azione `PutItem` sulla tabella quando la chiave e il valore del tag associati alla tabella corrispondono ai tag specificati nella policy.

4. Assumi il ruolo con le policy descritte nella Fase 3.
5. Usa il AWS CLI comando put-item per inserire un elemento in. `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. Scansiona la tabella per verificare se l'elemento è stato aggiunto alla tabella.

```
aws dynamodb scan --table-name myMusicTable  --region us-east-1
```

Using the AWS SDK for Java 2.x

1. Creare una tabella . L'esempio seguente utilizza l'CreateTableAPI per creare una tabella denominata. `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. Aggiungi un tag a questa tabella. L'TagResourceAPI nell'esempio seguente aggiunge la coppia chiave-valore del tag `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. Create una policy in linea e aggiungetela a un ruolo a cui è associata la policy AmazonDynamoDBReadOnlyAccess AWS gestita, come illustrato nell'esempio seguente.

   JSON

   ```
   {
     "Version":"2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
         "Action": "dynamodb:PutItem",
         "Resource": "arn:aws:dynamodb:*:*:table/*",
         "Condition": {
           "StringEquals": {
             "aws:ResourceTag/Title": "ProductManager"
           }
         }
       }
     ]
   }
   ```

   Questa policy consente l'azione `PutItem` sulla tabella quando la chiave e il valore del tag associati alla tabella corrispondono ai tag specificati nella policy.

4. Assumi il ruolo con le policy descritte nella Fase 3.

5. Usa l'PutItemAPI per inserire un elemento in. `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. Scansiona la tabella per verificare se l'elemento è stato aggiunto alla tabella.

```
ScanRequest scanRequest = ScanRequest.builder()
                .tableName(tableName)
                .build();

ScanResponse scanResponse = dynamoDB.scan(scanRequest);
```

Using the AWS SDK for Python (Boto3)

1. Creare una tabella . L'esempio seguente utilizza l'CreateTableAPI per creare una tabella denominata`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. Aggiungi un tag a questa tabella. L'TagResourceAPI nell'esempio seguente aggiunge la coppia chiave-valore del tag `Title: ProductManager` a. `myMusicTable`

```
tag_resouce_response = ddb_client.tag_resource(
    ResourceArn=table_arn,
    Tags=[
        {
            'Key': 'Title',
            'Value': 'ProductManager'
        },
    ]
)
```

3. Create una policy in linea e aggiungetela a un ruolo a cui è associata la policy AmazonDynamoDBReadOnlyAccess AWS gestita, come illustrato nell'esempio seguente.

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
        "Effect": "Allow",
        "Action": "dynamodb:PutItem",
        "Resource": "arn:aws:dynamodb:*:*:table/*",
        "Condition": {
            "StringEquals": {
            "aws:ResourceTag/Title": "ProductManager"
            }
        }
        }
    ]
    }
```

   Questa policy consente l'azione `PutItem` sulla tabella quando la chiave e il valore del tag associati alla tabella corrispondono ai tag specificati nella policy.

4. Assumi il ruolo con le policy descritte nella Fase 3.

5. Usa l'PutItemAPI per inserire un elemento in. `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. Scansiona la tabella per verificare se l'elemento è stato aggiunto alla tabella.

```
scan_response = client.scan(
    TableName='myMusicTable'
)
```

Senza ABAC

Se ABAC non è abilitato per te Account AWS, le condizioni dei tag nella policy IAM e nella tabella DynamoDB non corrispondono. Di conseguenza, l'azione `PutItem` restituirà una `AccessDeniedException` a causa dell'effetto della policy `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

Se ABAC è abilitato per la tua Account AWS, l'`put-item`azione viene completata correttamente e aggiunge un nuovo elemento alla tua tabella. Questo perché la policy in linea sulla tabella consente l'azione `PutItem` se le condizioni del tag nella policy IAM e nella tabella corrispondono.

Esempio 2: consenti un'azione utilizzando aws: RequestTag

Utilizzando la chiave di condizione aws: RequestTag /tag-key, puoi confrontare la coppia chiave-valore del tag passata nella tua richiesta con la coppia di tag specificata nella policy IAM. Ad esempio, è possibile consentire un'azione specifica, come `CreateTable`, usando `aws:RequestTag` se le condizioni del tag non corrispondono. A tal fine, segui questa procedura:

Using the AWS CLI

1. Crea una policy in linea e aggiungila a un ruolo a cui è associata la policy AmazonDynamoDBReadOnlyAccess AWS gestita, come mostrato nell'esempio seguente.

   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. Crea una tabella che contenga la coppia chiave-valore del tag di `"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. Create una policy in linea e aggiungetela a un ruolo a cui è associata la policy AmazonDynamoDBReadOnlyAccess AWS gestita, come illustrato nell'esempio seguente.

   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. Crea una tabella che contenga la coppia chiave-valore del tag di `"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'
        },
    ],
)
```

Senza ABAC

Se ABAC non è abilitato per te Account AWS, le condizioni dei tag nella politica in linea e nella tabella DynamoDB non corrispondono. Di conseguenza, la richiesta `CreateTable` non andrà a buon fine e la tabella non verrà creata.

```
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

Se ABAC è abilitato per la tua Account AWS, la richiesta di creazione della tabella viene completata correttamente. Poiché la coppia chiave-valore del tag `"Owner": "John"` è presente nella richiesta `CreateTable`, la policy in linea consente all'utente `John` di eseguire l'azione `CreateTable`.

Esempio 3: Negare un'azione utilizzando aws: TagKeys

Utilizzando la chiave aws: TagKeys condition, puoi confrontare le chiavi dei tag in una richiesta con le chiavi specificate nella policy IAM. Ad esempio, è possibile negare un'azione specifica, come `CreateTable`, utilizzando `aws:TagKeys` se una chiave di tag specifica non è presente nella richiesta. A tal fine, segui questa procedura:

Using the AWS CLI

1. Aggiungi una policy gestita dai clienti a un ruolo a cui è associata la policy AWS gestita di AmazonDynamoDBFullAccess, come illustrato nell'esempio seguente.

   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. Assumi il ruolo a cui è stata associata la policy e crea una tabella con la chiave del tag `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. Aggiungere una politica gestita dai clienti a un ruolo a cui è associata la politica di AWS gestione degli AmazonDynamoDBFullaccessi, come illustrato nell'esempio seguente.

   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. Assumi il ruolo a cui è stata associata la policy e crea una tabella con la chiave del tag `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'
        },
    ],
)
```

Senza ABAC

Se ABAC non è abilitato per il tuo Account AWS, DynamoDB non invia le chiavi dei tag nel `create-table` comando a IAM. La condizione `Null` garantisce che la condizione restituisca `false` in assenza di chiavi di tag nella richiesta. Poiché la policy `Deny` non corrisponde, il comando `create-table` verrà completato correttamente.

Con ABAC

Se ABAC è abilitato per il tuo Account AWS, le chiavi dei tag passate nel `create-table` comando vengono passate a IAM. La chiave del tag `Title` verrà valutata rispetto alla chiave del tag basata sulla condizione, `CostCenter`, presente nella policy `Deny`. La chiave del tag `Title` non corrisponde alla chiave del tag presente nella policy `Deny` a causa dell'operatore `StringNotEquals`. Pertanto, l'azione `CreateTable` non andrà a buon fine e la tabella non verrà creata. L'esecuzione del comando `create-table` restituisce una `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.
```

Risoluzione dei problemi più comuni dell'ABAC per le tabelle e gli indici DynamoDB

Questo argomento fornisce consigli per la risoluzione dei problemi relativi a errori e problemi comuni che potrebbero verificarsi durante l'implementazione dell'ABAC nelle tabelle o negli indici DynamoDB.

Le chiavi di condizione specifiche del servizio nelle policy generano un errore

Le chiavi di condizione specifiche del servizio non sono considerate chiavi di condizione valide. Se sono state utilizzate tali chiavi nelle policy, si verificherà un errore. Per risolvere questo problema, è necessario sostituire le chiavi di condizione specifiche del servizio con una chiave di condizione appropriata per implementare l'ABAC in DynamoDB.

Ad esempio, supponiamo di aver utilizzato la chiave `dynamodb:ResourceTag` condition in una policy in linea che esegue la richiesta. PutItem Si immagini che la richiesta abbia esito negativo con una `AccessDeniedException`. Nell'esempio seguente viene illustrata la policy in linea errata con la chiave di condizione `dynamodb:ResourceTag`.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:PutItem"
            ],
            "Resource": "arn:aws:dynamodb:*:*:table/*",
            "Condition": {
                "StringEquals": {
                    "dynamodb:ResourceTag/Owner": "John"
                }
            }
        }
    ]
}
```

Per risolvere questo problema, sostituisci la chiave di condizione `dynamodb:ResourceTag` con `aws:ResourceTag`, come illustrato nell'esempio seguente.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:PutItem"
            ],
            "Resource": "arn:aws:dynamodb:*:*:table/*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/Owner": "John"
                }
            }
        }
    ]
}
```

Impossibile disattivare l'ABAC

Se ABAC è stato abilitato per il tuo account tramite Supporto, non potrai disattivare ABAC tramite la console DynamoDB. Per disattivarlo, contatta il Supporto.

È possibile disattivare personalmente l'ABAC solo alle seguenti condizioni:

- È stata utilizzata la modalità self-service di adesione tramite la console DynamoDB.
- La disattivazione avviene entro sette giorni di calendario dall'adesione.

Protezione dei dati in DynamoDB

Amazon DynamoDB offre un'infrastruttura di archiviazione estremamente durevole, concepita per l'archiviazione dei dati mission-critical e primari. I dati vengono archiviati in modo ridondante su più dispositivi, in più strutture di una regione Amazon DynamoDB.

DynamoDB protegge i dati degli utenti archiviati a riposo e anche i dati in transito tra i client locali e DynamoDB e tra DynamoDB e altre risorse all'interno della stessa regione. AWS AWS

Argomenti

- Crittografia a riposo per DynamoDB
- Protezione delle connessioni DynamoDB attraverso endpoint VPC e policy IAM

Crittografia a riposo per DynamoDB

Tutti i dati utente memorizzati in Amazon DynamoDB sono completamente crittografati a riposo. La crittografia a riposo di DynamoDB offre sicurezza avanzata grazie alla crittografia dei dati mediante le chiavi di crittografia archiviate in AWS Key Management Service (AWS KMS). Questa funzionalità consente di ridurre gli oneri operativi e la complessità associati alla protezione dei dati sensibili. La crittografia dei dati inattivi consente di creare applicazioni ad alto livello di sicurezza che rispettano rigorosi requisiti normativi e di conformità per la crittografia.

La crittografia dei dati a riposo di DynamoDB fornisce un livello aggiuntivo di protezione dei dati proteggendo sempre i dati nella tabella crittografata che include la chiave primaria, gli indici secondari locali e globali, flussi, tabelle globali, backup e cluster DynamoDB Accelerator (DAX), ogni volta che i dati vengono archiviati in supporti fisici. Le policy aziendali, le normative di settore e del governo e i requisiti di conformità spesso esigono l'uso della crittografia dei dati inattivi per aumentare la sicurezza dei dati delle applicazioni. Per ulteriori informazioni sulla crittografia per le applicazioni per database, consulta AWS Database Encryption SDK.

Encryption at rest si integra con la gestione delle chiavi di crittografia utilizzate AWS KMS per crittografare le tabelle. Per ulteriori informazioni sui tipi e sugli stati delle chiavi, consulta i AWS Key Management Service concepts in the Guida per gli sviluppatori di AWS Key Management Service .

Quando si crea una nuova tabella, è possibile scegliere uno dei seguenti AWS KMS key tipi per crittografare la tabella. È possibile passare da un tipo di chiave all'altro in qualsiasi momento.

- Chiave di proprietà di AWS — Tipo di crittografia predefinito. La chiave è di proprietà di DynamoDB (nessun costo aggiuntivo).
- Chiave gestita da AWS — La chiave è memorizzata nel tuo account ed è gestita da AWS KMS (a AWS KMS pagamento).
- Chiave gestita dal cliente: la chiave è archiviata nel tuo account ed è creata da te, di tua proprietà e gestita da te. Hai il pieno controllo della chiave KMS (a AWS KMS pagamento).

Per ulteriori informazioni sui tipi di chiave, consulta Chiavi e AWS chiavi del cliente.

> **Note**
>
> - Quando si crea un nuovo cluster DAX con la crittografia a riposo abilitata, viene utilizzata una Chiave gestita da AWS per criptare i dati a riposo nel cluster.
> - Se la tabella dispone di chiavi di ordinamento, alcune di queste che contrassegnano i limiti dell'intervallo sono archiviate come testo non crittografato nei metadati della tabella.

Quando si accede a una tabella crittografata, DynamoDB decrittografa i dati della tabella in modo trasparente. Non è necessario modificare codice o applicazioni per utilizzare o gestire le tabelle crittografate. DynamoDB continua a offrire la stessa latenza di pochi millisecondi a cui si è abituati e tutte le query di DynamoDB funzionano senza problemi sui dati crittografati.

Puoi specificare una chiave di crittografia quando crei una nuova tabella o cambi le chiavi di crittografia su una tabella esistente utilizzando Console di gestione AWS, AWS Command Line Interface (AWS CLI) o l'API Amazon DynamoDB. Per scoprire come, consulta Gestione di tabelle crittografate in DynamoDB.

La crittografia a riposo utilizzando il Chiave di proprietà di AWS è offerta senza costi aggiuntivi. Tuttavia, vengono AWS KMS applicati dei costi per una chiave gestita dal cliente Chiave gestita da AWS e per una chiave gestita dal cliente. Per ulteriori informazioni sui prezzi, consulta Prezzi di AWS KMS.

La crittografia a riposo di DynamoDB è disponibile in AWS tutte le regioni, incluse le regioni AWS Cina (Pechino) e AWS Cina (Ningxia) e AWS GovCloud le regioni (Stati Uniti). Per ulteriori informazioni, consultare Crittografia dei dati a riposo di DynamoDB: come funziona e Note per l'utilizzo della crittografia dei dati inattivi di DynamoDB.

Crittografia dei dati a riposo di DynamoDB: come funziona

La crittografia a riposo di Amazon DynamoDB crittografa i dati tramite lo standard di crittografia avanzata a 256 bit (AES-256) che aiuta a proteggere i dati dall'accesso non autorizzato all'archiviazione sottostante.

Encryption at rest si integra con AWS Key Management Service (AWS KMS) per la gestione delle chiavi di crittografia utilizzate per crittografare le tabelle.

> **Note**
>
> A maggio 2022, AWS KMS ha modificato il programma di rotazione Chiavi gestite da AWS da ogni tre anni (circa 1.095 giorni) a ogni anno (circa 365 giorni).
> Chiavi gestite da AWS I nuovi vengono ruotati automaticamente un anno dopo la creazione e successivamente all'incirca ogni anno.
> Chiavi gestite da AWS Le versioni esistenti vengono ruotate automaticamente un anno dopo la rotazione più recente e successivamente ogni anno.

Chiavi di proprietà di AWS

Chiavi di proprietà di AWS non sono memorizzati nel tuo AWS account. Fanno parte di una raccolta di chiavi KMS che AWS possiede e gestisce per l'utilizzo in più AWS account. AWS i servizi possono essere utilizzati Chiavi di proprietà di AWS per proteggere i tuoi dati. Chiavi di proprietà di AWS utilizzati da DynamoDB vengono ruotati ogni anno (circa 365 giorni).

Non è possibile visualizzarne, gestirne Chiavi di proprietà di AWS, utilizzarne o controllarne l'utilizzo. Tuttavia, non è necessario effettuare alcuna operazione o modificare programmi per proteggere le chiavi che eseguono la crittografia dei dati.

Non ti viene addebitato un canone mensile o un canone di utilizzo per l'utilizzo di Chiavi di proprietà di AWS e questi non vengono conteggiati nelle AWS KMS quote relative al tuo account.

Chiavi gestite da AWS

Chiavi gestite da AWS sono chiavi KMS del tuo account che vengono create, gestite e utilizzate per tuo conto da un AWS servizio integrato con. AWS KMS Puoi visualizzare le Chiavi gestite da AWS nel tuo account, visualizzare le relative policy delle chiavi e verificare il loro utilizzo nei registri AWS CloudTrail . Tuttavia, non puoi gestire queste chiavi KMS o modificarne le autorizzazioni.

Encryption at rest si integra automaticamente con la gestione dei AWS KMS Chiavi gestite da AWS for DynamoDB `aws/dynamodb` () utilizzati per crittografare le tabelle. Se Chiave gestita da AWS non esiste un quando crei la tabella DynamoDB crittografata AWS KMS , crea automaticamente una nuova chiave per te. Questa chiave viene utilizzata con le tabelle crittografate che verranno create in futuro. AWS KMS combina hardware e software sicuri e ad alta disponibilità per fornire un sistema di gestione delle chiavi scalabile per il cloud.

Per ulteriori informazioni sulla gestione delle autorizzazioni di Chiave gestita da AWS, vedere Authorizing use of the Chiave gestita da AWSAWS Key Management Service nella Developer Guide.

Chiavi gestite dal cliente

Le chiavi gestite dal cliente sono chiavi KMS del tuo AWS account che crei, possiedi e gestisci. Hai il controllo completo di queste chiavi KMS, tra cui la definizione e il mantenimento delle policy delle chiavi, delle policy IAM e delle concessioni, l'abilitazione e la disabilitazione di tali chiavi, la rotazione del materiale crittografico, l'aggiunta di tag, la creazione di alias che fanno riferimento a esse e la pianificazione per l'eliminazione. Per ulteriori informazioni relative alla gestione delle autorizzazioni di una chiave gestita dal cliente, consulta Chiavi gestite dal cliente.

Quando specifichi una chiave gestita dal cliente come chiave di crittografia a livello di tabella, la tabella DynamoDB, gli indici secondari locale e globale e i flussi vengono crittografati con la stessa chiave gestita dal cliente. I backup on demand vengono crittografati con la chiave di crittografia a livello di tabella specificata al momento della creazione del backup. L'aggiornamento della chiave di crittografia a livello di tabella non modifica la chiave di crittografia associata con i backup on demand esistenti.

L'impostazione dello stato della chiave gestita dal cliente su disabilitata o la pianificazione per l'eliminazione impedisce a tutti gli utenti e al servizio DynamoDB di criptare o decriptare i dati e di eseguire operazioni di lettura e scrittura sulla tabella. DynamoDB deve poter accedere alla chiave di crittografia per assicurarsi di poter continuare ad accedere alla tabella e prevenire la perdita di dati.

Se disabiliti la chiave gestita dal cliente o ne pianifichi l'eliminazione, lo stato della tabella diventa Inaccessible (Inaccessibile). Per essere certi che sia possibile continuare a lavorare con la tabella, è necessario fornire a DynamoDB l'accesso alla chiave di crittografia specificata entro sette giorni. Non appena il servizio rileva che la chiave di crittografia non è accessibile, DynamoDB invia una notifica e-mail come avviso.

> **Note**
>
> - Se la chiave gestita dal cliente rimane inaccessibile al servizio DynamoDB per più di sette giorni, la tabella viene archiviata e non è più possibile accedervi. DynamoDB crea un backup on demand della tabella che viene addebitato. È possibile utilizzare il backup on demand per ripristinare i dati in una nuova tabella. Per avviare il ripristino, è necessario abilitare l'ultima chiave gestita dal cliente sulla tabella e DynamoDB deve potervi accedere.
> - Se non è possibile accedere alla chiave gestita dal cliente utilizzata per criptare una replica della tabella globale, DynamoDB rimuoverà questa replica dal gruppo di replica. La replica

non verrà eliminata e sarà interrotta da e verso questa regione 20 ore dopo aver rilevato che la chiave gestita dal cliente non è accessibile.

Per ulteriori informazioni, copri come attivare ed eliminare le chiavi.

Note sull'utilizzo Chiavi gestite da AWS

Amazon DynamoDB non può leggere i dati delle tabelle a meno che non abbia accesso alla chiave KMS memorizzata nel tuo account. AWS KMS DynamoDB utilizza la crittografia envelope e la gerarchia delle chiavi per criptare i dati. La tua chiave di AWS KMS crittografia viene utilizzata per crittografare la chiave principale di questa gerarchia di chiavi. Per ulteriori informazioni, consulta Crittografia envelope nella Guida per gli sviluppatori di AWS Key Management Service .

DynamoDB non AWS KMS richiede ogni operazione DynamoDB. La chiave viene aggiornata una volta ogni 5 minuti per chiamante con traffico attivo.

Assicurati di avere configurato l'SDK per riutilizzare le connessioni. In caso contrario, si verificheranno delle latenze dovute alla necessità di ristabilire nuove voci della AWS KMS cache per ogni operazione DynamoDB. Inoltre, potreste dover affrontare costi più elevati. AWS KMS CloudTrail Ad esempio, per fare ciò con l'SDK Node.js, puoi creare un nuovo agente HTTPS con `keepAlive` attivato. Per ulteriori informazioni, consulta Configurazione di keepAlive in Node.js nella Guida per lo sviluppatore di AWS SDK per JavaScript .

Note per l'utilizzo della crittografia dei dati inattivi di DynamoDB

Quando si utilizza la crittografia a riposo di Amazon DynamoDB, tenere in considerazione quanto riportato di seguito.

Tutti i dati della tabella vengono crittografati

La crittografia dei dati a riposo sul lato server è abilitata su tutti i dati della tabella DynamoDB e non può essere disabilitata. Non è possibile criptare solo un sottoinsieme di elementi in una tabella.

La crittografia dei dati inattivi crittografa i dati mentre è statica (dati inattivi) su media di storage persistente. Se la sicurezza dei dati è un problema per i dati in transito o i dati in utilizzo, è possibile adottare ulteriori misure:

- Dati in transito: tutti i dati in DynamoDB sono crittografati in transito. Per impostazione predefinita, le comunicazioni da e verso DynamoDB utilizzano il protocollo HTTPS, che protegge il traffico di rete tramite la crittografia Secure Sockets Layer (SSL)/Transport Layer Security (TLS).

- Dati in uso: proteggere i dati prima di inviarli a DynamoDB utilizzando la crittografia lato client. Per ulteriori informazioni, consulta Client-side and server-side encryption nella Guida per sviluppatori del client di crittografia di Amazon DynamoDB.

È possibile utilizzare i flussi con le tabelle crittografate. I flussi DynamoDB sono sempre crittografati con una chiave di crittografia a livello di tabella. Per ulteriori informazioni, consulta Acquisizione dei dati di modifica per DynamoDB Streams.

I backup DynamoDB sono crittografati e anche la tabella che viene ripristinata da un backup ha la crittografia abilitata. È possibile utilizzare la chiave o Chiave di proprietà di AWS Chiave gestita da AWS la chiave gestita dal cliente per crittografare i dati di backup. Per ulteriori informazioni, consulta Backup e ripristino per DynamoDB.

Gli indici secondari locali e gli indici secondari globali vengono crittografati utilizzando la stessa chiave della tabella di base.

Tipi di crittografia

> **Note**
>
> Le chiavi gestite dal cliente non sono supportate nelle tabelle globali versione 2017. Se desideri utilizzare una chiave gestita dal cliente in una tabella globale di DynamoDB, è necessario aggiornare la tabella alla versione 2019 della tabella globale e quindi abilitarla.

In Console di gestione AWS, il tipo di crittografia si intende `KMS` quando si utilizza la chiave Chiave gestita da AWS o la chiave gestita dal cliente per crittografare i dati. Il tipo di crittografia è `DEFAULT` quando utilizzi la Chiave di proprietà di AWS. Nell'API Amazon DynamoDB, il tipo di crittografia `KMS` è quando usi la chiave o la chiave gestita Chiave gestita da AWS dal cliente. In assenza del tipo di crittografia, i dati vengono criptati con la chiave Chiave di proprietà di AWS. Puoi passare dalla chiave gestita dal Chiave di proprietà di AWS cliente a quella gestita dal cliente in qualsiasi momento. Chiave gestita da AWS Puoi utilizzare la console, la AWS Command Line Interface (AWS CLI) o l'API Amazon DynamoDB per cambiare le chiavi di crittografia.

Notare le seguenti limitazioni quando si utilizzano le chiavi gestite dal cliente:

- Non puoi utilizzare una chiave gestita dal cliente con i cluster DynamoDB Accelerator (DAX). Per ulteriori informazioni, consulta Crittografia DAX dei dati inattivi.

- Puoi utilizzare una chiave gestita dal cliente per criptare le tabelle che utilizzano le transazioni. Tuttavia, per garantire la durabilità per la propagazione delle transazioni, una copia della richiesta di transazione viene archiviata temporaneamente dal servizio e criptata utilizzando una Chiave di proprietà di AWS. I dati di commit nelle tabelle e negli indici secondari vengono sempre criptati a riposo utilizzando la chiave gestita dal cliente.
- Puoi utilizzare una chiave gestita dal cliente per criptare le tabelle che utilizzano Contributor Insights. Tuttavia, i dati trasmessi a Amazon CloudWatch vengono crittografati con un. Chiave di proprietà di AWS
- Quando passi a una nuova chiave gestita dal cliente, assicurati di mantenere attiva la chiave originale fino al completamento del processo. AWS avrà comunque bisogno della chiave originale per decrittografare i dati prima di crittografarli con la nuova chiave. Il processo sarà completo quando SSEDescription lo stato della tabella sarà ABILITATO e verrà visualizzata la KMSMaster KeyArn nuova chiave gestita dal cliente. A questo punto, la chiave originale può essere disabilitata o pianificata per l'eliminazione.
- Una volta visualizzata la nuova chiave gestita dal cliente, la tabella e tutti i nuovi backup on demand vengono criptati con la nuova chiave.
- Tutti i backup on demand esistenti rimangono criptati con la chiave gestita dal cliente utilizzata al momento della creazione di tali backup. La stessa chiave è necessaria per ripristinare i backup. È possibile identificare la chiave per il periodo in cui è stato creato ogni backup utilizzando l' DescribeBackup API per visualizzare i backup SSEDescription.
- Se disabiliti la chiave gestita dal cliente o ne pianifichi l'eliminazione, tutti i dati in DynamoDB Streams avranno comunque una durata di 24 ore. Tutti i dati di attività non recuperati sono idonei per il taglio quando sono più vecchi di 24 ore.
- Se disabiliti la chiave gestita dal cliente o ne pianifichi l'eliminazione, le eliminazioni della durata (TTL) continuano per 30 minuti. Queste eliminazioni TTL continuano a essere emesse su DynamoDB Streams e sono soggette all'intervallo standard. trimming/retention

  Per ulteriori informazioni, copri come attivare ed eliminare le chiavi.

Utilizzo di chiavi KMS e chiavi dati

La funzionalità di crittografia a riposo di DynamoDB utilizza AWS KMS key una e una gerarchia di chiavi dati per proteggere i dati della tabella. DynamoDB utilizza la stessa gerarchia di chiavi per proteggere i flussi DynamoDB, tabelle globali e backup quando sono scritti su supporti durevoli.

Consigliamo di pianificare la strategia di crittografia prima di implementare la tabella in DynamoDB. Se archivi dati sensibili o riservati in DynamoDB, prendi in considerazione l'inclusione della crittografia lato client nel piano. In questo modo puoi crittografare i dati il più vicino possibile alla loro origine e garantirne la protezione per tutto il loro ciclo di vita. Per ulteriori informazioni su ciascun tipo, consultare la documentazione relativa al client di crittografia di DynamoDB.

AWS KMS key

La crittografia dei dati inattivi protegge le tabelle DynamoDB con una AWS KMS key. Per impostazione predefinita, DynamoDB utilizza una Chiave di proprietà di AWS, una chiave di crittografia multi-tenant creata e gestita in un account del servizio DynamoDB. Ma è possibile crittografare le tabelle DynamoDB in una chiave gestita dal cliente per DynamoDB (`aws/dynamodb`) nell' Account AWS. È possibile selezionare una chiave KMS differente per ogni tabella. La chiave KMS selezionata per una tabella viene anche utilizzata per crittografare gli indici secondari locali e globali, i flussi e i backup.

Quando si crea o si aggiorna la tabella, si seleziona la chiave KMS per una tabella. È possibile modificare la chiave KMS per una tabella in qualsiasi momento, nella console DynamoDB o utilizzando l'operazione. UpdateTable Il processo di scambio di chiavi è sicuro, non richiede tempi di inattività e non comporta alcun calo delle prestazioni del servizio.

> **Important**
>
> DynamoDB supporta soloChiavi KMS simmetriche. Non è possibile utilizzare una chiave KMS asimmetrica per crittografare le tabelle DynamoDB.

Utilizza una chiave gestita dal cliente per ottenere le seguenti caratteristiche:

- È possibile creare e gestire la chiave KMS, inclusa l'impostazione della Policy delle chiavi, la Policy IAM e le concessioni per controllare l'accesso alla chiave KMS. È possibile abilitare e disabilitare la chiave KMS, abilitare e disabilitare la rotazione automatica della chiave ed eliminare la chiave KMS quando non è più in uso.
- Puoi utilizzare una chiave gestita dal cliente con materiale di chiave importato o una chiave gestita dal cliente in un archivio delle chiavi personalizzate di cui sei proprietario e gestore.
- È possibile controllare la crittografia e la decrittografia della tabella DynamoDB esaminando le chiamate dell'API DynamoDB ai log. AWS KMSAWS CloudTrail

Utilizza Chiave gestita da AWS se hai bisogno di una delle seguenti funzionalità:

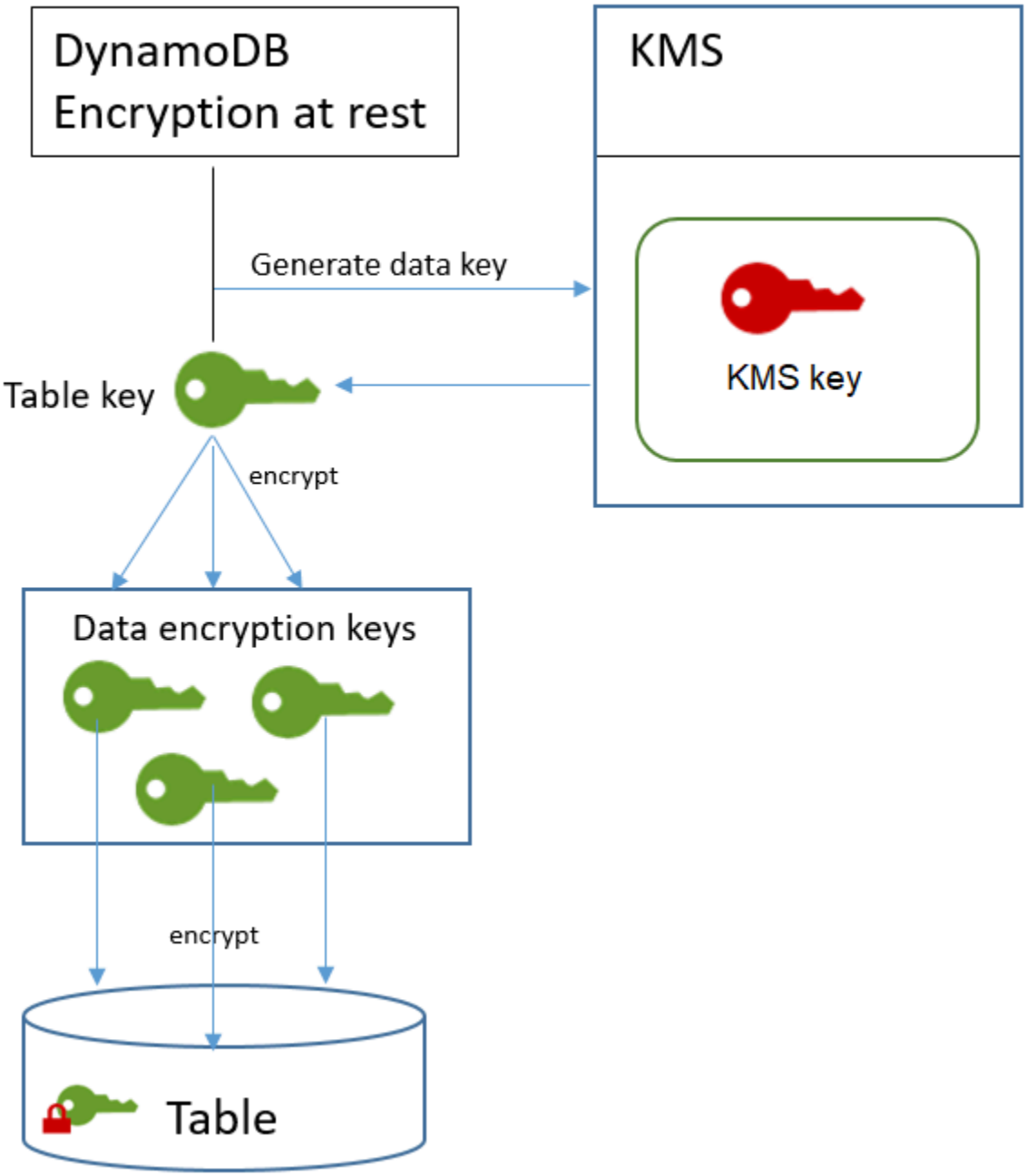
- Puoi visualizzare la chiave KMS e visualizzare la policy chiave. (non è possibile modificare la policy della chiave).
- È possibile controllare la crittografia e la decrittografia della tabella DynamoDB esaminando le chiamate dell'API DynamoDB ai log. AWS KMSAWS CloudTrail

Tuttavia, Chiave di proprietà di AWS è gratuito e il suo utilizzo non influisce sulle quote di risorse o richieste.AWS KMS Le chiavi gestite dal cliente Chiavi gestite da AWS comportano un costo per ogni chiamata API e le AWS KMS quote si applicano a queste chiavi KMS.

Chiavi delle tabelle

DynamoDB usa la chiave KMS per la tabella per generare e crittografare una chiave di dati univoca per la tabella, nota come la chiave della tabella. La chiave della tabella esiste per tutta la durata della tabella crittografata.

La chiave di tabella viene utilizzata come chiave di crittografia. DynamoDB utilizza questa chiave di tabella per proteggere le chiavi di crittografia dei dati utilizzate per crittografare i dati delle tabelle. DynamoDB genera una chiave di crittografia dei dati univoca per ogni struttura sottostante in una tabella, ma più di un elemento della tabella potrebbe essere protetto dalla stessa chiave di crittografia dei dati.



Quando si accede per la prima volta a una tabella crittografata, DynamoDB invia una richiesta AWS KMS per utilizzare la chiave KMS per decrittografare la chiave della tabella. Quindi, utilizza la chiave della tabella in testo normale per decrittografare le chiavi di crittografia dei dati e le chiavi di crittografia dei dati in testo normale per decrittografare i dati della tabella.

DynamoDB archivia e utilizza la chiave della tabella e le chiavi di crittografia dei dati all'esterno di. AWS KMS Protegge tutte le chiavi con la crittografia Advanced Encryption Standard (AES) e le

chiavi di crittografia a 256 bit. Quindi, archivia le chiavi crittografate con i dati crittografati, in modo che siano disponibili per decrittografare i dati della tabella on demand.

Quando viene modificata la chiave KMS per la tabella, DynamoDB genera una nuova chiave di tabella. Quindi, utilizza la nuova chiave per crittografare nuovamente le chiavi di crittografia dei dati.

Caching della chiave della tabella

Per evitare di chiamare AWS KMS per ogni operazione DynamoDB, DynamoDB memorizza nella cache le chiavi della tabella in testo semplice per ogni chiamante in memoria. Se DynamoDB riceve una richiesta per la chiave della tabella memorizzata nella cache dopo cinque minuti di inattività, invia una nuova richiesta AWS KMS per decrittografare la chiave della tabella. Questa chiamata acquisirà tutte le modifiche apportate alle politiche di accesso della chiave KMS in AWS KMS or AWS Identity and Access Management (IAM) dall'ultima richiesta di decrittografia della chiave della tabella.

Autorizzazione all'uso della propria chiave KMS

Se utilizzi una chiave gestita dal cliente o la Chiave gestita da AWS nel tuo account per proteggere la tabella DynamoDB, è necessario che le policy su tale chiave KMS forniscano a DynamoDB l'autorizzazione per usarla per tuo conto. Il contesto di autorizzazione di Chiave gestita da AWS for DynamoDB include la sua politica chiave e le autorizzazioni a delegare le autorizzazioni per utilizzarlo.

Hai il pieno controllo sulle policy e sulle concessioni di una chiave gestita dal cliente Poiché la Chiave gestita da AWS è presente nell'account, puoi visualizzare le relative policy e concessioni. Tuttavia, poiché è gestito da, non è possibile modificare AWS le politiche.

DynamoDB non necessita di autorizzazioni aggiuntive per utilizzare l'Chiave di proprietà di AWSimpostazione predefinita per proteggere le tabelle DynamoDB presenti nel tuo. Account AWS

Argomenti

- Politica chiave per un Chiave gestita da AWS
- Policy delle chiavi per una chiave gestita dal cliente
- Utilizzo di concessioni per autorizzare DynamoDB

Politica chiave per un Chiave gestita da AWS

Quando DynamoDB utilizza la Chiave gestita da AWS per DynamoDB (`aws/dynamodb`) in operazioni di crittografia, lo fa per conto dell'utente che accede alla risorsa DynamoDB. La politica chiave di Chiave gestita da AWS consente a tutti gli utenti dell'account di utilizzare Chiave gestita da AWS le operazioni specificate. Tuttavia, l'autorizzazione è concessa solo quando DynamoDB effettua la richiesta per conto dell'utente. La ViaService condizione nella policy chiave non consente a nessun utente di utilizzare la a Chiave gestita da AWS meno che la richiesta non provenga dal servizio DynamoDB.

Questa politica chiave, come le politiche comuni a tutti Chiavi gestite da AWS, è stabilita da. AWS Non è possibile modificarla, ma è possibile visualizzarla in qualsiasi momento. Per informazioni dettagliate, consulta Visualizzazione di una policy di chiave.

Le istruzioni di policy nella policy delle chiavi hanno l'effetto seguente:

- Consenti agli utenti dell'account di utilizzare Chiave gestita da AWS for DynamoDB nelle operazioni crittografiche quando la richiesta proviene da DynamoDB per loro conto. La policy, inoltre, consente agli utenti di creare concessioni per la chiave KMS.
- Consente alle identità IAM autorizzate nell'account di visualizzare le proprietà del Chiave gestita da AWS per DynamoDB e di revocare la concessione che consente a DynamoDB di utilizzare la chiave KMS. DynamoDB utilizza concessioni per le operazioni di manutenzione in corso.
- Consente a DynamoDB di eseguire operazioni di sola lettura per trovare il file per DynamoDB nel tuo account Chiave gestita da AWS .

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" : "*"
  } ]
}
```

Policy delle chiavi per una chiave gestita dal cliente

Quando selezioni una chiave gestita dal cliente per proteggere una tabella DynamoDB, DynamoDB ottiene l'autorizzazione per utilizzare la chiave KMS per conto del principale che effettua la selezione. Tale principale, un utente o un ruolo, deve disporre delle autorizzazioni per la chiave KMS richiesta da DynamoDB. È possibile fornire queste autorizzazioni in una policy chiave, in una policy IAM o mediante una concessione.

Le autorizzazioni minime richieste da DynamoDB per una chiave gestita dal cliente sono:

- kms:Encrypt
- kms:Decrypt
- kms: * (per e) ReEncrypt kms: ReEncryptFrom kms: ReEncryptTo
- kms: GenerateDataKey * (per kms: GenerateDataKey e kms:) GenerateDataKeyWithoutPlaintext

- km: DescribeKey
- km: CreateGrant

Ad esempio, la policy della chiave di esempio riportata di seguito fornisce solo le autorizzazioni necessarie. La policy ha i seguenti effetti:

- Consente a DynamoDB di utilizzare la chiave KMS nelle operazioni di crittografia e di creare concessioni, ma solo quando agisce per conto dei principali dell'account dotati dell'autorizzazione per l'utilizzo di DynamoDB. Se le entità specificate nell'istruzione della policy non dispongono dell'autorizzazione per l'utilizzo di DynamoDB, la chiamata non riesce, anche quando proviene dal servizio DynamoDB.
- La chiave kms: ViaService condition consente le autorizzazioni solo quando la richiesta proviene da DynamoDB per conto dei principali elencati nell'informativa. Tali entità principali non possono chiamare direttamente queste operazioni. Nota: il valore `kms:ViaService`,`dynamodb.*.amazonaws.com`, ha un asterisco (*) nella posizione Regione. DynamoDB richiede l'autorizzazione per essere indipendente da qualsiasi Regione AWS particolare in modo da poter effettuare chiamate interregionali per supportare le tabelle globali DynamoDB.
- Fornisce agli amministratori delle chiavi KMS (utenti che possono assumere il ruolo `db-team`) l'accesso in sola lettura alla chiave KMS e l'autorizzazione per revocare le concessioni, incluse le concessioni richieste da DynamoDB per proteggere la tabella.

Prima di utilizzare una policy chiave di esempio, sostituisci i principali di esempio con i principali effettivi del tuo. Account 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": "*"
    }
  ]
}
```

Utilizzo di concessioni per autorizzare DynamoDB

Oltre alle policy delle chiavi, DynamoDB usa le concessioni per impostare le autorizzazioni per una chiave gestita dal cliente o per la Chiave gestita da AWS per DynamoDB (`aws/dynamodb`). Per visualizzare le sovvenzioni relative a una chiave KMS nel tuo account, usa l'operazione. ListGrants DynamoDB non ha bisogno di concessioni, o di autorizzazioni aggiuntive, per utilizzare la Chiave di proprietà di AWS per proteggere la tabella.

DynamoDB usa le autorizzazioni per la concessione quando esegue la manutenzione dei sistemi in background e le attività di protezione dei dati continuative nel tempo. Utilizza inoltre concessioni per generare le chiavi delle tabelle.

Ogni concessione è specifica di una tabella. Se l'account include più tabelle crittografate con la stessa chiave KMS, è prevista una concessione di ciascun tipo per ogni tabella. La concessione è vincolata dal contesto di crittografia DynamoDB, che include il nome della tabella e l' Account AWS ID, e include l'autorizzazione a ritirare la concessione se non è più necessaria.

Per creare le concessioni, DynamoDB deve disporre dell'autorizzazione per chiamare `CreateGrant` per conto dell'utente che ha creato la tabella crittografata. Infatti Chiavi gestite da AWS, DynamoDB `kms:CreateGrant` ottiene l'autorizzazione dalla policy chiave, che consente agli utenti dell'account di CreateGrantrichiamare la chiave KMS solo quando DynamoDB effettua la richiesta per conto di un utente autorizzato.

La policy delle chiavi può anche consentire all'account di revocare la concessione sulla chiave KMS. Tuttavia, se revochi la concessione sulla tabella crittografata attiva, DynamoDB non sarà in grado di proteggere e mantenere la tabella.

Contesto di crittografia DynamoDB

Un contesto di crittografia è un set di coppie chiave-valore che contiene dati arbitrari non segreti. Quando includi un contesto di crittografia in una richiesta di crittografia dei dati, associa AWS KMS crittograficamente il contesto di crittografia ai dati crittografati. lo stesso contesto di crittografia sia necessario per decrittografare i dati.

DynamoDB utilizza lo stesso contesto di crittografia in AWS KMS tutte le operazioni crittografiche. Se utilizzi una chiave gestita dal cliente o una Chiave gestita da AWS per proteggere la tabella DynamoDB, puoi utilizzare il contesto di crittografia per identificare l'utilizzo della chiave KMS nei record e nei log di audit. Viene inoltre visualizzato in testo non crittografato nei log, ad esempio AWS CloudTrailAmazon Logs. CloudWatch

Il contesto di crittografia può anche essere usato come una condizione per le autorizzazioni in policy e concessioni. DynamoDB utilizza il contesto di crittografia per limitare le concessioni che consentono l'accesso alla chiave gestita dal cliente Chiave gestita da AWS o nell'account e nella regione dell'utente.

Nelle sue richieste a AWS KMS, DynamoDB utilizza un contesto di crittografia con due coppie chiave-valore.

```
"encryptionContextSubset": {
    "aws:dynamodb:tableName": "Books"
    "aws:dynamodb:subscriberId": "111122223333"
}
```

- Tabella — La prima coppia chiave-valore identificherà la tabella che DynamoDB sta crittografando. La chiave è `aws:dynamodb:tableName`. Il valore è il nome della tabella.

  ```
  "aws:dynamodb:tableName": "<table-name>"
  ```

  Ad esempio:

  ```
  "aws:dynamodb:tableName": "Books"
  ```

- Account — La seconda coppia chiave-valore identificherà l' Account AWS. La chiave è `aws:dynamodb:subscriberId`. Il valore è l'ID dell'account.

  ```
  "aws:dynamodb:subscriberId": "<account-id>"
  ```

  Ad esempio:

  ```
  "aws:dynamodb:subscriberId": "111122223333"
  ```

Monitoraggio dell'interazione DynamoDB con AWS KMS

Se utilizzi una chiave gestita dal cliente o una Chiave gestita da AWSper proteggere le tue tabelle DynamoDB, puoi AWS CloudTrail utilizzare i log per tenere traccia delle richieste a cui DynamoDB invia per tuo conto. AWS KMS

Le richieste `GenerateDataKey`, `Decrypt` e `CreateGrant` vengono illustrate in questa sezione. Inoltre, DynamoDB utilizza DescribeKeyun'operazione per determinare se la chiave KMS selezionata esiste nell'account e nella regione. Utilizza anche un'RetireGrantoperazione per rimuovere una concessione quando si elimina una tabella.

GenerateDataKey

Quando si abilita la crittografia dei dati inattivi su una tabella, DynamoDB crea una chiave della tabella univoca. Invia una GenerateDataKeyrichiesta a AWS KMS che specifica la chiave KMS per la tabella.

L'evento che registra l'operazione `GenerateDataKey` è simile a quello del seguente evento di esempio. L'utente è l'account di servizio DynamoDB. I parametri includono l'Amazon Resource Name (ARN) della chiave KMS, un identificatore della chiave che richiede una chiave a 256 bit e il contesto di crittografia che identifica la tabella e il Account 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"
}
```


Decrittografia

Quando si accede a una tabella DynamoDB crittografata, Dynamo DB deve decrittare la chiave della tabella per poter decrittare le chiavi sottostanti nella gerarchia. Quindi decrittografa i dati nella tabella. Per decrittare la chiave della tabella. DynamoDB invia una richiesta Decrypt AWS KMS a che specifica la chiave KMS per la tabella.

L'evento che registra l'operazione `Decrypt` è simile a quello del seguente evento di esempio. L'utente è il vostro Account AWS principale che accede alla tabella. I parametri includono la chiave crittografata della tabella (come blob di testo cifrato) e il contesto di crittografia che identifica la tabella e il. Account AWS AWS KMS ricava l'ID della chiave KMS dal testo cifrato.


```
{
    "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

Quando si utilizza una chiave gestita dal cliente o una Chiave gestita da AWS per proteggere la tabella DynamoDB, DynamoDB usa le concessioni per consentire al servizio di eseguire la protezione continua dei dati e le attività di manutenzione e durabilità. Queste concessioni non sono necessarie per le Chiave di proprietà di AWS.

Le concessioni che DynamoDB crea sono specifiche per una tabella. Il principale nella CreateGrantrichiesta è l'utente che ha creato la tabella.

L'evento che registra l'operazione `CreateGrant` è simile a quello del seguente evento di esempio. I parametri includono l'Amazon Resource Name (ARN) della chiave KMS per la tabella, il principale dell'assegnatario e il principale pianificato per il ritiro (il servizio DynamoDB), nonché le operazioni coperte dalla concessione. Include anche un vincolo che richiede che tutte le operazioni di crittografia utilizzino il contesto di crittografia specificato.

```
{
    "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"
}
```

Gestione di tabelle crittografate in DynamoDB

Puoi usare Console di gestione AWS o the AWS Command Line Interface (AWS CLI) per specificare la chiave di crittografia su nuove tabelle e aggiornare le chiavi di crittografia sulle tabelle esistenti in Amazon DynamoDB.

Argomenti

- Definizione della chiave di crittografia per una nuova tabella
- Aggiornamento di una chiave di crittografia

Definizione della chiave di crittografia per una nuova tabella

Completare la procedura riportata di seguito per specificare la chiave di crittografia su una nuova tabella utilizzando la console Amazon DynamoDB o la AWS CLI.

Creazione di una tabella crittografata (console)

1. Accedi Console di gestione AWS e apri la console DynamoDB all'indirizzo. https://console.aws.amazon.com/dynamodb/
2. Nel riquadro di navigazione sul lato sinistro della console scegli Tables (Tabelle).
3. Scegliere Create Table (Crea tabella). Nel campo Table name (Nome tabella) immetti **`Music`**. Per la chiave primaria, immetti **`Artist`** e per la chiave di ordinamento, immetti **`SongTitle`**, entrambe come stringhe.
4. In Impostazioni, verifica che Personalizza impostazioni sia selezionato.

> **Note**
>
> Se è selezionata l'opzione Usa impostazioni predefinite, le tabelle vengono crittografate quando sono inattive senza Chiave di proprietà di AWS costi aggiuntivi.

5. In Encryption at rest, scegli un tipo di crittografia o una chiave gestita dal cliente. Chiave di proprietà di AWS Chiave gestita da AWS

   - Di proprietà di Amazon DynamoDB. AWS chiave proprietaria, specificamente posseduta e gestita da DynamoDB. Non ti viene addebitato alcun costo aggiuntivo per l'utilizzo di questa chiave.
   - AWS chiave gestita. Alias della chiave: `aws/dynamodb`. La chiave è memorizzata nel tuo account ed è gestita da AWS Key Management Service (AWS KMS). AWS KMS si applicano costi.
   - Archiviato nell'account e di proprietà e gestito dal cliente. Chiave gestita dal cliente. La chiave è memorizzata nel tuo account ed è gestita da AWS Key Management Service (AWS KMS). AWS KMS si applicano costi.

   > **Note**
   >
   > Se scegli di possedere e gestire la tua chiave, assicurati che la policy delle chiavi KMS sia impostata in modo appropriato. Per ulteriori informazioni ed esempi, consulta Policy delle chiavi per una chiave gestita dal cliente.

6. Scegli Crea per creare la tabella crittografata. Per confermare il tipo di crittografia, seleziona i dettagli della tabella nella scheda Panoramica e rivedi la sezione Ulteriori dettagli.

Creazione di una tabella crittografata (AWS CLI)

Utilizza AWS CLI per creare una tabella con la chiave predefinita Chiave di proprietà di AWS Chiave gestita da AWS, la o una chiave gestita dal cliente per Amazon DynamoDB.

Per creare una tabella crittografata con le impostazioni predefinite Chiave di proprietà di AWS

- Crea la tabella `Music` crittografata come segue.

```
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**
>
> Questa tabella è ora crittografata utilizzando l'impostazione predefinita Chiave di proprietà di AWS nell'account del servizio DynamoDB.

Per creare una tabella crittografata con Chiave gestita da AWS for DynamoDB

- Crea la tabella `Music` crittografata come segue.

```
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
```

  Lo stato `SSEDescription` della descrizione della tabella è impostato su `ENABLED` e `SSEType` è impostato su `KMS`.

```
"SSEDescription": {
  "SSEType": "KMS",
  "Status": "ENABLED",
  "KMSMasterKeyArn": "arn:aws:kms:us-east-1:123456789012:key/abcd1234-abcd-1234-
a123-ab1234a1b234",
```

```
}
```

Per creare una tabella criptata con una chiave gestita dal cliente per DynamoDB

- Crea la tabella `Music` crittografata come segue.

```
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
```

Lo stato `SSEDescription` della descrizione della tabella è impostato su `ENABLED` e `SSEType` è impostato su `KMS`.

```
"SSEDescription": {
  "SSEType": "KMS",
  "Status": "ENABLED",
  "KMSMasterKeyArn": "arn:aws:kms:us-east-1:123456789012:key/abcd1234-abcd-1234-
a123-ab1234a1b234",
}
```

Aggiornamento di una chiave di crittografia

Puoi anche utilizzare la console DynamoDB o aggiornare AWS CLI le chiavi di crittografia di una tabella esistente tra una chiave e Chiave di proprietà di AWS una Chiave gestita da AWS chiave gestita dal cliente in qualsiasi momento.

Aggiornamento di una chiave di crittografia (console)

1. Accedi Console di gestione AWS e apri la console DynamoDB all'indirizzo. https://console.aws.amazon.com/dynamodb/

2. Nel riquadro di navigazione sul lato sinistro della console scegli Tables (Tabelle).
3. Scegli la policy da aggiornare.
4. Seleziona il menu a discesa Operazioni, quindi seleziona l'opzione Aggiornamento delle impostazioni.
5. Vai alla scheda Impostazioni aggiuntive.
6. In Crittografia, scegli Gestisci crittografia.
7. Scegli un tipo di crittografia:

   - Di proprietà di Amazon DynamoDB. La AWS KMS chiave è di proprietà e gestita da DynamoDB. Non ti viene addebitato alcun costo aggiuntivo per l'utilizzo di questa chiave.
   - AWS alias chiave gestita:. `aws/dynamodb` La chiave è memorizzata nel tuo account ed è gestita da AWS Key Management Service. (AWS KMS). AWS KMS si applicano costi.
   - Archiviato nell'account e di proprietà e gestito dal cliente. La chiave è memorizzata nel tuo account ed è gestita da AWS Key Management Service. (AWS KMS). AWS KMS si applicano costi.

     > **Note**
     >
     > Se scegli di possedere e gestire la tua chiave, assicurati che la policy delle chiavi KMS sia impostata in modo appropriato. Per ulteriori informazioni, consulta Policy delle chiavi per una chiave gestita dal cliente.

   Quindi, scegli Save (Salva) per aggiornare la tabella crittografata. Per confermare il tipo di crittografia, controlla i dettagli della tabella nella scheda Overview (Panoramica).

Aggiornamento di una chiave di crittografia (AWS CLI)

Gli esempi seguenti mostrano come aggiornare una tabella crittografata utilizzando AWS CLI.

Per aggiornare una tabella crittografata con quella predefinita Chiave di proprietà di AWS

- Aggiorna la tabella `Music` crittografata, come nell'esempio seguente.

```
aws dynamodb update-table \
  --table-name Music \
  --sse-specification Enabled=false
```

> **Note**
>
> Questa tabella è ora crittografata utilizzando l'impostazione predefinita Chiave di proprietà di AWS nell'account del servizio DynamoDB.

Per aggiornare una tabella crittografata con Chiave gestita da AWS for DynamoDB

- Aggiorna la tabella `Music` crittografata, come nell'esempio seguente.

```
aws dynamodb update-table \
  --table-name Music \
  --sse-specification Enabled=true
```

Lo stato `SSEDescription` della descrizione della tabella è impostato su `ENABLED` e `SSEType` è impostato su `KMS`.

```
"SSEDescription": {
  "SSEType": "KMS",
  "Status": "ENABLED",
  "KMSMasterKeyArn": "arn:aws:kms:us-east-1:123456789012:key/abcd1234-abcd-1234-
a123-ab1234a1b234",
}
```

Per aggiornare una tabella criptata con una chiave gestita dal cliente per DynamoDB

- Aggiorna la tabella `Music` crittografata, come nell'esempio seguente.

```
aws dynamodb update-table \
  --table-name Music \
  --sse-specification Enabled=true,SSEType=KMS,KMSMasterKeyId=abcd1234-abcd-1234-
a123-ab1234a1b234
```

Lo stato `SSEDescription` della descrizione della tabella è impostato su `ENABLED` e `SSEType` è impostato su `KMS`.

```
"SSEDescription": {
  "SSEType": "KMS",
```

```
  "Status": "ENABLED",
  "KMSMasterKeyArn": "arn:aws:kms:us-east-1:123456789012:key/abcd1234-abcd-1234-
a123-ab1234a1b234",
}
```

Protezione delle connessioni DynamoDB attraverso endpoint VPC e policy IAM

Le connessioni sono protette sia tra Amazon DynamoDB e le applicazioni locali sia tra DynamoDB e altre risorse all'interno della stessa regione. AWS AWS

Policy richieste per gli endpoint

Amazon DynamoDB fornisce un'API DescribeEndpoints che consente di enumerare le informazioni sugli endpoint regionali. Per le richieste agli endpoint DynamoDB pubblici, l'API risponde indipendentemente dalla policy IAM di DynamoDB configurata, anche se esiste una negazione esplicita o implicita nella policy IAM o negli endpoint VPC. Questo perché DynamoDB ignora intenzionalmente l'autorizzazione per l'API `DescribeEndpoints`.

Per le richieste da un endpoint VPC, sia le policy IAM che del cloud privato virtuale (VPC) degli endpoint devono autorizzare la chiamata API `DescribeEndpoints` per i principali di Identity and Access Management (IAM) richiedenti utilizzando l'operazione IAM `dynamodb:DescribeEndpoints`. In caso contrario, l'accesso all'API `DescribeEndpoints` verrà negato.

Di seguito è riportato un esempio di una policy di endpoint.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:root"
            },
            "Action": "dynamodb:DescribeEndpoints",
            "Resource": "*"
```

```
        }
    ]
}
```

Traffico tra servizio e applicazioni e client locali

Sono disponibili due opzioni di connettività tra la rete privata e: AWS

- Una AWS Site-to-Site VPN connessione. Per ulteriori informazioni, consulta Che cos'è AWS Site-to-Site VPN? nella Guida per l'utente AWS Site-to-Site VPN .
- Una Direct Connect connessione. Per ulteriori informazioni, consulta Che cos'è Direct Connect? nella Guida per l'utente Direct Connect .

L'accesso a DynamoDB tramite la rete avviene tramite published. AWS APIs I client devono supportare Transport Layer Security (TLS) 1.2. È consigliabile TLS 1.3. I client devono inoltre supportare le suite di cifratura con PFS (Perfect Forward Secrecy), ad esempio Ephemeral Diffie-Hellman (DHE) o Elliptic Curve Diffie-Hellman Ephemeral (ECDHE). La maggior parte dei sistemi moderni come Java 7 e versioni successive, supporta tali modalità. Inoltre, è necessario firmare le richieste utilizzando un ID chiave di accesso e la chiave di accesso segreta associate a un principale IAM, oppure è possibile utilizzare AWS Security Token Service (STS) per generare le credenziali di sicurezza temporanee per firmare le richieste.

Traffico tra risorse AWS nella stessa Regione

Un endpoint Amazon Virtual Private Cloud (Amazon VPC) per DynamoDB è un'entità logica all'interno di un VPC che consente la connettività solo a DynamoDB. Amazon VPC instrada le richieste ad DynamoDB e reindirizza le risposte al VPC. Per ulteriori informazioni, consultare Endpoint VPC nella Guida per l'utente di Amazon VPC. Per le policy di esempio che possono essere utilizzate per controllare l'accesso da endpoint VPC, consulta Utilizzo delle policy IAM per controllare l'accesso a DynamoDB.

> **Note**
>
> Gli endpoint Amazon VPC non sono accessibili tramite o. AWS Site-to-Site VPN Direct Connect

AWS Identity and Access Management (IAM) e DynamoDB

AWS Identity and Access Management è un AWS servizio che aiuta un amministratore a controllare in modo sicuro l'accesso alle risorse. AWS Gli amministratori controllano chi può essere autenticato (che ha effettuato l'accesso) e autorizzato (che dispone di autorizzazioni) a utilizzare risorse Amazon DynamoDB e DynamoDB Accelerator. È possibile utilizzare IAM per gestire le autorizzazioni di accesso e implementare le policy di sicurezza per Amazon DynamoDB e DynamoDB Accelerator. IAM è un AWS servizio che puoi utilizzare senza costi aggiuntivi.

Argomenti

- Identity and Access Management per Amazon DynamoDB
- Utilizzo di condizioni di policy IAM per il controllo granulare degli accessi

Identity and Access Management per Amazon DynamoDB

AWS Identity and Access Management (IAM) è un servizio Servizio AWS che aiuta un amministratore a controllare in modo sicuro l'accesso alle AWS risorse. Gli amministratori IAM controllano chi può essere autenticato (chi ha effettuato l'accesso) e autorizzato (chi dispone di autorizzazioni) a utilizzare le risorse DynamoDB. IAM è un software Servizio AWS che puoi utilizzare senza costi aggiuntivi.

Argomenti

- Destinatari
- Autenticazione con identità
- Gestione dell’accesso tramite policy
- Funzionamento di Amazon DynamoDB con IAM
- Esempi di policy basate su identità per Amazon DynamoDB
- Risoluzione dei problemi relativi all'identità e all'accesso di Amazon DynamoDB
- Policy IAM per impedire l'acquisto di capacità riservata DynamoDB

Destinatari

Il modo in cui utilizzi AWS Identity and Access Management (IAM) varia in base al tuo ruolo:

- Utente del servizio: richiedi le autorizzazioni all'amministratore se non riesci ad accedere alle funzionalità (vedi Risoluzione dei problemi relativi all'identità e all'accesso di Amazon DynamoDB)
- Amministratore del servizio: determina l'accesso degli utenti e invia le richieste di autorizzazione (vedi Funzionamento di Amazon DynamoDB con IAM)
- Amministratore IAM: scrivi policy per gestire l'accesso (vedi Esempi di policy basate su identità per Amazon DynamoDB)

Autenticazione con identità

L'autenticazione è il modo in cui accedi AWS utilizzando le tue credenziali di identità. Devi autenticarti come utente IAM o assumendo un ruolo IAM. Utente root dell'account AWS

Puoi accedere come identità federata utilizzando credenziali provenienti da una fonte di identità come AWS IAM Identity Center (IAM Identity Center), autenticazione Single Sign-On o credenziali. Google/ Facebook Per maggiori informazioni sull'accesso, consultare la sezione Come accedere a Account AWS nella Guida per l'utente di Accedi ad AWS .

Per l'accesso programmatico, AWS fornisce un SDK e una CLI per firmare crittograficamente le richieste. Per ulteriori informazioni, consulta AWS Signature Version 4 per le richieste API nella Guida per l'utente IAM.

Account AWS utente root

Quando si crea un Account AWS, si inizia con un'identità di accesso denominata utente Account AWS root che ha accesso completo a tutte Servizi AWS le risorse. Si consiglia vivamente di non utilizzare l'utente root per le attività quotidiane. Per le attività che richiedono le credenziali come utente root, consulta Attività che richiedono le credenziali dell'utente root nella Guida per l'utente IAM.

Identità federata

Come procedura ottimale, richiedi agli utenti umani di utilizzare la federazione con un provider di identità per accedere Servizi AWS utilizzando credenziali temporanee.

Un'identità federata è un utente della directory aziendale, del provider di identità Web o Directory Service che accede Servizi AWS utilizzando le credenziali di una fonte di identità. Le identità federate assumono ruoli che forniscono credenziali temporanee.

Per la gestione centralizzata degli accessi, consigliamo di utilizzare AWS IAM Identity Center. Per ulteriori informazioni, consulta Cos'è IAM Identity Center? nella Guida per l'utente di AWS IAM Identity Center .

Utenti e gruppi IAM

Un utente IAM è un'identità che dispone di autorizzazioni specifiche per una singola persona o applicazione. Ti consigliamo di utilizzare credenziali temporanee invece di utenti IAM con credenziali a lungo termine. Per ulteriori informazioni, consulta Richiedere agli utenti umani di utilizzare la federazione con un provider di identità per accedere AWS utilizzando credenziali temporanee nella Guida per l'utente IAM.

Un gruppo IAM specifica una raccolta di utenti IAM e semplifica la gestione delle autorizzazioni per gestire gruppi di utenti di grandi dimensioni. Per ulteriori informazioni, consulta Casi d'uso per utenti IAM nella Guida per l'utente IAM.

Ruoli IAM

Un ruolo IAM è un'identità con autorizzazioni specifiche che fornisce credenziali temporanee. Puoi assumere un ruolo passando da un ruolo utente a un ruolo IAM (console) o chiamando un'operazione AWS CLI o AWS API. Per ulteriori informazioni, consulta Metodi per assumere un ruolo nella Guida per l'utente IAM.

I ruoli IAM sono utili per l'accesso federato degli utenti, le autorizzazioni utente IAM temporanee, l'accesso tra account, l'accesso tra servizi e le applicazioni in esecuzione su Amazon. EC2 Per maggiori informazioni, consultare Accesso a risorse multi-account in IAM nella Guida per l'utente IAM.

Gestione dell'accesso tramite policy

Puoi controllare l'accesso AWS creando policy e collegandole a identità o risorse. AWS Una policy definisce le autorizzazioni quando è associata a un'identità o a una risorsa. AWS valuta queste politiche quando un preside effettua una richiesta. La maggior parte delle politiche viene archiviata AWS come documenti JSON. Per maggiori informazioni sui documenti delle policy JSON, consulta Panoramica delle policy JSON nella Guida per l'utente IAM.

Utilizzando le policy, gli amministratori specificano chi ha accesso a cosa definendo quale principale può eseguire azioni su quali risorse e in quali condizioni.

Per impostazione predefinita, utenti e ruoli non dispongono di autorizzazioni. Un amministratore IAM crea le policy IAM e le aggiunge ai ruoli, che gli utenti possono quindi assumere. Le policy IAM definiscono le autorizzazioni indipendentemente dal metodo utilizzato per eseguirle.

Policy basate sull'identità

Le policy basate su identità sono documenti di policy di autorizzazione JSON che è possibile collegare a un'identità (utente, gruppo o ruolo). Tali policy controllano le operazioni autorizzate per l'identità, nonché le risorse e le condizioni in cui possono essere eseguite. Per informazioni su come creare una policy basata su identità, consultare Definizione di autorizzazioni personalizzate IAM con policy gestite dal cliente nella Guida per l'utente IAM.

Le policy basate sull'identità possono essere policy in linea (incorporate direttamente in una singola identità) o policy gestite (policy autonome collegate a più identità). Per informazioni su come scegliere tra una policy gestita o una policy inline, consultare Scelta fra policy gestite e policy inline nella Guida per l'utente IAM.

Policy basate sulle risorse

Le policy basate su risorse sono documenti di policy JSON che è possibile collegare a una risorsa. Gli esempi includono le policy di trust dei ruoli IAM e le policy dei bucket di Amazon S3. Nei servizi che supportano policy basate sulle risorse, gli amministratori dei servizi possono utilizzarli per controllare l'accesso a una risorsa specifica. In una policy basata sulle risorse è obbligatorio specificare un'entità principale.

Le policy basate sulle risorse sono policy inline che si trovano in tale servizio. Non è possibile utilizzare le policy AWS gestite di IAM in una policy basata sulle risorse.

Altri tipi di policy

AWS supporta tipi di policy aggiuntivi che possono impostare le autorizzazioni massime concesse dai tipi di policy più comuni:

- Limiti delle autorizzazioni: impostano il numero massimo di autorizzazioni che una policy basata su identità ha la possibilità di concedere a un'entità IAM. Per ulteriori informazioni, consulta Limiti delle autorizzazioni per le entità IAM nella Guida per l'utente IAM.
- Politiche di controllo del servizio (SCPs): specificano le autorizzazioni massime per un'organizzazione o un'unità organizzativa in. AWS Organizations Per ulteriori informazioni, consultare Policy di controllo dei servizi nella Guida per l'utente di AWS Organizations .
- Politiche di controllo delle risorse (RCPs): imposta le autorizzazioni massime disponibili per le risorse nei tuoi account. Per ulteriori informazioni, consulta Politiche di controllo delle risorse (RCPs) nella Guida per l'AWS Organizations utente.

- Le policy di sessione sono policy avanzate che si passano come parametro quando si crea una sessione temporanea per un ruolo o un utente federato. Per maggiori informazioni, consultare Policy di sessione nella Guida per l'utente IAM.

Più tipi di policy

Quando a una richiesta si applicano più tipi di policy, le autorizzazioni risultanti sono più complicate da comprendere. Per scoprire come si AWS determina se consentire o meno una richiesta quando sono coinvolti più tipi di policy, consulta Logica di valutazione delle policy nella IAM User Guide.

Funzionamento di Amazon DynamoDB con IAM

Prima di utilizzare IAM per gestire l'accesso a DynamoDB, scopri quali funzionalità di IAM sono disponibili per l'uso con DynamoDB.

Funzionalità IAM	Supporto in DynamoDB
Policy basate sull'identità	Sì
Policy basate su risorse	Sì
Operazioni di policy	Sì
Risorse relative alle policy	Sì
Chiavi di condizione delle policy	Sì
ACLs	No
ABAC (tag nelle policy)	Sì
Credenziali temporanee	Sì
Autorizzazioni del principale	Sì
Ruoli di servizio	Sì
Ruoli collegati al servizio	Sì

Per avere una visione di alto livello di come DynamoDB e AWS altri servizi funzionano con la maggior parte delle funzionalità IAM, AWS consulta i servizi che funzionano con IAM nella IAM User Guide.

Policy basate su identità per DynamoDB

Supporta le policy basate sull'identità: sì

Le policy basate sull'identità sono documenti di policy di autorizzazione JSON che è possibile allegare a un'identità (utente, gruppo di utenti o ruolo IAM). Tali policy definiscono le operazioni che utenti e ruoli possono eseguire, su quali risorse e in quali condizioni. Per informazioni su come creare una policy basata su identità, consulta Definizione di autorizzazioni personalizzate IAM con policy gestite dal cliente nella Guida per l'utente di IAM.

Con le policy basate sull'identità di IAM, è possibile specificare quali operazioni e risorse sono consentite o respinte, nonché le condizioni in base alle quali le operazioni sono consentite o respinte. Per informazioni su tutti gli elementi utilizzabili in una policy JSON, consulta Guida di riferimento agli elementi delle policy JSON IAM nella Guida per l'utente IAM.

Esempi di policy basate su identità per DynamoDB

Per visualizzare esempi di policy basate su identità DynamoDB, consultare Esempi di policy basate su identità per Amazon DynamoDB.

Policy basate su risorse all'interno di DynamoDB

Supporta le policy basate sulle risorse: sì

Le policy basate su risorse sono documenti di policy JSON che è possibile collegare a una risorsa. Esempi di policy basate sulle risorse sono le policy di attendibilità dei ruoli di IAM e le policy dei bucket di Amazon S3. Nei servizi che supportano policy basate sulle risorse, gli amministratori dei servizi possono utilizzarli per controllare l'accesso a una risorsa specifica. Quando è collegata a una risorsa, una policy definisce le operazioni che un principale può eseguire su tale risorsa e a quali condizioni. In una policy basata sulle risorse è obbligatorio specificare un'entità principale. I principali possono includere account, utenti, ruoli, utenti federati o. Servizi AWS

Per consentire l'accesso multi-account, è possibile specificare un intero account o entità IAM in un altro account come entità principale in una policy basata sulle risorse. Per ulteriori informazioni, consulta Accesso a risorse multi-account in IAM nella Guida per l'utente IAM.

Operazioni delle policy per DynamoDB

Supporta le operazioni di policy: si

Gli amministratori possono utilizzare le policy AWS JSON per specificare chi ha accesso a cosa. In altre parole, quale entità principale può eseguire operazioni su quali risorse e in quali condizioni.

L'elemento `Action` di una policy JSON descrive le operazioni che è possibile utilizzare per consentire o negare l'accesso a un criterio. Includere le operazioni in una policy per concedere le autorizzazioni a eseguire l'operazione associata.

Per visualizzare un elenco di operazioni DynamoDB, consultare Operazioni definite da Amazon DynamoDB in Informazioni di riferimento sull'autorizzazione del servizio.

Le operazioni delle policy in DynamoDB utilizzano il seguente prefisso prima dell'operazione:

```
aws
```

Per specificare più operazioni in una sola istruzione, occorre separarle con la virgola.

```
"Action": [
      "aws:action1",
      "aws:action2"
         ]
```

Per visualizzare esempi di policy basate su identità DynamoDB, consultare Esempi di policy basate su identità per Amazon DynamoDB.

Operazioni delle policy per DynamoDB

Supporta le risorse relative alle policy: sì

Gli amministratori possono utilizzare le policy AWS JSON per specificare chi ha accesso a cosa. In altre parole, quale entità principale può eseguire operazioni su quali risorse e in quali condizioni.

L'elemento JSON `Resource` della policy specifica l'oggetto o gli oggetti ai quali si applica l'operazione. Come best practice, specifica una risorsa utilizzando il suo nome della risorsa Amazon (ARN). Per le azioni che non supportano le autorizzazioni a livello di risorsa, utilizzare un carattere jolly (*) per indicare che l'istruzione si applica a tutte le risorse.

```
"Resource": "*"
```

Per visualizzare un elenco dei tipi di risorse DynamoDB e ARNs relativi, consulta Resources defined by Amazon DynamoDB nel Service Authorization Reference. Per informazioni sulle operazioni con cui è possibile specificare l'ARN di ogni risorsa, consultare Operazioni definite da Amazon DynamoDB.

Per visualizzare esempi di policy basate su identità DynamoDB, consultare Esempi di policy basate su identità per Amazon DynamoDB.

Chiavi di condizione delle policy per DynamoDB

Supporta le chiavi di condizione delle policy specifiche del servizio: sì

Gli amministratori possono utilizzare le policy AWS JSON per specificare chi ha accesso a cosa. In altre parole, quale entità principale può eseguire operazioni su quali risorse e in quali condizioni.

L'elemento `Condition` specifica quando le istruzioni vengono eseguite in base a criteri definiti. È possibile compilare espressioni condizionali che utilizzano operatori di condizione, ad esempio uguale a o minore di, per soddisfare la condizione nella policy con i valori nella richiesta. Per visualizzare tutte le chiavi di condizione AWS globali, consulta le chiavi di contesto delle condizioni AWS globali nella Guida per l'utente IAM.

Per visualizzare un elenco di chiavi di condizione DynamoDB, consultare Chiavi di condizione per Amazon DynamoDB in Informazioni di riferimento sull'autorizzazione del servizio. Per informazioni su operazioni e risorse con cui è possibile utilizzare una chiave di condizione, consultare Operazioni definite da Amazon DynamoDB.

Per visualizzare esempi di policy basate su identità DynamoDB, consultare Esempi di policy basate su identità per Amazon DynamoDB.

Liste di controllo degli accessi (ACLs) in DynamoDB

Supporti ACLs: No

Le liste di controllo degli accessi (ACLs) controllano quali principali (membri dell'account, utenti o ruoli) dispongono delle autorizzazioni per accedere a una risorsa. ACLs sono simili alle politiche basate sulle risorse, sebbene non utilizzino il formato del documento di policy JSON.

Controllo degli accessi basato su attributi (ABAC) con DynamoDB

Supporta ABAC (tag nelle policy): sì

Il controllo degli accessi basato su attributi (ABAC) è una strategia di autorizzazione che definisce le autorizzazioni in base agli attributi, chiamati tag. È possibile allegare tag a entità e AWS risorse IAM, quindi progettare politiche ABAC per consentire operazioni quando il tag del principale corrisponde al tag sulla risorsa.

Per controllare l'accesso basato su tag, fornire informazioni sui tag nell'elemento condizione di una policy utilizzando le chiavi di condizione `aws:ResourceTag/key-name`, `aws:RequestTag/key-name` o `aws:TagKeys`.

Se un servizio supporta tutte e tre le chiavi di condizione per ogni tipo di risorsa, il valore per il servizio è Sì. Se un servizio supporta tutte e tre le chiavi di condizione solo per alcuni tipi di risorsa, allora il valore sarà Parziale.

Per maggiori informazioni su ABAC, consulta Definizione delle autorizzazioni con autorizzazione ABAC nella Guida per l'utente di IAM. Per visualizzare un tutorial con i passaggi per l'impostazione di ABAC, consulta Utilizzo del controllo degli accessi basato su attributi (ABAC) nella Guida per l'utente di IAM.

Utilizzo di credenziali temporanee con DynamoDB

Supporta le credenziali temporanee: sì

Le credenziali temporanee forniscono l'accesso a breve termine alle AWS risorse e vengono create automaticamente quando si utilizza la federazione o si cambia ruolo. AWS consiglia di generare dinamicamente credenziali temporanee anziché utilizzare chiavi di accesso a lungo termine. Per ulteriori informazioni, consulta Credenziali di sicurezza temporanee in IAM e Servizi AWS compatibili con IAM nella Guida per l'utente IAM.

Autorizzazioni del principale tra servizi per DynamoDB

Supporta l'inoltro delle sessioni di accesso (FAS): sì

Le sessioni di accesso inoltrato (FAS) utilizzano le autorizzazioni del principale che chiama e, in combinazione con la richiesta Servizio AWS, Servizio AWS per effettuare richieste ai servizi downstream. Per i dettagli delle policy relative alle richieste FAS, consulta Forward access sessions.

Ruoli di servizio per DynamoDB

Supporta i ruoli di servizio: sì

Un ruolo di servizio è un ruolo IAM che un servizio assume per eseguire operazioni per tuo conto. Un amministratore IAM può creare, modificare ed eliminare un ruolo di servizio dall'interno di IAM. Per

ulteriori informazioni, consulta la sezione Create a role to delegate permissions to an Servizio AWS nella Guida per l'utente IAM.

> **Warning**
>
> La modifica delle autorizzazioni per un ruolo di servizio potrebbe compromettere la funzionalità di DynamoDB. Modifica i ruoli di servizio solo quando DynamoDB fornisce le indicazioni per farlo.

Ruoli collegati ai servizi creati per DynamoDB

Supporta i ruoli collegati ai servizi: sì

Un ruolo collegato al servizio è un tipo di ruolo di servizio collegato a un. Servizio AWS Il servizio può assumere il ruolo per eseguire un'azione per tuo conto. I ruoli collegati al servizio vengono visualizzati nel tuo account Account AWS e sono di proprietà del servizio. Un amministratore IAM può visualizzare le autorizzazioni per i ruoli collegati al servizio, ma non modificarle.

Per ulteriori informazioni su come creare e gestire i ruoli collegati ai servizi, consulta Servizi AWS supportati da IAM. Trova un servizio nella tabella che include un `Yes` nella colonna Service-linked role (Ruolo collegato ai servizi). Scegli il collegamento Sì per visualizzare la documentazione relativa al ruolo collegato ai servizi per tale servizio.

Ruoli collegati al servizio supportati in DynamoDB

I seguenti ruoli collegati al servizio sono supportati in DynamoDB.

- DynamoDB utilizza il AWSServiceRoleForDynamoDBReplicationruolo collegato ai servizi per la replica globale delle tabelle su tutto il territorio. Regioni AWS Vedi Sicurezza per tabelle globali DynamoDB per ulteriori informazioni sul ruolo collegato al servizio. AWSServiceRoleForDynamoDBReplication
- DynamoDB Accelerator (DAX) utilizza il AWSServiceRoleForruolo DAX collegato ai servizi per la configurazione e la manutenzione di un cluster DAX. Per ulteriori informazioni sul ruolo collegato ai servizi DAX, vederethe section called "Utilizzo di ruoli collegati ai servizi per DAX". AWSService RoleFor

Oltre a questi ruoli collegati al servizio DynamoDB, DynamoDB utilizza il servizio Application Auto Scaling per gestire automaticamente le impostazioni di throughput nelle tabelle della modalità

con capacità allocata. Il servizio Application Auto Scaling utilizza il ruolo collegato al servizio AWSServiceRoleForApplicationAutoScaling_DynamoDBTableper gestire le impostazioni di throughput nelle tabelle DynamoDB che hanno l'auto scaling abilitato. Consulta Ruoli collegati ai servizi per Application Auto Scaling per ulteriori informazioni.

Esempi di policy basate su identità per Amazon DynamoDB

Per impostazione predefinita, gli utenti e i ruoli non dispongono dell'autorizzazione per creare o modificare risorse DynamoDB. Per concedere agli utenti l'autorizzazione a eseguire operazioni sulle risorse di cui hanno bisogno, un amministratore IAM può creare policy IAM.

Per informazioni su come creare una policy basata su identità IAM utilizzando questi documenti di policy JSON di esempio, consulta Creazione di policy IAM (console) nella Guida per l'utente IAM.

Per dettagli sulle azioni e sui tipi di risorse definiti da DynamoDB, incluso il formato di per ogni tipo di ARNs risorsa, consulta Azioni, risorse e chiavi di condizione per Amazon DynamoDB nel Service Authorization Reference.

Argomenti

- Best practice per le policy
- Utilizzo della console DynamoDB
- Consentire agli utenti di visualizzare le loro autorizzazioni
- Utilizzo di policy basate su identità con Amazon DynamoDB

Best practice per le policy

Le policy basate su identità determinano se qualcuno può creare, accedere o eliminare risorse DynamoDB nel tuo account. Queste operazioni possono comportare costi aggiuntivi per l' Account AWS. Quando si creano o modificano policy basate sull'identità, seguire queste linee guida e raccomandazioni:

- Inizia con le politiche AWS gestite e passa alle autorizzazioni con privilegi minimi: per iniziare a concedere autorizzazioni a utenti e carichi di lavoro, utilizza le politiche gestite che concedono le autorizzazioni per molti casi d'uso comuni.AWS Sono disponibili nel tuo. Account AWS Ti consigliamo di ridurre ulteriormente le autorizzazioni definendo politiche gestite dai AWS clienti specifiche per i tuoi casi d'uso. Per maggiori informazioni, consulta Policy gestite da AWS o Policy gestite da AWS per le funzioni dei processi nella Guida per l'utente di IAM.

- Applicazione delle autorizzazioni con privilegio minimo - Quando si impostano le autorizzazioni con le policy IAM, concedere solo le autorizzazioni richieste per eseguire un'attività. È possibile farlo definendo le azioni che possono essere intraprese su risorse specifiche in condizioni specifiche, note anche come autorizzazioni con privilegio minimo. Per maggiori informazioni sull'utilizzo di IAM per applicare le autorizzazioni, consulta Policy e autorizzazioni in IAM nella Guida per l'utente di IAM.
- Condizioni d'uso nelle policy IAM per limitare ulteriormente l'accesso - Per limitare l'accesso a operazioni e risorse è possibile aggiungere una condizione alle policy. Ad esempio, è possibile scrivere una condizione di policy per specificare che tutte le richieste devono essere inviate utilizzando SSL. Puoi anche utilizzare le condizioni per concedere l'accesso alle azioni del servizio se vengono utilizzate tramite uno specifico Servizio AWS, ad esempio CloudFormation. Per maggiori informazioni, consulta la sezione Elementi delle policy JSON di IAM: condizione nella Guida per l'utente di IAM.
- Utilizzo di IAM Access Analyzer per convalidare le policy IAM e garantire autorizzazioni sicure e funzionali - IAM Access Analyzer convalida le policy nuove ed esistenti in modo che aderiscano alla sintassi della policy IAM (JSON) e alle best practice di IAM. IAM Access Analyzer offre oltre 100 controlli delle policy e consigli utili per creare policy sicure e funzionali. Per maggiori informazioni, consulta Convalida delle policy per il Sistema di analisi degli accessi IAM nella Guida per l'utente di IAM.
- Richiedi l'autenticazione a più fattori (MFA): se hai uno scenario che richiede utenti IAM o un utente root nel Account AWS tuo, attiva l'MFA per una maggiore sicurezza. Per richiedere la MFA quando vengono chiamate le operazioni API, aggiungere le condizioni MFA alle policy. Per maggiori informazioni, consulta Protezione dell'accesso API con MFA nella Guida per l'utente di IAM.

Per maggiori informazioni sulle best practice in IAM, consulta Best practice di sicurezza in IAM nella Guida per l'utente di IAM.

Utilizzo della console DynamoDB

Per accedere alla console Amazon DynamoDB, è necessario disporre di un set di autorizzazioni minimo. Queste autorizzazioni devono consentirti di elencare e visualizzare i dettagli sulle risorse DynamoDB presenti nel tuo. Account AWS Se si cra una policy basata sull'identità più restrittiva rispetto alle autorizzazioni minime richieste, la console non funzionerà nel modo previsto per le entità (utenti o ruoli) associate a tale policy.

Non è necessario consentire autorizzazioni minime di console per gli utenti che effettuano chiamate solo verso o l' AWS CLI API. AWS Al contrario, è opportuno concedere l'accesso solo alle operazioni che corrispondono all'operazione API che stanno cercando di eseguire.

Per garantire che utenti e ruoli possano ancora utilizzare la console DynamoDB, collega anche `ConsoleAccess` DynamoDB o la policy gestita alle entità `ReadOnly` AWS . Per maggiori informazioni, consulta Aggiunta di autorizzazioni a un utente nella Guida per l'utente di IAM.

Consentire agli utenti di visualizzare le loro autorizzazioni

Questo esempio mostra in che modo è possibile creare una policy che consente agli utenti IAM di visualizzare le policy inline e gestite che sono collegate alla relativa identità utente. Questa policy include le autorizzazioni per completare questa azione sulla console o utilizzando programmaticamente l'API o. AWS CLI 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": "*"
        }
    ]
}
```

Utilizzo di policy basate su identità con Amazon DynamoDB

Questo argomento tratta l'uso di politiche basate sull'identità AWS Identity and Access Management (IAM) con Amazon DynamoDB e fornisce esempi. Gli esempi illustrano come un amministratore account può collegare policy di autorizzazioni a identità IAM (ovvero utenti, gruppi e ruoli) e quindi concedere autorizzazioni per eseguire operazioni sulle risorse di Amazon DynamoDB.

In questa sezione vengono trattati gli argomenti seguenti:

- Autorizzazioni IAM necessarie per utilizzare la console Amazon DynamoDB
- AWS politiche IAM gestite (predefinite) per Amazon DynamoDB
- Esempi di policy gestite dal cliente

Di seguito è riportato un esempio di policy delle autorizzazioni.

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 politica precedente contiene un'istruzione che concede le autorizzazioni per tre azioni DynamoDB (`dynamodb:DescribeTabledynamodb:Query`, e`dynamodb:Scan`) su una tabella nella `us-west-2` AWS regione, di proprietà dell'account specificato da. AWS *`account-id`* Il nome della risorsa Amazon (ARN) nel valore `Resource` specifica la tabella a cui si applicano le autorizzazioni.

Autorizzazioni IAM necessarie per utilizzare la console Amazon DynamoDB

Per lavorare con la console DynamoDB, un utente deve disporre di un set minimo di autorizzazioni che gli consentano di utilizzare le risorse DynamoDB del proprio AWS account. Oltre a queste autorizzazioni DynamoDB, la console richiede le autorizzazioni dei seguenti servizi:

- CloudWatch Autorizzazioni Amazon per visualizzare metriche e grafici.
- AWS Data Pipeline autorizzazioni per esportare e importare dati DynamoDB.
- AWS Identity and Access Management autorizzazioni per accedere ai ruoli necessari per le esportazioni e le importazioni.
- Autorizzazioni di Amazon Simple Notification Service per avvisarti ogni volta che viene attivato un CloudWatch allarme.
- AWS Lambda autorizzazioni per elaborare i record DynamoDB Streams.

Se decidi di creare una policy IAM più restrittiva delle autorizzazioni minime richieste, la console non funzionerà come previsto per gli utenti con tale policy IAM. Per garantire che tali utenti possano continuare a utilizzare la console DynamoDB, allega anche `AmazonDynamoDBReadOnlyAccess` AWS la policy gestita all'utente, come descritto in. AWS politiche IAM gestite (predefinite) per Amazon DynamoDB

Non è necessario consentire autorizzazioni minime per la console per gli utenti che effettuano chiamate solo verso l' AWS CLI API di Amazon DynamoDB.

> **Note**
>
> Se fai riferimento a un endpoint VPC, dovrai anche autorizzare la chiamata DescribeEndpoints API per i principali IAM richiedenti con l'azione IAM (dynamodb:). DescribeEndpoints Per ulteriori informazioni, consulta Policy richieste per gli endpoint.

AWS politiche IAM gestite (predefinite) per Amazon DynamoDB

AWS affronta alcuni casi d'uso comuni fornendo politiche IAM autonome create e amministrate da. AWS Queste policy AWS gestite concedono le autorizzazioni necessarie per i casi d'uso comuni in modo da evitare di dover verificare quali autorizzazioni sono necessarie. Per ulteriori informazioni, consulta Policy gestite da AWS nella Guida per l'utente di IAM.

Le seguenti policy AWS gestite, che puoi allegare agli utenti del tuo account, sono specifiche di DynamoDB e sono raggruppate per scenario d'uso:

- AmazonDynamoDBReadOnlyAccess— Concede l'accesso in sola lettura alle risorse DynamoDB tramite. Console di gestione AWS
- AmazonDynamoDBFullAccesso: garantisce l'accesso completo alle risorse DynamoDB tramite. Console di gestione AWS

Puoi rivedere queste politiche di autorizzazione AWS gestite accedendo alla console IAM e cercando lì policy specifiche.

> **Important**
>
> La best practice consiste nel creare policy IAM personalizzate che forniscano il privilegio minimo agli utenti, ai ruoli o ai gruppi che ne hanno bisogno.

Esempi di policy gestite dal cliente

In questa sezione sono riportati esempi di policy che concedono autorizzazioni per diverse operazioni DynamoDB. Queste politiche funzionano quando si utilizza AWS SDKs o il AWS CLI. Quando si utilizza la console, è necessario concedere autorizzazioni aggiuntive che sono specifiche della console. Per ulteriori informazioni, consulta Autorizzazioni IAM necessarie per utilizzare la console Amazon DynamoDB.

> **Note**
>
> Tutti i seguenti esempi di politiche utilizzano una delle AWS regioni e contengono nomi di account IDs e tabelle fittizi.

Esempi:

- Policy IAM per concedere autorizzazioni a tutte le azioni DynamoDB in una tabella
- Policy IAM per concedere le autorizzazioni di sola lettura su elementi in una tabella DynamoDB
- Policy IAM per concedere l'accesso a una tabella DynamoDB specifica e ai relativi indici
- Policy IAM per l'accesso in lettura, scrittura, aggiornamento ed eliminazione su una tabella DynamoDB
- Politica IAM per separare gli ambienti DynamoDB nello stesso account AWS
- Policy IAM per impedire l'acquisto di capacità riservata DynamoDB
- Policy IAM per concedere l'accesso in lettura solo per un flusso DynamoDB (non per la tabella)
- Politica IAM per consentire a una AWS Lambda funzione di accedere ai record di flusso DynamoDB
- Policy IAM per l'accesso in lettura e scrittura a un cluster DynamoDB Accelerator (DAX)

La Guida per l'utente di IAM, include tre ulteriori esempi di DynamoDB:

- Amazon DynamoDB: consente l'accesso a una tabella specifica
- Amazon DynamoDB: consente l'accesso a colonne specifiche
- Amazon DynamoDB: consente l'accesso a livello di riga a DynamoDB in base a un ID Amazon Cognito

Policy IAM per concedere autorizzazioni a tutte le azioni DynamoDB in una tabella

La seguente policy concede le autorizzazioni per tutte le operazioni DynamoDB su una tabella denominata `Books`. L'ARN della risorsa specificato in `Resource` identifica una tabella in una regione specifica. AWS Se sostituisci il nome della tabella `Books` nell'ARN`Resource` con un carattere jolly (*), tutte le operazioni DynamoDB saranno consentite su tutte le tabelle nell'account. Valuta attentamente le possibili implicazioni per la sicurezza prima di utilizzare un carattere jolly in questa o qualsiasi altra policy IAM.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllAPIActionsOnBooks",
```

```
            "Effect": "Allow",
            "Action": "dynamodb:*",
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
        }
    ]
}
```

> **Note**
>
> Questo è un esempio di utilizzo di un carattere jolly (*) per consentire tutte le operazioni, tra cui l'amministrazione, le operazioni dei dati, il monitoraggio e l'acquisto di capacità riservata DynamoDB. Invece, è una best practice specificare in modo esplicito ogni azione da concedere e solo ciò di cui ha bisogno un dato utente, ruolo o gruppo.

Policy IAM per concedere le autorizzazioni di sola lettura su elementi in una tabella DynamoDB

La seguente policy di autorizzazioni concede le autorizzazioni solo per le operazioni DynamoDB `GetItem`, `BatchGetItem`, `Scan`, `Query` e `ConditionCheckItem`, quindi imposta l'accesso in sola lettura sulla tabella `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"
        }
    ]
```

```
}
```

Policy IAM per concedere l'accesso a una tabella DynamoDB specifica e ai relativi indici

La policy seguente concede le autorizzazioni necessarie per le operazioni di modifica dei dati in una tabella DynamoDB denominata `Books` e tutti i relativi indici. Per ulteriori informazioni sul funzionamento degli indici, consulta Miglioramento dell'accesso ai dati con gli indici secondari in 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/*"
            ]
        }
    ]
}
```

Policy IAM per l'accesso in lettura, scrittura, aggiornamento ed eliminazione su una tabella DynamoDB

Utilizza questa policy per consentire all'applicazione di creare, leggere, aggiornare ed eliminare dati in tabelle, indici e flussi di Amazon DynamoDB. Sostituisci il nome AWS della regione, l'ID dell'account e il nome della tabella o il carattere jolly (*), se appropriato.

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/*"
            ]
        }
    ]
}
```

Per estendere questa policy a tutte le tabelle DynamoDB in AWS tutte le regioni per questo account, usa un carattere jolly (*) per la regione e il nome della tabella. Ad esempio:

```
"Resource":[
               "arn:aws:dynamodb:*:123456789012:table/*",
               "arn:aws:dynamodb:*:123456789012:table/*/index/*"
               ]
```

Politica IAM per separare gli ambienti DynamoDB nello stesso account AWS

Si supponga di avere ambienti separati in cui ogni ambiente mantiene la propria versione di una tabella denominata `ProductCatalog`. Se crei due `ProductCatalog` tabelle nello stesso AWS account, lavorare in un ambiente potrebbe influire sull'altro ambiente a causa del modo in cui sono impostate le autorizzazioni. Ad esempio, le quote sul numero di operazioni simultanee del piano di controllo (ad esempio`CreateTable`) vengono impostate a livello di account. AWS

Di conseguenza, ogni operazione in un ambiente riduce il numero di operazioni disponibili nell'altro ambiente. Vi è, inoltre, il rischio che il codice in un ambiente possa accedere per errore alle tabelle nell'altro ambiente.

> **Note**
>
> Se desideri separare i carichi di lavoro di produzione e di test per controllare il potenziale "raggio di esplosione" di un evento, la best practice consiste nel creare account AWS separati

per i carichi di lavoro di test e produzione. Per ulteriori informazioni, consulta Gestione e separazione degli account AWS.

Si supponga di avere due sviluppatori, Amit e Alice, che stanno eseguendo il test della tabella `ProductCatalog`. Invece di richiedere a ogni sviluppatore un AWS account separato, i tuoi sviluppatori possono condividere lo stesso account di test AWS . In questo account di test puoi creare per ogni sviluppatore una copia della stessa tabella su cui lavorare, ad esempio `Alice_ProductCatalog` e `Amit_ProductCatalog`. In questo caso, puoi creare gli utenti Alice e Amit nell' AWS account che hai creato per l'ambiente di test. È quindi possibile concedere a questi utenti le autorizzazioni per eseguire le operazioni DynamoDB sulle tabelle di loro proprietà.

Per concedere queste autorizzazioni utente IAM, è possibile completare una delle operazioni seguenti:

- Crea una policy separata per ogni utente e quindi collegare separatamente ogni policy al proprio utente. Ad esempio, è possibile collegare la seguente policy all'utente Alice per consentire l'accesso a tutte le operazioni DynamoDB sulla tabella `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/*"
        }
    ]
}
```

Quindi, è possibile creare una policy simile con una risorsa diversa (la tabella `Amit_ProductCatalog`) per l'utente Amit.

- Invece di collegare le policy ai singoli utenti, puoi utilizzare variabili di policy IAM per scrivere un'unica policy e collegarla a un gruppo. È necessario creare un gruppo e, per questo esempio, aggiungervi entrambi gli utenti Alice e Amit. Nel seguente esempio vengono concesse le autorizzazioni per eseguire tutte le operazioni DynamoDB sulla tabella `${aws:username}_ProductCatalog`. Quando la policy viene valutata, la variabile `${aws:username}` viene sostituita dal nome utente del richiedente. Ad esempio, se Alice invia una richiesta di aggiunta di un item, l'operazione è consentita solo se l'item viene aggiunto alla tabella `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**
>
> Quando si utilizzano le variabili di policy IAM, è necessario specificare esplicitamente la versione `2012-10-17` del linguaggio di policy IAM nella policy. La versione predefinita del linguaggio di policy IAM (`2008-10-17`) non supporta le variabili di policy.

Invece di identificare una tabella specifica come risorsa come si farebbe normalmente, è possibile utilizzare un carattere jolly (*) per concedere le autorizzazioni su tutte le tabelle il cui nome abbia il prefisso del nome dell'utente che sta effettuando la richiesta, come mostrato di seguito:

```
"Resource":"arn:aws:dynamodb:us-west-2:123456789012:table/${aws:username}_*"
```

Policy IAM per impedire l'acquisto di capacità riservata DynamoDB

Con la capacità riservata di Amazon DynamoDB, si paga una commissione anticipata una tantum e ci si impegna a pagare per un livello minimo di utilizzo, con un risparmio significativo, per un periodo di tempo. È possibile utilizzare il Console di gestione AWS per visualizzare e acquistare la capacità riservata. Tuttavia, si potrebbe volere che non tutti gli utenti nell'organizzazione abbiano la possibilità di acquistare capacità riservata. Per ulteriori informazioni sulla capacità riservata, consulta Prezzi di Amazon DynamoDB.

DynamoDB fornisce le seguenti operazioni API per controllare l'accesso alla gestione della capacità riservata:

- `dynamodb:DescribeReservedCapacity`: restituisce gli acquisti di capacità riservata attualmente in vigore.
- `dynamodb:DescribeReservedCapacityOfferings`: restituisce i dettagli sui piani di capacità riservata attualmente offerti da AWS.
- `dynamodb:PurchaseReservedCapacityOfferings`: esegue un acquisto effettivo della capacità riservata.

Console di gestione AWS Utilizza queste azioni API per visualizzare le informazioni sulla capacità riservata ed effettuare acquisti. Non puoi chiamare queste operazioni da un programma applicativo, poiché sono accessibili solo dalla console. Tuttavia, è possibile consentire o negare l'accesso a queste operazioni in una policy di autorizzazioni IAM.

La seguente politica consente agli utenti di visualizzare gli acquisti e le offerte di capacità riservata utilizzando il Console di gestione AWS , ma i nuovi acquisti vengono negati.

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:*"
        }
    ]
}
```

Ricorda che questa policy utilizza il carattere jolly (*) per consentire a tutti di descrivere le autorizzazioni, e per impedire a tutti l'acquisto della capacità riservata di DynamoDB.

Policy IAM per concedere l'accesso in lettura solo per un flusso DynamoDB (non per la tabella)

Quando si abilita DynamoDB Streams su una tabella, vengono acquisite le informazioni su ogni modifica apportata agli elementi nella tabella. Per ulteriori informazioni, consulta Acquisizione dei dati di modifica per DynamoDB Streams.

In alcuni casi, è possibile che si desideri impedire a un'applicazione di leggere i dati da una tabella DynamoDB, permettendo comunque l'accesso al flusso di tale tabella. Ad esempio, puoi configurare il AWS Lambda polling di uno stream e richiamare una funzione Lambda quando vengono rilevati aggiornamenti degli elementi, quindi eseguire un'elaborazione aggiuntiva.

Per controllare l'accesso a DynamoDB Streams sono disponibili le seguenti operazioni:

- `dynamodb:DescribeStream`
- `dynamodb:GetRecords`
- `dynamodb:GetShardIterator`
- `dynamodb:ListStreams`

Nella seguente policy esempio vengono concesse le autorizzazioni agli utenti per accedere ai flussi di una tabella denominata `GameScores`. Il carattere jolly (*) nell'ARN corrisponde a qualsiasi flusso associato a tale tabella.

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/*"
        }
    ]
}
```

Questa policy concede l'accesso ai flussi della tabella `GameScores`, ma non alla tabella stessa.

Politica IAM per consentire a una AWS Lambda funzione di accedere ai record di flusso DynamoDB

Se desideri eseguire determinate azioni in base agli eventi in un flusso DynamoDB, puoi scrivere AWS Lambda una funzione attivata da questi eventi. Una funzione Lambda come questa ha bisogno delle autorizzazioni per leggere i dati da un flusso di DynamoDB. Per ulteriori informazioni sull'uso di Lambda con DynamoDB Streams, consulta Streams e trigger DynamoDB AWS Lambda.

Per concedere autorizzazioni a Lambda, utilizza la policy di autorizzazione associata al ruolo IAM della funzione Lambda (nota anche come ruolo di esecuzione). È possibile specificare questa policy quando si crea la funzione Lambda.

Ad esempio, è possibile associare la seguente policy di autorizzazioni a un ruolo di esecuzione per concedere le autorizzazioni Lambda per eseguire le azioni DynamoDB Streams elencate.

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/*"
        }
    ]
}
```

Per ulteriori informazioni, consulta Autorizzazioni AWS Lambda nella Guida per gli sviluppatori di AWS Lambda .

Policy IAM per l'accesso in lettura e scrittura a un cluster DynamoDB Accelerator (DAX)

La seguente policy consente l'accesso in lettura, scrittura, aggiornamento ed eliminazione a un cluster DynamoDB Accelerator (DAX), ma non alla tabella DynamoDB associata. Per utilizzare questa politica, sostituisci il nome della AWS regione, l'ID dell'account e il nome del cluster DAX.

> **Note**
>
> Questa policy dà accesso al cluster DAX, ma non alla tabella DynamoDB associata. Assicurati che il cluster DAX disponga delle policy corrette per eseguire le stesse operazioni sulla tabella DynamoDB per conto dell'utente.

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"
        }
    ]
}
```

Per estendere questa politica all'accesso DAX per tutte le AWS regioni di un account, utilizza un carattere jolly (*) per il nome della regione.

```
"Resource": "arn:aws:dax:*:123456789012:cache/MyDAXCluster"
```

Risoluzione dei problemi relativi all'identità e all'accesso di Amazon DynamoDB

Utilizza le informazioni seguenti per diagnosticare e risolvere i problemi comuni che possono verificarsi durante l'utilizzo di DynamoDB e IAM.

Argomenti

- Non sono autorizzato a eseguire un'operazione in DynamoDB
- Non sono autorizzato a eseguire iam: PassRole
- Voglio consentire a persone esterne a me di accedere Account AWS alle mie risorse DynamoDB

Non sono autorizzato a eseguire un'operazione in DynamoDB

Se ti Console di gestione AWS dice che non sei autorizzato a eseguire un'azione, devi contattare l'amministratore per ricevere assistenza. L'amministratore è la persona da cui si sono ricevuti il nome utente e la password.

Il seguente esempio di errore si verifica quando l'utente `mateojackson` prova a utilizzare la console per visualizzare i dettagli relativi a una risorsa *`my-example-widget`* fittizia, ma non dispone di autorizzazioni `aws:`*`GetWidget`* fittizie.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 aws:GetWidget on resource: my-example-widget
```

In questo caso, Mateo richiede al suo amministratore di aggiornare le policy per poter accedere alla risorsa *`my-example-widget`* utilizzando l'operazione `aws:`*`GetWidget`*.

Non sono autorizzato a eseguire iam: PassRole

Se si riceve un errore che indica che non si è autorizzati a eseguire l'operazione `iam:PassRole`, è necessario aggiornare le policy per poter passare un ruolo a DynamoDB.

Alcuni Servizi AWS consentono di passare un ruolo esistente a quel servizio invece di creare un nuovo ruolo di servizio o un ruolo collegato al servizio. Per eseguire questa operazione, è necessario disporre delle autorizzazioni per trasmettere il ruolo al servizio.

L'errore di esempio seguente si verifica quando un utente IAM denominato `marymajor` cerca di utilizzare la console per eseguire un'operazione in DynamoDB. Tuttavia, l'azione richiede che il servizio disponga delle autorizzazioni concesse da un ruolo di servizio. Mary non dispone delle autorizzazioni per trasmettere il ruolo al servizio.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

In questo caso, le policy di Mary devono essere aggiornate per poter eseguire l'operazione `iam:PassRole`.

Se hai bisogno di aiuto, contatta il tuo AWS amministratore. L'amministratore è la persona che ti ha fornito le credenziali di accesso.

Voglio consentire a persone esterne a me di accedere Account AWS alle mie risorse DynamoDB

È possibile creare un ruolo con il quale utenti in altri account o persone esterne all'organizzazione possono accedere alle tue risorse. È possibile specificare chi è attendibile per l'assunzione del ruolo. Per i servizi che supportano politiche basate sulle risorse o liste di controllo degli accessi (ACLs), puoi utilizzare tali politiche per concedere alle persone l'accesso alle tue risorse.

Per maggiori informazioni, consulta gli argomenti seguenti:

- Per capire se DynamoDB supporta queste funzionalità, consultare Funzionamento di Amazon DynamoDB con IAM.
- Per scoprire come fornire l'accesso alle tue risorse attraverso Account AWS le risorse di tua proprietà, consulta Fornire l'accesso a un utente IAM in un altro Account AWS di tua proprietà nella IAM User Guide.
- Per scoprire come fornire l'accesso alle tue risorse a terze parti Account AWS, consulta Fornire l'accesso a soggetti Account AWS di proprietà di terze parti nella Guida per l'utente IAM.
- Per informazioni su come fornire l'accesso tramite la federazione delle identità, consulta Fornire l'accesso a utenti autenticati esternamente (Federazione delle identità) nella Guida per l'utente di IAM.
- Per informazioni sulle differenze di utilizzo tra ruoli e policy basate su risorse per l'accesso multi-account, consulta Accesso a risorse multi-account in IAM nella Guida per l'utente di IAM.

Policy IAM per impedire l'acquisto di capacità riservata DynamoDB

Con la capacità riservata di Amazon DynamoDB, si paga una commissione anticipata una tantum e ci si impegna a pagare per un livello minimo di utilizzo, con un risparmio significativo, per un periodo di tempo. Puoi utilizzare il Console di gestione AWS per visualizzare e acquistare la capacità riservata. Tuttavia, si potrebbe volere che non tutti gli utenti nell'organizzazione abbiano la possibilità di acquistare capacità riservata. Per ulteriori informazioni sulla capacità riservata, consulta Prezzi di Amazon DynamoDB.

DynamoDB fornisce le seguenti operazioni API per controllare l'accesso alla gestione della capacità riservata:

- `dynamodb:DescribeReservedCapacity`: restituisce gli acquisti di capacità riservata attualmente in vigore.

- `dynamodb:DescribeReservedCapacityOfferings`: restituisce i dettagli sui piani di capacità riservata attualmente offerti da AWS.
- `dynamodb:PurchaseReservedCapacityOfferings`: esegue un acquisto effettivo della capacità riservata.

Console di gestione AWS Utilizza queste azioni API per visualizzare le informazioni sulla capacità riservata ed effettuare acquisti. Non puoi chiamare queste operazioni da un programma applicativo, poiché sono accessibili solo dalla console. Tuttavia, è possibile consentire o negare l'accesso a queste operazioni in una policy di autorizzazioni IAM.

La seguente politica consente agli utenti di visualizzare gli acquisti e le offerte di capacità riservata utilizzando il Console di gestione AWS , ma i nuovi acquisti vengono negati.

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:*"
        }
    ]
}
```

Ricorda che questa policy utilizza il carattere jolly (*) per consentire a tutti di descrivere le autorizzazioni, e per impedire a tutti l'acquisto della capacità riservata di DynamoDB.

Utilizzo di condizioni di policy IAM per il controllo granulare degli accessi

Quando si concedono le autorizzazioni in DynamoDB, è possibile specificare le condizioni che determinano il modo in cui una policy di autorizzazioni viene applicata.

Panoramica

In DynamoDB, si ha la possibilità di specificare le condizioni nel momento in cui si concedono le autorizzazioni utilizzando una policy IAM (consultare Identity and Access Management per Amazon DynamoDB). Ad esempio, puoi:

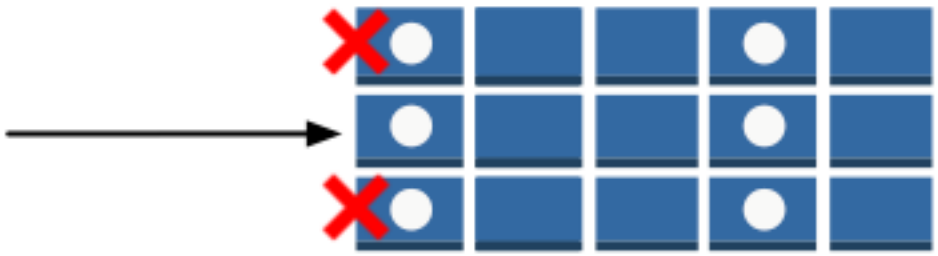
- Concedere autorizzazioni per permettere agli utenti accesso in sola lettura a determinati elementi e attributi in una tabella o indice secondario.
- Concedere autorizzazioni per permettere agli utenti di accedere in sola scrittura a determinati attributi in una tabella, in base all'identità di tale utente.

In DynamoDB è possibile specificare le condizioni in una policy IAM utilizzando chiavi di condizione, come illustrato nel caso d'uso nella sezione seguente.

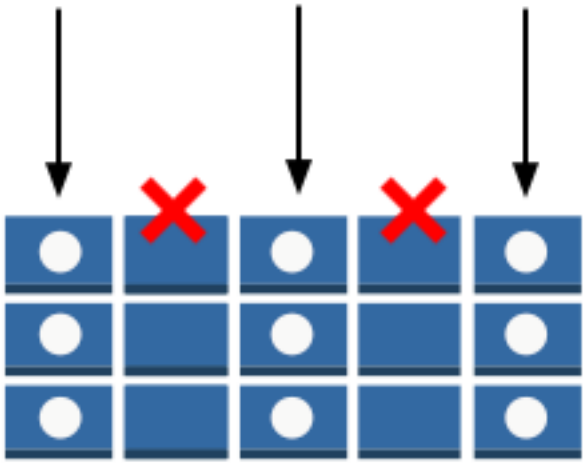
Caso d'uso delle autorizzazioni

Oltre a controllare l'accesso alle operazioni API DynamoDB, è possibile controllare anche l'accesso a singoli elementi e attributi di dati. Ad esempio, puoi eseguire le operazioni seguenti:

- Concedere autorizzazioni su una tabella, ma limitare l'accesso ad item specifici in tale tabella sulla base di determinati valori delle chiavi primarie. Un esempio potrebbe essere un'app di social network per i giochi, in cui tutti i dati di gioco degli utenti vengono archiviati in un'unica tabella, ma nessun utente può accedere agli elementi dei dati di cui non sono proprietari, come mostrato nell'illustrazione seguente:



- Nascondi le informazioni in modo che solo un sottoinsieme di attributi sia visibile all'utente. Un esempio potrebbe essere un'app che mostra dati dei voli per gli aeroporti nelle vicinanze, in base alla posizione dell'utente. I nomi delle compagnie aeree, gli orari di arrivo e di partenza e il numero dei voli vengono tutti visualizzati. Tuttavia, gli attributi come i nomi dei piloti, o il numero dei passeggeri, sono nascosti, come mostrato nell'illustrazione seguente:



Per implementare questo tipo di controllo degli accessi granulare, scrivere una policy di autorizzazioni IAM che specifichi le condizioni per accedere alle credenziali di sicurezza e le autorizzazioni associate. Quindi applichi la policy agli utenti, ai gruppi o ai ruoli che crei utilizzando la console IAM. La policy IAM può limitare l'accesso ai singoli elementi in una tabella, agli attributi in tali elementi o a entrambi nello stesso tempo.

Puoi utilizzare la federazione delle identità sul Web per controllare l'accesso degli utenti che si autenticano con Facebook, Google o Login with Amazon. Per ulteriori informazioni, consulta Utilizzo della federazione delle identità Web.

Puoi utilizzare l'elemento `Condition` IAM per implementare una policy di controllo degli accessi fine-grained. Aggiungendo un elemento `Condition` a una policy di autorizzazioni, è possibile consentire o negare l'accesso agli elementi e agli attributi nelle tabelle e negli indici DynamoDB, in base ai tuoi particolari requisiti aziendali.

Ad esempio, considera un'app di gioco per dispositivi mobili che consente ai giocatori di scegliere e giocare a una varietà di giochi diversi. L'app utilizza una tabella DynamoDB denominata `GameScores` per tenere traccia dei punteggi elevati o di altri dati dell'utente. Ogni item nella tabella è identificato univocamente da un ID utente e dal nome del gioco a cui l'utente

ha giocato. La tabella `GameScores` ha una chiave primaria costituita da una chiave di partizione (`UserId`) e da una chiave di ordinamento (`GameTitle`). Gli utenti possono avere accesso solo ai dati di gioco associati al proprio ID utente. Un utente che desidera giocare deve appartenere a un ruolo IAM denominato `GameRole`, a cui è collegata una policy di sicurezza.

Per gestire le autorizzazioni degli utenti in quest'app, potresti scrivere una policy di autorizzazioni come la seguente:

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"
            }
         }
      }
   ]
}
```

Oltre a concedere le autorizzazioni per operazioni DynamoDB specifiche (elemento `Action`) sulla tabella `GameScores` (elemento `Resource`), l'elemento `Condition` utilizza le seguenti chiavi di condizione specifiche di DynamoDB che limitano le autorizzazioni, come illustrato di seguito:

- `dynamodb:LeadingKeys`: questa chiave di condizione consente agli utenti di accedere solo agli elementi in cui il valore della chiave di partizione corrisponde al proprio ID utente. Questo ID, `${www.amazon.com:user_id}`, è una variabile di sostituzione. Per ulteriori informazioni sulle variabili di sostituzione, consulta Utilizzo della federazione delle identità Web.
- `dynamodb:Attributes`: questa chiave di condizione limita l'accesso agli attributi specificati in modo che solo le operazioni elencate nella policy di autorizzazione possano restituire valori per questi attributi. Inoltre, la clausola `StringEqualsIfExists` garantisce che l'app debba fornire sempre un elenco di attributi specifici su cui agire e che l'app non possa richiedere tutti gli attributi.

Quando viene valutata una policy IAM, il risultato è sempre o true (l'accesso viene consentito) o false (l'accesso viene negato). Se una parte dell'elemento `Condition` è false, l'intera policy restituisce false e l'accesso viene quindi negato.

> **Important**
>
> Se utilizzi `dynamodb:Attributes`, devi specificare i nomi di tutti gli attributi della chiave primaria e della chiave di indicizzazione per la tabella e qualsiasi indice secondario che sia elencato nella policy. In caso contrario, DynamoDB non potrà utilizzare questi attributi della chiave per effettuare l'operazione richiesta.

I documenti delle policy IAM possono contenere solo i seguenti caratteri Unicode: tabulatore orizzontale (U+0009), segno di avanzamento riga (U+000A), ritorno a capo (U+000D), e i caratteri nell'intervallo da U+0020 a U+00FF.

Specifica delle condizioni: Uso delle chiavi di condizione

AWS fornisce un set di chiavi di condizione predefinite (AWS-wide condition keys) per tutti i AWS servizi che supportano IAM per il controllo degli accessi. Ad esempio, puoi utilizzare la chiave di condizione `aws:SourceIp` per controllare l'indirizzo IP del richiedente prima che un'operazione venga effettuata. Per ulteriori informazioni e un elenco delle chiavi AWS-wide, consulta Available Keys for Conditions nella IAM User Guide.

La seguente tabella mostra le chiavi di condizione specifiche dei servizi DynamoDB che si applicano a DynamoDB.

Chiave di condizione DynamoDB	Description
`dynamodb:LeadingKeys`	Rappresenta il primo attributo chiave di una tabella, ovvero la chiave di partizione. Il nome della chiave `LeadingKeys` è plurale, anche se la chiave viene utilizzata con operazioni con item singolo. Inoltre, devi utilizzare il modificatore `ForAllValues` quando utilizzi `LeadingKeys` in una condizione.
`dynamodb:FirstPartitionKeyValues`	Rappresenta il primo attributo chiave di una tabella, in altre parole, la prima chiave di partizione. Il nome della chiave `FirstPartitionKeyValues` è plurale, anche se la chiave viene utilizzata con operazioni con item singolo. Inoltre, è necessario utilizzare il `ForAllValues` modificatore quando si utilizza in una condizione. `FirstPartitionKeyValues` e `LeadingKeys` può essere utilizzato come intercambiabile.
`dynamodb:SecondPartitionKeyValues`	Simile a `dynamodb:FirstPartitionKeyValues` . Rappresenta la seconda chiave di partizione delle risorse. Il nome della chiave `SecondPartitionKeyValues` è plurale, anche se la chiave viene utilizzata con operazioni con item singolo.
`dynamodb:ThirdPartitionKeyValues`	Simile a `dynamodb:FirstPartitionKeyValues` . Rappresenta la chiave di terza partizione delle risorse. Il nome della chiave `ThirdPartitionKeyValues` è plurale, anche se la chiave viene utilizzata con operazioni con item singolo.

Chiave di condizione DynamoDB	Description
dynamodb:FourthPartitionKeyValues	Simile a dynamodb:FirstPartitionKeyValues . Rappresenta la quarta chiave di partizione delle risorse. Il nome della chiave FourthPartitionKeyValues è plurale, anche se la chiave viene utilizzata con operazioni con item singolo.
dynamodb:Select	Rappresenta il parametro Select di una richiesta Query o Scan. Select può essere uno qualsiasi dei seguenti valori: • ALL_ATTRIBUTES • ALL_PROJECTED_ATTRIBUTES • SPECIFIC_ATTRIBUTES • COUNT
dynamodb:Attributes	Rappresenta un elenco degli attributi di primo livello a cui si accede tramite una richiesta. Un attributo di primo livello è accessibile tramite una richiesta se esso, o qualsiasi attributo annidato in esso contenuto , è specificato nei parametri della richiesta o restituito dalla richiesta . Ad esempio, una GetItem richiesta che specifica un ProjectionExpression di"Name, Address.City" , l'dynamodb:Attributes elenco includerebbe «Nome» e «Indirizzo».
dynamodb:ReturnValues	Rappresenta il parametro ReturnValues di una richiesta. ReturnValues può essere uno qualsiasi dei seguenti valori: • ALL_OLD • UPDATED_OLD • ALL_NEW • UPDATED_NEW • NONE

Chiave di condizione DynamoDB	Description
`dynamodb:ReturnConsumedCapacity`	Rappresenta il parametro `ReturnConsumedCapacity` di una richiesta. `ReturnConsumedCapacity` può essere uno qualsiasi dei seguenti valori: • `TOTAL` • `NONE`

Limitazione dell'accesso utente

Molte policy di autorizzazioni IAM permettono agli utenti di accedere solo a quegli elementi in una tabella in cui il valore della chiave di partizione corrisponde all'identificatore dell'utente. Ad esempio, l'app di gioco menzionata in precedenza limita l'accesso in questo modo, tanto che gli utenti possono solo accedere ai dati di gioco collegati al loro ID utente. Le variabili di sostituzione IAM `${www.amazon.com:user_id}`, `${graph.facebook.com:id}` e `${accounts.google.com:sub}` contengono identificatori degli utenti per Login with Amazon, Facebook e Google. Per scoprire come un'applicazione effettua l'accesso a uno di questi provider di identità e ottiene gli identificatori, consulta Utilizzo della federazione delle identità Web.

Important

Il controllo granulare degli accessi non è supportato per limitare la replica delle tabelle globali. L'applicazione di condizioni di policy per il controllo granulare degli accessi ai principali di servizio o ai ruoli collegati al servizio DynamoDB utilizzati per la replica delle tabelle globali può interrompere la replica all'interno di una tabella globale.

Note

Ognuno degli esempi presenti nella sezione seguente imposta la clausola `Effect` su `Allow` e specifica solo le operazioni, le risorse e i parametri permessi. L'accesso è consentito solo a ciò che è elencato esplicitamente nella policy IAM.
In alcuni casi è possibile riscrivere queste policy in modo che si basino sul rifiuto (vale a dire impostare la clausola `Effect` su `Deny` e invertire tutta la logica nella policy). Tuttavia, si consiglia di evitare di utilizzare le policy basate sul rifiuto con DynamoDB poiché la loro

scrittura corretta è difficile rispetto alle policy basate sulle concessioni. Inoltre, le future modifiche all'API DynamoDB (o le modifiche agli input API esistenti) possono rendere una policy basata sul rifiuto inefficace.

Policy di esempio: utilizzo di condizioni per il controllo granulare degli accessi

In questa sezione vengono illustrate varie policy per implementare il controllo degli accessi granulare sulle tabelle e gli indici di DynamoDB.

> **Note**
>
> Tutti gli esempi utilizzano la regione us-west-2 e contengono account fittizi. IDs

Il video seguente spiega il controllo granulare degli accessi in DynamoDB attraverso le condizioni delle policy IAM.

1: Concessione delle autorizzazioni che limitano l'accesso agli elementi con un valore di chiave di partizione specifico

La seguente policy di autorizzazioni concede le autorizzazioni per permettere un insieme di operazioni DynamoDB sulla tabella `GamesScore`. Utilizza la chiave di condizione `dynamodb:LeadingKeys` per limitare le operazioni degli utenti solo sugli elementi il cui valore di chiave di partizione `UserID` corrisponda all'ID utente univoco di Login with Amazon per questa app.

> **Important**
>
> L'elenco delle operazioni non include le autorizzazioni per l'operazione `Scan` poiché `Scan` restituisce tutti gli elementi, indipendentemente dalle chiavi principali.

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**
>
> Quando si utilizzano le variabili di policy, è necessario specificare esplicitamente la versione `2012-10-17` nella policy. La versione di default della sintassi della/e policy di accesso, `2008-10-17`, non supporta le variabili di policy.

Per implementare l'accesso in sola lettura, puoi rimuovere le operazioni che possono modificare i dati. Nella policy seguente solo quelle operazioni che forniscono accesso in sola lettura sono incluse nella condizione.

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**
>
> Se si utilizza `dynamodb:Attributes`, è necessario specificare i nomi di tutti gli attributi della chiave primaria e della chiave di indicizzazione per la tabella e qualsiasi indice secondario che sia elencato nella policy. In caso contrario, DynamoDB non potrà utilizzare questi attributi della chiave per effettuare l'operazione richiesta.

2: Concessione delle autorizzazioni che limitano l'accesso ad attributi specifici in una tabella

La seguente policy di autorizzazioni permette l'accesso solo a due attributi specifici in una tabella aggiungendo la chiave di condizione `dynamodb:Attributes`. Questi attributi possono essere letti, scritti o valutati in una scrittura condizionale o un filtro di scansione.

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 policy adotta un approccio allow list, che permette l'acceso a un insieme denominato di attributi. Puoi scrivere, invece, una policy equivalente che rifiuti l'accesso ad altri attributi. Questo approccio deny list non è consigliato. Gli utenti possono determinare i nomi di questi attributi negati seguendo il principio del privilegio minimo, come spiegato in Wikipedia all'indirizzo http://en.wikipedia. org/wiki/Principle_of_least_privilege e utilizza un approccio

basato sulla lista degli indirizzi consentiti per enumerare tutti i valori consentiti, anziché specificare gli attributi negati.

Questa policy non permette `PutItem`, `DeleteItem` o `BatchWriteItem`. Queste operazioni sostituiscono sempre l'intero item precedente, il che permetterebbe agli utenti di eliminare i valori precedenti per gli attributi a cui non hanno il permesso di accedere.

La clausola `StringEqualsIfExists` nella policy di autorizzazioni garantisce quanto segue:

- Se l'utente specifica il parametro `Select`, allora il suo valore deve essere `SPECIFIC_ATTRIBUTES`. Questo requisito previene che l'operazione API restituisca attributi per cui non si ha il permesso, come ad esempio da una proiezione di indice.
- Se l'utente specifica il parametro `ReturnValues`, allora il suo valore deve essere `NONE`, `UPDATED_OLD` o `UPDATED_NEW`. Questo viene richiesto poiché l'operazione `UpdateItem` effettua anche operazioni di lettura implicite per verificare che un item esista prima di sostituirlo, in modo tale che i valori di attributo precedenti possano essere restituiti se richiesto. Limitare `ReturnValues` in questo modo garantisce che gli utenti possano solo leggere o scrivere gli attributi permessi.
- La clausola `StringEqualsIfExists` assicura che, nel contesto delle operazioni consentite, per ogni richiesta possa essere utilizzato solo uno di questi parametri, `Select` o `ReturnValues`.

Di seguito vengono elencate alcune variazioni a questa policy:

- Per permettere le operazioni di sola lettura, puoi rimuovere `UpdateItem` dall'elenco delle operazioni consentite. Poiché nessuna delle operazioni rimanenti accetta `ReturnValues`, puoi rimuovere `ReturnValues` dalla condizione. Puoi anche modificare `StringEqualsIfExists` in `StringEquals` poiché il parametro `Select` ha sempre un valore (`ALL_ATTRIBUTES`, se non diversamente specificato).
- Per consentire le operazioni di sola scrittura, puoi rimuovere tutto tranne `UpdateItem` dall'elenco delle operazioni consentite. Poiché `UpdateItem` non accetta il parametro `Select`, puoi rimuovere `Select` dalla condizione. È necessario anche modificare `StringEqualsIfExists` in `StringEquals` poiché il parametro `ReturnValues` ha sempre un valore (`NONE`, se non diversamente specificato).

- Per permettere tutti gli attributi il cui nome corrisponde a un modello, utilizza `StringLike` anziché `StringEquals` e utilizza un carattere jolly (*) che trovi una corrispondenza di modelli con più caratteri.

3: Concessione delle autorizzazioni per impedire gli aggiornamenti su determinati attributi

La seguente policy di autorizzazioni limita l'accesso degli utenti in modo che possa aggiornare solo gli attributi specifici identificati dalla chiave di condizione `dynamodb:Attributes`. La condizione `StringNotLike` impedisce a un'applicazione di aggiornare gli attributi specificati utilizzando la chiave di condizione `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"
               ]
            }
         }
      }
   ]
}
```

Tenere presente quanto segue:

- L'operazione `UpdateItem`, come altre operazioni di scrittura, richiede l'accesso in lettura agli elementi, in modo che possa restituire valori prima e dopo l'aggiornamento. Nella policy limiti l'operazione per poter accedere solo agli attributi che è permesso aggiornare specificando la chiave di condizione `dynamodb:ReturnValues`. La chiave di condizione limita il valore di `ReturnValues` nella richiesta mentre specifica solo `NONE`, `UPDATED_OLD` o `UPDATED_NEW` e non include `ALL_OLD` o `ALL_NEW`.
- Le operazioni `PutItem` e `DeleteItem` sostituiscono un item intero, permettendo quindi alle applicazioni di modificare qualsiasi attributo. Pertanto, quando si limita un'applicazione all'aggiornamento solo di attributi specifici, non è necessario concedere l'autorizzazione per questi. APIs

4: Concessione delle autorizzazioni per eseguire query solo sugli attributi proiettati in un indice

La seguente policy di autorizzazioni consente le query su un indice secondario (`TopScoreDateTimeIndex`), utilizzando la chiave di condizione `dynamodb:Attributes`. Inoltre limita le query in modo che possano richiedere solo gli attributi specifici che sono stati proiettati nell'indice.

Per richiedere all'applicazione di specificare un elenco di attributi nella query, la policy specifica anche la chiave di condizione `dynamodb:Select` in modo che richieda che il parametro `Select` dell'operazione DynamoDB `Query` sia `SPECIFIC_ATTRIBUTES`. L'elenco di attributi è limitato a un elenco specifico che viene fornito utilizzando la chiave di condizione `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 policy di autorizzazione seguente è simile, però la query deve richiedere tutti gli attributi che sono stati proiettati nell'indice.

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: Concessione delle autorizzazioni per limitare l'accesso a determinati attributi e ai valori di chiave di partizione

La policy di autorizzazione seguente specifica le operazioni di DynamoDB (specificate nell'elemento `Action`) su una tabella e un indice di tabella (specificati nell'elemento `Resource`). La policy utilizza la chiave di condizione `dynamodb:LeadingKeys` per limitare l'accesso solo agli elementi il cui valore della chiave di partizione corrisponde all'ID Facebook dell'utente.

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"
               ]
            }
         }
      }
   ]
}
```

Tenere presente quanto segue:

- Le operazioni di scrittura permesse dalla policy (`UpdateItem`) possono solo modificare `attribute-A` o `attribute-B`.
- Poiché la policy permette `UpdateItem`, un'applicazione può inserire nuovi item e gli attributi nascosti saranno null nei nuovi item. Se questi attributi sono proiettati in `TopScoreDateTimeIndex`, la policy ha il vantaggio aggiuntivo di impedire le query che causeranno operazioni di recupero dalla tabella.
- Le applicazioni non possono leggere nessun attributo che non sia elencato in `dynamodb:Attributes`. Con questa policy in atto, un'applicazione deve impostare il parametro `Select` su `SPECIFIC_ATTRIBUTES` nelle richieste di lettura e solo gli attributi nell'allow list possono essere richiesti. Per le richieste di lettura, l'applicazione non può impostare `ReturnValues` su `ALL_OLD` o `ALL_NEW` e non può effettuare operazioni di scrittura condizionali basate su attributi che non siano questi.

Argomenti correlati

- Identity and Access Management per Amazon DynamoDB
- Autorizzazioni API DynamoDB: riferimento a operazioni, risorse e condizioni

Utilizzo della federazione delle identità Web

Quando scrivi un'applicazione destinata a un gran numero di utenti, puoi scegliere di utilizzare la federazione delle identità Web per l'autenticazione e l'autorizzazione. La federazione delle identità Web rimuove la necessità di creare singoli utenti . Gli utenti possono invece accedere a un provider di identità e quindi ottenere credenziali di sicurezza temporanee da AWS Security Token Service ()AWS STS. L'app può quindi utilizzare queste credenziali per accedere ai servizi AWS .

La federazione delle identità Web supporta i seguenti provider di identità:

- Login with Amazon
- Facebook
- Google

Risorse aggiuntive per la federazione delle identità Web

Le risorse seguenti possono fornire ulteriori informazioni sulla federazione delle identità Web:

- Il post Federazione delle identità Web tramite AWS SDK per .NETsul blog di AWS Developer illustra come utilizzare la federazione delle identità Web con Facebook. Include frammenti di codice in C# che mostrano come assumere un ruolo IAM con l'identità web e come utilizzare credenziali di sicurezza temporanee per accedere a una risorsa. AWS
- AWS Mobile SDK for iOS e AWS Mobile SDK for Android contengono app di esempio. Includono codice che mostra come invocare i provider di identità e quindi come utilizzare le informazioni provenienti da questi provider per ottenere e usare credenziali di sicurezza temporanee.
- L'articolo Web Identity Federation with Mobile Applications illustra la federazione delle identità Web e mostra un esempio di come utilizzare la federazione delle identità Web per accedere a una risorsa. AWS

Policy di esempio per la federazione delle identità Web

Per mostrare come utilizzare la federazione delle identità web con DynamoDB, rivedi GameScoresla tabella introdotta in. Utilizzo di condizioni di policy IAM per il controllo granulare degli accessi Ecco la chiave principale per. GameScores

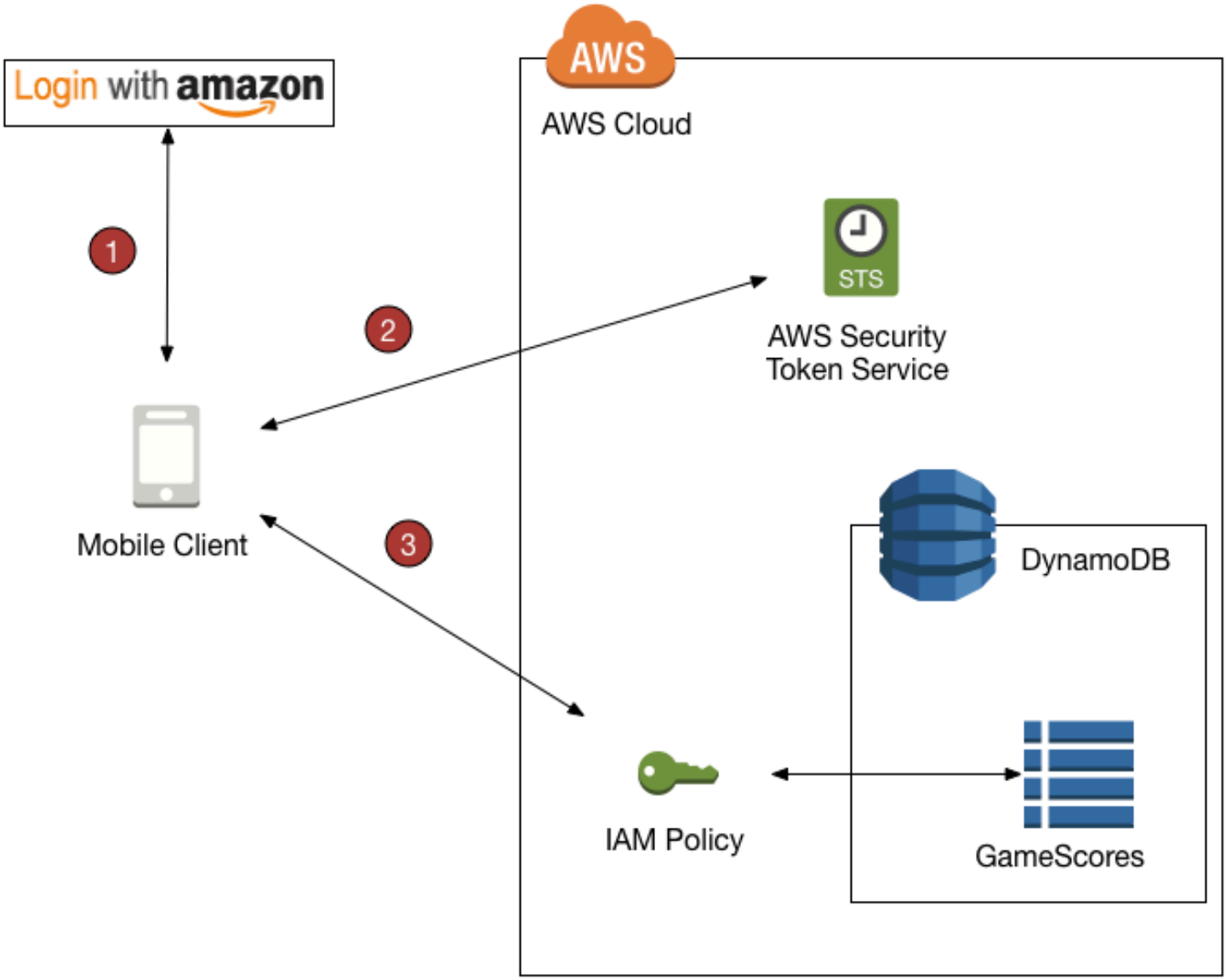
Nome tabella	Tipo di chiave primaria	Nome e tipo di chiave di partizione	Nome e tipo di chiave di ordinamento
GameScores (UserId, GameTitle, ...)	Composita	Nome dell'attributo: UserId ▪Tipo: stringa	Nome dell'attributo: GameTitle ▪Tipo: stringa

Immagina ora che un'app di gioco per dispositivi mobili utilizzi questa tabella e che debba supportare migliaia, o persino milioni, di utenti. Su questa scala, diventa molto difficile gestire i singoli utenti delle app e garantire che ogni utente possa accedere solo ai propri dati nella GameScorestabella. Per fortuna, molti utenti dispongono già di account con un provider di identità di terze parti, come Facebook, Google o Login with Amazon. Pertanto, è opportuno utilizzare uno di questi provider per attività di autenticazione.

Per poterlo fare utilizzando la federazione delle identità Web, lo sviluppatore deve registrare l'app con un provider di identità (come Login with Amazon) e ottenere un ID app univoco. Successivamente, lo sviluppatore dovrà creare un ruolo IAM. (In questo esempio, questo ruolo è denominato GameRole.) Al ruolo deve essere allegato un documento di policy IAM, che specifichi le condizioni in base alle quali l'app può accedere alla GameScorestabella.

Quando un utente desidera giocare, accede al proprio account di Login with Amazon dall'interno dell'app di gioco. L'app chiama quindi AWS Security Token Service (AWS STS), fornendo l'ID dell'app Login GameRolewith Amazon e richiedendo l'iscrizione. AWS STS restituisce AWS le credenziali temporanee all'app e le consente di accedere alla GameScorestabella, in base al documento relativo alla GameRolepolicy.

Il seguente diagramma mostra come questi elementi di integrano fra loro.



Panoramica sulla federazione delle identità Web

1. L'app chiama un provider di identità di terza parte per autenticare l'utente e l'app. Il provider di identità restituisce all'app un token di identità Web.
2. L'app chiama AWS STS e trasmette il token di identità web come input. AWS STS autorizza l'app e le fornisce credenziali di AWS accesso temporanee. L'app può assumere un ruolo IAM (GameRole) e accedere alle AWS risorse in conformità con la politica di sicurezza del ruolo.
3. L'app chiama DynamoDB per accedere alla tabella. GameScores Poiché ha assunto il GameRole, l'app è soggetta alla politica di sicurezza associata a quel ruolo. Il documento della policy impedisce all'app di accedere a dati che non appartengono all'utente.

Ancora una volta, ecco la politica di sicurezza GameRoleche è stata mostrata in[Utilizzo di condizioni di policy IAM per il controllo granulare degli accessi](#):

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 `Condition` clausola determina quali elementi GameScoressono visibili all'app. Per farlo, confronta l'ID Login with Amazon con i valori della chiave di partizione `UserId` in `GameScores`. Solo gli elementi appartenenti all'utente corrente possono essere elaborati utilizzando una delle operazioni DynamoDB elencate in questa policy. Altri item nella tabella non sono accessibili. Inoltre, è possibile accedere solo agli attributi specifici elencati nella policy.

Preparazione per l'utilizzo della federazione delle identità Web

Se sei uno sviluppatore app e desideri utilizzare la federazione delle identità Web per la sua app, procedi nel seguente modo:

1. Registrati come sviluppatore con un provider di identità di terza parte. I seguenti link esterni forniscono informazioni sulla registrazione con i provider di identità supportati:
   - Centro Sviluppatori di Login with Amazon
   - Registrazione sul sito di Facebook
   - Utilizzo della OAuth versione 2.0 per accedere a Google APIs sul sito di Google
2. Registra l'app con il provider di identità. Quando lo fai, il provider ti fornisce un ID univoco per la tua app. Se desideri che la tua app funzioni con più provider di identità, è necessario ottenere un ID app da ogni provider.
3. Crea uno o più ruoli IAM. È necessario un ruolo per ogni provider di identità di ogni app. Ad esempio, puoi creare un ruolo che può essere assunto da un'app in cui l'utente ha effettuato l'accesso tramite Login with Amazon, un secondo ruolo per la stessa app in cui l'utente ha effettuato l'accesso tramite Facebook e un terzo ruolo per l'app in cui gli utenti effettuano l'eccesso tramite Google.

   Come parte della procedura di creazione dei ruoli, è necessario collegare una policy IAM al ruolo. Il documento di policy dovrebbe definire le risorse di richieste dall'app e le autorizzazioni per accedere a tali risorse.

Per ulteriori informazioni, consulta Informazioni sulla federazione delle identità Web nella Guida per l'utente di IAM.

> **Note**
>
> In alternativa AWS Security Token Service, puoi usare Amazon Cognito. Amazon Cognito è il servizio più adatto alla gestione delle credenziali provvisorie per le applicazioni per dispositivi mobili. Per ulteriori informazioni, consulta Ottenere le credenziali nella Guida per gli sviluppatori di Amazon Cognito.

Generazione di una policy IAM tramite la console DynamoDB

La console DynamoDB consente di creare una policy IAM da usare con la federazione delle identità Web. Per poterlo fare, scegli una tabella DynamoDB e specifica il provider di identità, le operazioni e gli attributi da includere nella policy. La console DynamoDB genera quindi una policy che può essere collegata a un ruolo IAM.

1. Accedi Console di gestione AWS e apri la console DynamoDB all'indirizzo. https://console.aws.amazon.com/dynamodb/
2. Nel pannello di navigazione, seleziona Tabelle.
3. Nell'elenco di tabelle, seleziona la tabella per la quale si desidera creare la policy IAM.
4. Seleziona il ppulsante Azioni e scegli Crea una policy di controllo degli accessi.
5. Seleziona il provider di identità, le operazioni e gli attributi per la policy.

   Dopo aver selezionato le impostazioni desiderate, scegliere Genera policy Verrà visualizzata la policy generata;
6. Scegli Vedi la documentazione e segui i passaggi necessari per associare la policy generata a un ruolo IAM.

Scrittura dell'app per l'utilizzo della federazione delle identità Web

Per utilizzare la federazione delle identità Web, l'app deve assumere il ruolo IAM creato. Da quel punto in poi, l'app applica la policy d'accesso collegata al ruolo.

In fase di runtime, se l'app utilizza la federazione delle identità Web, deve seguire la procedura seguente:

1. Autenticarsi con un provider di identità di terza parte. L'app deve chiamare il provider di identità utilizzando un'interfaccia fornita da quest'ultimo. Il modo esatto in cui si effettua l'autenticazione

dell'utente dipende dal provider e dalla piattaforma su cui si esegue l'app. In genere, se l'utente non ha già effettuato l'accesso, il provider di identità si occupa di mostrare una pagina di accesso per tale provider.

Dopo aver autenticato l'utente, il provider di identità restituisce alla tua app un token di identità Web. Il formato di questo token dipende dal provider, ma in genere è una stringa di caratteri molto lunga.

2. Ottieni credenziali di AWS sicurezza temporanee. Per fare ciò, l'app invia una richiesta `AssumeRoleWithWebIdentity` a AWS Security Token Service AWS STS. Questa richiesta contiene i seguenti elementi:
   - Il token dell'identità Web ottenuto nella fase precedente;
   - L'ID app fornito dal provider di identità;
   - L'Amazon Resource Name (ARN) del ruolo IAM che è stato creato per questo provider di identità per questa app.

   AWS STS restituisce un set di credenziali di AWS sicurezza che scadono dopo un determinato periodo di tempo (3.600 secondi, per impostazione predefinita).

   L'esempio seguente mostra una richiesta e la risposta da un'operazione `AssumeRoleWithWebIdentity` in AWS STS. Il token dell'identità Web è stato ottenuto dal provider di identità 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. Accedi alle risorse. AWS La risposta da AWS STS contiene le informazioni che sono richieste dall'app per poter accedere alle risorse DynamoDB:

   - I campi `AccessKeyID`, `SecretAccessKey` e `SessionToken` contengono le credenziali di sicurezza valide solo per questo utente e quest'app.
   - Il campo `Expiration` indica il limite di tempo dopo il quale queste credenziali non sono più valide.
   - Il campo `AssumedRoleId` contiene il nome di un ruolo IAM specifico della sessione che stato assunto dall'app. L'app rispetta i controlli degli accessi nel documento della policy IAM per la durata di questa sessione.
   - Il campo `SubjectFromWebIdentityToken` contiene l'ID univoco presente in una variabile di policy IAM per questo provider di identità specifico. Di seguito sono riportate le variabili di policy IAM per i provider supportati e alcuni valori di esempio:

Variabile di policy	Valore di esempio
`${www.amazon.com:user_id}`	`amzn1.account.AGJZDKHJKAUUS W6C44CHPEXAMPLE`
`${graph.facebook.com:id}`	`123456789`
`${accounts.google.com:sub}`	`123456789012345678901`

Per le policy IAM di esempio in cui vengono utilizzate queste variabili di policy, consulta Policy di esempio: utilizzo di condizioni per il controllo granulare degli accessi.

Per ulteriori informazioni su come AWS STS generare credenziali di accesso temporanee, consulta Requesting Temporary Security Credentials nella IAM User Guide.

Autorizzazioni API DynamoDB: riferimento a operazioni, risorse e condizioni

Quando si configura Identity and Access Management per Amazon DynamoDB e si scrive una policy di autorizzazioni che può essere collegata a un'identità IAM (policy basate su identità), è possibile utilizzare l'elenco di Operazioni, risorse e chiavi di condizioni per Amazon DynamoDB nella Guida per l'utente di IAM come riferimento. La pagina elenca ogni operazione dell'API DynamoDB, le azioni corrispondenti per le quali è possibile concedere le autorizzazioni per eseguire l'azione e la risorsa per la quale è possibile concedere AWS le autorizzazioni. Puoi specificare le azioni nel campo `Action` della policy e il valore della risorsa nel campo `Resource`.

È possibile utilizzare le chiavi di condizione AWS-wide nelle policy di DynamoDB per esprimere condizioni. Per un elenco completo delle chiavi AWS-wide, consulta il riferimento agli elementi della policy IAM JSON nella IAM User Guide.

Oltre alle chiavi di condizione AWS-wide, DynamoDB dispone di chiavi specifiche che è possibile utilizzare nelle condizioni. Per ulteriori informazioni, consulta Utilizzo di condizioni di policy IAM per il controllo granulare degli accessi.

Argomenti correlati

- Identity and Access Management per Amazon DynamoDB
- Utilizzo di condizioni di policy IAM per il controllo granulare degli accessi

Convalida della conformità per settore per DynamoDB

Per sapere se un Servizio AWS programma rientra nell'ambito di specifici programmi di conformità, consulta Servizi AWS la sezione Ambito per programma di conformitàServizi AWS di conformità e scegli il programma di conformità che ti interessa. Per informazioni generali, consulta Programmi di AWS conformità Programmi di di .

È possibile scaricare report di audit di terze parti utilizzando AWS Artifact. Per ulteriori informazioni, consulta Scaricamento dei report in AWS Artifact .

La vostra responsabilità di conformità durante l'utilizzo Servizi AWS è determinata dalla sensibilità dei dati, dagli obiettivi di conformità dell'azienda e dalle leggi e dai regolamenti applicabili. Per ulteriori informazioni sulla responsabilità di conformità durante l'utilizzo Servizi AWS, consulta la Documentazione AWS sulla sicurezza.

Resilienza e ripristino di emergenza in Amazon DynamoDB

L'infrastruttura AWS globale è costruita attorno AWS a regioni e zone di disponibilità. AWS Le regioni forniscono più zone di disponibilità fisicamente separate e isolate, collegate con reti a bassa latenza, ad alto throughput e altamente ridondanti. Con le zone di disponibilità, è possibile progettare e gestire applicazioni e database che eseguono il failover automatico tra zone di disponibilità senza interruzioni. Le zone di disponibilità sono più disponibili, tolleranti ai guasti e scalabili rispetto alle infrastrutture tradizionali a data center singolo o multiplo.

Se occorre replicare i dati o le applicazioni su distanze geografiche più ampie, utilizza le regioni locali AWS . Una regione AWS locale è un singolo data center progettato per integrare una regione esistente. AWS Come tutte le AWS regioni, le regioni AWS locali sono completamente isolate dalle altre AWS regioni.

Per ulteriori informazioni su AWS regioni e zone di disponibilità, consulta infrastruttura AWS globale.

Oltre all'infrastruttura AWS globale, Amazon DynamoDB offre diverse funzionalità per supportare le esigenze di resilienza e backup dei dati.

Backup e ripristino on demand

DynamoDB fornisce funzionalità di backup on demand. Consente di creare backup completi delle tabelle per la conservazione e l'archiviazione a lungo termine. Per ulteriori informazioni, consulta Backup e ripristino on demand per DynamoDB.

Point-in-time recupero

Point-in-time il ripristino aiuta a proteggere le tabelle DynamoDB da operazioni di scrittura o eliminazione accidentali. Grazie al ripristino point-in-time, non dovrai preoccuparti di creare, gestire o programmare i backup on-demand. Per ulteriori informazioni, consulta Point-in-time recovery for DynamoDB.

Tabelle globali sincronizzate tra regioni AWS

DynamoDB distribuisce automaticamente i dati e il traffico per la tabella su un numero di server sufficiente per gestire i requisiti di velocità effettiva e archiviazione, garantendo al contempo

prestazioni rapide e costanti. Tutti i dati sono archiviati su dischi a stato solido (SSDs) e vengono replicati automaticamente su più zone di disponibilità in una AWS regione, garantendo elevata disponibilità e durabilità dei dati integrate. È possibile utilizzare le tabelle globali per mantenere le tabelle DynamoDB sincronizzate tra le regioni. AWS

Sicurezza dell'infrastruttura in Amazon DynamoDB

In quanto servizio gestito, Amazon DynamoDB è protetto dalle procedure di sicurezza di rete globali descritte nella sezione Protezione AWS dell'infrastruttura situata nel Well-Architected Framework. AWS

Si utilizzano chiamate API AWS pubblicate per accedere a DynamoDB attraverso la rete. I client possono utilizzare la versione 1.2 o 1.3 di TLS (Transport Layer Security). I client devono inoltre supportare le suite di cifratura con PFS (Perfect Forward Secrecy), ad esempio Ephemeral Diffie-Hellman (DHE) o Elliptic Curve Diffie-Hellman Ephemeral (ECDHE). La maggior parte dei sistemi moderni, come Java 7 e versioni successive, supporta tali modalità. Inoltre, le richieste devono essere firmate tramite un ID chiave di accesso e una chiave di accesso segreta associata a un principal IAM. In alternativa, è possibile utilizzare AWS Security Token Service (AWS STS) per generare le credenziali di sicurezza temporanee per firmare le richieste.

Puoi anche utilizzare un endpoint di cloud privato virtuale (VPC) per DynamoDB per consentire alle istanze Amazon EC2 nel tuo VPC di utilizzare i loro indirizzi IP privati per accedere a DynamoDB senza esposizione alla rete Internet pubblica. Per ulteriori informazioni, consulta Utilizzo di endpoint Amazon VPC per accedere a DynamoDB.

Utilizzo di endpoint Amazon VPC per accedere a DynamoDB

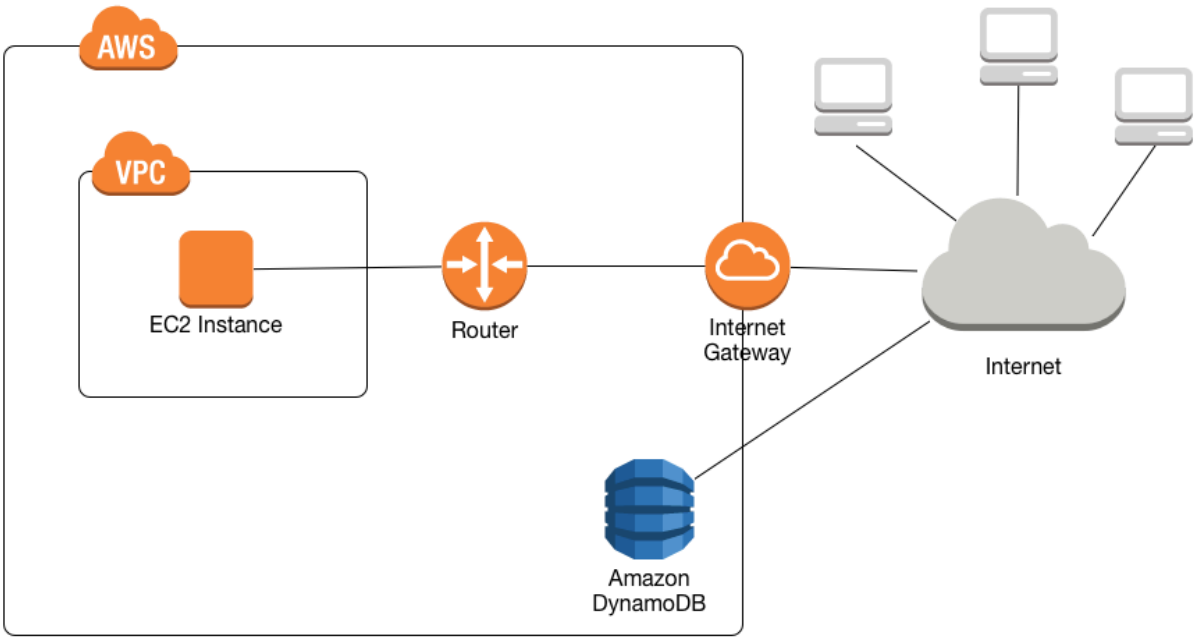
Per motivi di sicurezza, molti AWS clienti eseguono le proprie applicazioni all'interno di un ambiente Amazon Virtual Private Cloud (Amazon VPC). Con Amazon VPC, puoi avviare EC2 istanze Amazon in un cloud privato virtuale, logicamente isolato da altre reti, inclusa la rete Internet pubblica. Con un Amazon VPC, puoi controllarne l'intervallo di indirizzi IP, le sottoreti, le tabelle di routing e i gateway di rete, nonché le impostazioni di sicurezza.

> **Note**
>
> Se hai creato il tuo Account AWS dopo il 4 dicembre 2013, hai già un VPC predefinito in ciascuno di essi. Regione AWS Un VPC predefinito è pronto per l'uso: può essere utilizzato immediatamente senza dover eseguire ulteriori operazioni di configurazione.

Per ulteriori informazioni, consulta VPC predefinito e sottoreti predefinite nella Guida per l'utente di Amazon VPC.

Per accedere a Internet, il VPC deve disporre di un gateway Internet, ovvero di un router virtuale che connette il VPC a Internet. Ciò consente alle applicazioni in esecuzione su Amazon EC2 nel tuo VPC di accedere a risorse Internet, come Amazon DynamoDB.

Per impostazione predefinita, le comunicazioni da e verso DynamoDB utilizzano il protocollo HTTPS, che protegge il traffico di rete utilizzando la crittografia. SSL/TLS Il diagramma seguente mostra un' EC2 istanza Amazon in un VPC che accede a DynamoDB, facendo in modo che DynamoDB utilizzi un gateway Internet anziché endpoint VPC.



Molti clienti esprimono legittime preoccupazioni in materia di sicurezza e di privacy quando si tratta di inviare e ricevere dati tramite la rete Internet pubblica. I clienti possono risolvere questi problemi usando una rete privata virtuale (VPN) per instradare tutto il traffico di rete di DynamoDB tramite la propria infrastruttura di rete aziendale. Questo approccio tuttavia può introdurre problematiche relative alla larghezza di banda e alla disponibilità.

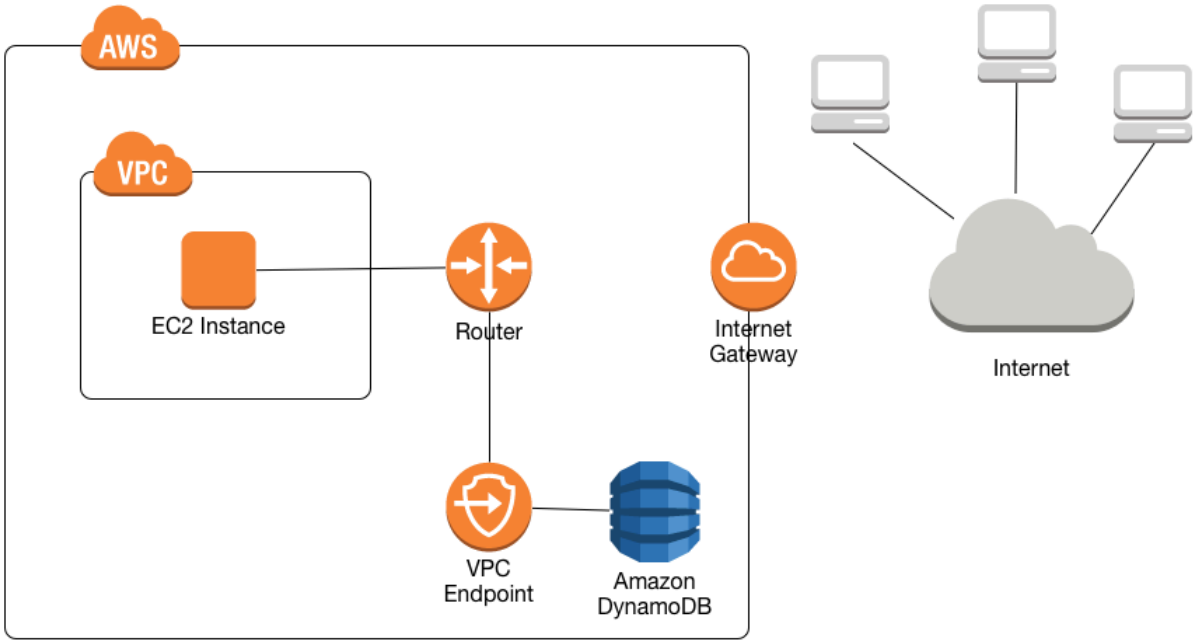
Gli endpoint VPC per DynamoDB possono mitigare queste difficoltà. Un endpoint VPC per DynamoDB consente alle EC2 istanze Amazon nel tuo VPC di utilizzare i propri indirizzi IP privati

per accedere a DynamoDB senza alcuna esposizione alla rete Internet pubblica. Le tue EC2 istanze non richiedono indirizzi IP pubblici e non hai bisogno di un gateway Internet, un dispositivo NAT o un gateway privato virtuale nel tuo VPC. Le policy degli endpoint vengono utilizzate per controllare l'accesso a DynamoDB. Il traffico tra il tuo VPC e il AWS servizio non esce dalla rete Amazon.

> **Note**
>
> Anche quando utilizzi indirizzi IP pubblici, tutte le comunicazioni VPC tra istanze e servizi ospitati vengono mantenute private all'interno della rete. AWS AWS I pacchetti che provengono dalla AWS rete con una destinazione sulla AWS rete rimangono sulla rete AWS globale, ad eccezione del traffico da o verso le regioni della AWS Cina.

Quando si crea un endpoint VPC per DynamoDB, qualsiasi richiesta a un endpoint DynamoDB all'interno della regione (ad esempio dynamodb.us-west-2.amazonaws.com) viene instradata verso un endpoint DynamoDB privato all'interno della rete Amazon. Non è necessario modificare le applicazioni in esecuzione su EC2 istanze nel VPC. Il nome dell'endpoint rimane invariato, ma l'instradamento verso DynamoDB rimane interamente all'interno della rete Amazon e non accede alla rete Internet pubblica.

Il diagramma seguente mostra come un' EC2 istanza in un VPC può utilizzare un endpoint VPC per accedere a DynamoDB.



Per ulteriori informazioni, consulta the section called “Tutorial: Utilizzo di un endpoint VPC per DynamoDB”.

Condivisione di endpoint Amazon VPC e DynamoDB

Per abilitare l'accesso al servizio DynamoDB tramite l'endpoint gateway di una sottorete VPC, devi disporre delle autorizzazioni dell'account proprietario per tale sottorete VPC.

Dopo che all'endpoint gateway della sottorete VPC è stato concesso l'accesso a DynamoDB, qualsiasi account AWS con accesso a tale sottorete può utilizzare DynamoDB. Ciò significa che tutti gli utenti dell'account all'interno della sottorete VPC possono utilizzare qualsiasi tabella DynamoDB a cui hanno accesso. Ciò include le tabelle DynamoDB associate a un account diverso rispetto alla sottorete VPC. Il proprietario della sottorete VPC può comunque impedire a qualsiasi utente specifico all'interno della sottorete di utilizzare il servizio DynamoDB tramite l'endpoint del gateway, a sua discrezione.

Tutorial: Utilizzo di un endpoint VPC per DynamoDB

Questa sezione guida attraverso la configurazione e l'utilizzo di un endpoint VPC per DynamoDB.

Argomenti

- Passaggio 1: avviare un' EC2 istanza Amazon
- Passaggio 2: configura la tua EC2 istanza Amazon
- Fase 3: creazione di un endpoint VPC per DynamoDB
- Fase 4: pulizia (opzionale)

Passaggio 1: avviare un' EC2 istanza Amazon

In questa fase, avvii un' EC2 istanza Amazon nel tuo Amazon VPC predefinito. È quindi possibile creare e utilizzare un endpoint VPC per DynamoDB.

1. Apri la EC2 console Amazon all'indirizzo https://console.aws.amazon.com/ec2/.
2. Seleziona Avvia istanza e completa le seguenti operazioni:

   Fase 1: scelta di un'Amazon Machine Image (AMI)

   - Nella parte superiore dell'elenco AMIs, vai su Amazon Linux AMI e scegli Seleziona.

   Fase 2: scelta di un tipo di istanza

   - All'inizio dell'elenco dei tipi di istanza, scegli t2.micro.
   - Scegliere Next: Configure Instance Details (Successivo: Configura i dettagli dell'istanza).

   Fase 3: configurare i dettagli dell'istanza

   - Vai a Network (Rete) e scegli il VPC predefinito.

     Scegli Passaggio successivo: aggiunta dello storage.

   Fase 4: aggiungere storage

   - Salta questa fase scegliendo Next: Tag Instance (Successivo: Assegna un tag all'istanza).

   Fase 5: assegnazione di un tag all'istanza

   - Ignora questa fase scegliendo Next: Configure Security Group (Successivo: Configura il gruppo di sicurezza).

Fase 6: configura il gruppo di sicurezza

- Scegli Seleziona un gruppo di sicurezza esistente.
- Nell'elenco dei gruppi di sicurezza, scegli default (predefinito). Questo è il gruppo di sicurezza di default per il tuo ambiente VPC.
- Scegli Next: Review and Launch (Successivo: Verifica e avvia).

Fase 7: verifica il lancio dell'istanza

- Scegli Avvia.

3. Nella finestra Select an existing key pair or create a new key pair (Seleziona una coppia di chiavi esistente oppure crea una nuova coppia di chiavi), effettua una delle seguenti operazioni:

   - Se non disponi di una coppia di EC2 chiavi Amazon, scegli Crea una nuova coppia di chiavi e segui le istruzioni. Ti verrà chiesto di scaricare un file di chiave privata (file.pem); ti servirà in seguito quando accederai alla tua istanza Amazon EC2.
   - Se disponi già di una coppia di EC2 chiavi Amazon, vai a Seleziona una coppia di chiavi e scegli la tua coppia di chiavi dall'elenco. Devi già avere il file della chiave privata (file.pem) disponibile per accedere alla tua istanza Amazon EC2 .

4. Quando hai configurato la tua coppia di chiavi, scegli Launch Instances (Avvia istanze).
5. Torna alla home page della EC2 console Amazon e scegli l'istanza che hai lanciato. Nel riquadro inferiore, nella scheda Descrizione, individuare il DNS pubblico per l'istanza. Ad esempio: `ec2-00-00-00-00.us-east-1.compute.amazonaws.com`.

   Prendere nota di questo nome DNS pubblico, perché sarà necessario nella fase successiva di questa esercitazione (Passaggio 2: configura la tua EC2 istanza Amazon).

> **Note**
>
> Ci vorranno alcuni minuti prima che la tua EC2 istanza Amazon diventi disponibile. Prima di continuare, verificare che Stato istanza sia `running` e che tutti i controlli di verifica stato siano stati superati.

Passaggio 2: configura la tua EC2 istanza Amazon

Quando la tua EC2 istanza Amazon sarà disponibile, potrai accedervi e prepararla per il primo utilizzo.

> **Note**
>
> I passaggi seguenti presuppongono che ti stia connettendo alla tua EC2 istanza Amazon da un computer che esegue Linux. Per altri modi di connessione, consulta Connect to Your Linux Instance nella Amazon EC2 User Guide.

1. Dovrai autorizzare il traffico SSH in entrata verso la tua istanza Amazon. EC2 Per fare ciò, creerai un nuovo gruppo di EC2 sicurezza e poi assegnerai il gruppo di sicurezza alla tua istanza. EC2

   a. Fare clic su Security Groups (Gruppi di sicurezza) nel pannello di navigazione.
   b. Scegli Crea gruppo di sicurezza. Nella finestra Crea gruppo di sicurezza effettua le operazioni seguenti:

      - Nome gruppo di sicurezza: immettere un nome per il gruppo di sicurezza. Ad esempio: `my-ssh-access`
      - Descrizione: specificare una descrizione breve per il gruppo di sicurezza.
      - VPC: scegli il VPC predefinito.
      - Nella sezione Regole del gruppo di sicurezza, seleziona Aggiungi regole e completare le operazioni descritte di seguito.
        - Tipo: seleziona SSH.
        - Origine: scegli Il mio IP.

      Dopo aver selezionato tutte le impostazioni desiderate, scegli Crea.
   c. Nel riquadro di navigazione, scegliere Instances (Istanze).
   d. Scegli l' EC2 istanza Amazon in cui hai effettuato il lancioPassaggio 1: avviare un' EC2 istanza Amazon.
   e. Seleziona Operazioni --> Reti --> Modifica i gruppi di sicurezza.
   f. In Modifica gruppi di sicurezza, seleziona il gruppo di sicurezza creato in precedenza in questa procedura (ad esempio, `my-ssh-access`). Dovrebbe essere selezionato anche il

gruppo di sicurezza `default` esistente. Dopo aver selezionato le impostazioni desiderate, seleziona Assegna i gruppi di sicurezza.

2. Usa il `ssh` comando per accedere alla tua EC2 istanza Amazon, come nell'esempio seguente.

```
ssh -i my-keypair.pem ec2-user@public-dns-name
```

È necessario specificare il file di chiave privata (file .pem) e il nome DNS pubblico dell'istanza. Consulta Passaggio 1: avviare un' EC2 istanza Amazon.

L'ID accesso è `ec2-user`. Non è richiesta una password.

3. Configura AWS le tue credenziali come mostrato nell'esempio seguente. Quando richiesto, immettere l'ID chiave di accesso AWS , la chiave segreta e nome della regione predefinita.

```
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]:
```

È ora possibile creare un endpoint VPC per DynamoDB.

Fase 3: creazione di un endpoint VPC per DynamoDB

In questa fase, verrà creato un endpoint VPC per DynamoDB e ne sarà eseguito il test per verificare che funzioni.

1. Prima di iniziare, verifica di poter comunicare con DynamoDB utilizzando l'endpoint pubblico.

```
aws dynamodb list-tables
```

L'output mostrerà un elenco di tabelle DynamoDB di proprietà. (Se non si disponi di alcuna tabella, l'elenco sarà vuoto).

2. Verifica che DynamoDB sia un servizio disponibile per la creazione di endpoint VPC nella regione corrente. AWS Il comando è mostrato in grassetto, seguito dall'output di esempio.

```
aws ec2 describe-vpc-endpoint-services
```

```
{
    "ServiceNames": [
        "com.amazonaws.us-east-1.s3",
        "com.amazonaws.us-east-1.dynamodb"
    ]
}
```

Nell'output di esempio, DynamoDB è uno dei servizi disponibili, quindi è possibile procedere con la creazione di un endpoint VPC.

3. Determinare l'identificatore VPC.

```
aws ec2 describe-vpcs
```

```
{
    "Vpcs": [
        {
            "VpcId": "vpc-0bbc736e",
            "InstanceTenancy": "default",
            "State": "available",
            "DhcpOptionsId": "dopt-8454b7e1",
            "CidrBlock": "172.31.0.0/16",
            "IsDefault": true
        }
    ]
}
```

Nell'output di esempio, l'ID VPC è `vpc-0bbc736e`.

4. Crea l'endpoint VPC. Per il parametro `--vpc-id`, specificare l'ID VPC della fase precedente. Utilizza il parametro `--route-table-ids` per associare l'endpoint alle tabelle di routing.

```
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. Verificare di poter accedere a DynamoDB tramite l'endpoint VPC.

```
aws dynamodb list-tables
```

Se vuoi, puoi provare altri AWS CLI comandi per DynamoDB. Per ulteriori informazioni, consulta la sezione relativa alle informazioni di riferimento ai comandi di AWS CLI.

Fase 4: pulizia (opzionale)

Se desideri eliminare le risorse create in questo tutorial, seguire queste procedure:

Come rimuovere l'endpoint VPC per DynamoDB

1. Accedi alla tua EC2 istanza Amazon.
2. Determinare l'ID endpoint 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"
    }
}
```

Nell'output di esempio, l'ID endpoint VPC è `vpce-9b15e2f2`.

3. Eliminare l'endpoint VPC.

```
aws ec2 delete-vpc-endpoints --vpc-endpoint-ids vpce-9b15e2f2
```

```
{
    "Unsuccessful": []
}
```

L'array vuoto `[]` indica la riuscita dell'operazione (non ci sono state richieste non riuscite).

Per terminare la tua istanza Amazon EC2

1. Apri la EC2 console Amazon all'indirizzo https://console.aws.amazon.com/ec2/.
2. Nel riquadro di navigazione, scegliere Instances (Istanze).
3. Scegli la tua EC2 istanza Amazon.
4. Seleziona Actions (Operazioni), Instance State (Stato istanza), Terminate (Termina).
5. Nella finestra di conferma scegli Sì, termina.

AWS PrivateLink per DynamoDB

Con AWS PrivateLink for DynamoDB, puoi effettuare il provisioning degli endpoint Amazon VPC di interfaccia (endpoint di interfaccia) nel tuo cloud privato virtuale (Amazon VPC). Questi endpoint sono accessibili direttamente dalle applicazioni locali tramite VPN e/o in un altro modo Regione AWS tramite il peering Amazon VPC. Direct Connect Utilizzando AWS PrivateLink e gli endpoint di interfaccia è possibile semplificare la connettività di rete privata dalle applicazioni a DynamoDB.

Le applicazioni presenti nel VPC non richiedono indirizzi IP pubblici per comunicare con DynamoDB utilizzando gli endpoint di interfaccia VPC per le operazioni DynamoDB. Gli endpoint dell'interfaccia sono rappresentati da una o più interfacce di rete elastiche (ENIs) a cui vengono assegnati indirizzi IP privati dalle sottoreti del tuo Amazon VPC. Le richieste a DynamoDB tramite gli endpoint di interfaccia rimangono nella rete Amazon. Puoi anche accedere agli endpoint di interfaccia nel tuo Amazon VPC da applicazioni locali AWS Direct Connect tramite AWS Virtual Private Network o ().Site-to-Site VPN Per ulteriori informazioni su come connettere l'Amazon VPC alla rete on-premises, consulta la Guida per l'utente di Direct Connect e la Guida per l'utente di AWS Site-to-Site VPN.

Per informazioni generali sugli endpoint di interfaccia, consulta Interface Amazon VPC endpoints AWS PrivateLink() nella Guida.AWS PrivateLink AWS PrivateLink è supportato anche per gli endpoint Amazon DynamoDB Streams. Per ulteriori informazioni, consulta the section called “AWS PrivateLink per DynamoDB Streams”.

Argomenti

- Tipi di endpoint Amazon VPC per Amazon DynamoDB
- Considerazioni sull'utilizzo AWS PrivateLink per Amazon DynamoDB
- Creazione di un endpoint Amazon VPC
- Accesso agli endpoint di interfaccia Amazon DynamoDB
- Accesso alle tabelle DynamoDB e alle operazioni API di controllo dagli endpoint di interfaccia DynamoDB
- Aggiornamento di una configurazione DNS locale
- Creazione di una policy sugli endpoint Amazon VPC per DynamoDB
- Utilizzo degli endpoint DynamoDB con accesso privato a Console di gestione AWS
- AWS PrivateLink per DynamoDB Streams
- Utilizzo AWS PrivateLink per DynamoDB Accelerator (DAX)

Tipi di endpoint Amazon VPC per Amazon DynamoDB

Puoi utilizzare due tipi di endpoint Amazon VPC per accedere ad Amazon DynamoDB: endpoint gateway ed endpoint di interfaccia (utilizzando). AWS PrivateLink Un endpoint gateway è un gateway specificato nella tabella di routing per accedere a DynamoDB dal tuo Amazon VPC attraverso la rete. AWS Gli endpoint di interfaccia estendono la funzionalità degli endpoint gateway utilizzando indirizzi IP privati per instradare le richieste a DynamoDB dall'interno del tuo Amazon VPC, in locale o da un Amazon VPC in un altro tramite il peering Amazon VPC o. Regione AWS AWS Transit Gateway Per ulteriori informazioni, consulta What is Amazon VPC peering? e Transit Gateway vs Amazon VPC peering.

Gli endpoint di interfaccia sono compatibili con gli endpoint gateway. Se si dispone di un endpoint gateway nell’Amazon VPC, è possibile utilizzare entrambi i tipi di endpoint nello stesso Amazon VPC.

Endpoint gateway per DynamoDB	Endpoint di interfaccia per DynamoDB
In entrambi i casi, il traffico di rete rimane sulla rete. AWS	

Endpoint gateway per DynamoDB	Endpoint di interfaccia per DynamoDB
Utilizzo di indirizzi IP pubblici per Amazon DynamoDB	Utilizzo di indirizzi IP privati del proprio Amazon VPC per accedere ad Amazon DynamoDB
Non consente l'accesso da on-premise	Consente l'accesso da On-Premise
Non consentire l'accesso da un altro Regione AWS	Consenti l'accesso da un endpoint Amazon VPC a un altro Regione AWS utilizzando il peering Amazon VPC o AWS Transit Gateway
Non fatturata	Fatturata

Per ulteriori informazioni sugli endpoint gateway, consulta Gateway Amazon VPC endpoints nella Guida ad AWS PrivateLink .

Considerazioni sull'utilizzo AWS PrivateLink per Amazon DynamoDB

Le considerazioni relative ad Amazon VPC si applicano ad Amazon AWS PrivateLink DynamoDB. Per ulteriori informazioni, consulta Considerazioni di un endpoint di interfaccia e Quote di AWS PrivateLink nella Guida di AWS PrivateLink . Inoltre, si applicano le limitazioni seguenti:

AWS PrivateLink per Amazon DynamoDB non supporta quanto segue:

- Transport Layer Security (TLS) 1.1
- Servizi del sistema dei nomi di dominio (DNS) privati e ibridi

Puoi inviare fino a 50.000 richieste al secondo per ogni AWS PrivateLink endpoint abilitato.

> **Note**
>
> I timeout di connettività di rete verso gli AWS PrivateLink endpoint non rientrano nell'ambito delle risposte di errore di DynamoDB e devono essere gestiti in modo appropriato dalle applicazioni che si connettono agli endpoint. PrivateLink

Creazione di un endpoint Amazon VPC

Per creare un endpoint Amazon VPC di interfaccia, consulta Create an Amazon VPC endpoint nella Guida ad AWS PrivateLink .

Accesso agli endpoint di interfaccia Amazon DynamoDB

Quando si crea un endpoint di interfaccia, DynamoDB genera due tipi di nomi DNS DynamoDB specifici dell'endpoint: Regionale e zonale.

- Un nome DNS regionale include un ID endpoint Amazon VPC univoco, un identificatore di servizio, Regione AWS`vpce.amazonaws.com` la e nel nome. Ad esempio, per l'ID dell'endpoint Amazon VPC *`vpce-1a2b3c4d`*, il nome DNS generato potrebbe essere simile a *`vpce-1a2b3c4d-5e6f`*`.dynamodb.us-east-1.vpce.amazonaws.com`.
- I nomi DNS zonali includono la zona di disponibilità, ad esempio *`vpce-1a2b3c4d-5e6f`*`-us-east-1a.dynamodb.us-east-1.vpce.amazonaws.com`. Puoi utilizzare questa opzione se l'architettura isola le zone di disponibilità. Ad esempio, puoi utilizzarla per il contenimento degli errori o per ridurre i costi di trasferimento dei dati a livello regionale.

> **Note**
>
> Per ottenere un'affidabilità ottimale, si consiglia di implementare il servizio su un minimo di tre zone di disponibilità.

Accesso alle tabelle DynamoDB e alle operazioni API di controllo dagli endpoint di interfaccia DynamoDB

È possibile utilizzare AWS CLI o AWS SDKs per accedere alle tabelle DynamoDB e controllare le operazioni delle API tramite gli endpoint dell'interfaccia DynamoDB.

AWS CLI esempi

Per accedere alle tabelle DynamoDB o alle operazioni dell'API di controllo DynamoDB tramite gli endpoint dell'interfaccia DynamoDB nei comandi, utilizza i parametri and. AWS CLI `--region` `--endpoint-url`

Esempio: creazione di un endpoint 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
```

Esempio: modifica di un endpoint 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
```

Esempio: creazione di un elenco di tabelle utilizzando un URL dell'endpoint

Nell'esempio seguente, sostituisci la Regione `us-east-1` e il nome DNS dell'ID dell'endpoint VPC `vpce-1a2b3c4d-5e6f.dynamodb.us-east-1.vpce.amazonaws.com` con le informazioni appropriate.

```
aws dynamodb --region us-east-1 --endpoint https://vpce-1a2b3c4d-5e6f.dynamodb.us-
east-1.vpce.amazonaws.com list-tables
```

AWS Esempi SDK

Per accedere alle tabelle DynamoDB o alle operazioni dell'API di controllo DynamoDB tramite gli endpoint dell'interfaccia DynamoDB quando si utilizza DynamoDB, è necessario aggiornare la versione alla versione più recente. AWS SDKs SDKs Quindi configura i client per utilizzare un URL dell'endpoint per accedere a una tabella o a un'operazione API di controllo DynamoDB tramite gli endpoint di interfaccia DynamoDB.

SDK for Python (Boto3)

Esempio: utilizzo di un URL dell'endpoint per accedere a una tabella DynamoDB

Nell'esempio seguente, sostituisci la Regione `us-east-1` e l'ID endpoint VPC `https://vpce-1a2b3c4d-5e6f.dynamodb.us-east-1.vpce.amazonaws.com` con le informazioni appropriate.

```
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

Esempio: utilizzo di un URL dell'endpoint per accedere a una tabella DynamoDB

Nell'esempio seguente, sostituisci la Regione `us-east-1` e l'ID endpoint VPC `https://vpce-1a2b3c4d-5e6f.dynamodb.us-east-1.vpce.amazonaws.com` con le informazioni appropriate.

```
//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

Esempio: utilizzo di un URL dell'endpoint per accedere a una tabella DynamoDB

Nell'esempio seguente, sostituisci la regione us-east-1 e l'ID https://vpce-1a2b3c4d-5e6f.dynamodb.us-east-1.vpce.amazonaws.com endpoint VPC con le tue informazioni.

```
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()
```

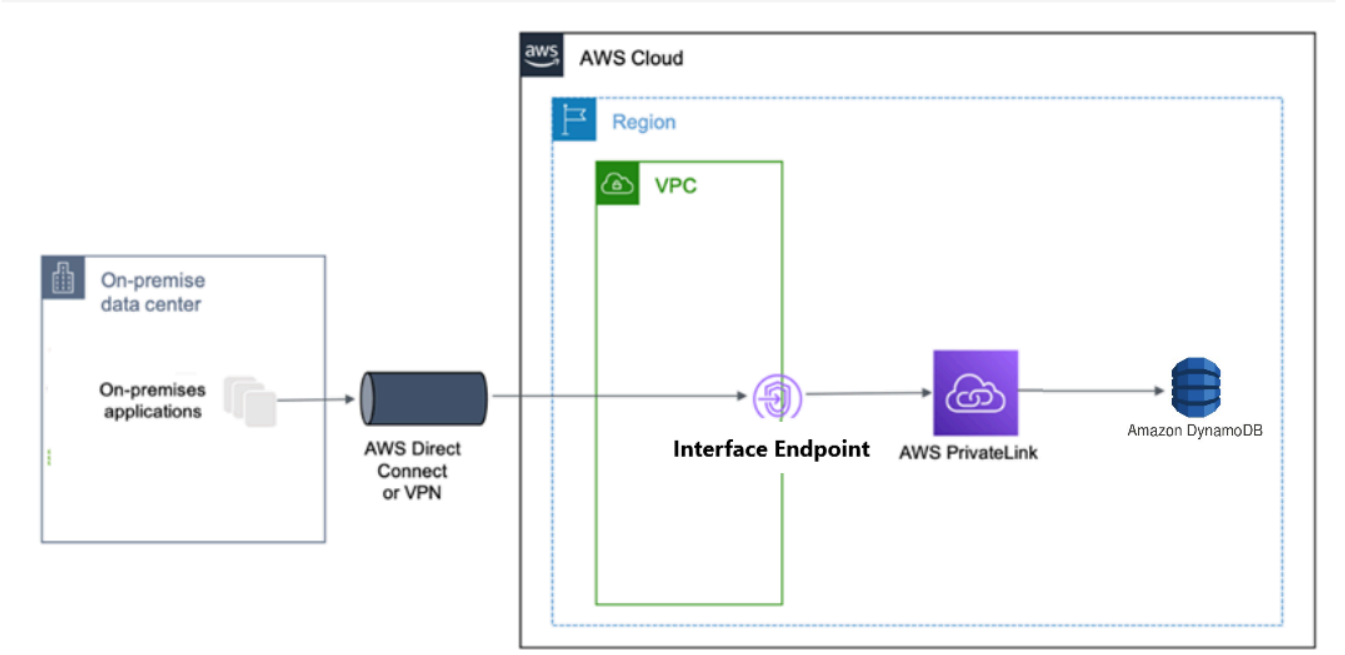
Aggiornamento di una configurazione DNS locale

Quando si utilizzano nomi DNS specifici degli endpoint per accedere agli endpoint di interfaccia per DynamoDB, non è necessario aggiornare il resolver DNS on-premises. È possibile risolvere il nome

DNS specifico dell'endpoint con l'indirizzo IP privato dell'endpoint di interfaccia dal dominio DNS pubblico di DynamoDB.

Utilizzo degli endpoint di interfaccia per accedere a DynamoDB senza un endpoint gateway o un gateway Internet nell'Amazon VPC

Gli endpoint di interfaccia nell'Amazon VPC possono instradare sia le applicazioni nell'Amazon VPC che le applicazioni on-premises a DynamoDB sulla rete Amazon, come illustrato nel diagramma seguente.

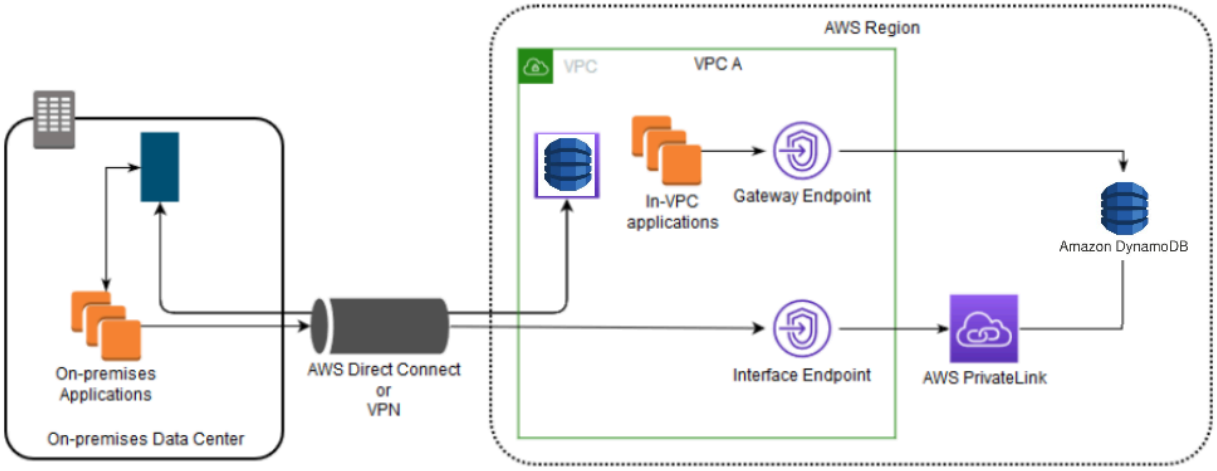


Il diagramma illustra quanto segue:

- La tua rete locale utilizza Direct Connect o Site-to-Site VPN per connettersi ad Amazon VPC A.
- Le applicazioni on-premises e nell'Amazon VPC A utilizzano nomi DNS specifici degli endpoint per accedere a DynamoDB tramite l'endpoint di interfaccia DynamoDB.
- Le applicazioni locali inviano dati all'endpoint di interfaccia in Amazon VPC tramite Direct Connect (o). Site-to-Site VPN AWS PrivateLink sposta i dati dall'endpoint dell'interfaccia a DynamoDB attraverso la rete. AWS
- Le applicazioni VPC di Amazon inviano inoltre traffico all'endpoint dell'interfaccia. AWS PrivateLink sposta i dati dall'endpoint dell'interfaccia a DynamoDB attraverso la rete. AWS

Utilizzo di endpoint gateway e di endpoint di interfaccia insieme nello stesso Amazon VPC per accedere a DynamoDB

È possibile creare endpoint di interfaccia e mantenere l'endpoint gateway esistente nello stesso Amazon VPC, come illustrato nel diagramma seguente. Con questo approccio si consente alle applicazioni nell'Amazon VPC di continuare ad accedere a DynamoDB tramite l'endpoint gateway senza incorrere nella fatturazione. Pertanto, solo le applicazioni on-premises utilizzerebbero gli endpoint di interfaccia per accedere a DynamoDB. Per accedere a DynamoDB in questo modo, è necessario aggiornare le applicazioni on-premises per utilizzare nomi DNS specifici degli endpoint per DynamoDB.



Il diagramma illustra quanto segue:

- Le applicazioni locali utilizzano nomi DNS specifici dell'endpoint per inviare dati all'endpoint di interfaccia all'interno di Amazon VPC tramite (o). Direct Connect Site-to-Site VPN AWS PrivateLink sposta i dati dall'endpoint dell'interfaccia a DynamoDB attraverso la rete. AWS
- Utilizzando nomi DynamoDB regionali predefiniti, le applicazioni VPC interne ad Amazon inviano dati all'endpoint gateway che si connette a DynamoDB tramite la rete. AWS

Per ulteriori informazioni sugli endpoint gateway, consulta Gateway Amazon VPC endpoints nella Guida per l'utente di Amazon VPC.

Creazione di una policy sugli endpoint Amazon VPC per DynamoDB

È possibile collegare una policy sugli endpoint al proprio endpoint Amazon VPC che controlla l'accesso a DynamoDB. Questa policy specifica le informazioni riportate di seguito:

- Il principale AWS Identity and Access Management (IAM) che può eseguire azioni
- Le azioni che possono essere eseguite
- Le risorse sui cui si possono eseguire le azioni

Argomenti

- Esempio: limitazione dell'accesso a una tabella specifica da un endpoint Amazon VPC

Esempio: limitazione dell'accesso a una tabella specifica da un endpoint Amazon VPC

È possibile creare una policy sugli endpoint che limita l'accesso solo a una tabella DynamoDB specifica. Questo tipo di policy è utile se Servizi AWS nel tuo Amazon VPC ne hai altre che utilizzano tabelle. La seguente policy sulle tabella limita l'accesso solo a *DOC-EXAMPLE-TABLE*. Per utilizzare questa policy sugli endpoint, sostituisci *DOC-EXAMPLE-TABLE* con il nome della tabella.

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/*"]
    }
  ]
}
```

Utilizzo degli endpoint DynamoDB con accesso privato a Console di gestione AWS

È necessario configurare la configurazione DNS per DynamoDB e i flussi DynamoDB quando si utilizzano endpoint VPC con la console DynamoDB nell'Accesso privato alla Console di gestione AWS.

Per configurare DynamoDB in modo che sia accessibile Console di gestione AWS in Private Access, è necessario creare i due endpoint VPC seguenti:

- `com.amazonaws.<region>.dynamodb`
- `com.amazonaws.<region>.dynamodb-streams`

Durante la creazione degli endpoint VPC, accedi alla console Route53 e crea una zona ospitata privata per DynamoDB utilizzando l'endpoint Regionale `dynamodb.us-east-1.amazonaws.com`.

Crea i seguenti due record di alias nella zona ospitata privata:

- `dynamodb.<region>.amazonaws.com` che instrada il traffico verso l'endpoint VPC `com.amazonaws.<region>.dynamodb`.
- `streams.dynamodb.<region>.amazonaws.com` che instrada il traffico verso l'endpoint VPC `com.amazonaws.<region>.dynamodb-streams`.

AWS PrivateLink per DynamoDB Streams

Con AWS PrivateLink Amazon DynamoDB Streams, puoi fornire endpoint Amazon VPC di interfaccia (endpoint di interfaccia) nel tuo cloud privato virtuale (Amazon VPC). Questi endpoint sono accessibili direttamente dalle applicazioni locali tramite VPN e/o in un altro modo Regione AWS tramite il peering Amazon VPC. Direct Connect Utilizzando AWS PrivateLink e interfacciando gli endpoint, puoi semplificare la connettività di rete privata dalle tue applicazioni a DynamoDB Streams.

Le applicazioni presenti in Amazon VPC non richiedono indirizzi IP pubblici per comunicare con i flussi DynamoDB tramite gli endpoint Amazon VPC di interfaccia per le operazioni con i flussi DynamoDB. Gli endpoint dell'interfaccia sono rappresentati da una o più interfacce di rete elastiche (ENIs) a cui vengono assegnati indirizzi IP privati dalle sottoreti del tuo Amazon VPC. Le richieste ai flussi DynamoDB tramite gli endpoint di interfaccia rimangono nella rete Amazon. Puoi anche accedere agli endpoint di interfaccia nel tuo Amazon VPC da applicazioni locali Direct Connect

tramite AWS Virtual Private Network o AWS (VPN). Per ulteriori informazioni su come connettersi alla rete locale, AWS Virtual Private Network consulta la Guida per l'utente e la Guida per Direct Connect l'utente.AWS Site-to-Site VPN

Per informazioni sulla creazione di endpoint di interfaccia, consulta Endpoint di interfaccia di Amazon VPC (AWS PrivateLink).

> **Note**
>
> Per i flussi DynamoDB sono supportati solo gli endpoint di interfaccia. Gli endpoint gateway non sono supportati.

Argomenti

- Considerazioni sull'utilizzo di AWS PrivateLink per i flussi Amazon DynamoDB
- Creazione di un endpoint Amazon VPC
- Accesso agli endpoint di interfaccia dei flussi Amazon DynamoDB
- Accesso alle operazioni API dei flussi DynamoDB dagli endpoint di interfaccia dei flussi DynamoDB
- AWS Esempi SDK
- Creazione di una policy sugli endpoint Amazon VPC per i flussi DynamoDB
- Utilizzo degli endpoint DynamoDB con accesso privato a Console di gestione AWS

Considerazioni sull'utilizzo di AWS PrivateLink per i flussi Amazon DynamoDB

Le considerazioni relative ad Amazon VPC si applicano ad Amazon AWS PrivateLink DynamoDB Streams. Per ulteriori informazioni, consulta interface endpoint considerations e AWS PrivateLink quotas. Le restrizioni si applicano come segue.

AWS PrivateLink per Amazon DynamoDB Streams non supporta quanto segue:

- Transport Layer Security (TLS) 1.1
- Servizi del sistema dei nomi di dominio (DNS) privati e ibridi

> **Note**
>
> I timeout di connettività di rete verso gli AWS PrivateLink endpoint non rientrano nell'ambito delle risposte di errore di DynamoDB Streams e devono essere gestiti in modo appropriato dalle applicazioni che si connettono agli endpoint. AWS PrivateLink

Creazione di un endpoint Amazon VPC

Per creare un endpoint Amazon VPC di interfaccia, consulta Create an Amazon VPC endpoint nella Guida ad AWS PrivateLink .

Accesso agli endpoint di interfaccia dei flussi Amazon DynamoDB

Quando si crea un endpoint di interfaccia, DynamoDB genera due tipi di nomi DNS dei flussi DynamoDB specifici dell'endpoint: Regionale e zonale.

- Un nome DNS regionale include un ID endpoint Amazon VPC univoco, un identificatore di servizio, Regione AWS`vpce.amazonaws.com` la e nel nome. Ad esempio, per l'ID dell'endpoint Amazon VPC *`vpce-1a2b3c4d`*, il nome DNS generato potrebbe essere simile a *`vpce-1a2b3c4d-5e6f`*`.streams.dynamodb.us-east-1.vpce.amazonaws.com`.
- I nomi DNS zonali includono la zona di disponibilità, ad esempio *`vpce-1a2b3c4d-5e6f`***`-us-east-1a.`**`streams.dynamodb.us-east-1.vpce.amazonaws.com`. Puoi utilizzare questa opzione se l'architettura isola le zone di disponibilità. Ad esempio, puoi utilizzarla per il contenimento degli errori o per ridurre i costi di trasferimento dei dati a livello regionale.

Accesso alle operazioni API dei flussi DynamoDB dagli endpoint di interfaccia dei flussi DynamoDB

È possibile utilizzare AWS CLI o AWS SDKs per accedere alle operazioni dell'API DynamoDB Streams tramite gli endpoint dell'interfaccia DynamoDB Streams.

AWS CLI esempi

Per accedere a DynamoDB Streams o alle operazioni API tramite gli endpoint dell'interfaccia DynamoDB Streams nei comandi, utilizza i parametri and. AWS CLI `--region` `--endpoint-url`

Esempio: creazione di un endpoint 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
```

Esempio: modifica di un endpoint 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
```

Esempio: creazione di un elenco di flussi utilizzando un URL dell'endpoint

Nell'esempio seguente, sostituisci la Regione `us-east-1` e il nome DNS dell'ID dell'endpoint VPC `vpce-1a2b3c4d-5e6f.streams.dynamodb.us-east-1.vpce.amazonaws.com` con le informazioni appropriate.

```
aws dynamodbstreams --region us-east-1 –endpoint https://
vpce-1a2b3c4d-5e6f.streams.dynamodb.us-east-1.vpce.amazonaws.com list-streams
```

AWS Esempi SDK

Per accedere alle operazioni dell'API Amazon DynamoDB Streams tramite gli endpoint dell'interfaccia DynamoDB Streams quando si utilizza la, è necessario aggiornare la versione alla versione più recente. AWS SDKs SDKs Quindi configura i client per utilizzare un URL dell'endpoint per un'operazione API dei flussi DynamoDB tramite gli endpoint di interfaccia dei flussi DynamoDB.

SDK for Python (Boto3)

Esempio: utilizzo di un URL dell'endpoint per accedere a un flusso DynamoDB

Nell'esempio seguente, sostituisci la Regione `us-east-1` e l'ID endpoint VPC `https://vpce-1a2b3c4d-5e6f.streams.dynamodb.us-east-1.vpce.amazonaws.com` con le informazioni appropriate.

```
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

Esempio: utilizzo di un URL dell'endpoint per accedere a un flusso DynamoDB

Nell'esempio seguente, sostituisci la Regione `us-east-1` e l'ID endpoint VPC `https://vpce-1a2b3c4d-5e6f.streams.dynamodb.us-east-1.vpce.amazonaws.com` con le informazioni appropriate.

```
//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

Esempio: utilizzo di un URL dell'endpoint per accedere a un flusso DynamoDB

Nell'esempio seguente, sostituisci la Regione `us-east-1` e l'ID endpoint VPC `https://vpce-1a2b3c4d-5e6f.streams.dynamodb.us-east-1.vpce.amazonaws.com` con le informazioni appropriate.

```
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()
```

Creazione di una policy sugli endpoint Amazon VPC per i flussi DynamoDB

È possibile collegare una policy sugli endpoint all'endpoint Amazon VPC che controlla l'accesso ai flussi DynamoDB. Questa policy specifica le informazioni riportate di seguito:

- Il principio AWS Identity and Access Management (IAM) che può eseguire azioni
- Le azioni che possono essere eseguite
- Le risorse sui cui si possono eseguire le azioni

Argomenti

- Esempio: limitazione dell'accesso a un flusso specifico da un endpoint Amazon VPC

Esempio: limitazione dell'accesso a un flusso specifico da un endpoint Amazon VPC

È possibile creare una policy sugli endpoint che limita l'accesso solo a flussi DynamoDB specifici. Questo tipo di policy è utile se Servizi AWS nel tuo Amazon VPC ne hai altre che utilizzano DynamoDB Streams. La seguente policy sui flussi limita l'accesso solo al flusso *2025-02-20T11:22:33.444* collegato a *DOC-EXAMPLE-TABLE*. Per utilizzare questa policy sugli endpoint, sostituisci *DOC-EXAMPLE-TABLE* con il nome della tabella e *2025-02-20T11:22:33.444* con l'etichetta del flusso.

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**
>
> Gli endpoint gateway non sono supportati nei flussi DynamoDB.

Utilizzo degli endpoint DynamoDB con accesso privato a Console di gestione AWS

È necessario configurare la configurazione DNS per DynamoDB e i flussi DynamoDB quando si utilizzano endpoint VPC con la console DynamoDB nell'Accesso privato alla Console di gestione AWS.

Per configurare DynamoDB in modo che sia accessibile Console di gestione AWS in Private Access, è necessario creare i due endpoint VPC seguenti:

- `com.amazonaws.<region>.dynamodb`
- `com.amazonaws.<region>.dynamodb-streams`

Durante la creazione degli endpoint VPC, accedi alla console Route53 e crea una zona ospitata privata per DynamoDB utilizzando l'endpoint Regionale `dynamodb.us-east-1.amazonaws.com`.

Crea i seguenti due record di alias nella zona ospitata privata:

- `dynamodb.<region>.amazonaws.com` che instrada il traffico verso l'endpoint VPC `com.amazonaws.<region>.dynamodb`.
- `streams.dynamodb.<region>.amazonaws.com` che instrada il traffico verso l'endpoint VPC `com.amazonaws.<region>.dynamodb-streams`.

Utilizzo AWS PrivateLink per DynamoDB Accelerator (DAX)

AWS PrivateLink for DynamoDB Accelerator (DAX) consente di accedere in modo sicuro alla APIs gestione DAX, ad esempio `CreateCluster` `DeleteCluster` e tramite indirizzi IP privati`DescribeClusters`, all'interno del cloud privato virtuale (VPC). Questa funzionalità consente di accedere ai servizi DAX in modo privato dalle applicazioni senza esporre il traffico alla rete Internet pubblica.

DAX PrivateLink supporta gli endpoint dual-stack (), abilitando entrambi e la connettività. `dax.{region}.api.aws` IPv4 IPv6 Con AWS PrivateLink for DAX, i clienti possono accedere al

servizio utilizzando nomi DNS privati. Il supporto dual-stack per endpoint garantisce una connettività trasparente mantenendo al contempo la privacy della rete. Ciò consente di accedere a DAX tramite Internet pubblico e endpoint VPC senza apportare modifiche alla configurazione dell'SDK.

Considerazioni sull'utilizzo AWS PrivateLink per DynamoDB Accelerator (DAX)

Durante l'implementazione AWS PrivateLink di DynamoDB Accelerator (DAX), è necessario tenere conto di diverse considerazioni importanti.

Prima di configurare un endpoint di interfaccia per DAX, considerate quanto segue:

- Gli endpoint dell'interfaccia DAX supportano solo l'accesso alla gestione DAX all'interno dello stesso. APIs Regione AWS Non è possibile utilizzare un endpoint di interfaccia per accedere alla gestione DAX in altre regioni. APIs
- Per accedere alla gestione Console di gestione AWS privata di DAX, potrebbe essere necessario creare endpoint VPC aggiuntivi per servizi simili e servizi correlati. `com.amazonaws.region.console`
- La creazione e l'utilizzo di un endpoint di interfaccia verso DAX sono a pagamento. Per informazioni sui prezzi, consulta Prezzi di AWS PrivateLink.

Come AWS PrivateLink funziona con DAX

Quando si crea un endpoint di interfaccia per DAX:

1. AWS crea un'interfaccia di rete endpoint in ogni sottorete abilitata per l'endpoint dell'interfaccia.
2. Si tratta di interfacce di rete gestite dai richiedenti che fungono da punti di ingresso per il traffico destinato a DAX.
3. Puoi quindi accedere a DAX tramite indirizzi IP privati all'interno del tuo VPC.
4. Questa architettura consente di utilizzare i gruppi di sicurezza VPC per gestire l'accesso agli endpoint.
5. Le applicazioni possono accedere sia a DynamoDB che a DAX tramite i rispettivi endpoint di interfaccia all'interno di un VPC, permettendo anche alle applicazioni locali di connettersi tramite Direct Connect o VPN.
6. Ciò fornisce un modello di connettività coerente per entrambi i servizi, semplifica l'architettura e migliora la sicurezza mantenendo il traffico all'interno della rete. AWS

Creazione di endpoint di interfaccia per DAX

È possibile creare un endpoint di interfaccia per connettersi a DAX utilizzando l' Console di gestione AWS AWS SDK o l'API. CloudFormation AWS

Per creare un endpoint di interfaccia per DAX utilizzando la console

1. Accedi alla console Amazon VPC all'indirizzo. https://console.aws.amazon.com/vpc/
2. Nel pannello di navigazione, seleziona Endpoints (Endpoint).
3. Scegliere Create Endpoint (Crea endpoint).
4. Per la categoria del servizio, scegli Servizi AWSe per Nome del servizio, cerca e seleziona`com.amazonaws.region.dax`.
5. Per VPC, seleziona il VPC da cui desideri accedere a DAX e per le sottoreti, seleziona le sottoreti in cui verranno create le interfacce di rete degli endpoint. AWS
6. Per i gruppi di sicurezza, seleziona o crea gruppi di sicurezza da associare alle interfacce di rete degli endpoint.
7. Per Policy, mantieni l'accesso completo predefinito o personalizzalo secondo necessità.
8. Seleziona Abilita nome DNS per abilitare il DNS privato per l'endpoint. Mantieni abilitato il nome DNS privato per impedire modifiche alla configurazione dell'SDK. Se abilitato, le applicazioni possono continuare a utilizzare il nome DNS del servizio standard (esempio:). `dax.region.amazonaws.com` AWS crea una zona ospitata privata nel tuo VPC che risolve questo nome nell'indirizzo IP privato dell'endpoint.

   > **Note**
   >
   > Se necessario, utilizzate i nomi DNS regionali. L'uso di nomi DNS zonali non è consigliato. Inoltre, seleziona 3 o più sottoreti AZs per garantire la massima disponibilità. PrivateLink

9. Seleziona Crea endpoint.

Per creare un endpoint di interfaccia per DAX utilizzando AWS CLI

Utilizzare il `create-vpc-endpoint` comando con il `vpc-endpoint-type` parametro impostato su `Interface` e il `service-name` parametro impostato su. `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
```

Risorse aggiuntive

Per ulteriori informazioni sugli AWS PrivateLink endpoint VPC, consulta le seguenti risorse:

- AWS PrivateLink per DynamoDB
- AWS PrivateLink per DynamoDB Streams
- Connect il tuo VPC ai servizi utilizzando AWS PrivateLink
- Semplifica la connettività privata a DynamoDB con AWS PrivateLink
- AWS PrivateLink Whitepaper

Analisi della configurazione e delle vulnerabilità in Amazon DynamoDB

AWS gestisce le attività di sicurezza di base come l'applicazione di patch al sistema operativo (OS) guest e al database, la configurazione del firewall e il disaster recovery. Queste procedure sono state riviste e certificate dalle terze parti appropriate. Per ulteriori dettagli, consulta le seguenti risorse :

- Convalida della conformità per Amazon DynamoDB
- Modello di responsabilità condivisa
- Amazon Web Services: panoramica dei processi di sicurezza (whitepaper)

Le best practice di sicurezza seguenti gestiscono anche l'analisi di configurazione e vulnerabilità in Amazon DynamoDB:

- Monitora la conformità di DynamoDB con Regole di AWS Config
- Monitora la configurazione di DynamoDB con AWS Config

Best practice di sicurezza per Amazon DynamoDB

Amazon DynamoDB fornisce una serie di caratteristiche di sicurezza che occorre valutare durante lo sviluppo e l'implementazione delle policy di sicurezza. Le seguenti best practice sono linee guida generali e non rappresentano una soluzione di sicurezza completa. Poiché queste best practice potrebbero non essere appropriate o sufficienti per l'ambiente, sono da considerare come considerazioni utili anziché prescrizioni.

Argomenti

- Best practice relative alla sicurezza di prevenzione di DynamoDB
- Best practice relative alla sicurezza di rilevamento di DynamoDB

Best practice relative alla sicurezza di prevenzione di DynamoDB

Le seguenti best practice consentono di anticipare ed evitare incidenti di sicurezza in Amazon DynamoDB.

Crittografia dei dati a riposo

DynamoDB esegue la crittografia dei dati a riposo per tutti i dati utente archiviati in tabelle, indici, flussi e backup utilizzando chiavi di crittografia archiviate in AWS Key Management Service (AWS KMS). Questo fornisce un livello aggiuntivo di protezione dei dati dagli accessi non autorizzati allo storage sottostante.

È possibile specificare se DynamoDB deve utilizzare Chiave di proprietà di AWS una chiave (tipo di crittografia predefinito), una o Chiave gestita da AWS una chiave gestita dal cliente per crittografare i dati degli utenti. Per ulteriori informazioni, consulta la sezione relativa alla crittografia dei dati inattivi di Amazon DynamoDB.

Utilizzo di ruoli IAM per autenticare l'accesso a DynamoDB

Affinché utenti, applicazioni e altri AWS servizi possano accedere a DynamoDB, devono includere credenziali AWS valide nelle loro richieste API. AWS Non è necessario archiviare AWS le credenziali direttamente nell'applicazione o nell'istanza. EC2 Si tratta di credenziali a lungo termine che non vengono automaticamente ruotate e, pertanto, potrebbero avere un impatto aziendale significativo se compromesse. Un ruolo IAM consente di ottenere chiavi di accesso temporanee che possono essere utilizzate per accedere a AWS servizi e risorse.

Per ulteriori informazioni, consulta Identity and Access Management per Amazon DynamoDB.

Utilizzo di policy IAM per autorizzazione di base di DynamoDB

Quando concedi le autorizzazioni, sei tu a decidere chi le ottiene, per quale DynamoDB APIs ottengono le autorizzazioni e le azioni specifiche che desideri consentire su tali risorse. L'implementazione del privilegio minimo è fondamentale per ridurre i rischi di sicurezza e l'impatto che può risultare da errori o intenzioni dannose.

Collegare policy di autorizzazione a identità IAM (ovvero, utenti, gruppi e ruoli) e concedere in tal modo le autorizzazioni per eseguire operazioni su risorse DynamoDB.

Puoi farlo usando quanto segue:

- AWS Politiche gestite (predefinite)
- Policy gestite dal cliente

Utilizzo di condizioni di policy IAM per il controllo granulare degli accessi

Quando si concedono le autorizzazioni in DynamoDB, è possibile specificare le condizioni che determinano il modo in cui una policy di autorizzazioni viene applicata. L'implementazione del privilegio minimo è fondamentale per ridurre i rischi di sicurezza e l'impatto che può risultare da errori o intenzioni dannose.

Puoi specificare le condizioni durante la concessione delle autorizzazioni utilizzando una policy IAM. Ad esempio, puoi eseguire le operazioni seguenti:

- Concedere autorizzazioni per permettere agli utenti accesso in sola lettura a determinati elementi e attributi in una tabella o indice secondario.
- Concedere autorizzazioni per permettere agli utenti di accedere in sola scrittura a determinati attributi in una tabella, in base all'identità di tale utente.

Per ulteriori informazioni, consulta Utilizzo di condizioni di policy IAM per il controllo degli accessi fine-grained.

Utilizzo di un endpoint VPC e delle policy per accedere a DynamoDB

Se si richiede l'accesso a DynamoDB solo da un virtual private cloud (VPC), è necessario utilizzare un endpoint VPC per limitare l'accesso solo dal VPC richiesto. Questo impedisce che il traffico attraversi l'Internet aperto e che sia soggetto a tale ambiente.

L'utilizzo di un endpoint VPC per DynamoDB consente di controllare e imitare l'accesso utilizzando quanto segue:

- Policy di endpoint VPC: queste policy vengono applicate all'endpoint VPC di DynamoDB. Consentono inoltre di controllare e limitare l'accesso API alla tabella DynamoDB.
- Policy IAM: utilizzando la condizione `aws:sourceVpce` sulle policy collegate a utenti, gruppi o ruoli, è possibile imporre che tutti gli accessi alla tabella DynamoDB avvengano tramite l'endpoint VPC specificato.

Per ulteriori informazioni, consulta Endpoint per Amazon DynamoDB.

Valutazione della crittografia lato client

Consigliamo di pianificare la strategia di crittografia prima di implementare la tabella in DynamoDB. Se archivi dati sensibili o riservati in DynamoDB, prendi in considerazione l'inclusione della crittografia lato client nel piano. In questo modo puoi crittografare i dati il più vicino possibile alla loro origine e garantirne la protezione per tutto il loro ciclo di vita. Grazie alla crittografia dei dati sensibili in transito e inattivi, puoi accertarti che i dati di testo non crittografato non siano disponibili per terze parti.

AWS Database Encryption SDK for DynamoDB (SDK di crittografia del database AWS per DynamoDB) è una libreria software che consente di proteggere i dati della tabella prima di inviarli a DynamoDB. Crittografa, firma, verifica e decrittografa gli elementi della tabella DynamoDB. L'utente controlla quali attributi sono crittografati e firmati.

Considerazioni sulla chiave primaria

Non utilizzare nomi sensibili o dati sensibili in testo normale nella chiave primaria per la tabella e gli indici secondari globali. I nomi delle chiavi verranno visualizzati nella definizione della tabella. Ad esempio, i nomi delle chiavi primarie sono accessibili a chiunque disponga delle autorizzazioni per effettuare chiamate. DescribeTable I valori delle chiavi possono essere visualizzati nei tuoi registri AWS CloudTraile in altri. Inoltre, DynamoDB utilizza i valori chiave per distribuire i dati e instradare le richieste AWS e gli amministratori possono osservare i valori per mantenere l'integrità del servizio.

Se è necessario utilizzare dati sensibili nella tabella o nei valori della chiave GSI, si consiglia di utilizzare la crittografia client. end-to-end Ciò consente di eseguire riferimenti chiave-valore ai dati garantendo al contempo che non appaiano mai non crittografati nei log relativi a DynamoDB. Un modo per farlo è utilizzare l'AWS Database Encryption SDK for DynamoDB, sebbene non sia obbligatorio. In caso di utilizzo di una soluzione personalizzata, sarebbe sempre opportuno utilizzare un algoritmo di crittografia sufficientemente sicuro. Non è opportuno utilizzare un'opzione non crittografica come un hash, poiché nella maggior parte delle situazioni non sono considerate sufficientemente sicure.

Se i nomi delle chiavi primarie sono sensibili, si consiglia di utilizzare invece `pk` e `sk`. Questa è una best practice generale che lascia flessibile la progettazione della chiave di partizione.

Consulta sempre i tuoi esperti di sicurezza o il team addetto all' AWS account se sei preoccupato sulla scelta giusta.

Best practice relative alla sicurezza di rilevamento di DynamoDB

Le best practice seguenti per Amazon DynamoDB consentono di rilevare potenziali debolezze e incidenti di sicurezza.

Utilizzalo AWS CloudTrail per monitorare l'utilizzo delle chiavi KMS AWS gestite

Se si utilizza un file Chiave gestita da AWSper la crittografia a riposo, viene registrato l'utilizzo di questa chiave. AWS CloudTrail CloudTrail fornisce visibilità sull'attività degli utenti registrando le azioni intraprese sul tuo account. CloudTrail registra informazioni importanti su ogni azione, tra cui chi ha effettuato la richiesta, i servizi utilizzati, le azioni eseguite, i parametri delle azioni e gli elementi di risposta restituiti dal AWS servizio. Queste informazioni aiutano a tenere traccia delle modifiche apportate alle AWS risorse e a risolvere i problemi operativi. CloudTrail semplifica la garanzia della conformità alle politiche interne e agli standard normativi.

È possibile utilizzare CloudTrail per controllare l'utilizzo delle chiavi. CloudTrail crea file di registro che contengono una cronologia delle chiamate AWS API e degli eventi correlati per il tuo account. Questi file di registro includono tutte le richieste AWS KMS API effettuate utilizzando gli strumenti da riga di comando e quelle effettuate tramite AWS servizi integrati. Console di gestione AWS AWS SDKs Puoi utilizzare questi file di log per ottenere informazioni su quando la chiave KMS è stata usata, l'operazione richiesta, l'identità del richiedente, l'indirizzo IP da cui proviene la richiesta e così via. Per ulteriori informazioni, consulta Registrazione di chiamate API di AWS KMS con AWS CloudTrail nella Guida per l'utente di AWS CloudTrail.

Monitora le operazioni di DynamoDB utilizzando CloudTrail

CloudTrail può monitorare sia gli eventi del piano di controllo che gli eventi del piano dati. Le operazioni del piano di controllo consentono di creare e gestire le tabelle DynamoDB. Ti permettono anche di utilizzare indici, flussi e altri oggetti che dipendono dalle tabelle. Le operazioni del piano dati consentono di eseguire operazioni di creazione, lettura, aggiornamento ed eliminazione (chiamate anche CRUD) sui dati in una tabella. Alcune operazioni del piano dati consentono inoltre di leggere i dati da un indice secondario. Per abilitare la registrazione degli

eventi del piano dati CloudTrail, è necessario abilitare la registrazione dell'attività dell'API del piano dati in. CloudTrail Per ulteriori informazioni, consulta Logging data events for trails.

Quando si verifica un'attività in DynamoDB, tale attività viene registrata in CloudTrail un evento insieme ad AWS altri eventi di servizio nella cronologia degli eventi. Per ulteriori informazioni, consulta Registrazione delle operazioni di DynamoDB con AWS CloudTrail. Puoi visualizzare, cercare e scaricare gli eventi recenti nel tuo AWS account. Per ulteriori informazioni, consulta Visualizzazione degli eventi con cronologia degli CloudTrail eventi nella Guida AWS CloudTrail per l'utente.

Per una registrazione continua degli eventi nel tuo AWS account, inclusi gli eventi per DynamoDB, crea un percorso. Un trail consente di CloudTrail inviare file di log a un bucket Amazon Simple Storage Service (Amazon S3). Per impostazione predefinita, quando crei un percorso sulla console, il percorso si applica a tutte le AWS regioni. Il percorso registra gli eventi di tutte le regioni nella partizione AWS e distribuisce i file di log nel bucket S3 specificato. Inoltre, puoi configurare altri AWS servizi per analizzare ulteriormente e agire in base ai dati sugli eventi raccolti nei CloudTrail log.

Utilizzo di DynamoDB Streams per monitorare le operazioni del piano dati

DynamoDB è integrato AWS Lambda in modo da poter creare trigger, parti di codice che rispondono automaticamente agli eventi in DynamoDB Streams. Con i trigger è possibile creare applicazioni che rispondono alle modifiche di dati nelle tabelle DynamoDB.

Se si abilita DynamoDB Streams su una tabella, è possibile associare l'Amazon Resource Name (ARN) del flusso a una funzione Lambda da te scritta. Immediatamente dopo la modifica di un elemento della tabella, viene visualizzato un nuovo record nello stream della tabella. AWS Lambda interroga lo stream e richiama la funzione Lambda in modo sincrono quando rileva nuovi record di stream. La funzione Lambda può eseguire qualsiasi operazione specificata, come l'invio di una notifica o l'inizializzazione di un flusso di lavoro.

Per un esempio, consulta Tutorial: utilizzo di AWS Lambda con Amazon DynamoDB Streams. Questo esempio riceve l'input di un evento DynamoDB, elabora i messaggi in esso contenuti e scrive alcuni dei dati degli eventi in entrata su Amazon Logs. CloudWatch

Monitora la configurazione di DynamoDB con AWS Config

Grazie a AWS Config, puoi monitorare continuamente e registrare le modifiche alla configurazione delle risorse AWS . Puoi anche usarla AWS Config per inventariare le tue AWS risorse. Quando viene rilevata una modifica rispetto a uno stato precedente, è possibile distribuire una notifica di Amazon Simple Notification Service (Amazon SNS) in modo che possa essere rivista e si possa

intervenire. Segui le indicazioni in Configurazione AWS Config con la console, assicurandoti che i tipi di risorse DynamoDB siano inclusi.

Puoi configurare lo streaming delle modifiche AWS Config alla configurazione e delle notifiche su un argomento di Amazon SNS. Ad esempio, quando una risorsa viene aggiornata, puoi ricevere una notifica al tuo indirizzo email, in modo che tu possa visualizzare tali modifiche. Puoi anche ricevere notifiche quando AWS Config valuti le tue regole personalizzate o gestite rispetto alle tue risorse.

Per un esempio, consulta l'argomento Notifiche AWS Config inviate a un Amazon SNS nella AWS Config Developer Guide.

Monitora la conformità di DynamoDB alle regole AWS Config

AWS Config monitora continuamente le modifiche alla configurazione che si verificano tra le tue risorse. controllando se queste modifiche violano eventuali condizioni nelle regole. Se una risorsa viola una regola, AWS Config contrassegna la risorsa e la regola come non conformi.

Utilizzando AWS Config per valutare le configurazioni delle risorse, è possibile valutare la conformità delle configurazioni delle risorse alle pratiche interne, alle linee guida del settore e alle normative. AWS Config fornisce regole AWS gestite, che sono regole predefinite e personalizzabili che vengono AWS Config utilizzate per valutare se le AWS risorse sono conformi alle migliori pratiche comuni.

Assegnazione di tag alle risorse DynamoDB per identificazione e automazione

Puoi assegnare metadati alle tue AWS risorse sotto forma di tag. Ogni tag è una semplice etichetta composta da una chiave definita dal cliente e un valore facoltativo che può semplificare la gestione, la ricerca e il filtro delle risorse.

Il tagging consente l'implementazione di gruppi controllati. Anche se non ci sono tipi di tag inerenti, è possibile suddividere le risorse in base a scopo, proprietario, ambiente o altri criteri. Di seguito vengono mostrati alcuni esempi:

- Sicurezza - Utilizzato per determinare requisiti quali la crittografia.
- Riservatezza - Un identificatore per il livello di riservatezza dei dati specifico supportato da una risorsa.
- Ambiente: utilizzato per differenziare tra infrastruttura di sviluppo, test e produzione.

Per ulteriori informazioni, consulta Strategie di assegnazione di tag di AWS e Assegnazione di tag per DynamoDB.

Monitora l'utilizzo di Amazon DynamoDB in relazione alle best practice di sicurezza utilizzando. AWS Security Hub CSPM

Security Hub utilizza controlli di sicurezza per valutare le configurazioni delle risorse e gli standard di sicurezza per aiutarti a rispettare vari framework di conformità.

Per ulteriori informazioni sull'utilizzo di Security Hub volto a valutare le risorse DynamoDB, consulta Controlli di Amazon DynamoDB nella Guida per l'utente di AWS Security Hub CSPM .

Monitoraggio e registrazione di log in DynamoDB

Il monitoraggio è una parte importante per mantenere l'affidabilità, la disponibilità e le prestazioni di DynamoDB e delle soluzioni AWS. È necessario raccogliere i dati sul monitoraggio da tutte le parti delle soluzioni AWS per semplificare il debug di un guasto in più punti.

Argomenti

- Piano di monitoraggio
- Baseline delle prestazioni
- Servizi integrati
- Strumenti di monitoraggio automatici
- Monitoraggio di metriche in DynamoDB con Amazon CloudWatch
- Registrazione delle operazioni di DynamoDB con AWS CloudTrail
- Analisi dell'accesso ai dati utilizzando CloudWatch Contributor Insights per DynamoDB

Piano di monitoraggio

Prima di iniziare il monitoraggio di DynamoDB, crea un piano di monitoraggio che includa le risposte alle seguenti domande:

- Quali sono gli obiettivi del monitoraggio?
- Di quali risorse si intende eseguire il monitoraggio?
- Con quale frequenza sarà eseguito il monitoraggio di queste risorse?
- Quali strumenti di monitoraggio verranno utilizzati?
- Chi eseguirà i processi di monitoraggio?
- Chi deve ricevere una notifica quando si verifica un problema?

Baseline delle prestazioni

Stabilisci una baseline per le prestazioni normali di DynamoDB nell’ambiente, misurando le prestazioni in diversi momenti e con condizioni di carico differenti. Quando si esegue il monitoraggio di DynamoDB, si dovrebbe considerare di archiviare i dati storici sul monitoraggio. Questi dati archiviati forniranno una baseline rispetto a cui confrontare i dati sulle prestazioni correnti e

identificare i normali modelli o le anomalie di prestazioni e ideare metodi per risolvere i problemi. Per stabilire una baseline, è necessario monitorare almeno gli elementi seguenti:

- Il numero di unità di capacità di lettura o scrittura utilizzate nel periodo di tempo specificato, in modo da tenere traccia quanto throughput assegnato viene utilizzato.
- Le richieste che hanno superato la capacità di lettura o scrittura assegnata a una tabella durante il periodo di tempo specificato, in modo da determinare quali richieste superano le quote di throughput assegnato della tabella.
- Errori di sistema, in modo da determinare se qualche richiesta ha provocato un errore.

Servizi integrati

DynamoDB monitora automaticamente le tabelle per conto dell'utente e segnala le metriche tramite Amazon CloudWatch. Inoltre, DynamoDB si integra con i Servizi AWS seguenti per aiutare a monitorare e risolvere i problemi relativi alle proprie risorse DynamoDB.

- AWS CloudTrail acquisisce le chiamate API e gli eventi correlati effettuati da o per conto del tuo Account AWSe fornisce i file di log a un bucket Simple Storage Service (Amazon S3) specificato. Per ulteriori informazioni, consulta Registrazione delle operazioni di DynamoDB con AWS CloudTrail.
- Contributor Insights è uno strumento diagnostico per l'identificazione rapida delle chiavi con maggiori accessi e limitazione (della larghezza di banda della rete) nella tabella o nell'indice. Per ulteriori informazioni, consulta Analisi dell'accesso ai dati utilizzando CloudWatch Contributor Insights per DynamoDB.

Strumenti di monitoraggio automatici

AWS fornisce vari strumenti che possono essere utilizzati per monitorare DynamoDB. Si consiglia di automatizzare il più possibile i processi di monitoraggio. Per controllare DynamoDB e segnalare l'eventuale presenza di problemi, è possibile usare i seguenti strumenti di monitoraggio automatici:

- Allarmi Amazon CloudWatch: controllano una singola metrica per un periodo di tempo specificato ed eseguono una o più operazioni in base alla relazione tra il valore della metrica e una determinata soglia per più periodi di tempo.

L'operazione è una notifica inviata a un topic Amazon Simple Notification Service (Amazon SNS) o alla policy di Dimensionamento automatico Amazon EC2. Gli allarmi Amazon CloudWatch non invocano operazioni semplicemente perché sono in un determinato stato. È necessario che lo stato sia cambiato e che sia rimasto invariato per una serie specificata di periodi. Per ulteriori informazioni, consulta Monitoraggio di metriche in DynamoDB con Amazon CloudWatch.

- Monitoraggio dei log AWS CloudTrail: è possibile condividere file di log tra gli account, monitorare i file di log AWS CloudTrail in tempo reale inviandoli a AWS CloudTrail Logs, scrivere applicazioni di elaborazione dei log in Java e verificare che i file di log non siano cambiati dopo la distribuzione da parte di AWS CloudTrail. Per ulteriori informazioni, consulta What is Amazon CloudWatch Logs nella Guida per l'utente di AWS CloudTrail.

Monitoraggio di metriche in DynamoDB con Amazon CloudWatch

È possibile monitorare DynamoDB utilizzando CloudWatch, che raccoglie i dati non elaborati da DynamoDB e li elabora trasformandoli in parametri leggibili quasi in tempo reale. Queste statistiche vengono mantenute per un periodo di tempo, per permettere l'accesso alle informazioni storiche e offrire una prospettiva migliore sulle prestazioni del servizio o dell'applicazione web. Per impostazione predefinita, i dati dei parametri di DynamoDB vengono inviati automaticamente a CloudWatch. Per maggiori informazioni, consulta Cos'è Amazon CloudWatch? e Conservazione dei parametri nella Guida per l'utente di Amazon CloudWatch.

Argomenti

- Come si utilizzano i parametri di DynamoDB?
- Visualizzazione dei parametri nella console CloudWatch
- Visualizzazione delle metriche nella AWS CLI
- Parametri e dimensioni di DynamoDB
- Creazione di CloudWatch allarmi in DynamoDB

Come si utilizzano i parametri di DynamoDB?

I parametri forniti da DynamoDB offrono informazioni che possono essere analizzate in diversi modi. L'elenco seguente mostra alcuni usi comuni dei parametri. Questi suggerimenti sono solo introduttivi e non costituiscono un elenco completo.

Come si utilizzano i parametri di DynamoDB?

In che modo?	Parametri rilevanti
Come faccio a monitorare la frequenza di eliminazioni TTL nella tabella?	È possibile monitorare `TimeToLiveDeletedItemCount` nel periodo di tempo specificato per tenere traccia della frequenza di eliminazioni TTL nella tabella. Per un esempio di un'applicazione serverless che utilizza la metrica `TimeToLiveDeletedItemCount`, consulta Automatically archive items to S3 using DynamoDB Time to Live (TTL) with AWS Lambda and Amazon Data Firehose.
Come faccio a determinare quanta parte del throughput allocato viene utilizzata?	È possibile monitorare `ConsumedReadCapacityUnits` o `ConsumedWriteCapacityUnits` nel periodo di tempo specificato per tenere traccia di quanto del throughput assegnato viene utilizzato.
Come faccio a stabilire quali richieste superano le quote di throughput allocato di una tabella?	Il parametro `ThrottledRequests` viene incrementato di uno se qualsiasi evento in una richiesta supera le quote di throughput assegnato. Quindi, per scoprire quale evento limita una richiesta, confrontare `ThrottledRequests` con i parametri `ReadThrottleEvents` e `WriteThrottleEvents` della tabella e degli indici.
Come faccio a stabilire se si sono verificati errori di sistema?	È possibile monitorare `SystemErrors` per determinare se eventuali richieste hanno causato un codice HTTP 500 (errore del server). In genere, questo parametro deve essere uguale a zero. In caso contrario, è opportuno analizzare la situazione.
Come faccio a monitorare il valore di latenza per le operazioni sulle tabelle?	È possibile monitorare `SuccessfulRequestLatency` tenendo traccia della latenza media e della latenza mediana tramite metriche percentili (p50). I picchi di latenza occasionali non devono essere motivo di preoccupazione. Tuttavia, se la latenza media o p50 (mediana) è elevata, potrebbe esserci un problema di fondo da risolvere. Per ulteriori informazioni, consulta Risoluzione dei problemi di latenza in Amazon DynamoDB.

Visualizzazione dei parametri nella console CloudWatch

Le metriche vengono raggruppate prima in base al namespace del servizio e successivamente in base alle diverse combinazioni di dimensioni all'interno di ogni namespace.

Per visualizzare i parametri nella console di CloudWatch

1. Apri la console CloudWatch all'indirizzo https://console.aws.amazon.com/cloudwatch/.
2. Nel pannello di navigazione, seleziona Metriche, Tutte le metriche.
3. Seleziona lo spazio dei nomi DynamoDB. È possibile anche selezionare lo spazio dei nomi Utilizzo per visualizzare le metriche di utilizzo di DynamoDB. Per informazioni sui parametri di utilizzo, consulta parametri di utilizzo di AWS.
4. La scheda Sfoglia mostra tutte le metriche per tale dimensione nel namespace.
5. (Facoltativo) Per aggiungere il grafico delle metriche a una dashboard CloudWatch, seleziona Operazioni, Aggiungi a pannello di controllo.

Visualizzazione delle metriche nella AWS CLI

Per ottenere informazioni sulle metriche utilizzando la AWS CLI, usa il comando CloudWatch `list-metrics`. Nell'esempio seguente, vengono elencati tutti i parametri nello spazio dei nomi `AWS/DynamoDB`.

```
aws cloudwatch list-metrics --namespace "AWS/DynamoDB"
```

Per ottenere statistiche sui parametri, utilizza il comando `get-metric-statistics`. Il seguente comando ottiene statistiche `ConsumedReadCapacityUnits` per la tabella `ProductCatalog` nel periodo specifico di 24 ore, con una granularità di 5 minuti.

```
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
```

L'output di esempio viene visualizzato come segue:

```
{
    "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 "
}
```

Parametri e dimensioni di DynamoDB

Quando interagisci con DynamoDB, invia metriche e dimensioni a. CloudWatch

DynamoDB fornisce il throughput allocato utilizzato per periodi di un minuto. La scalabilità automatica si attiva quando la capacità consumata supera l'utilizzo previsto configurato per due minuti consecutivi. CloudWatch gli allarmi potrebbero avere un breve ritardo fino a qualche minuto prima di attivare il ridimensionamento automatico. Questo ritardo garantisce una valutazione metrica accurata CloudWatch . Se i picchi di throughput utilizzato distano più di un minuto, il dimensionamento automatico potrebbe non attivarsi. Analogamente, un evento di riduzione verticale può avvenire quando 15 punti dati consecutivi sono inferiori all'utilizzo di destinazione. In entrambi i casi, dopo i trigger di auto scaling, viene richiamata l'UpdateTableAPI. Sono quindi necessari alcuni minuti per aggiornare la capacità allocata per la tabella o l'indice. Durante questo periodo, tutte le richieste

che superano la capacità allocata precedente delle tabelle verranno sottoposte a limitazione (della larghezza di banda della rete).

Visualizzazione di parametri e dimensioni

CloudWatch visualizza le seguenti metriche per DynamoDB:

Parametri di DynamoDB

> **Note**
>
> Amazon CloudWatch aggrega questi parametri a intervalli di un minuto:
>
> - `ConditionalCheckFailedRequests`
> - `ConsumedReadCapacityUnits`
> - `ConsumedWriteCapacityUnits`
> - `ReadAccountLimitThrottleEvents`
> - `ReadKeyRangeThroughputThrottleEvents`
> - `ReadMaxOnDemandThroughputThrottleEvents`
> - `ReadProvisionedThroughputThrottleEvents`
> - `ReadThrottleEvents`
> - `ReturnedBytes`
> - `ReturnedItemCount`
> - `ReturnedRecordsCount`
> - `SuccessfulRequestLatency`
> - `SystemErrors`
> - `TimeToLiveDeletedItemCount`
> - `ThrottledRequests`
> - `TransactionConflict`
> - `UserErrors`
> - `WriteAccountLimitThrottleEvents`
> - `WriteKeyRangeThroughputThrottleEvents`
> - `WriteMaxOnDemandThroughputThrottleEvents`
> - `WriteProvisionedThroughputThrottleEvents`

- `WriteThrottleEvents`

Per tutti i parametri di DynamoDB, la granularità dell'aggregazione è di cinque minuti.

Non tutte le statistiche, come Average (Media) o Sum (Somma), si applicano a tutti i parametri. Tuttavia, tutti questi valori sono disponibili tramite la console Amazon DynamoDB o utilizzando la console o per tutte CloudWatch le AWS CLI metriche AWS SDKs .

Nella tabella seguente ciascun parametro presenta un elenco di statistiche valide applicabile a quel parametro.

Elenco di parametri disponibili

- AccountMaxReads
- AccountMaxTableLevelReads
- AccountMaxTableLevelWrites
- AccountMaxWrites
- AccountProvisionedReadCapacityUtilization
- AccountProvisionedWriteCapacityUtilization
- AgeOfOldestUnreplicatedRecord
- ConditionalCheckFailedRequests
- 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

Il numero massimo di unità di capacità di lettura che possono essere utilizzate da un account. Questo limite non si applica alle tabelle on demand o agli indici secondari globali.

Unità: `Count`

Statistiche valide:

- `Maximum`: il numero massimo di unità di capacità di lettura che possono essere utilizzate da un account.

AccountMaxTableLevelReads

Il numero massimo di unità di capacità di lettura che possono essere utilizzate da una tabella o un indice secondario globale di un account. Per le tabelle on demand, questo valore limita il numero massimo di unità di richiesta di lettura che possono essere utilizzate da una tabella o un indice secondario globale.

Unità: `Count`

Statistiche valide:

- `Maximum`: il numero massimo di unità di capacità di lettura che possono essere utilizzate da una tabella o un indice secondario globale di un account.

AccountMaxTableLevelWrites

Il numero massimo di unità di capacità di scrittura che possono essere utilizzate da una tabella o un indice secondario globale di un account. Per le tabelle on demand, questo valore limita il numero massimo di unità di richiesta di scrittura che possono essere utilizzate da una tabella o un indice secondario globale.

Unità: `Count`

Statistiche valide:

- `Maximum`: il numero massimo di unità di capacità di scrittura che possono essere utilizzate da una tabella o un indice secondario globale di un account.

AccountMaxWrites

Il numero massimo di unità di capacità di scrittura che possono essere utilizzate da un account. Questo limite non si applica alle tabelle on demand o agli indici secondari globali.

Unità: `Count`

Statistiche valide:

- `Maximum`: il numero massimo di unità di capacità di scrittura che possono essere utilizzate da un account.

AccountProvisionedReadCapacityUtilization

La percentuale di unità di capacità di lettura assegnata utilizzate dall'account.

Unità: `Percent`

Statistiche valide:

- `Maximum`: la percentuale massima di unità di capacità di lettura assegnata utilizzate dall'account.
- `Minimum`: la percentuale minima di unità di capacità di lettura assegnata utilizzate dall'account.
- `Average`: la percentuale minima di unità di capacità di lettura assegnata utilizzate dall'account. Il parametro viene pubblicato ad intervalli di cinque minuti. Pertanto, se si modificano rapidamente le unità di capacità di lettura assegnata, questa statistica potrebbe non corrispondere alla media reale.

AccountProvisionedWriteCapacityUtilization

La percentuale di unità di capacità di scrittura assegnata utilizzate dall'account.

Unità: `Percent`

Statistiche valide:

- `Maximum`: la percentuale massima di unità di capacità di scrittura assegnata utilizzate dall'account.
- `Minimum`: la percentuale minima di unità di capacità di scrittura assegnata utilizzate dall'account.
- `Average`: la percentuale media di unità di capacità di scrittura assegnata utilizzate dall'account. Il parametro viene pubblicato ad intervalli di cinque minuti. Pertanto, se si modificano rapidamente le unità di capacità di scrittura assegnata, questa statistica potrebbe non corrispondere alla media reale.

AgeOfOldestUnreplicatedRecord

Il tempo trascorso da quando un record ancora da replicare in Kinesis Data Streams è apparso per la prima volta nella tabella DynamoDB.

Unità: `Milliseconds`

Dimensioni: `TableName, DelegatedOperation`

Statistiche valide:

- `Maximum`.
- `Minimum`.
- `Average`.

ConditionalCheckFailedRequests

Il numero di tentativi di esecuzione di scritture condizionali non riusciti. Le operazioni `PutItem`, `UpdateItem` e `DeleteItem` consentono di fornire una condizione logica che deve essere considerata true affinché l'operazione possa procedere. Se questa condizione restituisce il valore false, `ConditionalCheckFailedRequests` viene incrementato di uno. Anche `ConditionalCheckFailedRequests` viene incrementato di uno per le istruzioni PartiQL Update and Delete in cui viene fornita una condizione logica e tale condizione restituisce il valore false.

> **Note**
>
> Una scrittura condizionale non riuscita comporterà un errore HTTP 400 (Richiesta errata). Questi eventi si riflettono nel parametro `ConditionalCheckFailedRequests` ma non nel parametro `UserErrors`.

Unità: `Count`

Dimensioni: `TableName`

Statistiche valide:

- `Minimum`
- `Maximum`
- `Average`
- `SampleCount`
- `Sum`

ConsumedChangeDataCaptureUnits

Il numero di unità di acquisizione dati di modifica consumate.

Unità: `Count`

Dimensioni: `TableName, DelegatedOperation`

Statistiche valide:

- `Minimum`
- `Maximum`
- `Average`

ConsumedReadCapacityUnits

Numero di unità di capacità di lettura utilizzate nel periodo di tempo specificato per la capacità sia in provisioning che on-demand, in modo da tenere traccia quanta velocità di trasmissione effettiva viene utilizzata. È possibile recuperare la capacità di lettura totale consumata per una tabella e tutti i relativi indici secondari globali o per un determinato indice secondario globale. Per ulteriori informazioni, consulta Modalità per la capacità di lettura e scrittura.

La dimensione `TableName` restituisce il parametro `ConsumedReadCapacityUnits` per la tabella, ma non per gli indici secondari globali. Per visualizzare il parametro `ConsumedReadCapacityUnits` per un indice secondario globale, è necessario specificare anche i parametri `TableName` e `GlobalSecondaryIndexName`.

> **Note**
>
> Ciò significa che picchi brevi e intensi nel consumo di capacità che durano solo un secondo potrebbero non essere rispecchiati con precisione nel CloudWatch grafico, portando potenzialmente a un tasso di consumo apparente inferiore per quel minuto.
> Utilizza la statistica `Sum` per calcolare la velocità effettiva consumata. Ad esempio, ottieni il valore `Sum` su un intervallo di un minuto e dividilo per il numero di secondi in un minuto (60) per calcolare il parametro medio `ConsumedReadCapacityUnits` al secondo. È possibile confrontare il valore calcolato con il valore di velocità effettiva assegnato fornito da DynamoDB.

Unità: `Count`

Dimensioni: `TableName, GlobalSecondaryIndexName`

Statistiche valide:

- `Minimum`: il numero minimo di unità di capacità di lettura utilizzate da qualsiasi richiesta individuale alla tabella o all'indice.
- `Maximum`: il numero massimo di unità di capacità di lettura utilizzate da qualsiasi richiesta individuale alla tabella o all'indice.
- `Average`: la capacità di lettura media per richiesta consumata.

> **Note**
>
> Il valore `Average` è influenzato dai periodi di inattività in cui il valore del campione sarà zero.

- `Sum`: le unità di capacità di lettura totali consumate. Questa è la statistica più utile per il parametro `ConsumedReadCapacityUnits`.
- `SampleCount`: rappresenta la frequenza con cui viene emessa la metrica. Anche per le tabelle con traffico pari a zero verrà emesso il `SampleCount`, ma i valori di esempio saranno sempre zero.

> **Note**
>
> Il valore `SampleCount` è influenzato dai periodi di inattività in cui il valore del campione sarà zero.

ConsumedWriteCapacityUnits

Numero di unità di capacità di scrittura utilizzate nel periodo di tempo specificato per la capacità sia in provisioning che on-demand, in modo da tenere traccia quanta velocità di trasmissione effettiva viene utilizzata. È possibile recuperare la capacità di scrittura totale consumata per una tabella e tutti i relativi indici secondari globali o per un determinato indice secondario globale. Per ulteriori informazioni, consulta Modalità per la capacità di lettura e scrittura.

La dimensione `TableName` restituisce il parametro `ConsumedWriteCapacityUnits` per la tabella, ma non per gli indici secondari globali. Per visualizzare il parametro

`ConsumedWriteCapacityUnits` per un indice secondario globale, è necessario specificare anche i parametri `TableName` e `GlobalSecondaryIndexName`.

> **Note**
>
> Utilizza la statistica `Sum` per calcolare la velocità effettiva consumata. Ad esempio, recupera il valore `Sum` su un intervallo di un minuto e dividilo per il numero di secondi in un minuto (60) per calcolare le `ConsumedWriteCapacityUnits` medie al secondo (riconoscendo che questa media non evidenzia picchi elevati ma brevi nell'attività di scrittura che si è verificata durante quel minuto). È possibile confrontare il valore calcolato con il valore di velocità effettiva assegnato fornito da DynamoDB.

Unità: `Count`

Dimensioni: `TableName, GlobalSecondaryIndexName`

Statistiche valide:

- `Minimum`: il numero minimo di unità di capacità di scrittura utilizzate da qualsiasi richiesta individuale alla tabella o all'indice.
- `Maximum`: il numero massimo di unità di capacità di scrittura utilizzate da qualsiasi richiesta individuale alla tabella o all'indice.
- `Average`: la capacità di scrittura media per richiesta consumata.

  > **Note**
  >
  > Il valore `Average` è influenzato dai periodi di inattività in cui il valore del campione sarà zero.

- `Sum`: le unità di capacità di scrittura totali consumate. Questa è la statistica più utile per il parametro `ConsumedWriteCapacityUnits`.
- `SampleCount`: rappresenta la frequenza con cui viene emessa la metrica. Anche per le tabelle con traffico pari a zero verrà emesso il `SampleCount`, ma i valori di esempio saranno sempre zero.

> **Note**
>
> Il valore `SampleCount` è influenzato dai periodi di inattività in cui il valore del campione sarà zero.

FailedToReplicateRecordCount

Il numero di registri che DynamoDB non è riuscito a replicare nel flusso dei dati Kinesis.

Unità: `Count`

Dimensioni: `TableName`, `DelegatedOperation`

Statistiche valide:

- `Sum`

MaxProvisionedTableReadCapacityUtilization

La percentuale di unità di capacità di lettura assegnata utilizzata dalla tabella di lettura assegnata più elevata o dall'indice secondario globale di un account.

Unità: `Percent`

Statistiche valide:

- `Maximum`: la percentuale massima di unità di capacità di lettura assegnata utilizzate dalla tabella di lettura assegnata più elevata o dall'indice secondario globale di un account.
- `Minimum`: la percentuale minima di unità di capacità di lettura assegnata utilizzate dalla tabella di lettura assegnata più elevata o dall'indice secondario globale di un account.
- `Average`: la percentuale media di unità di capacità di lettura assegnata utilizzata dalla tabella di lettura assegnata più elevata o dall'indice secondario globale dell'account. Il parametro viene pubblicato ad intervalli di cinque minuti. Pertanto, se si modificano rapidamente le unità di capacità di lettura assegnata, questa statistica potrebbe non corrispondere alla media reale.

MaxProvisionedTableWriteCapacityUtilization

La percentuale di capacità di scrittura assegnata utilizzata dalla tabella di scrittura assegnata più elevata o dall'indice secondario globale di un account.

Unità: `Percent`

Statistiche valide:

- `Maximum`: la percentuale massima di capacità di scrittura assegnata utilizzata dalla tabella di scrittura assegnata più elevata o dall'indice secondario globale di un account.
- `Minimum`: la percentuale minima di capacità di scrittura assegnata utilizzata dalla tabella di scrittura assegnata più elevata o dall'indice secondario globale di un account.
- `Average`: la percentuale media di capacità di scrittura assegnata utilizzata dalla tabella di scrittura assegnata più elevata o dall'indice secondario globale di un account. Il parametro viene pubblicato ad intervalli di cinque minuti. Pertanto, se si modificano rapidamente le unità di capacità di scrittura assegnata, questa statistica potrebbe non corrispondere alla media reale.

OnDemandMaxReadRequestUnits

Il numero di unità di richiesta di lettura on demand specificato per una tabella o un indice secondario globale.

Per visualizzare le `OnDemandMaxReadRequestUnits` per una tabella è necessario specificare il `TableName`. Per visualizzare il parametro `OnDemandMaxReadRequestUnits` per un indice secondario globale, è necessario specificare anche i parametri `TableName` e `GlobalSecondaryIndexName`.

Unità: numero

Dimensioni: `TableName`, `GlobalSecondaryIndexName`

Statistiche valide:

- `Minimum`: l’impostazione più bassa per le unità di richiesta di lettura on demand. Se si utilizza `UpdateTable` per aumentare le unità di richiesta di lettura, questa metrica mostrerà il valore più basso di `ReadRequestUnits` on demand durante questo periodo di tempo.
- `Maximum`: l’impostazione più alta per le unità di richiesta di lettura on demand. Se si utilizza `UpdateTable` per diminuire le unità di richiesta di lettura, questa metrica mostrerà il valore più alto di `ReadRequestUnits` on demand durante questo periodo di tempo.

- `Average`: le unità medie di richiesta di lettura on demand. Il parametro `OnDemandMaxReadRequestUnits` viene pubblicato a intervalli di cinque minuti. Pertanto, se si modificano rapidamente le unità di capacità di lettura on demand, questa statistica potrebbe non riflettere la media reale.

OnDemandMaxWriteRequestUnits

Il numero di unità di richiesta di scrittura on demand specificato per una tabella o un indice secondario globale.

Per visualizzare le `OnDemandMaxWriteRequestUnits` per una tabella è necessario specificare il `TableName`. Per visualizzare il parametro `OnDemandMaxWriteRequestUnits` per un indice secondario globale, è necessario specificare anche i parametri `TableName` e `GlobalSecondaryIndexName`.

Unità: `Count`

Dimensioni: `TableName`, `GlobalSecondaryIndexName`

Statistiche valide:

- `Minimum`: l'impostazione più bassa per le unità di richiesta di scrittura on demand. Se si utilizza `UpdateTable` per aumentare le unità di richiesta di scrittura, questa metrica mostrerà il valore più basso di `WriteRequestUnits` on demand durante questo periodo di tempo.
- `Maximum`: l'impostazione più alta per le unità di richiesta di scrittura on demand. Se si utilizza `UpdateTable` per diminuire le unità di richiesta di scrittura, questa metrica mostrerà il valore più alto di `WriteRequestUnits` on demand durante questo periodo di tempo.
- `Average`: le unità medie di richiesta di scrittura on demand. Il parametro `OnDemandMaxWriteRequestUnits` viene pubblicato a intervalli di cinque minuti. Pertanto, se si modificano rapidamente le unità di capacità di scrittura on demand, questa statistica potrebbe non riflettere la media reale.

OnlineIndexConsumedWriteCapacity

Il numero di unità di capacità di scrittura consumate quando si aggiunge un nuovo indice secondario globale a una tabella. Se la capacità di scrittura dell'indice è troppo bassa, l'attività di scrittura in entrata durante la fase di backfill potrebbe essere ridotta. Ciò può aumentare il tempo necessario

per creare l'indice. È consigliabile monitorare questa statistica durante la creazione dell'indice per determinare se la capacità di scrittura dell'indice è stata sottofornita.

È possibile regolare la capacità di scrittura dell'indice utilizzando l'operazione `UpdateTable` anche mentre l'indice è ancora in fase di creazione.

La metrica `ConsumedWriteCapacityUnits` per l'indice non include il throughput di scrittura utilizzato durante la creazione dell'indice.

> **Note**
>
> Questo parametro non può essere emesso se la fase di backfill del nuovo indice secondario globale viene completata rapidamente (meno di pochi minuti), cosa che può verificarsi se la tabella di base contiene pochi o nessun elemento nell'indice da sottoporre al backfill.

Unità: `Count`

Dimensioni: `TableName, GlobalSecondaryIndexName`

Statistiche valide:

- `Minimum`
- `Maximum`
- `Average`
- `SampleCount`
- `Sum`

OnlineIndexPercentageProgress

La percentuale di completamento quando un nuovo indice secondario globale viene aggiunto a una tabella. DynamoDB deve prima allocare le risorse per il nuovo indice e quindi eseguire il backfill degli attributi dalla tabella nell'indice. Per tabelle di grandi dimensioni, questo processo potrebbe richiedere molto tempo. È necessario monitorare questa statistica per visualizzare l'avanzamento relativo man mano che DynamoDB crea l'indice.

Unità: `Count`

Dimensioni: `TableName, GlobalSecondaryIndexName`

Statistiche valide:

- `Minimum`
- `Maximum`
- `Average`
- `SampleCount`
- `Sum`

OnlineIndexThrottleEvents

Il numero di eventi di limitazione di scrittura che si verificano quando si aggiunge un nuovo indice secondario globale a una tabella. Questi eventi indicano che il completamento della creazione dell'indice richiederà più tempo, poiché l'attività di scrittura in ingresso supera la velocità effettiva di scrittura prevista per l'indice.

È possibile regolare la capacità di scrittura dell'indice utilizzando l'operazione `UpdateTable` anche mentre l'indice è ancora in fase di creazione.

La metrica `WriteThrottleEvents` per l'indice non include gli eventi di limitazione (della larghezza di banda della rete) che si verificano durante la creazione dell'indice.

Unità: `Count`

Dimensioni: `TableName, GlobalSecondaryIndexName`

Statistiche valide:

- `Minimum`
- `Maximum`
- `Average`
- `SampleCount`
- `Sum`

PendingReplicationCount

Metrica per Tabelle globali versione 2017.11.29 (Legacy) (solo tabelle globali). Il numero di aggiornamenti degli elementi scritti in una tabella di replica, ma che non sono stati ancora scritti in un'altra replica nella tabella globale.

Unità: `Count`

Dimensioni: `TableName, ReceivingRegion`

Statistiche valide:

- `Average`
- `Sample Count`
- `Sum`

ProvisionedReadCapacityUnits

Il numero di unità di capacità in lettura assegnata per una tabella o un indice secondario globale. La dimensione `TableName` restituisce il parametro `ProvisionedReadCapacityUnits` per la tabella, ma non per gli indici secondari globali. Per visualizzare il parametro `ProvisionedReadCapacityUnits` per un indice secondario globale, è necessario specificare anche i parametri `TableName` e `GlobalSecondaryIndexName`.

Unità: `Count`

Dimensioni: `TableName`, `GlobalSecondaryIndexName`

Statistiche valide:

- `Minimum`: l'impostazione più bassa per la capacità di lettura assegnata. Se si utilizza `UpdateTable` per aumentare la capacità di lettura, questo parametro mostra il valore più basso di `ReadCapacityUnits` assegnato durante questo periodo di tempo.
- `Maximum`: l'impostazione più alta per la capacità di lettura assegnata. Se si utilizza `UpdateTable` per aumentare la capacità di lettura, questo parametro mostra il valore più alto di `ReadCapacityUnits` assegnato durante questo periodo di tempo.
- `Average`: la capacità media di lettura assegnata. Il parametro `ProvisionedReadCapacityUnits` viene pubblicato a intervalli di cinque minuti. Pertanto, se si modificano rapidamente le unità di capacità di lettura assegnata, questa statistica potrebbe non corrispondere alla media reale.

ProvisionedWriteCapacityUnits

Il numero di unità di capacità di scrittura assegnata per una tabella o un indice secondario globale.

La dimensione `TableName` restituisce il parametro `ProvisionedWriteCapacityUnits` per la tabella, ma non per gli indici secondari globali. Per visualizzare il parametro `ProvisionedWriteCapacityUnits` per un indice secondario globale, è necessario specificare anche i parametri `TableName` e `GlobalSecondaryIndexName`.

Unità: `Count`

Dimensioni: `TableName`, `GlobalSecondaryIndexName`

Statistiche valide:

- `Minimum`: l'impostazione più bassa per la capacità di scrittura assegnata. Se si utilizza `UpdateTable` per aumentare la capacità di scrittura, questo parametro mostra il valore più basso di `WriteCapacityUnits` assegnato durante questo periodo di tempo.
- `Maximum`: l'impostazione più alta per la capacità di scrittura assegnata. Se si utilizza `UpdateTable` per aumentare la capacità di scrittura, questo parametro mostra il valore più alto di `WriteCapacityUnits` assegnato durante questo periodo di tempo.
- `Average`: la capacità media di scrittura assegnata. Il parametro `ProvisionedWriteCapacityUnits` viene pubblicato a intervalli di cinque minuti. Pertanto, se si modificano rapidamente le unità di capacità di scrittura assegnata, questa statistica potrebbe non corrispondere alla media reale.

ReadAccountLimitThrottleEvents

Il numero di richieste di lettura sottoposte a limitazione (della larghezza di banda della rete) a causa dei limiti dell’account.

Unità: `Count`

Dimensioni: `TableName`, `GlobalSecondaryIndexName`

Statistiche valide:

- `Sum`: numero totale di eventi sottoposti a limitazione (della larghezza di banda della rete).
- `SampleCount`: numero di occorrenze di limitazione (della larghezza di banda della rete).
- `Minimum`: numero minimo di eventi sottoposti a limitazione (della larghezza di banda della rete) in un dato campione.
- `Maximum`: numero massimo di eventi sottoposti a limitazione (della larghezza di banda della rete) in un dato campione.

ReadKeyRangeThroughputThrottleEvents

Il numero di richieste di lettura sottoposte a limitazione (della larghezza di banda della rete) a causa dei limiti della partizione.

Unità: `Count`

Dimensioni: `TableName`, `GlobalSecondaryIndexName`

Statistiche valide:

- `Sum`: numero totale di eventi sottoposti a limitazione (della larghezza di banda della rete).
- `SampleCount`: numero di occorrenze di limitazione (della larghezza di banda della rete).
- `Minimum`: numero minimo di eventi sottoposti a limitazione (della larghezza di banda della rete) in un dato campione.
- `Maximum`: numero massimo di eventi sottoposti a limitazione (della larghezza di banda della rete) in un dato campione.

ReadMaxOnDemandThroughputThrottleEvents

Il numero di richieste di lettura sottoposte a limitazione (della larghezza di banda della rete) a causa del throughput massimo on demand.

Unità: `Count`

Dimensioni: `TableName`, `GlobalSecondaryIndexName`

Statistiche valide:

- `Sum`: numero totale di eventi sottoposti a limitazione (della larghezza di banda della rete).
- `SampleCount`: numero di occorrenze di limitazione (della larghezza di banda della rete).
- `Minimum`: numero minimo di eventi sottoposti a limitazione (della larghezza di banda della rete) in un dato campione.
- `Maximum`: numero massimo di eventi sottoposti a limitazione (della larghezza di banda della rete) in un dato campione.

ReadProvisionedThroughputThrottleEvents

Il numero di richieste di lettura sottoposte a limitazione (della larghezza di banda della rete) a causa dei limiti di throughput allocato.

Unità: `Count`

Dimensioni: `TableName`, `GlobalSecondaryIndexName`

Statistiche valide:

- `Sum`: numero totale di eventi sottoposti a limitazione (della larghezza di banda della rete).
- `SampleCount`: numero di occorrenze di limitazione (della larghezza di banda della rete).
- `Minimum`: numero minimo di eventi sottoposti a limitazione (della larghezza di banda della rete) in un dato campione.
- `Maximum`: numero massimo di eventi sottoposti a limitazione (della larghezza di banda della rete) in un dato campione.

ReadThrottleEvents

Richieste a DynamoDB che superano le unità di capacità di lettura assegnata per una tabella o un indice secondario globale.

Una singola richiesta può comportare più eventi. Ad esempio, un parametro `BatchGetItem` che legge 10 elementi viene elaborato come 10 eventi `GetItem`. Per ogni evento, un parametro `ReadThrottleEvents` viene incrementato di uno se quell'evento è limitato. Il parametro `ThrottledRequests` per l'intero `BatchGetItem` non viene incrementato a meno che non vengano limitati tutti e 10 gli eventi `GetItem`.

La dimensione `TableName` restituisce il parametro `ReadThrottleEvents` per la tabella, ma non per gli indici secondari globali. Per visualizzare il parametro `ReadThrottleEvents` per un indice secondario globale, è necessario specificare anche i parametri `TableName` e `GlobalSecondaryIndexName`.

Unità: `Count`

Dimensioni: `TableName`, `GlobalSecondaryIndexName`

Statistiche valide:

- `SampleCount`

- `Sum`

ReplicationLatency

Questo parametro è per le tabelle globali DynamoDB. Il tempo trascorso tra un elemento aggiornato visualizzato nel flusso DynamoDB per una tabella di replica e l'elemento visualizzato in un'altra replica nella tabella globale.

Unità: `Milliseconds`

Dimensioni: `TableName, ReceivingRegion`

Statistiche valide:

- `Average`
- `Minimum`
- `Maximum`

ReturnedBytes

Il numero di byte restituiti dalle operazioni `GetRecords`(Amazon DynamoDB Streams) durante il periodo di tempo specificato.

Unità: `Bytes`

Dimensioni: `Operation, StreamLabel, TableName`

Statistiche valide:

- `Minimum`
- `Maximum`
- `Average`
- `SampleCount`
- `Sum`

ReturnedItemCount

Il numero di elementi restituiti dalle operazioni `Query`, `Scan` o`ExecuteStatement` (select) operazioni durante il periodo di tempo specificato.

Il numero di elementi restituito non coincide necessariamente con il numero di elementi valutati. Si supponga, ad esempio, di aver richiesto un parametro `Scan` su una tabella o un indice che conteneva 100 elementi, ma di aver specificato un parametro `FilterExpression` che ha ristretto i risultati in modo che venissero restituiti solo 15 elementi. In questo caso, la risposta dal parametro `Scan` conterrà un parametro `ScanCount` di 100 e un parametro `Count` con 15 elementi restituiti.

Unità: `Count`

Dimensioni: `TableName, Operation`

Statistiche valide:

- `Minimum`
- `Maximum`
- `Average`
- `SampleCount`
- `Sum`

ReturnedRecordsCount

Il numero di record di flusso restituiti dalle operazioni `GetRecords` (Amazon DynamoDB Streams) durante il periodo di tempo specificato.

Unità: `Count`

Dimensioni: `Operation, StreamLabel, TableName`

Statistiche valide:

- `Minimum`
- `Maximum`
- `Average`
- `SampleCount`
- `Sum`

SuccessfulRequestLatency

La latenza delle richieste a DynamoDB o ai flussi Amazon DynamoDB con esito positivo durante il periodo di tempo specificato. `SuccessfulRequestLatency` può fornire due tipi diversi di informazioni:

- Il tempo trascorso per le richieste con esito positivo (`Minimum`, `Maximum`, `Sum`, `Average` o `Percentile`).
- Il numero di richieste eseguite correttamente (`SampleCount`).

`SuccessfulRequestLatency` riflette l'attività solo all'interno di DynamoDB o dei flussi Amazon DynamoDB e non tiene conto della latenza di rete o dell'attività lato client.

> **Note**
>
> Per analizzare valori percentili personalizzati (come p99.9), puoi inserire manualmente il percentile desiderato (ad esempio, p99.9) nel campo delle statistiche metriche. CloudWatch Ciò consente di valutare le distribuzioni di latenza oltre i percentili predefiniti elencati nel menu a discesa.

Unità: `Milliseconds`

Dimensioni: `TableName, Operation, StreamLabel`

Statistiche valide:

- `Minimum`
- `Maximum`
- `Sum`
- `Average`
- `Percentile`
- `SampleCount`

SystemErrors

Le richieste a DynamoDB o Amazon DynamoDB Streams che generano un codice di stato HTTP 500 durante il periodo di tempo specificato. Un codice HTTP 500 indica in genere un errore di servizio interno.

> **Note**
>
> Quando DynamoDB restituisce un errore di sistema (HTTP 500), esegue in modo automatico un AWS SDKs numero configurabile di tentativi. Se il problema si risolve durante la ripetizione di un tentativo, l'applicazione continua senza visualizzare l'errore e si potrebbe notare un aumento della latenza percepita sul lato client. Se l'errore persiste dopo tutte le ripetizioni di tentativi, si propaga al codice dell'applicazione.

Unità: `Count`

Dimensioni: `TableName, Operation`

Statistiche valide:

- `Sum`
- `SampleCount`

TimeToLiveDeletedItemCount

Il numero di elementi eliminati in base alla durata (TTL, Time to Live) nel periodo di tempo specificato. Questo parametro consente di monitorare la frequenza di eliminazioni TTL nella tabella.

Unità: `Count`

Dimensioni: TableName

Statistiche valide:

- `Sum`

ThrottledPutRecordCount

Il numero di record che sono stati limitati dal flusso dei dati Kinesis a causa della capacità insufficiente di Kinesis Data Streams.

Unità: `Count`

Dimensioni: TableName, DelegatedOperation

Statistiche valide:

- `Minimum`
- `Maximum`
- `Average`
- `SampleCount`

ThrottledRequests

Le richieste a DynamoDB che superano i limiti di velocità effettiva assegnata su una risorsa (ad esempio una tabella o un indice).

`ThrottledRequests` viene incrementato di uno se qualsiasi evento in una richiesta supera il limite di velocità effettiva assegnata. Ad esempio, se si aggiorna un elemento in una tabella con indici secondari globali, sono presenti più eventi, ovvero una scrittura nella tabella e una scrittura in ciascun indice. Se uno o più di questi eventi sono limitati, il parametro `ThrottledRequests` viene incrementato di uno.

> **Note**
>
> In una richiesta batch (`BatchGetItem` o `BatchWriteItem`), il parametro `ThrottledRequests` viene incrementato solo se ogni richiesta nel batch è limitata. Se qualsiasi singola richiesta all'interno del batch è limitata, viene incrementato uno dei seguenti parametri:
>
> - `ReadThrottleEvents`: per un evento `GetItem` limitato in `BatchGetItem`.
> - `WriteThrottleEvents`: per un evento `PutItem` o `DeleteItem` limitato in `BatchWriteItem`.

Per scoprire quale evento limita una richiesta, confrontare `ThrottledRequests` con i parametri `ReadThrottleEvents` e `WriteThrottleEvents` della tabella e i relativi indici.

> **Note**
>
> Una richiesta limitata comporterà un codice di stato HTTP 400. Tutti questi eventi si riflettono nel parametro `ThrottledRequests` ma non nel parametro `UserErrors`.

Unità: `Count`

Dimensioni: `TableName, Operation`

Statistiche valide:

- `Sum`
- `SampleCount`

TransactionConflict

Le richieste a livello di elemento vengono rifiutate a causa di conflitti di transazioni tra richieste simultanee sugli stessi elementi. Per ulteriori informazioni, consulta Gestione dei conflitti nelle transazioni in DynamoDB.

Unità: `Count`

Dimensioni: `TableName`

Statistiche valide:

- `Sum`: il numero di richieste a livello di elemento rifiutate a causa di conflitti di transazione.

> **Note**
>
> Se più richieste a livello di elemento all'interno di una chiamata ai parametri `TransactWriteItems` o `TransactGetItems` sono state rifiutate, il parametro `Sum` viene incrementato di uno per ogni richiesta di `Put`, `Update`, `Delete` oppure di `Get` a livello di elemento.

- `SampleCount`: il numero di richieste rifiutate a causa di conflitti di transazione.

> **Note**
>
> Se più richieste a livello di elemento all'interno di una chiamata ai parametri `TransactWriteItems` o `TransactGetItems` sono state rifiutate, il parametro `SampleCount` viene incrementato di uno.

- `Min`: il numero minimo di richieste a livello di elemento rifiutate all'interno di una chiamata ai parametri `TransactWriteItems`, `TransactGetItems`, `PutItem`, `UpdateItem` oppure `DeleteItem`.
- `Max`: il numero massimo di richieste a livello di elemento rifiutate all'interno di una chiamata ai parametri `TransactWriteItems`, `TransactGetItems`, `PutItem`, `UpdateItem` oppure `DeleteItem`.
- `Average`: il numero medio di richieste a livello di elemento rifiutate all'interno di una chiamata ai parametri `TransactWriteItems`, `TransactGetItems`, `PutItem`, `UpdateItem` oppure `DeleteItem`.

UserErrors

Le richieste a DynamoDB o Amazon DynamoDB Streams che generano un codice di stato HTTP 400 durante il periodo di tempo specificato. Un codice HTTP 400 in genere indica un errore sul lato client, ad esempio una combinazione non valida di parametri, un tentativo di aggiornamento di una tabella inesistente o una firma della richiesta non corretta.

Alcuni esempi di eccezioni che registreranno le metriche di log relative a `UserErrors`:

- `ResourceNotFoundException`
- `ValidationException`
- `TransactionConflict`

Tutti questi eventi si riflettono nel parametro `UserErrors`, ad eccezione di quanto segue:

- ProvisionedThroughputExceededException— Vedi la `ThrottledRequests` metrica in questa sezione.
- ConditionalCheckFailedException— Vedi la `ConditionalCheckFailedRequests` metrica in questa sezione.

`UserErrors`rappresenta l'aggregato degli errori HTTP 400 per le richieste DynamoDB o Amazon DynamoDB Streams per la regione corrente e l'account corrente. AWS AWS

Unità: `Count`

Statistiche valide:

- `Sum`
- `SampleCount`

WriteAccountLimitThrottleEvents

Il numero di richieste di scrittura sottoposte a limitazione (della larghezza di banda della rete) a causa dei limiti dell'account.

Unità: `Count`

Dimensioni: `TableName`

Statistiche valide:

- `Sum`: numero totale di eventi sottoposti a limitazione (della larghezza di banda della rete).
- `SampleCount`: numero di occorrenze di limitazione (della larghezza di banda della rete).
- `Minimum`: numero minimo di eventi sottoposti a limitazione (della larghezza di banda della rete) in un dato campione.
- `Maximum`: numero massimo di eventi sottoposti a limitazione (della larghezza di banda della rete) in un dato campione.

WriteKeyRangeThroughputThrottleEvents

Il numero di richieste di scrittura sottoposte a limitazione (della larghezza di banda della rete) a causa dei limiti di partizione.

Unità: `Count`

Dimensioni: `TableName`, `GlobalSecondaryIndexName`

Statistiche valide:

- `Sum`: numero totale di eventi sottoposti a limitazione (della larghezza di banda della rete).
- `SampleCount`: numero di occorrenze di limitazione (della larghezza di banda della rete).
- `Minimum`: numero minimo di eventi sottoposti a limitazione (della larghezza di banda della rete) in un dato campione.
- `Maximum`: numero massimo di eventi sottoposti a limitazione (della larghezza di banda della rete) in un dato campione.

WriteMaxOnDemandThroughputThrottleEvents

Il numero di richieste di scrittura sottoposte a limitazione (della larghezza di banda della rete) a causa del throughput massimo on demand.

Unità: `Count`

Dimensioni: `TableName`, `GlobalSecondaryIndexName`

Statistiche valide:

- `Sum`: numero totale di eventi sottoposti a limitazione (della larghezza di banda della rete).
- `SampleCount`: numero di occorrenze di limitazione (della larghezza di banda della rete).
- `Minimum`: numero minimo di eventi sottoposti a limitazione (della larghezza di banda della rete) in un dato campione.
- `Maximum`: numero massimo di eventi sottoposti a limitazione (della larghezza di banda della rete) in un dato campione.

WriteProvisionedThroughputThrottleEvents

Il numero di richieste di scrittura sottoposte a limitazione (della larghezza di banda della rete) a causa dei limiti di throughput allocato.

Unità: `Count`

Dimensioni: `TableName`, `GlobalSecondaryIndexName`

Statistiche valide:

- `Sum`: numero totale di eventi sottoposti a limitazione (della larghezza di banda della rete).

- `SampleCount`: numero di occorrenze di limitazione (della larghezza di banda della rete).
- `Minimum`: numero minimo di eventi sottoposti a limitazione (della larghezza di banda della rete) in un dato campione.
- `Maximum`: numero massimo di eventi sottoposti a limitazione (della larghezza di banda della rete) in un dato campione.

WriteThrottleEvents

Le richieste a DynamoDB che superano le unità di capacità di scrittura assegnata per una tabella o un indice secondario globale.

Una singola richiesta può comportare più eventi. Ad esempio, una richiesta `PutItem` su una tabella con tre indici secondari globali produce quattro eventi: la tabella di scrittura e ciascuna delle tre scritture sull'indice. Per ogni evento, il parametro `WriteThrottleEvents` viene incrementato di uno se quell'evento è limitato. Per le singole richieste `PutItem`, se uno qualsiasi degli eventi è limitato, il parametro `ThrottledRequests` viene incrementato di uno. Per `BatchWriteItem`, il parametro `ThrottledRequests` per l'intero `BatchWriteItem` non viene incrementato a meno che non siano limitati tutti i singoli eventi `PutItem` o `DeleteItem`.

La dimensione `TableName` restituisce il parametro `WriteThrottleEvents` per la tabella, ma non per gli indici secondari globali. Per visualizzare il parametro `WriteThrottleEvents` per un indice secondario globale, è necessario specificare anche i parametri `TableName` e `GlobalSecondaryIndexName`.

Unità: `Count`

Dimensioni: `TableName, GlobalSecondaryIndexName`

Statistiche valide:

- `Sum`
- `SampleCount`

Parametri di utilizzo

I parametri di utilizzo CloudWatch consentono di gestire in modo proattivo l'utilizzo visualizzando i parametri nella CloudWatch console, creando dashboard personalizzati, rilevando i cambiamenti di

attività con il rilevamento delle CloudWatch anomalie e configurando allarmi che avvisano quando l'utilizzo si avvicina a una soglia.

DynamoDB integra anche queste metriche di utilizzo con Service Quotas. Puoi utilizzarlo per gestire l'utilizzo delle quote di servizio da parte del tuo CloudWatch account. Per ulteriori informazioni, vedere Visualizzazione delle service quotas e impostazione degli avvisi

Elenco di parametri di utilizzo disponibili

- AccountProvisionedWriteCapacityUnits
- AccountProvisionedReadCapacityUnits
- TableCount

AccountProvisionedWriteCapacityUnits

La somma di unità di capacità in scrittura assegnata per tutte le tabelle e gli indici secondari globali di un account.

Unità: `Count`

Statistiche valide:

- `Minimum` - Il numero più basso di unità di capacità in scrittura assegnato durante un periodo di tempo.
- `Maximum` - Il numero più elevato di unità di capacità in scrittura assegnato durante un periodo di tempo.
- `Average` - Il numero medio di unità di capacità di scrittura assegnato durante un periodo di tempo.

Questa metrica viene pubblicata a intervalli di cinque minuti. Pertanto, se si modificano rapidamente le unità di capacità di scrittura assegnata, questa statistica potrebbe non corrispondere alla media reale.

AccountProvisionedReadCapacityUnits

La somma di unità di capacità in lettura assegnata per tutte le tabelle e gli indici secondari globali di un account.

Unità: `Count`

Statistiche valide:

- `Minimum` - Il numero più basso di unità di capacità in lettura assegnato durante un periodo di tempo.
- `Maximum` - Il numero più elevato di unità di capacità in lettura assegnato durante un periodo di tempo.
- `Average` - Il numero medio di unità di capacità di lettura assegnato durante un periodo di tempo.

Questa metrica viene pubblicata a intervalli di cinque minuti. Pertanto, se si modificano rapidamente le unità di capacità di lettura assegnata, questa statistica potrebbe non corrispondere alla media reale.

TableCount

Il numero di tabelle attive di un account.

Unità: `Count`

Statistiche valide:

- `Minimum` - Il numero più basso di tabelle durante un periodo di tempo.
- `Maximum` - Il numero più alto di tabelle durante un periodo di tempo.
- `Average` - Il numero medio di tabelle durante un periodo di tempo.

Comprendere i parametri e le dimensioni per DynamoDB

I parametri per DynamoDB sono qualificati mediante i valori dell'account, il nome della tabella, il nome dell'indice secondario globale o l'operazione. Puoi utilizzare la CloudWatch console per recuperare i dati DynamoDB lungo una qualsiasi delle dimensioni nella tabella seguente.

Elenco di dimensioni disponibili

- DelegatedOperation
- GlobalSecondaryIndexName
- Operazioni
- OperationType
- Verb
- ReceivingRegion
- StreamLabel

- TableName

DelegatedOperation

Questa dimensione limita i dati alle operazioni eseguite da DynamoDB per conto tuo. Vengono catturate le seguenti operazioni:

- Modificare l'acquisizione dei dati per Kinesis Data Streams.

GlobalSecondaryIndexName

Questa dimensione limita i dati a un indice secondario globale su una tabella. Se si specifica `GlobalSecondaryIndexName`, è necessario specificare anche `TableName`.

Operazioni

Questa dimensione limita i dati a uno dei seguenti verbi PartiQL di DynamoDB:

- `PutItem`
- `DeleteItem`
- `UpdateItem`
- `GetItem`
- `BatchGetItem`
- `Scan`
- `Query`
- `BatchWriteItem`
- `TransactWriteItems`
- `TransactGetItems`
- `ExecuteTransaction`
- `BatchExecuteStatement`
- `ExecuteStatement`

Inoltre, è possibile limitare i dati alla seguente operazione di Amazon DynamoDB Streams:

- `GetRecords`

OperationType

Questa dimensione limita i dati a uno dei seguenti tipi di operazione:

- `Read`
- `Write`

Questa dimensione viene emessa per le richieste `ExecuteTransaction` e `BatchExecuteStatement`.

Verb

Questa dimensione limita i dati a uno dei seguenti verbi PartiQL di DynamoDB:

- Inserimento: `PartiQLInsert`
- Selezionare: `PartiQLSelect`
- Aggiornare: `PartiQLUpdate`
- Eliminare: `PartiQLDelete`

Questa dimensione viene emessa per l'operazione `ExecuteStatement`.

ReceivingRegion

Questa dimensione limita i dati a una particolare regione. AWS Viene utilizzata con i parametri provenienti da tabelle di replica all'interno di una tabella globale DynamoDB.

StreamLabel

Questa dimensione limita i dati a un'etichetta di flusso specifica. Viene utilizzata con i parametri provenienti dalle operazioni `GetRecords` di Amazon DynamoDB Streams.

TableName

Questa dimensione limita i dati a una tabella specifica. Questo valore può essere qualsiasi nome di tabella nella regione corrente e nell' AWS account corrente.

Creazione di CloudWatch allarmi in DynamoDB

Un CloudWatch allarme controlla una singola metrica in un periodo di tempo specificato ed esegue una o più azioni specifiche, in base al valore della metrica rispetto a una soglia nel tempo. L'azione

è una notifica inviata a un argomento di Amazon SNS o a una politica di Amazon Auto EC2 Scaling. Puoi anche aggiungere allarmi alle dashboard in modo da monitorare e ricevere avvisi sulle tue AWS risorse e applicazioni in più regioni. Non c'è limite al numero di allarmi che puoi creare. CloudWatch gli allarmi non richiamano azioni semplicemente perché si trovano in uno stato particolare; lo stato deve essere cambiato e mantenuto per un determinato numero di periodi. Per un elenco degli allarmi DynamoDB consigliati, consulta Allarmi consigliati.

> **Note**
>
> È necessario specificare tutte le dimensioni richieste durante la creazione dell' CloudWatch allarme, poiché non CloudWatch verranno aggregate le metriche relative a una dimensione mancante. La creazione di un CloudWatch allarme con una dimensione mancante non genererà un errore durante la creazione dell'allarme.

Si supponga una tabella predisposta con cinque unità di capacità di lettura. Desideri ricevere una notifica prima di consumare l'intera capacità di lettura assegnata, quindi decidi di creare un CloudWatch allarme per ricevere una notifica quando la capacità consumata raggiunge l'80% di quella prevista per la tabella. È possibile creare allarmi nella CloudWatch console o utilizzando il. AWS CLI

Creazione di un allarme nella console CloudWatch

Per creare un allarme nella CloudWatch console

1. Accedi a Console di gestione AWS e apri la CloudWatch console all'indirizzo https://console.aws.amazon.com/cloudwatch/.
2. Nel pannello di navigazione, scegli Alarms (Allarmi), All alarms (Tutti gli allarmi).
3. Scegli Crea allarme.
4. Trova la riga con la tabella che desideri monitorare e **`ConsumeReadCapacityUnits`** nella colonna Nome della metrica. Seleziona la casella di spunta accanto a questa riga e scegli Seleziona metrica.
5. In Specifica parametri e condizioni, per Statistica seleziona Somma. Per Periodo, seleziona 1 minuto.
6. In Conditions (Condizioni), specifica quanto segue:

    a. For Threshold type (Tipo di soglia), scegli Static (Statica).

   b. Per Ogni volta che **`ConsumedReadCapacityUnits`** è, seleziona Maggiore/Uguale e inserisci come soglia 240.
7. Scegli Next (Successivo).
8. In Notifica, seleziona **`In alarm`** e seleziona un argomento SNS per notificare quando l'allarme è in stato `ALARM`.
9. Al termine, scegli Apply (Applica).
10. Inserisci un nome e una descrizione per l'allarme, quindi scegli Next (Successivo).
11. In Preview and create (Visualizza anteprima e crea), conferma che le informazioni e le condizioni sono quelle desiderate, quindi scegli Create alarm (Crea allarme).

Creazione di un allarme in 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
```

Testa l'allarme.

```
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
```

Altri AWS CLI esempi

La procedura seguente descrive come ricevere una notifica in caso di richieste che superano le quote di throughput allocato per una tabella.

1. Crea un argomento Amazon SNS. `arn:aws:sns:us-east-1:123456789012:requests-exceeding-throughput` Per ulteriori informazioni, consulta la pagina relativa alla configurazione di Amazon Simple Notification Service.
2. Crea l'allarme.

```
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. Testa l'allarme.

```
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
```

La procedura seguente descrive come ricevere una notifica in caso di errori di sistema.

1. Crea un argomento Amazon SNS. `arn:aws:sns:us-east-1:123456789012:notify-on-system-errors` Per ulteriori informazioni, consulta la pagina relativa alla configurazione di Amazon Simple Notification Service.
2. Crea l'allarme.

```
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. Testa l'allarme.

```
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
```

Registrazione delle operazioni di DynamoDB con AWS CloudTrail

DynamoDB è integrato con AWS CloudTrail, un servizio che offre un record delle operazioni eseguite da un utente, un ruolo o un servizio AWS in DynamoDB. CloudTrail acquisisce tutte le chiamate API per DynamoDB come eventi. Le chiamate acquisite includono chiamate dalla console DynamoDB e chiamate di codice alle operazioni API DynamoDB, utilizzando sia PartiQL che l'API classica. Se si crea un percorso, è possibile abilitare la distribuzione continua di eventi CloudTrail in un bucket Amazon S3, inclusi gli eventi per DynamoDB. Se non si configura un percorso, è comunque possibile visualizzare gli eventi più recenti nella console di CloudTrail in Cronologia eventi. Grazie alle informazioni raccolte da CloudTrail è possibile determinare la richiesta effettuata a DynamoDB, l'indirizzo IP da cui è partita la richiesta, l'autore della richiesta, il momento in cui è stata effettuata e altri dettagli.

Per un monitoraggio e una segnalazione affidabili, gli eventi CloudTrail possono essere aggregati anche con Amazon CloudWatch Logs. Per migliorare l'analisi dell'attività del servizio DynamoDB e identificare le modifiche nelle attività per un account AWS, è possibile eseguire query sui log AWS CloudTrail utilizzando Amazon Athena. Ad esempio, è possibile utilizzare le query per identificare le tendenze e isolare con maggiore precisione le attività in base ad attributi specifici, ad esempio l'indirizzo IP di origine o un utente.

Per ulteriori informazioni su CloudTrail, incluso come configurarlo e abilitarlo, consulta la Guida per l'utente di AWS CloudTrail.

Argomenti

- Informazioni su DynamoDB in CloudTrail
- Informazioni sulle voci dei file di registro di DynamoDB

Informazioni su DynamoDB in CloudTrail

CloudTrail è abilitato sull'account AWS al momento della sua creazione. Quando si verifica un'attività supportata in DynamoDB, questa viene registrata in un evento CloudTrail insieme ad altri eventi di servizio AWS in Cronologia eventi. È possibile visualizzare, cercare e scaricare gli eventi recenti nell'account AWS. Per ulteriori informazioni, consulta Utilizzo della cronologia degli eventi di CloudTrail.

Per una registrazione continua degli eventi nell'account AWS, compresi gli eventi per DynamoDB, creare un percorso. Un percorso consente a CloudTrail di distribuire i file di log in un bucket Amazon S3. Per impostazione predefinita, quando si crea un trail nella console, il trail sarà valido in tutte le Regioni AWS. Il trail registra gli eventi di tutte le Regioni nella partizione AWS e distribuisce i file di log nel bucket Amazon S3 specificato. Inoltre, è possibile configurare altri servizi AWS per analizzare con maggiore dettaglio e usare i dati evento raccolti nei log CloudTrail. Per ulteriori informazioni, consulta gli argomenti seguenti:

- Panoramica della creazione di un percorso
- Servizi e integrazioni CloudTrail supportati
- Configurazione delle notifiche Amazon SNS per CloudTrail
- Ricezione di file di log CloudTrail da più Regioni e Ricezione di file di log CloudTrail da più account

Eventi del piano di controllo in CloudTrail

Le seguenti operazioni API vengono registrate come eventi nei file di 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

Eventi del piano dati DynamoDB in CloudTrail

Per abilitare la registrazione delle seguenti operazioni API nei file CloudTrail, è necessario abilitare la registrazione dell'attività API del piano dati in CloudTrail. Per ulteriori informazioni, consulta Registrazione di eventi di dati per i percorsi.

Gli eventi del piano dati possono essere filtrati in base al tipo di risorsa per un controllo granulare sulle chiamate API DynamoDB che vuoi registrare e pagare in modo selettivo in CloudTrail. Ad

esempio, specificando `AWS::DynamoDB::Stream` come tipo di risorsa, puoi registrare solo le chiamate alle API dei flussi DynamoDB. Per le tabelle con flussi abilitati, il campo delle risorse nell'evento del piano dati contiene sia `AWS::DynamoDB::Stream` che `AWS::DynamoDB::Table`. Se si specifica `AWS::DynamoDB::Table` come tipo di risorsa, per impostazione predefinita verranno registrati sia gli eventi della tabella DynamoDB che quelli dei flussi DynamoDB. Puoi aggiungere un filtro aggiuntivo per escludere gli eventi di streaming, se non desideri che vengano registrati. Per ulteriori informazioni, consulta DataResource nella documentazione di riferimento dell'API di AWS CloudTrail.

Amazon DynamoDB

- BatchExecuteStatement
- BatchGetItem
- BatchWriteItem
- DeleteItem
- ExecuteStatement
- ExecuteTransaction
- GetItem
- PutItem
- Query
- Scan
- TransactGetItems
- TransactWriteItems
- UpdateItem

> **Note**
>
> Le operazioni del piano dati relative alla durata (TTL, Time to Live) di DynamoDB non vengono registrate da CloudTrail.

DynamoDB Streams

- GetRecords
- GetShardIterator

Informazioni sulle voci dei file di registro di DynamoDB

Un trail è una configurazione che consente la distribuzione di eventi come i file di log in un bucket Amazon S3 specificato. I file di log di CloudTrail possono contenere una o più voci di log. Un evento rappresenta una singola richiesta da un'origine e include informazioni sull'operazione richiesta, data e ora dell'operazione, parametri della richiesta e così via.

Ogni evento o voce di log contiene informazioni sull'utente che ha generato la richiesta. Le informazioni di identità consentono di determinare quanto segue:

- Se la richiesta è stata effettuata con credenziali utente o root.
- Se la richiesta è stata effettuata con le credenziali di sicurezza temporanee per un ruolo o un utente federato.
- Se la richiesta è stata effettuata da un altro servizio AWS.

> **Note**
>
> I valori degli attributi non chiave verranno revisionati nei log CloudTrail delle operazioni che utilizzano l'API PartiQL e non saranno visualizzati nei log delle azioni che utilizzano l'API classica.

Per ulteriori informazioni, consulta Elemento CloudTrail userIdentity.

Negli esempi seguenti vengono illustrati i log CloudTrail di questi tipi di eventi:

Amazon DynamoDB

- `UpdateTable`
- `DeleteTable`
- `CreateCluster`
- `PutItem` (riuscito)
- `UpdateItem` (non riuscito)
- `TransactWriteItems` (riuscito)
- `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 (riuscito)

```
{
    "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` (non riuscito)

```
{
    "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` (riuscito)

```
{
    "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"
        }
    ]
```

```
}
```

Analisi dell'accesso ai dati utilizzando CloudWatch Contributor Insights per DynamoDB

Amazon CloudWatch Contributor Insights per Amazon DynamoDB è uno strumento diagnostico per identificare a colpo d'occhio le chiavi con accesso più frequente e quelle con limitazioni nella tabella o nell'indice. Questo strumento utilizza le informazioni dei contributori. CloudWatch

Abilitando CloudWatch Contributor Insights for DynamoDB su una tabella o su un indice secondario globale, è possibile visualizzare gli elementi più accessibili e con limitazioni di tali risorse.

> **Note**
>
> CloudWatch si applicano costi per Contributor Insights for DynamoDB. Per ulteriori informazioni sui prezzi, consulta la pagina CloudWatchdei prezzi di Amazon.

Argomenti

- CloudWatch approfondimenti sui contributori per DynamoDB: come funziona
- Nozioni di base su Contributor Insights di CloudWatch per DynamoDB
- Utilizzo di IAM con informazioni sui CloudWatch contributori per DynamoDB

CloudWatch approfondimenti sui contributori per DynamoDB: come funziona

Amazon DynamoDB si integra CloudWatch con Contributor Insights per fornire informazioni sugli elementi più accessibili e limitati in una tabella o in un indice secondario globale. DynamoDB ti fornisce queste informazioni CloudWatch tramite regole, report e grafici dei dati dei report di Contributor Insights.

CloudWatch Contributor Insights for DynamoDB è progettato per non avere alcun impatto sulle prestazioni della tabella DynamoDB.

Per ulteriori informazioni su CloudWatch Contributor Insights, consulta Usare Contributor Insights per analizzare dati ad alta cardinalità nella Amazon User Guide. CloudWatch

Le seguenti sezioni descrivono i concetti e il comportamento principali di CloudWatch Contributor Insights for DynamoDB.

Argomenti

- CloudWatch modalità di analisi dei contributori per DynamoDB
- CloudWatch approfondimenti sui contributori per le regole di DynamoDB
- Comprendere le informazioni sui CloudWatch contributori per i grafici di DynamoDB
- Interazioni con altre caratteristiche di DynamoDB
- CloudWatch approfondimenti sui collaboratori per la fatturazione in DynamoDB

CloudWatch modalità di analisi dei contributori per DynamoDB

CloudWatch Contributor Insights for DynamoDB offre due modalità distinte per soddisfare diverse esigenze di monitoraggio.

Modalità Chiavi con limitazione (della larghezza di banda della rete)

Questa modalità si concentra esclusivamente sulle richieste sottoposte a limitazione (della larghezza di banda della rete) ed elabora gli eventi solo quando si verifica la limitazione. Fornisce informazioni sui problemi di prestazioni senza il sovraccarico di tracciare tutti i modelli di accesso. In questa modalità, DynamoDB tiene traccia solo di:

- Elementi con maggiore limitazione (della larghezza di banda della rete): elementi che subiscono il maggior numero di eventi di limitazione

Questa modalità è ideale quando:

- L'obiettivo principale è identificare e risolvere i problemi di limitazione (della larghezza di banda della rete)
- Si desidera mantenere Contributor Insights sempre attivo per il rilevamento delle limitazioni (della larghezza di banda della rete) in tempo reale
- Si desidera un approccio ottimizzato in termini di costi per monitorare i problemi di limitazione (della larghezza di banda della rete)

> **Note**
>
> La modalità Chiavi con limitazione (della larghezza di banda della rete) elabora gli eventi solo quando si verifica una limitazione di questo tipo, il che la rende conveniente per il monitoraggio continuo. Questo approccio mirato consente di lasciare la funzionalità abilitata in modo permanente con un impatto minimo sui costi, pur garantendo una visibilità immediata sui problemi di limitazione (della larghezza di banda della rete) non appena si verificano.

Se la tabella non presenta alcuna limitazione (della larghezza di banda della rete), non verrà visualizzato alcun dato nei grafici di Contributor Insights, il che indica prestazioni soddisfacenti. Quando viene rilevata una limitazione (della larghezza di banda della rete), i grafici generati aiutano a identificare modelli di accesso specifici che causano problemi di prestazioni. Queste informazioni possono aiutare a implementare strategie per affrontare modelli di accesso non uniformi.

Per strategie di monitoraggio complete, puoi integrare queste informazioni sulla limitazione con altre CloudWatch metriche per creare dashboard unificate che correlano gli eventi di limitazione con le prestazioni complessive delle tabelle.

Modalità Chiavi con accessi e limitazione (della larghezza di banda della rete)

Questa modalità offre un monitoraggio completo sia degli elementi a cui si accede che di quelli sottoposti a limitazione (della larghezza di banda della rete). In questa modalità, DynamoDB tiene traccia di:

- Elementi con maggiori accessi: elementi che consumano la maggior parte della capacità di lettura e scrittura
- Elementi con maggiore limitazione (della larghezza di banda della rete): elementi che subiscono il maggior numero di eventi di limitazione

Questa modalità è ideale quando si ha bisogno di una visibilità completa sui modelli di accesso alla tabella e si desidera comprendere sia gli elementi ad alto traffico che i problemi di limitazione (della larghezza di banda della rete).

Passaggio da una modalità all’altra

È possibile passare da una modalità all'altra in qualsiasi momento utilizzando la console AWS CLI DynamoDB, oppure. APIs Durante il passaggio tra modalità:

- CloudWatch Le regole esistenti vengono aggiornate in base alla nuova modalità
- CloudWatch Le regole relative ai tasti limitati rimangono invariate e mantengono i dati storici continui per le metriche di limitazione:
  - Passando dalla modalità Chiavi con limitazione (della larghezza di banda della rete) alla modalità Chiavi con accessi e limitazione (della larghezza di banda della rete), le regole esistenti relative alle chiavi limitate vengono mantenute e vengono create nuove regole relative alle chiavi con accessi
  - Passando dalla modalità Chiavi con accessi e limitazione (della larghezza di banda della rete) alla modalità Chiavi con limitazione (della larghezza di banda della rete), vengono mantenute solo le regole esistenti relative alle chiavi limitate e vengono rimosse le regole relative alle chiavi con accessi
- La fatturazione viene modificata immediatamente per riflettere l'elaborazione degli eventi della nuova modalità

CloudWatch approfondimenti sui contributori per le regole di DynamoDB

Quando abiliti CloudWatch Contributor Insights for DynamoDB su una tabella o un indice secondario globale, DynamoDB crea regole per tuo conto in base alla modalità selezionata.

> **Note**
>
> Abilitando Contributor Insights su una tabella DynamoDB si è soggetti ai limiti delle regole di Contributor Insights. Per ulteriori informazioni, consulta CloudWatch service quotas.

Regole per la modalità Chiavi con accessi e limitazione (della larghezza di banda della rete)

In modalità Chiavi con accessi e limitazione (della larghezza di banda della rete), DynamoDB crea le seguenti regole:

- Elementi più accessibili (chiavi di partizione): identifica le chiavi di partizione degli elementi più accessibili nella tabella o nell'indice secondario globale.

  CloudWatch formato del nome della regola: `DynamoDBContributorInsights-PKC-[resource_name]-[creationtimestamp]`
- La maggior parte delle chiavi con limitazioni (chiavi di partizione): identifica le chiavi di partizione degli elementi con più limitazioni nella tabella o nell'indice secondario globale.

CloudWatch formato del nome della regola: `DynamoDBContributorInsights-PKT-[resource_name]-[creationtimestamp]`

Se la tabella o l'indice secondario globale ha chiavi di ordinamento, DynamoDB crea anche le seguenti regole specifiche per le chiavi di ordinamento:

- Chiavi con maggiori accessi (chiavi di partizione e di ordinamento): identifica le chiavi di partizione e di ordinamento degli elementi con maggiori accessi nella tabella o nell'indice secondario globale.

  CloudWatch formato del nome della regola: `DynamoDBContributorInsights-SKC-[resource_name]-[creationtimestamp]`
- Chiavi con maggiore throttling (chiavi di partizione e di ordinamento): identifica le chiavi di partizione e di ordinamento degli elementi con maggiore throttling nella tabella o nell'indice secondario globale.

  CloudWatch formato del nome della regola: `DynamoDBContributorInsights-SKT-[resource_name]-[creationtimestamp]`

Regole per la modalità Chiavi con limitazione (della larghezza di banda della rete)

In modalità Chiavi con limitazione (della larghezza di banda della rete), DynamoDB crea solo le regole relative alla limitazione (della larghezza di banda della rete):

- Chiavi con maggiore throttling (chiave di partizione): identifica le chiavi di partizione degli elementi con maggiore throttling nella tabella o nell'indice secondario globale.

  CloudWatch formato del nome della regola: `DynamoDBContributorInsights-PKT-[resource_name]-[creationtimestamp]`

Se la tabella o l'indice secondario globale ha chiavi di ordinamento, DynamoDB crea anche:

- Chiavi con maggiore throttling (chiavi di partizione e di ordinamento): identifica le chiavi di partizione e di ordinamento degli elementi con maggiore throttling nella tabella o nell'indice secondario globale.

  CloudWatch formato del nome della regola: `DynamoDBContributorInsights-SKT-[resource_name]-[creationtimestamp]`

Questo approccio mirato riduce il numero di regole attive e diminuisce il volume di eventi elaborati per diagnosticare meglio gli eventi di limitazione (della larghezza di banda della rete).

> **Note**
>
> - Quando si utilizza la CloudWatch console o APIs si visualizza CloudWatch Contributor Insights for DynamoDB, vengono visualizzate solo le regole corrispondenti alla modalità selezionata.
> - Non puoi utilizzare la CloudWatch console o modificare o APIs eliminare direttamente le regole create da CloudWatch Contributor Insights per DynamoDB. La disabilitazione di CloudWatch Contributor Insights for DynamoDB su una tabella o su un indice secondario globale elimina automaticamente le regole create per quella tabella o indice secondario globale.
> - Quando si utilizza l'GetInsightRuleReportoperazione con le regole di CloudWatch Contributor Insights create da DynamoDB, si ottengono `MaxContributorValue` solo `Maximum` statistiche utili. Le altre statistiche dell'elenco non restituiscono valori significativi.
> - CloudWatch Contributor Insights for DynamoDB ha un limite di 25 collaboratori. La richiesta per più di 25 contributori restituisce un errore.

È possibile creare CloudWatch allarmi utilizzando le regole di CloudWatch Contributor Insights for DynamoDB. In tal modo è possibile ricevere una notifica quando un articolo supera o soddisfa una soglia specifica per `ConsumedThroughputUnits` o `ThrottleCount`. Per ulteriori informazioni, consulta Impostazione di un allarme sui dati delle metriche Contributor Insights.

Comprendere le informazioni sui CloudWatch contributori per i grafici di DynamoDB

CloudWatch Contributor Insights for DynamoDB visualizza diversi tipi di grafici sia su DynamoDB CloudWatch che sulle console a seconda della modalità selezionata.

Disponibilità di grafici per modalità

I grafici visualizzati dipendono dalla modalità Contributor Insights selezionata.

- La modalità Chiavi con accessi e limitazione (della larghezza di banda della rete) mostra sia i grafici Elementi con accesso più frequente che quelli degli Elementi con maggiore limitazione (della larghezza di banda della rete)

- La modalità Chiavi con limitazione (della larghezza di banda della rete) mostra solo i grafici degli Elementi con maggiore limitazione (della larghezza di banda della rete)

Elementi con maggiori accessi

Questo grafico è disponibile solo in modalità Chiavi con accessi e limitazione (della larghezza di banda della rete. Utilizza questo grafico per identificare gli elementi con maggiori accessi nella tabella o nell'indice secondario globale. Il grafico visualizza `ConsumedThroughputUnits` sull'asse y e l'orario sull'asse x. Ognuna delle chiavi N principali viene visualizzata nel proprio colore con una legenda visualizzata sotto l'asse x.

DynamoDB misura la frequenza di accesso alla chiave mediante `ConsumedThroughputUnits`, che misura il traffico di lettura e scrittura combinato. `ConsumedThroughputUnits` viene definito come segue:

- Assegnato: (3 unità di capacità di scrittura utilizzate) + unità di capacità di lettura utilizzate
- On demand: (3 unità di richiesta di scrittura) + unità di richiesta di lettura

Nella console DynamoDB ciascun punto dati nel grafico rappresenta il massimo di `ConsumedThroughputUnits` nell'arco di un minuto. Ad esempio, un valore di grafico di 180.000 `ConsumedThroughputUnits` indica che l'elemento ha avuto accessi continui a un throughput massimo per elemento di 1.000 unità di richiesta di scrittura o di 3.000 unità di richiesta di lettura per un intervallo di 60 secondi nell'arco del periodo di un minuto (60 x 3.000 secondi). In altre parole, i valori del grafico rappresentano il traffico massimo per ogni periodo di un minuto. È possibile modificare la granularità temporale della `ConsumedThroughputUnits` metrica (ad esempio, per visualizzare metriche da 5 minuti anziché da 1 minuto) sulla console. CloudWatch

Se si vedono diverse linee strettamente raggruppate senza evidenti valori anomali, indica che il carico di lavoro è relativamente bilanciato tra gli elementi in una determinata finestra temporale. Se nel grafico si vedono punti isolati invece di linee collegate, indica un elemento che ha avuto maggiori accessi solo per un breve periodo.

Se la tabella o l'indice secondario globale contiene chiavi di ordinamento, DynamoDB crea due grafici: uno per le chiavi di partizione più accessibili e uno per le coppie partizione+chiavi di ordinamento più accessibili. È possibile visualizzare il traffico a livello delle chiavi di partizione nel grafico contenente solo le chiavi di partizione. Puoi vedere il traffico a livello di elemento nei grafici partition + sort keys.

Elementi con maggiore throttling

Questo grafico è disponibile in entrambe le modalità. Utilizza questo grafico per identificare gli elementi con maggiore throttling nella tabella o nell'indice secondario globale. Il grafico visualizza `ThrottleCount` sull'asse y e l'orario sull'asse x. Ognuna delle chiavi N principali viene visualizzata nel proprio colore con una legenda visualizzata sotto l'asse x.

DynamoDB misura la frequenza di throttling utilizzando `ThrottleCount`, che è il calcolo degli errori `ProvisionedThroughputExceededException`, `ThrottlingException` e `RequestLimitExceeded`.

La limitazione di scrittura causata da capacità di scrittura insufficiente per un indice secondario globale non viene misurata. È possibile utilizzare il grafico Elementi con maggiori accessi dell'indice secondario globale per identificare i modelli di accesso sbilanciati che possono causare la limitazione della scrittura. Per ulteriori informazioni, consulta Considerazioni sul throughput assegnato per indici secondari globali.

Nella console DynamoDB, ogni punto dati nel grafico rappresenta il conteggio degli eventi di limitazione in un periodo di un minuto.

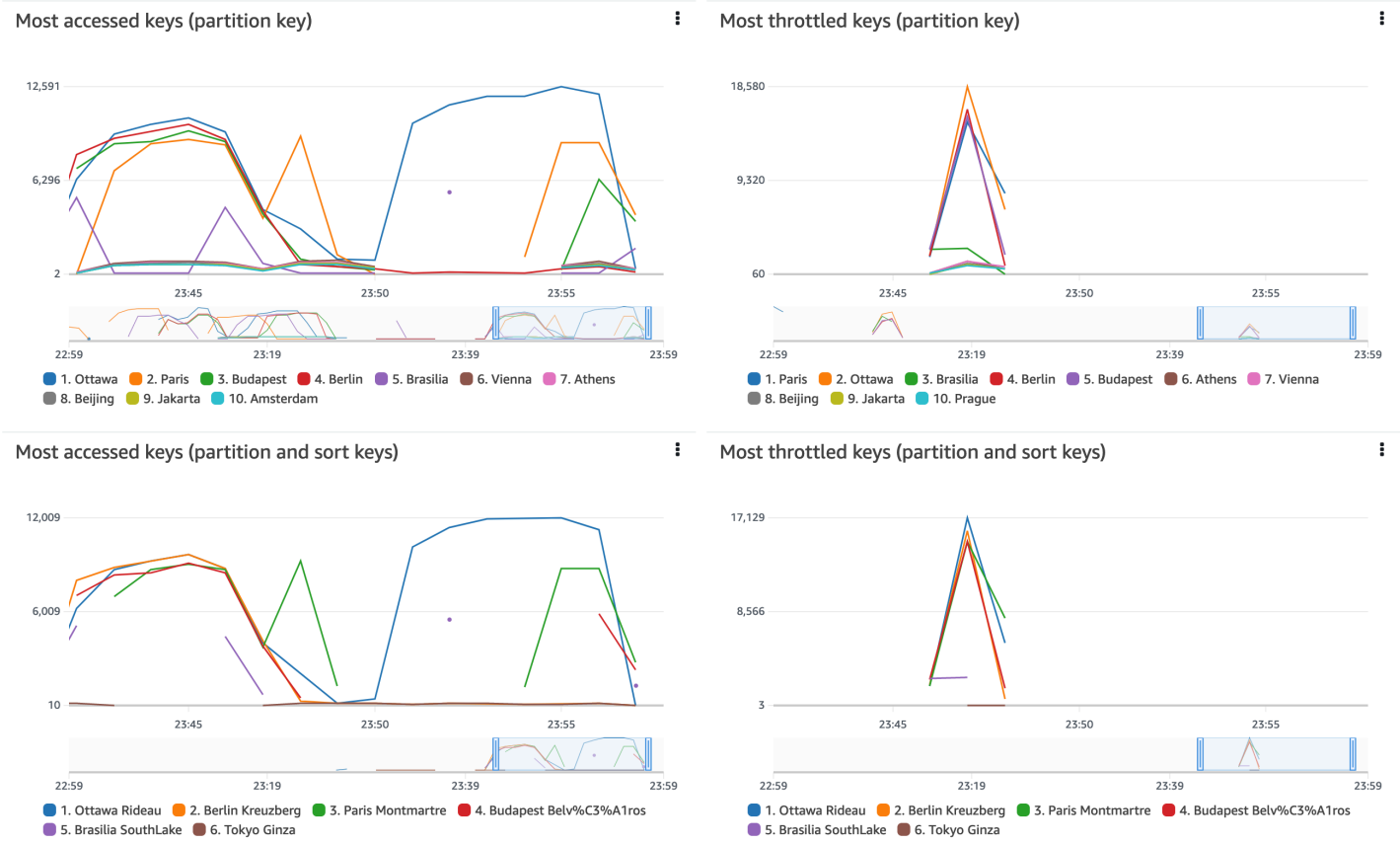
Se nel grafico non sono presenti dati, indica che le richieste non vengono sottoposte a throttling. Se nel grafico si vedono punti isolati invece di linee collegate, indica un elemento che ha avuto maggiore throttling solo per un breve periodo.

Se la tabella o l'indice secondario globale contiene chiavi di ordinamento, DynamoDB crea due grafici: uno per la maggior parte delle chiavi di partizione con limitazione e uno per la maggior parte delle coppie di chiavi di ordinamento e partizione limitate. Puoi vedere il numero di acceleratori a livello di chiavi di partizione nel grafico relativo alle sole chiavi di partizione e il conteggio dell'acceleratore a livello di elemento nei grafici partition + sort keys.

> **Note**
>
> In modalità Chiavi con limitazione (della larghezza di banda della rete, questo è l'unico tipo di grafico visibile. L'assenza di dati in questi grafici indica buone prestazioni della tabella senza che si verifichi limitazione (della larghezza di banda della rete).

Esempi di report

L'esempio seguente mostra i report generati per una tabella con chiavi di partizione e chiavi di ordinamento in modalità tasti con accesso limitato. In modalità Chiavi con limitazione (della larghezza di banda della rete), viene visualizzata solo la parte di questo report relativa alla limitazione.



Interazioni con altre caratteristiche di DynamoDB

Le seguenti sezioni descrivono come CloudWatch Contributor Insights for DynamoDB si comporta e interagisce con diverse altre funzionalità di DynamoDB. Vale a dire che questi comandi sono validi per entrambe le modalità, salvo diversamente specificato.

Tabelle globali

CloudWatch Contributor Insights for DynamoDB monitora le repliche globali delle tabelle come tabelle distinte. I grafici di Contributor Insights per una replica in una AWS regione potrebbero non mostrare gli stessi modelli di un'altra regione. Ciò perché i dati di scrittura vengono replicati in tutte le repliche in una tabella globale, ma ciascuna replica può fornire il traffico di lettura collegato a una regione.

Ogni replica può essere configurata con una modalità Contributor Insights diversa in modo indipendente. Ad esempio, è possibile utilizzare la modalità Chiavi con accessi e limitazione (della larghezza di banda della rete nell'area principale per un monitoraggio completo, mentre la modalità Chiavi con limitazione (della larghezza di banda della rete nelle aree secondarie per mantenere la visibilità sui problemi di prestazioni.

DynamoDB Accelerator (DAX)

CloudWatch Contributor Insights for DynamoDB non mostra le risposte della cache DAX. Mostra solo le risposte all'accesso a una tabella o a un indice secondario globale.

> **Note**
>
> CloudWatch DynamoDB Contributor Insights non supporta le richieste PartiQL.

Crittografia dei dati a riposo

CloudWatch Contributor Insights for DynamoDB non influisce sul funzionamento della crittografia in DynamoDB. I dati chiave primari pubblicati in CloudWatch sono crittografati con. Chiave di proprietà di AWS Tuttavia, DynamoDB supporta anche la chiave gestita dal cliente e una chiave Chiave gestita da AWS gestita dal cliente.

CloudWatch Contributor Insights for DynamoDB visualizza le chiavi di partizione e le chiavi di ordinamento (se applicabile) degli elementi a cui si accede di frequente e che sono limitati. Sebbene CloudWatch Contributor Insights funzioni con tabelle DynamoDB crittografate, è importante notare che utilizza il proprio contesto di crittografia di proprietà di Amazon, che è separato dalla crittografia configurata della tabella.

Se la chiave primaria della tabella DynamoDB contiene informazioni sensibili e le politiche di sicurezza dell'organizzazione richiedono il pieno controllo sui processi di crittografia, l' CloudWatch attivazione di Contributor Insights potrebbe non essere adatta.

Controllo granulare degli accessi

CloudWatch Contributor Insights for DynamoDB non funziona diversamente per le tabelle con controllo granulare degli accessi (FGAC). In altre parole, qualsiasi utente che dispone delle autorizzazioni appropriate può visualizzare le chiavi primarie protette da FGAC nei grafici di CloudWatch Contributor Insights. CloudWatch

Se la chiave primaria della tabella contiene dati protetti da FGAC su cui non vuoi pubblicare CloudWatch, non dovresti abilitare CloudWatch Contributor Insights for DynamoDB per quella tabella.

Controllo accessi

Puoi controllare l'accesso a CloudWatch Contributor Insights for DynamoDB AWS Identity and Access Management utilizzando (IAM) limitando le autorizzazioni del piano di controllo di DynamoDB e le autorizzazioni del piano dati. CloudWatch Per ulteriori informazioni, consulta Using IAM with CloudWatch Contributor Insights for DynamoDB.

CloudWatch approfondimenti sui collaboratori per la fatturazione in DynamoDB

I costi per CloudWatch Contributor Insights for DynamoDB vengono visualizzati nella sezione CloudWatchdella fattura mensile. Questi addebiti vengono calcolati in base al numero di eventi DynamoDB elaborati e alla modalità selezionata.

Fatturazione in base alla modalità

Le due modalità di Contributor Insights hanno caratteristiche di fatturazione diverse.

- Fatturazione della modalità Chiavi con accessi e limitazione (della larghezza di banda della rete): in questa modalità, ogni elemento scritto o letto tramite un'operazione sul piano dati rappresenta un evento, indipendentemente dal fatto che la richiesta abbia esito positivo o sia sottoposta a limitazione (della larghezza di banda della rete). Se una tabella o un indice secondario globale include chiavi di ordinamento, ogni elemento letto o scritto rappresenta due eventi. Questo perché DynamoDB sta identificando i principali contributori da serie temporali separate: uno solo per le chiavi di partizione e uno per le coppie di chiavi di partizione e ordinamento.
- Fatturazione della modalità Chiavi con limitazione (della larghezza di banda della rete): in questa modalità, solo le richieste con limitazione (della larghezza di banda della rete) generano eventi fatturabili. Gli eventi vengono generati solo quando le richieste generano errori `ProvisionedThroughputExceededException`, `ThrottlingException` o `RequestLimitExceeded`. Se una tabella o un indice secondario globale include chiavi di ordinamento, ogni elemento limitato rappresenta due eventi (tracciamento delle chiavi di partizione e tracciamento delle chiavi di ordinamento della partizione e delle chiavi di ordinamento).

Esempi di fatturazione

Ad esempio, si supponga che l'applicazione esegua le seguenti operazioni DynamoDB: una `GetItem`, una `PutItem` e una `BatchWriteItem` che inserisce cinque elementi. Si supponga inoltre

che l'operazione `PutItem` venga sottoposta a limitazione (della larghezza di banda della rete), ma che tutte le altre operazioni abbiano esito positivo.

- Modalità Chiavi con accessi e limitazione (della larghezza di banda della rete)
  - Se la tabella o l'indice secondario globale ha solo una chiave di partizione, vengono generati 7 eventi (1 per`GetItem`, 1 per e 5 per il`PutItem`). `BatchWriteItem`
  - Se la tabella o l'indice secondario globale dispone di chiavi di partizione e chiavi di ordinamento, vengono generati 14 eventi (2 per`GetItem`, 2 per e 10 per). `PutItem` `BatchWriteItem`
- Modalità Chiavi con limitazione (della larghezza di banda della rete)
  - Se la tabella o l'indice secondario globale ha solo una chiave di partizione, viene generato 1 evento (solo per quelli limitati`PutItem`).
  - Se la tabella o l'indice secondario globale ha chiavi di partizione e chiavi di ordinamento, vengono generati 2 eventi (2 per quelli limitati). `PutItem`

  Le operazioni `GetItem` e `BatchWriteItem` con esito positivo non generano alcun evento in modalità Chiavi con limitazione (della larghezza di banda della rete).

Fattori di fatturazione comuni

Un'operazione `Query` produrrà sempre 1 evento, indipendentemente dalla modalità o dal numero di elementi restituiti.

A differenza di altre funzionalità di DynamoDB CloudWatch , la fatturazione di Contributor Insights for DynamoDB non varia in base a quanto segue:

- La modalità di capacità (con provisioning rispetto a on-demand)
- Indica se si eseguono richieste di lettura o scrittura
- La dimensione (KB) degli elementi letti o scritti

Nozioni di base su Contributor Insights di CloudWatch per DynamoDB

In questa sezione viene descritto come abilitare e utilizzare Contributor Insights di Amazon CloudWatch in modalità differenti per soddisfare le esigenze di monitoraggio attraverso la console Amazon DynamoDB o AWS Command Line Interface (AWS CLI).

Nei seguenti esempi, viene utilizzata la tabella DynamoDB che viene definita nel tutorial Nozioni di base su DynamoDB.

Argomenti

- Scelta della modalità di Contributor Insights
- Utilizzo di Contributor Insights (console)
- Utilizzo di Contributor Insights (AWS CLI)

Scelta della modalità di Contributor Insights

Prima di abilitare Contributor Insights, è necessario comprendere le due modalità disponibili. Esamina il confronto tra le modalità per selezionare l'opzione più adatta ai requisiti specifici.

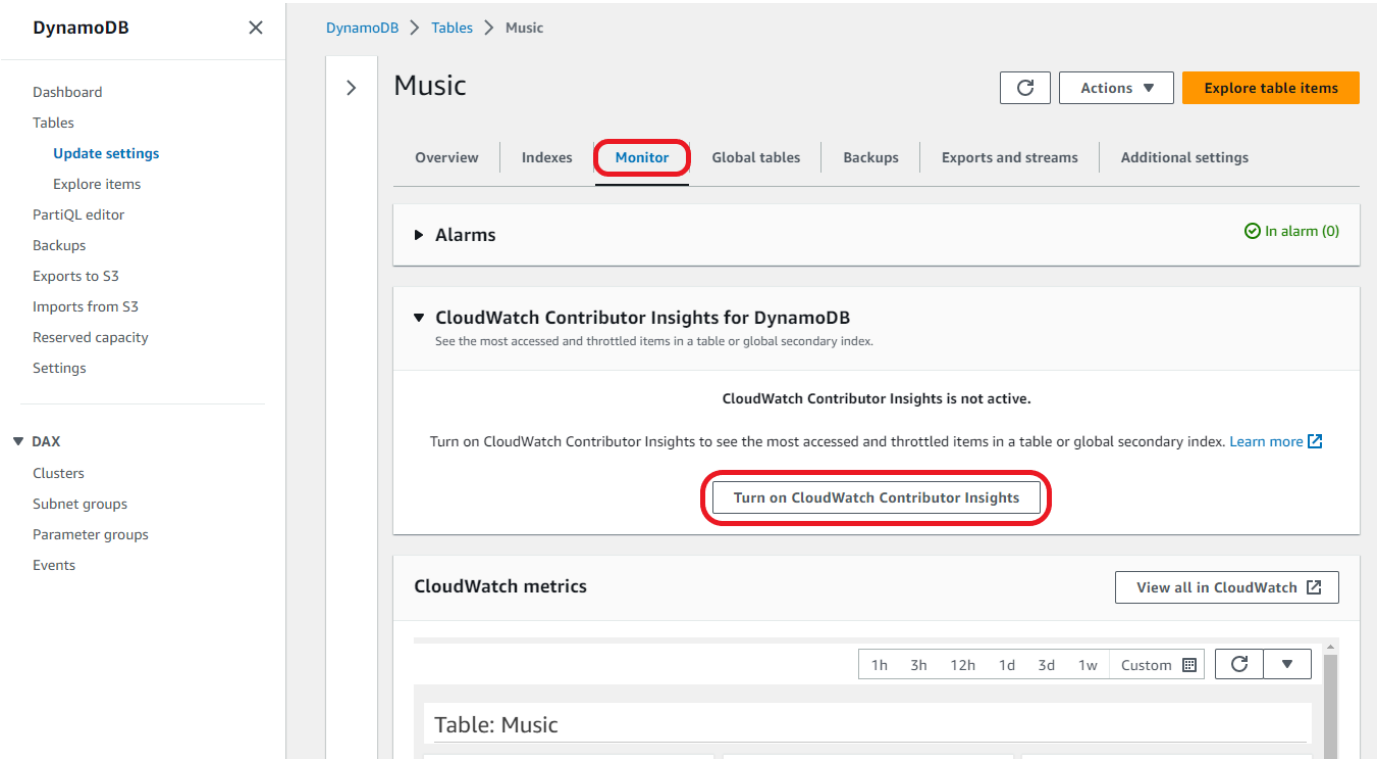
Aspetto	Modalità Chiavi con accessi e limitazione (della larghezza di banda della rete)	Modalità Chiavi con limitazione (della larghezza di banda della rete)
Monitoraggi	Tutte le richieste (eseguite con successo e limitate)	Solo richieste con limitazione (della larghezza di banda della rete)
Grafici	Elementi con maggiore accesso + Elementi con maggiore limitazione (della larghezza di banda della rete)	Solo elementi con maggiore limitazione (della larghezza di banda della rete)
Ideale per	Analisi e ottimizzazione mirate	Monitoraggio della limitazione (della larghezza di banda della rete)
Da usare quando	È necessaria una visibilità completa sui modelli di accesso. Si stanno eseguendo analisi o debug a breve termine.	L'obiettivo principale è identificare e risolvere i problemi di limitazione (della larghezza di banda della rete) Si desidera mantenere Contributor Insights sempre attivo per avvisi sulla limitazione (della larghezza di banda della rete) in tempo reale.

Utilizzo di Contributor Insights (console)

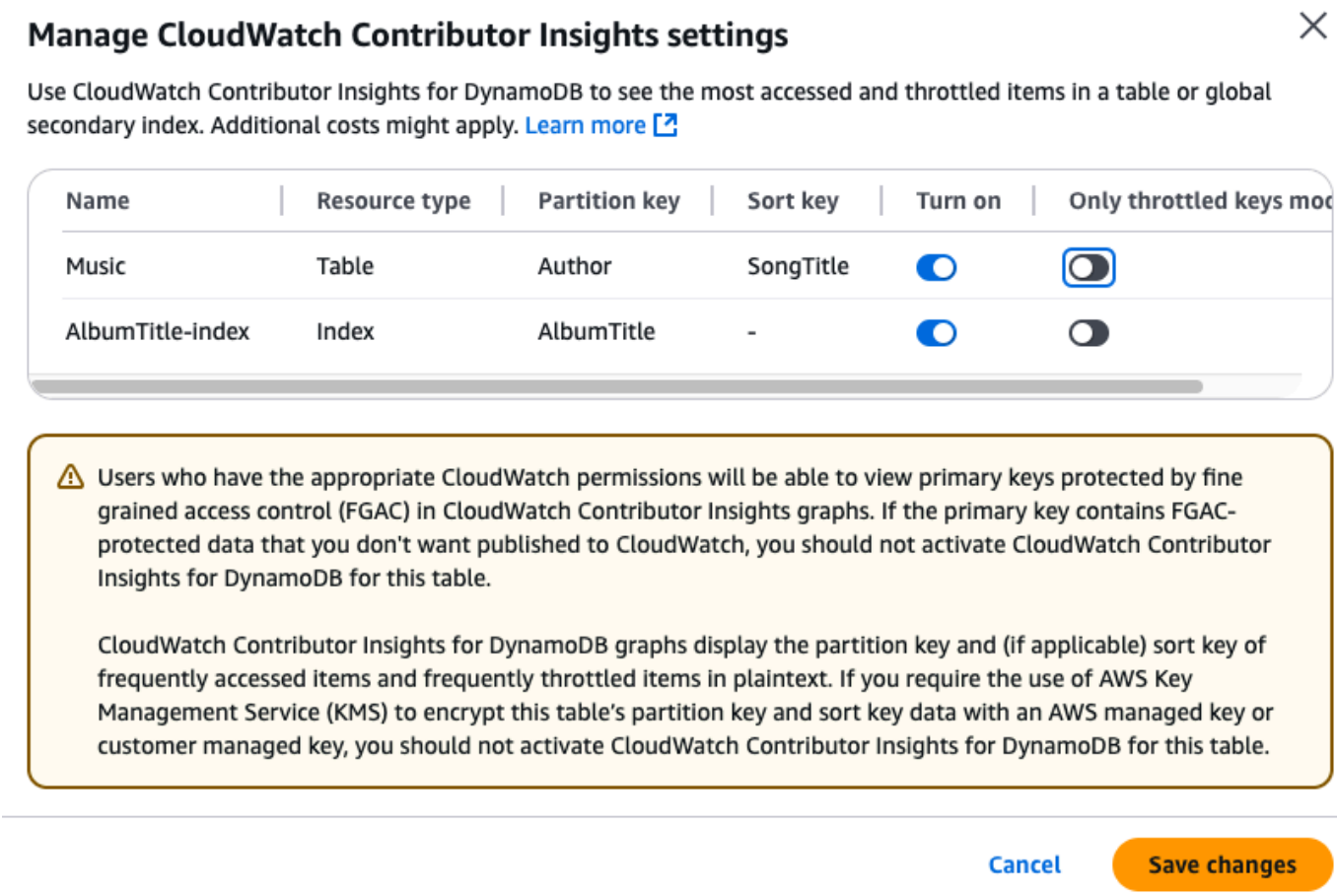
La console offre un modo intuitivo per abilitare Contributor Insights e selezionare la modalità appropriata per le esigenze di monitoraggio.

Per utilizzare Contributor Insights nella console

1. Accedi alla console Console di gestione AWS e apri la console DynamoDB all'indirizzo https://console.aws.amazon.com/elasticache/.
2. Nel riquadro di navigazione sul lato sinistro della console scegli Tables (Tabelle).
3. Seleziona la tabella `Music`.
4. Selezionare la scheda Monitor (Monitora).
5. Scegli Attiva Contributor Insights di CloudWatch.



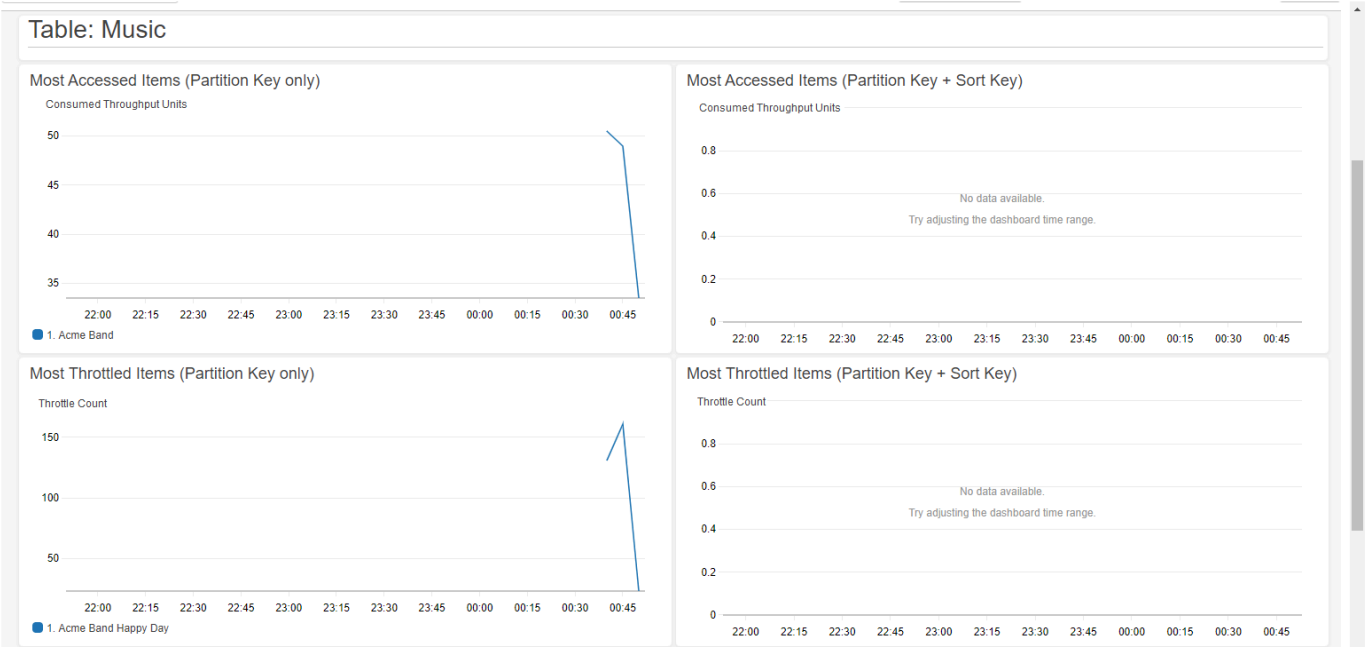
6. Nella finestra di dialogo delle impostazioni Gestire CloudWatch Contributor Insights, abilita l'opzione Attiva sia per la tabella di base `Music` che per l'indice secondario globale `AlbumTitle-index`.
7. Lascia disattivata la Modalità solo chiavi limitate per entrambi, quindi seleziona Salva modifiche.



Ciò abilita la modalità predefinita Chiavi con accessi e limitazione (della larghezza di banda della rete) sia per la tabella che per il GSI, che fornisce il monitoraggio sia degli elementi con accessi che di quelli sottoposti a limitazione (della larghezza di banda della rete). L'attivazione dell'opzione Modalità solo chiavi limitate abilita la modalità Chiavi con limitazione (della larghezza di banda della rete).

Se l'operazione non riesce, consulta DescribeContributorInsights FailureException nella Documentazione di riferimento delle API di Amazon DynamoDB per le cause possibili.

8. I grafici di Contributor Insights di CloudWatch sono ora visibili nella scheda Monitoraggio della tabella `Music`. Poiché è stata attivata la modalità Chiavi con accessi e limitazione (della larghezza di banda della rete), vengono visualizzati sia i grafici degli elementi con accessi che di quelli sottoposti a limitazione (della larghezza di banda della rete).



Passaggio da una modalità all'altra

È possibile passare da una modalità all'altra in qualsiasi momento senza disabilitare Contributor Insights.

Come cambiare modalità di Contributor Insights

1. Nella scheda Monitoraggio della tabella, seleziona Gestire CloudWatch Contributor Insights.
2. Nella finestra di dialogo delle impostazioni Gestisci Contributor Insights, per ogni tabella di base o GSI:

   - Attiva o disattiva la Modalità solo chiavi limitate per abilitare la modalità Chiavi con limitazione (della larghezza di banda della rete) o torna alla modalità predefinita Chiavi con accessi e limitazione (della larghezza di banda della rete).
   - Disattiva l'opzione Attiva per disabilitare CloudWatch Contributor Insights per una tabella o un GSI.
3. Scegli Save changes (Salva modifiche).

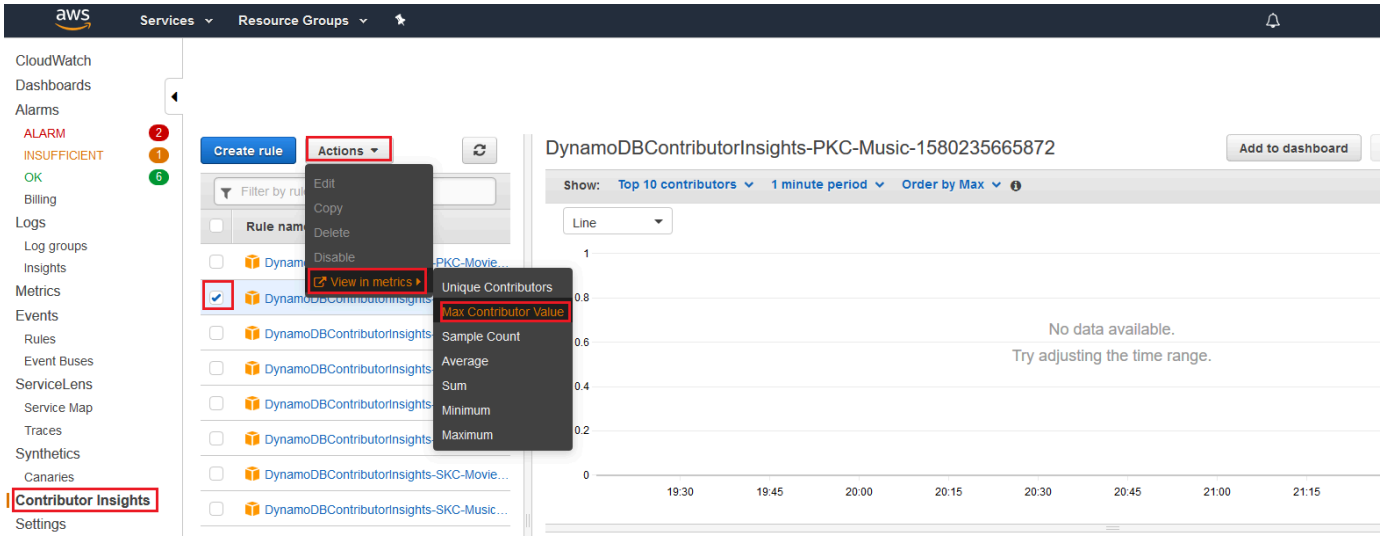
   Una volta completata l'operazione, i grafici rifletteranno la nuova modalità.

Creazione di allarmi CloudWatch

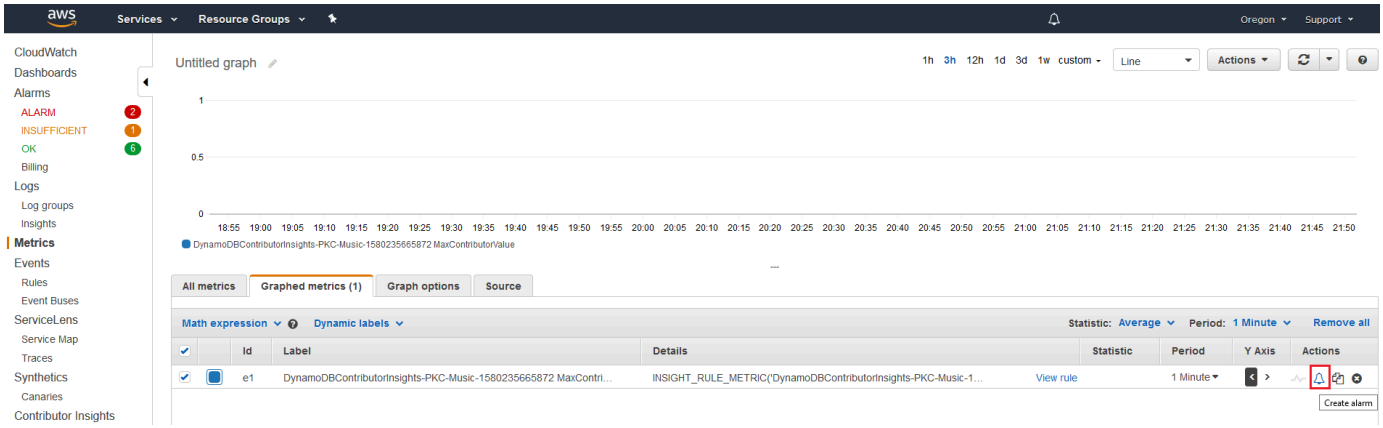
Attieniti alla seguente procedura per creare un allarme CloudWatch e ricevere una notifica quando una chiave di partizione utilizza più di 50.000 ConsumedThroughputUnits o viene sottoposta a limitazione (della larghezza di banda della rete).

1. Accedi alla Console di gestione AWS e apri la console CloudWatch all'indirizzo https://console.aws.amazon.com/cloudwatch/.
2. Nel riquadro di navigazione sul lato sinistro della console scegli Contributor Insights.
3. Seleziona la regola appropriata in base alla modalità e a ciò che desideri monitorare:
   - Per il monitoraggio degli elementi con accessi (solo modalità Chiavi con accessi e limitazione (della larghezza di banda della rete)): seleziona DynamoDBContributorInsights-PKC-Music
   - Per il monitoraggio degli elementi sottoposti a limitazione (della larghezza di banda della rete) (entrambe le modalità): seleziona DynamoDBContributorInsights-PKT-Music
4. Seleziona il menu a tendina Operazioni.
5. Scegli Visualizza nei parametri.
6. Scegli Valore massimo Contributor.

> **Note**
>
> Solo `Max Contributor Value` e `Maximum` restituiscono statistiche utili. Le altre statistiche dell'elenco non restituiscono valori significativi.

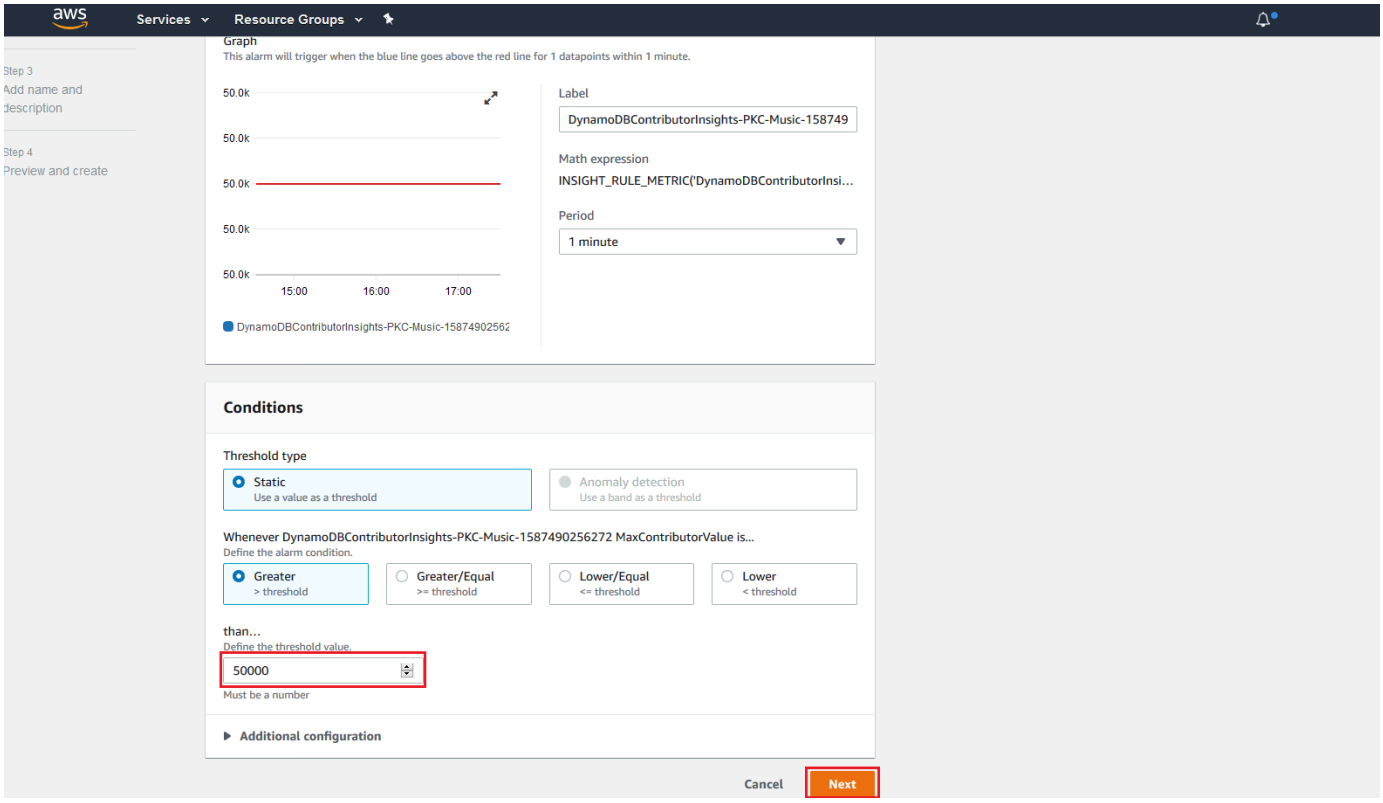


7. Nella colonna Operazioni scegli Crea allarme.



8. Immetti un valore di soglia appropriato e seleziona Successivo:

   - Per gli elementi con accessi (regole PKC): immetti 50000 per `ConsumedThroughputUnits`
   - Per gli elementi sottoposti a limitazione (della larghezza di banda della rete) (regole PKT): immetti 1 per `ThrottleCount` per ricevere un avviso in caso di limitazione



9. Consulta Utilizzo degli allarmi Amazon CloudWatch per i dettagli su come configurare la notifica per l'allarme.

Utilizzo di Contributor Insights (AWS CLI)

AWS CLI fornisce l'accesso programmatico a Contributor Insights con supporto completo per entrambe le modalità. È possibile specificare la modalità quando si abilita Contributor Insights o cambiare modalità in un secondo momento.

Operazioni di base con la modalità predefinita

Come utilizzare Contributor Insights con le impostazioni predefinite

1. Abilita Contributor Insights di CloudWatch per DynamoDB sulla tabella di base `Music` con la modalità Chiavi con accessi e limitazione (della larghezza di banda della rete). Poiché `ACCESSED_AND_THROTTLED_KEYS` è la modalità predefinita, è possibile omettere il parametro `--contributor-insights-mode=ACCESSED_AND_THROTTLED_KEYS`.

   ```
   aws dynamodb update-contributor-insights \
                       --table-name Music \
                       --contributor-insights-action=ENABLE
   ```

2. Abilitare Contributor Insights per DynamoDB sull'indice secondario globale `AlbumTitle-index`.

   ```
   aws dynamodb update-contributor-insights \
                       --table-name Music \
                       --index-name AlbumTitle-index \
                       --contributor-insights-action=ENABLE
   ```

3. Ottenere lo stato e le regole per la tabella `Music` e tutti i relativi indici.

   ```
   aws dynamodb describe-contributor-insights
                       --table-name Music
   ```

   La risposta includerà il campo `ContributorInsightsMode` che mostra `ACCESSED_AND_THROTTLED_KEYS`.

4. Ottieni lo stato per la tabella `Music` e tutti i relativi indici.

   ```
   aws dynamodb list-contributor-insights --table-name Music
   ```

Abilitazione della modalità Chiavi con limitazione (della larghezza di banda della rete)

Come abilitare Contributor Insights in modalità Chiavi con limitazione (della larghezza di banda della rete

1. Abilita Contributor Insights di CloudWatch per DynamoDB sulla tabella di base `Music` con la modalità Chiavi con limitazione (della larghezza di banda della rete.

```
aws dynamodb update-contributor-insights \
    --table-name Music \
    --contributor-insights-action=ENABLE \
    --contributor-insights-mode=THROTTLED_KEYS
```

2. Abilita Contributor Insights nella modalità Chiavi con limitazione (della larghezza di banda della rete per l'indice secondario globale `AlbumTitle-index`.

```
aws dynamodb update-contributor-insights \
    --table-name Music \
    --index-name AlbumTitle-index \
    --contributor-insights-action=ENABLE \
    --contributor-insights-mode=THROTTLED_KEYS
```

3. Verifica la modalità descrivendo la configurazione di Contributor Insights.

```
aws dynamodb describe-contributor-insights --table-name Music
```

   La risposta mostrerà `ContributorInsightsMode` come `THROTTLED_KEYS` e un numero minore di regole rispetto alla modalità predefinita.

Passaggio da una modalità all'altra

Come cambiare modalità di Contributor Insights

1. Passa dalla modalità Chiavi con limitazione (della larghezza di banda della rete alla modalità Chiavi con accessi e limitazione (della larghezza di banda della rete).

```
aws dynamodb update-contributor-insights \
    --table-name Music \
    --contributor-insights-action=ENABLE \
    --contributor-insights-mode=ACCESSED_AND_THROTTLED_KEYS
```

2. Passa dalla modalità Chiavi con accessi e limitazione (della larghezza di banda della rete) alla modalità Chiavi con limitazione (della larghezza di banda della rete).

```
aws dynamodb update-contributor-insights \
    --table-name Music \
    --contributor-insights-action=ENABLE \
    --contributor-insights-mode=THROTTLED_KEYS
```

3. Controlla lo stato durante la transizione.

```
aws dynamodb describe-contributor-insights --table-name Music
```

Durante il cambio di modalità, `ContributorInsightsStatus` verrà visualizzato come `ENABLING`. Una volta completata l'operazione, verrà visualizzato come `ENABLED` nella nuova modalità.

Gestione di Contributor Insights

Come gestire le impostazioni di Contributor Insights

1. Disabilitare CloudWatch Contributor Insights per DynamoDB sull'indice secondario globale `AlbumTitle-index`.

```
aws dynamodb update-contributor-insights \
                    --table-name Music --index-name AlbumTitle-index \
                    --contributor-insights-action=DISABLE
```

2. Elenca tutte le configurazioni di Contributor Insights nell'account.

```
aws dynamodb list-contributor-insights
```

Questo mostra tutte le tabelle e gli indici con Contributor Insights abilitato, insieme alle relative modalità.

3. Ottieni informazioni dettagliate su una configurazione specifica.

```
aws dynamodb describe-contributor-insights \
                    --table-name Music \
                    --index-name AlbumTitle-index
```

Risposte di esempio

Ecco alcuni esempi di risposte che mostrano le differenze tra le modalità:

Risposta in modalità Chiavi con accessi e limitazione (della larghezza di banda della rete)

```
{
    "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"
}
```

Risposta in modalità Chiavi con limitazione (della larghezza di banda della rete)

```
{
    "TableName": "Music",
    "ContributorInsightsRuleList": [
        "DynamoDBContributorInsights-PKT-Music-1234567890123",
        "DynamoDBContributorInsights-SKT-Music-1234567890123"
    ],
    "ContributorInsightsStatus": "ENABLED",
    "ContributorInsightsMode": "THROTTLED_KEYS",
    "LastUpdateDateTime": "2024-01-15T10:35:00.000Z"
}
```

Si noti che la modalità Chiavi con limitazione (della larghezza di banda della rete) ha un numero minore di regole (solo PKT e SKT), il che corrisponde a un monitoraggio più mirato.

Utilizzo di IAM con informazioni sui CloudWatch contributori per DynamoDB

La prima volta che abiliti Amazon CloudWatch Contributor Insights per Amazon DynamoDB, DynamoDB crea AWS Identity and Access Management automaticamente un ruolo collegato al servizio (IAM) per te. Questo ruolo consente a DynamoDB di CloudWatch gestire le regole di Contributor Insights per tuo conto. `AWSServiceRoleForDynamoDBCloudWatchContributorInsights` Non eliminare questo ruolo

collegato al servizio. Se lo si elimina, tutte le regole gestite non verranno più pulite quando si elimina la tabella o l'indice secondario globale.

Per ulteriori informazioni sui ruoli collegati al servizio, consulta Utilizzo dei ruoli collegati al servizio nella Guida per l'utente IAM.

Sono richieste le seguenti autorizzazioni:

- Per abilitare o disabilitare CloudWatch Contributor Insights for DynamoDB, è necessario `dynamodb:UpdateContributorInsights` disporre dell'autorizzazione per la tabella o l'indice.
- Per visualizzare i grafici di CloudWatch Contributor Insights for DynamoDB, è necessario disporre dell'autorizzazione. `cloudwatch:GetInsightRuleReport`
- Per descrivere CloudWatch Contributor Insights for DynamoDB per una determinata tabella o indice DynamoDB, è necessario disporre dell'autorizzazione. `dynamodb:DescribeContributorInsights`
- Per elencare gli stati di CloudWatch Contributor Insights for DynamoDB per ogni tabella e indice secondario globale, è necessario disporre dell'autorizzazione. `dynamodb:ListContributorInsights`

Esempio: abilitare o disabilitare le informazioni sui CloudWatch contributori per DynamoDB

La seguente policy IAM concede le autorizzazioni per abilitare o disabilitare 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/*"
        }
    ]
}
```

Per le tabelle crittografate tramite chiave KMS, l'utente deve disporre delle autorizzazioni kms:Decrypt per aggiornare 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"
            ]
        }
    ]
}
```

Esempio: recupera il rapporto sulle regole di Contributor Insights CloudWatch

La seguente policy IAM concede le autorizzazioni per recuperare CloudWatch il rapporto sulle regole di Contributor Insights.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:GetInsightRuleReport"
            ],
            "Resource": "arn:aws:cloudwatch:*:*:insight-rule/
DynamoDBContributorInsights*"
        }
    ]
}
```

Esempio: applica in modo selettivo le informazioni sui CloudWatch collaboratori per le autorizzazioni di DynamoDB in base alla risorsa

La seguente policy IAM concede le autorizzazioni per le operazioni `ListContributorInsights` e `DescribeContributorInsights` e nega l'operazione `UpdateContributorInsights` per un indice secondario globale specifico.

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"
        }
    ]
}
```

Utilizzo di ruoli collegati ai servizi per CloudWatch Contributor Insights for DynamoDB

CloudWatch Contributor Insights for DynamoDB AWS Identity and Access Management utilizza ruoli collegati ai servizi (IAM). Un ruolo collegato ai servizi è un tipo unico di ruolo IAM collegato direttamente a CloudWatch Contributor Insights for DynamoDB. I ruoli collegati ai servizi sono predefiniti da CloudWatch Contributor Insights per DynamoDB e includono tutte le autorizzazioni richieste dal servizio per chiamare altri servizi per tuo conto. AWS

Un ruolo collegato al servizio semplifica la configurazione di CloudWatch Contributor Insights per DynamoDB perché non è necessario aggiungere manualmente le autorizzazioni necessarie. CloudWatch Contributor Insights for DynamoDB definisce le autorizzazioni dei suoi ruoli collegati ai servizi e, se non diversamente definito, solo CloudWatch Contributor Insights for DynamoDB può assumerne i ruoli. Le autorizzazioni definite includono la policy di attendibilità e la policy delle autorizzazioni che non può essere collegata a nessun'altra entità IAM.

Per informazioni sugli altri servizi che supportano i ruoli collegati ai servizi, consulta Servizi AWS che funzionano con IAM e cerca i servizi che riportano Sì nella colonna Ruolo associato ai servizi. Scegli Sì in corrispondenza di un link per visualizzare la documentazione relativa al ruolo collegato ai servizi per tale servizio.

Autorizzazioni di ruolo collegate ai servizi per CloudWatch Contributor Insights for DynamoDB

CloudWatch Contributor Insights for DynamoDB utilizza il ruolo collegato al servizio denominato. AWSServiceRoleForDynamoDBCloudWatchContributorInsights Lo scopo del ruolo collegato ai servizi è consentire ad Amazon DynamoDB di gestire CloudWatch le regole di Amazon Contributor Insights create per le tabelle DynamoDB e gli indici secondari globali, per tuo conto.

Ai fini dell'assunzione del ruolo, il ruolo collegato ai servizi `AWSServiceRoleForDynamoDBCloudWatchContributorInsights` considera attendibili i seguenti servizi:

- `contributorinsights.dynamodb.amazonaws.com`

La politica di autorizzazione dei ruoli consente a CloudWatch Contributor Insights for DynamoDB di completare le seguenti azioni sulle risorse specificate:

- Operazione: `Create and manage Insight Rules` su `DynamoDBContributorInsights`

Per consentire a un'entità IAM (come un utente, un gruppo o un ruolo) di creare, modificare o eliminare un ruolo collegato ai servizi devi configurare le relative autorizzazioni. Per ulteriori informazioni, consulta Autorizzazioni del ruolo collegato ai servizi nella Guida per l'utente di IAM.

Creazione di un ruolo collegato ai servizi per CloudWatch Contributor Insights for DynamoDB

Non hai bisogno di creare manualmente un ruolo collegato ai servizi. Quando abiliti Contributor Insights nella Console di gestione AWS, o nell' AWS API AWS CLI, CloudWatch Contributor Insights for DynamoDB crea automaticamente il ruolo collegato al servizio.

Se elimini questo ruolo collegato ai servizi, è possibile ricrearlo seguendo lo stesso processo utilizzato per ricreare il ruolo nell'account. Quando abiliti Contributor Insights, CloudWatch Contributor Insights for DynamoDB crea nuovamente il ruolo collegato al servizio per te.

Modifica di un ruolo collegato ai servizi per CloudWatch Contributor Insights for DynamoDB

CloudWatch Contributor Insights for DynamoDB non consente di modificare il ruolo collegato al servizio. `AWSServiceRoleForDynamoDBCloudWatchContributorInsights` Dopo aver creato un ruolo collegato al servizio, non potrai modificarne il nome perché varie entità potrebbero farvi riferimento. È possibile tuttavia modificarne la descrizione utilizzando IAM. Per ulteriori informazioni, consulta Modifica di un ruolo collegato ai servizi nella Guida per l'utente di IAM.

Eliminazione di un ruolo collegato al servizio per CloudWatch Contributor Insights for DynamoDB

Non è necessario eliminare manualmente il ruolo `AWSServiceRoleForDynamoDBCloudWatchContributorInsights`. Quando disabiliti Contributor Insights nella Console di gestione AWS, o nell' AWS API AWS CLI, CloudWatch Contributor Insights for DynamoDB pulisce le risorse.

Puoi anche utilizzare la console IAM, AWS CLI o l' AWS API per eliminare manualmente il ruolo collegato al servizio. Per farlo, sarà necessario prima eseguire manualmente la pulizia delle risorse associate al ruolo collegato ai servizi e poi eliminarlo manualmente.

> **Note**
>
> Se il servizio CloudWatch Contributor Insights for DynamoDB utilizza il ruolo quando si tenta di eliminare le risorse, l'eliminazione potrebbe non riuscire. In questo caso, attendi alcuni minuti e quindi ripeti l'operazione.

Per eliminare manualmente il ruolo collegato ai servizi mediante IAM

Utilizza la console IAM AWS CLI, il o l' AWS API per eliminare il ruolo collegato al `AWSServiceRoleForDynamoDBCloudWatchContributorInsights` servizio. Per ulteriori informazioni, consultare Eliminazione del ruolo collegato ai servizi nella Guida per l'utente di IAM.

Best practice per la progettazione e l'architettura con DynamoDB

Utilizza questa sezione per individuare rapidamente le raccomandazioni per massimizzare le prestazioni e ridurre al minimo i costi di throughput durante l'utilizzo di DynamoDB.

Argomenti

- Progettazione NoSQL per DynamoDB
- Utilizzo di DynamoDB Well-Architected Lens per ottimizzare il carico di lavoro di DynamoDB
- Best practice per la progettazione delle chiavi di partizione e relativo uso in DynamoDB
- Best practice per l'uso delle chiavi di ordinamento per organizzare i dati in DynamoDB
- Best practice per l'uso di indici secondari in DynamoDB.
- Best practice per l'archiviazione di elementi e attributi di grandi dimensioni in DynamoDB
- Best practice per la gestione dei dati di serie temporali in DynamoDB
- Le migliori pratiche per la gestione delle many-to-many relazioni nelle tabelle DynamoDB
- Best practice per eseguire query e scansioni dei dati in DynamoDB
- Best practice per la progettazione di tabelle DynamoDB
- Utilizzo delle tabelle globali DynamoDB
- Best practice per la gestione del piano di controllo (control plane) in DynamoDB
- Best practice per l'utilizzo di operazioni sui dati in blocco in DynamoDB
- Best practice per l'implementazione del controllo delle versioni in DynamoDB
- Le migliori pratiche per comprendere i report di AWS fatturazione e utilizzo in DynamoDB
- Migrazione di una tabella DynamoDB da un account a un altro
- Prontuario per l'integrazione di DAX con le applicazioni DynamoDB

- Considerazioni sull'utilizzo AWS PrivateLink per Amazon DynamoDB

Progettazione NoSQL per DynamoDB

I sistemi di database NoSQL come Amazon DynamoDB utilizzano modelli alternativi per la gestione dei dati, come coppie chiave-valore o archiviazione di documenti. Quando si passa da un sistema di

gestione di database relazionali a un sistema di database NoSQL come DynamoDB, è importante comprendere le differenze chiave e gli approcci di progettazione specifici.

Argomenti

- Differenze tra la progettazione di dati relazionali e NoSQL.
- Due concetti chiave per la progettazione di NoSQL.
- Approccio alla progettazione NoSQL.
- NoSQL Workbench per DynamoDB

Differenze tra la progettazione di dati relazionali e NoSQL.

I sistemi di gestione dei database relazionali (RDBMS) e i database NoSQL, hanno diversi vantaggi e svantaggi:

- Nei sistemi RDBMS, si possono eseguire query sui dati in modo flessibile ma le query sono relativamente costose e non si ridimensionano bene in situazioni di traffico elevato (consulta Fase iniziale per la modellazione dei dati relazionali in DynamoDB).
- In un database NoSQL come DynamoDB, le query possono essere eseguite in modo efficiente in un numero limitato di modi, al di fuori dei quali possono essere costose e lente.

Queste differenze rendono la progettazione del database diversa tra i due sistemi:

- Nei sistemi RDBMS, si progetta tenendo a mente la flessibilità senza doversi preoccupare dei dettagli dell'implementazione o delle prestazioni. L'ottimizzazione delle query in genere non ha ripercussioni sulla progettazione dello schema, ma la normalizzazione è importante.
- In DynamoDB, si progetta lo schema in modo specifico per rendere le query più comuni e importanti il più veloci ed economiche possibile. Le tue strutture di dati sono programmate per i requisiti specifici dei casi d'uso del tuo business.

Due concetti chiave per la progettazione di NoSQL.

La progettazione di un NoSQL richiede un approccio diverso rispetto alla progettazione di un RDBMS. Per un sistema RDBMS puoi creare un modello di dati normalizzati senza pensare ai modelli di accesso. Puoi estenderlo in seguito quando si presentano nuovi quesiti e requisiti di query. Puoi organizzare ogni tipo di dati nella sua tabella.

In che modo la progettazione NoSQL è diversa

- Per contro, non è necessario iniziare a progettare lo schema per DynamoDB finché non si sa a quali domande si deve rispondere. È essenziale capire da subito i problemi di business e i casi d'uso dell'applicazione.
- In un'applicazione DynamoDB è necessario mantenere il minor numero possibile di tabelle. Un numero inferiore di tabelle rende le cose più scalabili, richiede una minore gestione delle autorizzazioni e riduce il sovraccarico dell'applicazione DynamoDB. Può anche aiutare a mantenere i costi di backup complessivi più bassi.

Approccio alla progettazione NoSQL.

La prima fase nella progettazione di un'applicazione DynamoDB consiste nell'identificare i modelli di query specifici che il sistema deve soddisfare.

In particolare, è importante capire le tre proprietà fondamentali dei modelli di accesso all'applicazione prima di iniziare:

- Dimensioni dei dati: sapere quanti dati verranno archiviati e richiesti in uno specifico momento aiuterà a determinare la maniera più efficace per distribuire i dati.
- Forma dei dati: invece di modificare i dati quando viene elaborata una query (come nei sistemi RDBMS)., un database NoSQL organizza i dati in modo che la loro forma nel database corrisponda al soggetto della query. Questo è un fattore chiave nell'aumentare la velocità e la scalabilità.
- Velocità dei dati: DynamoDB si dimensiona aumentando il numero di partizioni fisiche disponibili per eseguire le query e distribuendo efficacemente i dati nelle partizioni. Conoscere in anticipo quali saranno i picchi di carico delle query può aiutare a determinare come partizionare i dati per sfruttare al meglio la capacità. I/O

Dopo aver identificato i requisiti specifici della query, puoi organizzare i dati in casi ai principi generali che definiscono le prestazioni:

- Raggruppamento dei dati correlati. La ricerca ha dimostrato che il principio della “località di riferimento”, che consiste nel tenere insieme i dati correlati in un unico posto, è un fattore chiave per migliorare le prestazioni e i tempi di risposta nei sistemi NoSQL, proprio come si è rivelato importante per l’ottimizzazione delle tabelle di routing molti anni fa.

Come regola generale, è necessario mantenere il minor numero possibile di tabelle in un'applicazione DynamoDB.

Le eccezioni sono i casi in cui sono coinvolti dati di serie temporali a volumi elevati o i set di dati che hanno modelli di accesso molto diversi. Una tabella singola con indici invertiti possono solitamente abilitare query semplici per creare ed estrarre strutture di dati gerarchici complesse richieste dalla tua applicazione.

- Utilizzo dell'ordine di selezione. Gli elementi correlati possono essere raggruppati e le query possono essere eseguite facilmente se la progettazione della chiave prevede che siano raggruppati. Questa è una strategia di progettazione NoSQL importante.
- Distribuzione delle query. È inoltre importante che un volume elevato di query non si concentri su una sola parte del database, dove possono superare I/O la capacità. Invece, è necessario progettare chiavi di dati per distribuire il traffico il più possibile in maniera uniforme tra le partizioni, evitando “hot spot”.
- Utilizzo di indici secondari globali. Creando indici secondati globali specifici, puoi abilitare query diverse rispetto a quelle supportate dalla tua tabella principale, che sono comunque veloci e non tanto costose.

Questi principi generali si traducono in alcuni modelli di progettazione comuni che puoi utilizzare per modellare i dati in maniera efficiente in DynamoDB.

NoSQL Workbench per DynamoDB

NoSQL Workbench per DynamoDB è un'applicazione GUI lato client multipiattaforma per operazioni e sviluppo di database moderni. È disponibile per Windows, macOS e Linux. NoSQL Workbench è uno strumento visuale di sviluppo che offre funzionalità di modellazione, visualizzazione dei dati, generazione di dati di esempio e funzionalità di sviluppo di query che consentono di progettare, creare, eseguire query e gestire le tabelle DynamoDB. Grazie a NoSQL Workbench per DynamoDB, è possibile creare nuovi modelli di dati o progettare modelli in base ai modelli di dati esistenti che soddisfino i pattern di accesso ai dati delle applicazioni. Puoi anche importare ed esportare il modello di dati progettato alla fine del processo. Per ulteriori informazioni, consulta Creazione di modelli di dati con NoSQL Workbench.

Utilizzo di DynamoDB Well-Architected Lens per ottimizzare il carico di lavoro di DynamoDB

Questa sezione descrive Amazon DynamoDB Well-Architected Lens, una raccolta di principi di progettazione e linee guida per la progettazione di carichi di lavoro DynamoDB correttamente strutturati.

Ottimizzazione dei costi sulle tabelle DynamoDB

Questa sezione illustra le best practice su come ottimizzare i costi per le tabelle DynamoDB esistenti. Esaminare le seguenti strategie per vedere quale strategia di ottimizzazione dei costi si adatta meglio alle proprie esigenze e affrontarle in modo iterativo. Ogni strategia fornirà una panoramica di ciò che potrebbe influire sui costi, quali segnali cercare e le indicazioni prescrittive su come ridurli.

Argomenti

- Valutazione dei costi a livello di tabella
- Valutazione della modalità di capacità della tabella DynamoDB
- Valutazione delle impostazioni di dimensionamento automatico di una tabella DynamoDB
- Valutazione della selezione della classe di tabella DynamoDB
- Identificazione di risorse inutilizzate in DynamoDB
- Valutazione dei modelli di utilizzo delle tabelle DynamoDB
- Valutazione dell'utilizzo dei flussi DynamoDB
- Valutazione della capacità allocata per un provisioning di dimensioni adeguate in una tabella DynamoDB

Valutazione dei costi a livello di tabella

Lo strumento Cost Explorer presente all'interno Console di gestione AWS di consente di visualizzare i costi suddivisi per tipo, come i costi di lettura, scrittura, archiviazione e backup. Puoi anche visualizzare questi costi riepilogati per periodo, ad esempio per mese o per giorno.

Una sfida che gli amministratori devono affrontare si presenta quando è necessario esaminare i costi di una singola tabella specifica. Alcuni di questi dati sono disponibili tramite la console DynamoDB o tramite chiamate all'API `DescribeTable`, tuttavia, per impostazione predefinita, Esploratore dei

costi non consente di filtrare o raggruppare in base ai costi associati a una tabella specifica. Questa sezione mostra come utilizzare i tag per eseguire l'analisi dei costi di singole tabelle nello strumento.

Argomenti

- Come visualizzare i costi di una singola tabella DynamoDB
- Visualizzazione predefinita di Esploratore dei costi
- Come utilizzare e applicare i tag di tabella in Esploratore dei costi

Come visualizzare i costi di una singola tabella DynamoDB

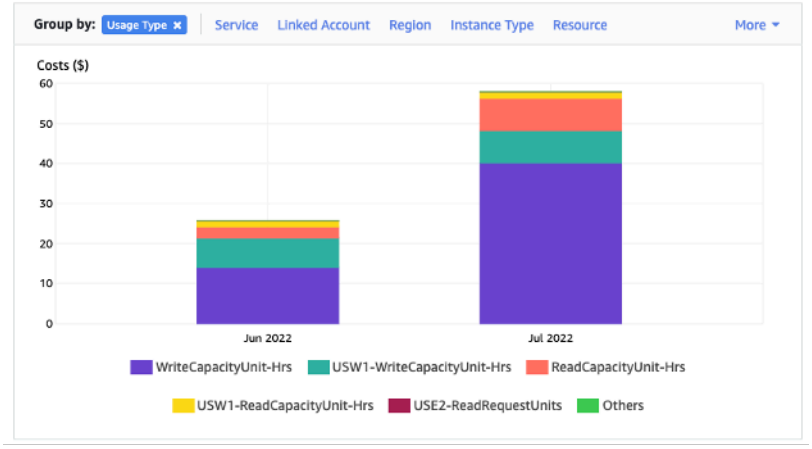
Sia Amazon Console di gestione AWS DynamoDB che `DescribeTable` l'API ti mostreranno informazioni su una singola tabella, incluso lo schema della chiave primaria, gli eventuali indici sulla tabella e le dimensioni e il numero di elementi della tabella e degli eventuali indici. Le dimensioni della tabella, più le dimensioni degli indici, possono essere utilizzate per calcolare il costo di archiviazione mensile per la tabella. Ad esempio, 0,25 USD per GB nella regione us-east-1.

Se la tabella è in modalità di capacità assegnata, vengono restituite anche le impostazioni correnti RCU e WCU. Queste potrebbero essere utilizzate per calcolare i costi correnti di lettura e scrittura per la tabella, ma questi costi potrebbero cambiare, soprattutto se per la tabella è stata configurata la scalabilità automatica.

> **Note**
>
> Se la tabella è in modalità di capacità on demand, `DescribeTable` non sarà utile per stimare i costi di velocità di trasmissione effettiva, poiché vengono fatturati in base all'utilizzo effettivo e non su quello con assegnazione in un determinato periodo.

Visualizzazione predefinita di Esploratore dei costi

La visualizzazione predefinita di Esploratore dei costi fornisce grafici che mostrano il costo delle risorse consumate, come la velocità di trasmissione effettiva e l'archiviazione. Puoi scegliere di raggruppare i costi per periodo, ad esempio i totali per mese o per giorno. È inoltre possibile suddividere e confrontare i costi di archiviazione, lettura, scrittura e altre caratteristiche.



Come utilizzare e applicare i tag di tabella in Esploratore dei costi

Per impostazione predefinita, Esploratore dei costi non fornisce un riepilogo dei costi per una tabella specifica, poiché combina i costi di più tabelle in un totale. Tuttavia, puoi utilizzare l'assegnazione di tag alle risorse AWS per identificare ogni tabella tramite un tag di metadati. I tag sono coppie chiave-valore che è possibile utilizzare per diversi scopi, ad esempio per identificare tutte le risorse appartenenti a un progetto o a un reparto. Per questo esempio, supponiamo che tu abbia una tabella denominata. MyTable

1. Imposta un tag con la chiave di table_name e il valore di. MyTable
2. Attiva il tag in Esploratore dei costi, quindi filtra il valore del tag per ottenere maggiore visibilità sui costi di ogni tabella.

> **Note**
>
> Potrebbero essere necessari uno o due giorni prima che il tag inizi a comparire in Esploratore dei costi

Puoi impostare tu stesso i tag dei metadati nella console o tramite l'automazione come la AWS CLI o l'SDK AWS . Valuta la possibilità di richiedere l'impostazione di un tag table_name come parte del nuovo processo di creazione delle tabelle dell'organizzazione. Per le tabelle esistenti, è disponibile un'utilità Python che troverà e applicherà questi tag a tutte le tabelle esistenti in una determinata regione del tuo account. Vedi Eponymous Table Tagger su per maggiori dettagli. GitHub

Valutazione della modalità di capacità della tabella DynamoDB

In questa sezione viene fornita una panoramica su come selezionare la modalità di capacità appropriata per la tabella DynamoDB. Ogni modalità è ottimizzata per soddisfare le esigenze di un carico di lavoro diverso in termini di capacità di risposta alle variazioni della velocità di trasmissione effettiva e di fatturazione dell'utilizzo. Per prendere una decisione è necessario bilanciare questi fattori.

Argomenti

- Modalità di capacità della tabella disponibili
- Quando selezionare la modalità di capacità on demand
- Quando selezionare la modalità di capacità assegnata
- Fattori aggiuntivi da valutare nella scelta di una modalità di capacità della tabella

Modalità di capacità della tabella disponibili

Quando crei una tabella DynamoDB, devi selezionare la modalità di capacità on demand o la modalità di capacità assegnata.

È possibile cambiare le tabelle dalla modalità con capacità allocata alla modalità on demand fino a quattro volte in una finestra variabile di 24 ore. È possibile passare le tabelle dalla modalità on demand alla modalità con capacità allocata in qualsiasi momento.

Modalità di capacità on demand

La modalità con capacità on demand è progettata per eliminare la necessità di pianificare o eseguire il provisioning della capacità di una tabella DynamoDB. In questa modalità, le richieste alla tabella verranno soddisfate istantaneamente senza la necessità di dimensionare le risorse (fino al doppio del picco di velocità di trasmissione effettiva precedente della tabella).

DynamoDB on-demand pay-per-request offre prezzi per le richieste di lettura e scrittura in modo da pagare solo per ciò che si utilizza.

Modalità di capacità assegnata

La modalità con capacità allocata è un modello più tradizionale in cui è necessario definire la capacità disponibile per le richieste alla tabella direttamente o sfruttando il dimensionamento automatico. Poiché per la tabella viene allocata una capacità specifica in un dato momento, la fatturazione si basa sulla capacità allocata totale anziché sul numero di richieste utilizzate. Se si supera la capacità

assegnata la tabella potrebbe rifiutare le richieste, con conseguente impatto negativo sull'esperienza utente delle applicazioni.

La modalità con capacità allocata richiede un monitoraggio continuo per trovare un bilanciamento al fine di evitare il provisioning eccessivo o insufficiente della tabella per mantenere contenuti sia la limitazione (della larghezza di banda della rete) che i costi.

Quando selezionare la modalità di capacità on demand

Se l'obiettivo è ottimizzare i costi e il carico di lavoro è simile a quello nei seguenti grafici, la modalità on demand è la scelta migliore.

I seguenti fattori contribuiscono a questo tipo di carico di lavoro:

- Modello di traffico che si evolve nel tempo
- Volume variabile di richieste (derivante da carichi di lavoro in batch)
- Tempi imprevedibili delle richieste (con conseguenti picchi di traffico)
- Scende a zero o al di sotto del 30% del picco in una determinata ora

Per i carichi di lavoro che presentano i fattori indicati in precedenza, l'utilizzo del dimensionamento automatico per mantenere nella tabella una capacità sufficiente per rispondere ai picchi di traffico comporterà probabilmente un provisioning eccessivo della tabella e un costo superiore al necessario oppure il fatto che la tabella presenti un provisioning insufficiente e che le richieste vengano sottoposte a limitazione (della larghezza di banda della rete) inutilmente. La modalità con capacità on demand è la scelta migliore perché è in grado di gestire le fluttuazioni del traffico senza dover prevedere o regolare la capacità.

Con il modello di pay-per-request prezzo della modalità on demand, non devi preoccuparti della capacità inattiva perché paghi solo per il throughput effettivamente utilizzato. La fatturazione viene effettuata in base alle richieste di lettura o scrittura utilizzate, quindi i costi riflettono direttamente

l'utilizzo effettivo, semplificando il bilanciamento delle spese e delle prestazioni. Facoltativamente è anche possibile configurare il throughput massimo di lettura o scrittura (o entrambe) al secondo per singole tabelle on demand e indici secondari globali al fine di mantenere costi e utilizzo limitati. Per ulteriori informazioni, consulta Throughput massimo per tabelle on demand.

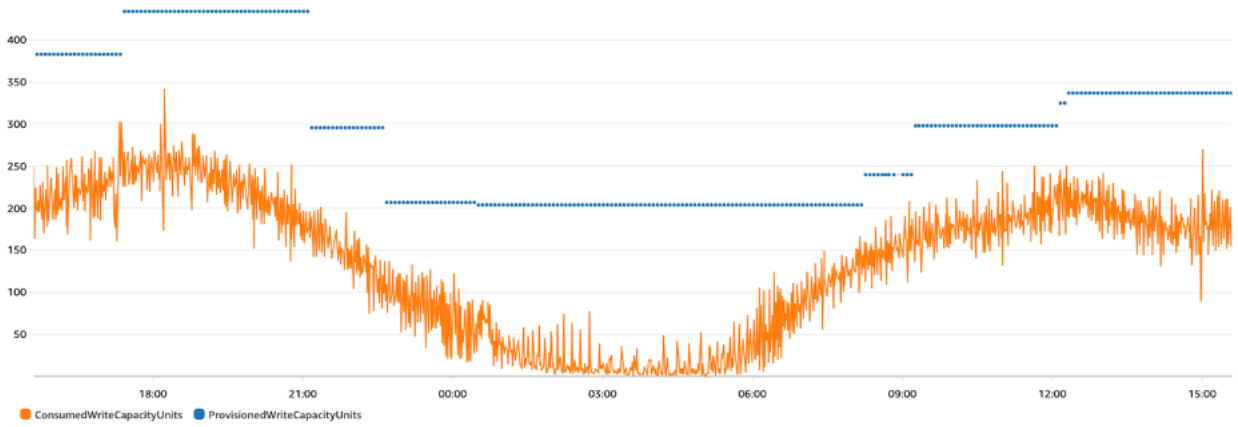
Quando selezionare la modalità di capacità assegnata

Un carico di lavoro ideale per la modalità con capacità allocata presenta un modello di utilizzo più stabile e prevedibile, come mostrato nel grafico seguente.

> **Note**
>
> Si consiglia di esaminare le metriche con una certa granularità, ad esempio per 14 giorni, prima di intervenire sulla capacità allocata.

I seguenti fattori contribuiscono a questo tipo di carico di lavoro:

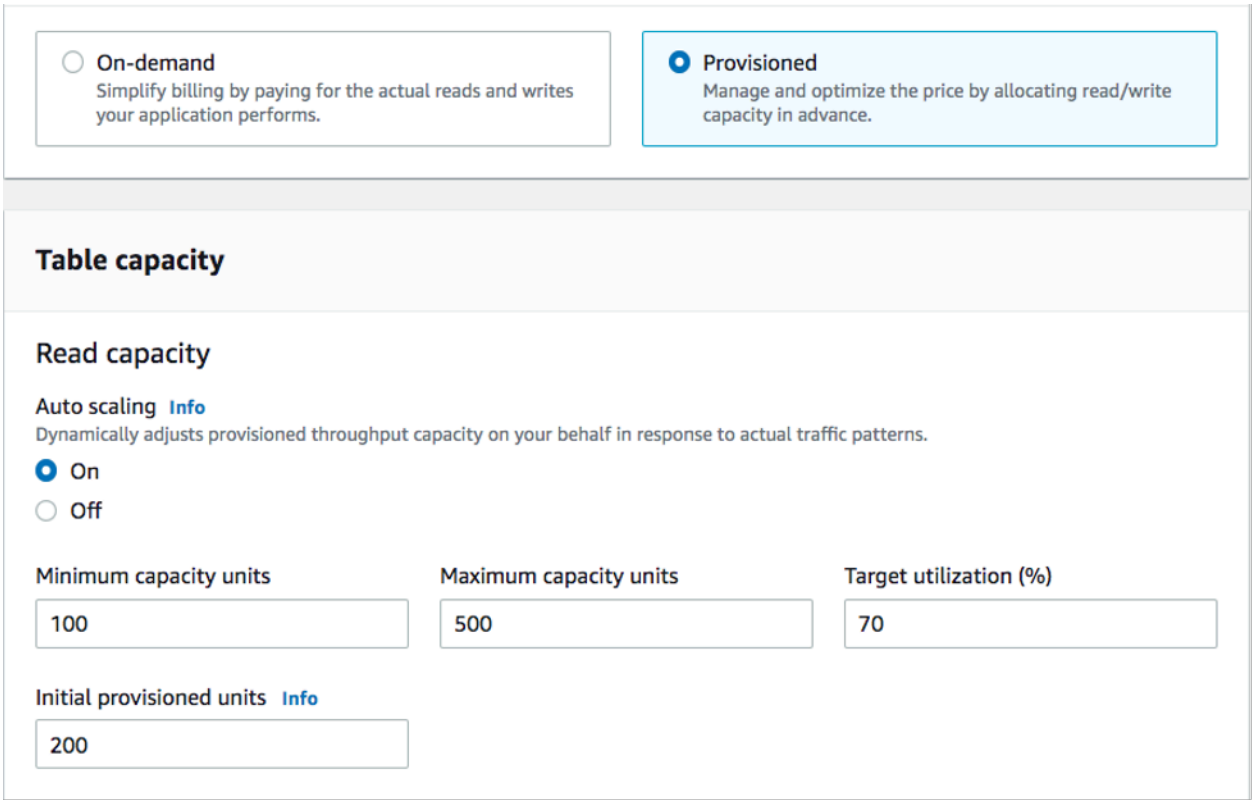
- Traffico costante, prevedibile e ciclico per una determinata ora o giorno
- Picchi di traffico limitati e di breve durata



Poiché i volumi di traffico in una determinata ora o giorno sono più stabili, è possibile impostare la capacità allocata della tabella perché sia relativamente vicina alla capacità effettivamente utilizzata. In definitiva, ottimizzare i costi di una tabella con capacità assegnata significa avvicinare il più possibile la capacità assegnata (linea blu) alla capacità consumata (linea arancione) senza aumentare la metrica `ThrottledRequests` nella tabella. Lo spazio tra le due linee è sia uno spreco di capacità che un'assicurazione contro una cattiva esperienza utente dovuta alla limitazione della larghezza di banda della rete. Se si è in grado di prevedere i requisiti di throughput dell'applicazione e si preferisce

la prevedibilità dei costi derivante dal controllo della capacità di lettura e scrittura, allora può essere opportuno continuare a utilizzare le tabelle con provisioning.

DynamoDB fornisce la scalabilità automatica per le tabelle con capacità assegnata, che bilancerà automaticamente questo valore per conto dell'utente. Ciò consente di tenere traccia della capacità consumata durante il giorno e di impostare la capacità della tabella in base a poche variabili. Quando si utilizza il dimensionamento automatico, la tabella subirà un provisioning eccessivo e sarà necessario regolare con precisione il rapporto tra il numero di limitazioni (della larghezza di banda della rete) e le unità di capacità con provisioning eccessivo per soddisfare le esigenze dei carichi di lavoro.



Unità di capacità minima

È possibile impostare la capacità minima di una tabella per ridurre la limitazione della larghezza di banda della rete, ma ciò non ridurrà il costo della tabella. Se la tabella presenta periodi di basso utilizzo seguiti da un improvviso picco di utilizzo elevato, l'impostazione del valore minimo può impedire che la scalabilità automatica imposti la capacità della tabella su un valore troppo basso.

Unità di capacità massima

È possibile impostare la capacità massima di una tabella per limitare un dimensionamento della tabella maggiore del previsto. Valuta la possibilità di applicare un valore massimo per le tabelle

di sviluppo o di test in cui non si desideri eseguire test di carico su larga scala. Puoi impostare un numero massimo per qualsiasi tabella, ma assicurati di valutare regolarmente questa impostazione rispetto alla capacità di base della tabella quando la usi in produzione per evitare una limitazione della larghezza di banda della rete accidentale.

Utilizzo di destinazione

L'impostazione dell'utilizzo di destinazione della tabella è il mezzo principale per l'ottimizzazione dei costi per una tabella con capacità assegnata. L'impostazione di un valore di percentuale inferiore aumenterà il livello di assegnazione eccessiva della tabella, aumentando i costi ma riducendo il rischio di limitazione della larghezza di banda della rete. L'impostazione di un valore di percentuale superiore diminuirà l'assegnazione eccessiva della tabella, aumentando però il rischio di limitazione della larghezza di banda della rete.

Fattori aggiuntivi da valutare nella scelta di una modalità di capacità della tabella

Quando si decide tra le due modalità, ci sono alcuni fattori ulteriori che vale la pena valutare.

Utilizzo della capacità allocata

Per determinare quando la modalità on demand costerebbe meno della capacità allocata, è utile esaminare l'utilizzo della capacità allocata, che si riferisce all'efficienza con cui vengono utilizzate le risorse allocate (o "con provisioning"). La modalità on demand costa meno per i carichi di lavoro con un utilizzo medio della capacità allocata inferiore al 35%. In molti casi, anche per carichi di lavoro con un utilizzo della capacità allocata superiore al 35%, può essere più conveniente utilizzare la modalità on demand, soprattutto se il carico di lavoro presenta periodi di bassa attività combinati a picchi occasionali.

Capacità riservata

Per le tabelle con capacità assegnata, DynamoDB offre la possibilità di acquistare capacità riservata per lettura e scrittura (le unità di capacità di scrittura replicate e le tabelle Standard-IA attualmente non sono idonee). La capacità riservata offre sconti significativi rispetto ai prezzi standard della capacità allocata.

Quando decidi tra le due modalità della tabella, valuta quanto questo ulteriore sconto influirà sul costo della tabella. In alcuni casi, può essere meno economicamente oneroso eseguire un carico di lavoro relativamente imprevedibile su una tabella con provisioning in eccesso e con capacità riservata.

Miglioramento della prevedibilità del carico di lavoro

In alcune situazioni, un carico di lavoro può avere un modello sia prevedibile che imprevedibile. Sebbene questo possa essere facilmente supportato da una tabella on demand, i costi saranno probabilmente migliori se è possibile migliorare i modelli imprevedibili nel carico di lavoro.

Una delle cause più comuni di questi modelli è l'importazione in batch. Questo tipo di traffico può spesso superare la capacità di base della tabella a tal punto che nell'esecuzione si potrebbe verificare una limitazione della larghezza di banda della rete. Per mantenere un carico di lavoro come questo in esecuzione su una tabella con capacità assegnata, valuta le seguenti opzioni:

- Se il batch viene eseguito in orari pianificati, è possibile pianificare un aumento della capacità di scalabilità automatica prima dell'esecuzione
- Se il batch si verifica in modo casuale, prova a prolungare il tempo di esecuzione anziché eseguirlo il più velocemente possibile
- Aggiungi un periodo di incremento all'importazione in cui la velocità dell'importazione inizia lentamente ma aumenta in modo graduale nell'arco di pochi minuti, fino a quando il dimensionamento automatico non abbia avuto l'opportunità di iniziare a regolare la capacità della tabella

Valutazione delle impostazioni di dimensionamento automatico di una tabella DynamoDB

In questa sezione viene fornita una panoramica su come valutare le impostazioni di dimensionamento automatico sulle tabelle DynamoDB. Il dimensionamento automatico di Amazon DynamoDB è una funzionalità che gestisce la velocità di trasmissione effettiva delle tabelle e dell'indice secondario globale (GSI) in base al traffico dell'applicazione e alle metriche di utilizzo di destinazione. Ciò garantisce che le tabelle o le vostre tabelle GSIs dispongano della capacità richiesta per i modelli di applicazione.

Il servizio di scalabilità AWS automatica monitorerà l'utilizzo corrente della tabella e lo confronterà con il valore di utilizzo target:. `TargetValue` Invierà una notifica se è necessario aumentare o diminuire la capacità allocata.

Argomenti

- Comprendere le impostazioni di dimensionamento automatico
- Come identificare le tabelle con un basso utilizzo di destinazione (<=50%)
- Come gestire i carichi di lavoro con varianza stagionale
- Come affrontare carichi di lavoro con picchi di lavoro con pattern sconosciuti

- Come gestire i carichi di lavoro con applicazioni collegate

Comprendere le impostazioni di dimensionamento automatico

La definizione del valore corretto per l'utilizzo di destinazione, la fase iniziale e i valori finali è un'attività che richiede il coinvolgimento del team operativo. Ciò consente di definire correttamente i valori in base all'utilizzo storico dell'applicazione, che verranno utilizzati per attivare le policy AWS di dimensionamento automatico. Il target di utilizzo è la percentuale della capacità totale che deve essere raggiunta durante un periodo di tempo prima che si applichino le regole di dimensionamento automatico.

Quando si imposta un target di utilizzo elevato (intorno al 90%), significa che il traffico deve essere superiore al 90% per un periodo di tempo prima che si attivi il dimensionamento automatico. Non dovresti utilizzare un target di utilizzo elevato a meno che l'applicazione non sia molto costante e non riceva picchi di traffico.

Quando si imposta un utilizzo molto basso (un target inferiore al 50%), significa che l'applicazione deve raggiungere il 50% della capacità fornita prima di attivare una policy di dimensionamento automatico. A meno che il traffico delle applicazioni non cresca a un ritmo molto aggressivo, questo di solito si traduce in capacità inutilizzata e risorse sprecate.

Come identificare le tabelle con un basso utilizzo di destinazione (<=50%)

Puoi utilizzare AWS CLI o Console di gestione AWS per monitorare e identificare le `TargetValues` policy di auto scaling nelle tue risorse DynamoDB:

AWS CLI

1. Restituisce l'intero elenco di risorse eseguendo il seguente comando:

   ```
   aws application-autoscaling describe-scaling-policies --service-namespace
    dynamodb
   ```

   Questo comando restituirà l'intero elenco di policy di dimensionamento automatico emesse per qualsiasi risorsa DynamoDB. Se desideri recuperare solo le risorse da una particolare tabella, puoi aggiungere `–resource-id parameter`. Ad esempio:

   ```
   aws application-autoscaling describe-scaling-policies --service-namespace
    dynamodb --resource-id "table/<table-name>”
   ```

2. Restituisce solo le policy di dimensionamento automatico per un particolare GSI eseguendo il seguente comando

```
aws application-autoscaling describe-scaling-policies --service-namespace
 dynamodb --resource-id "table/<table-name>/index/<gsi-name>”
```

I valori che ci interessano per le policy di dimensionamento automatico sono evidenziati di seguito. Vogliamo assicurarci che il valore di destinazione sia superiore al 50% per evitare un provisioning eccessivo. Viene visualizzato un risultato simile al seguente:

```
{
    "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"
        }
    ]
}
```

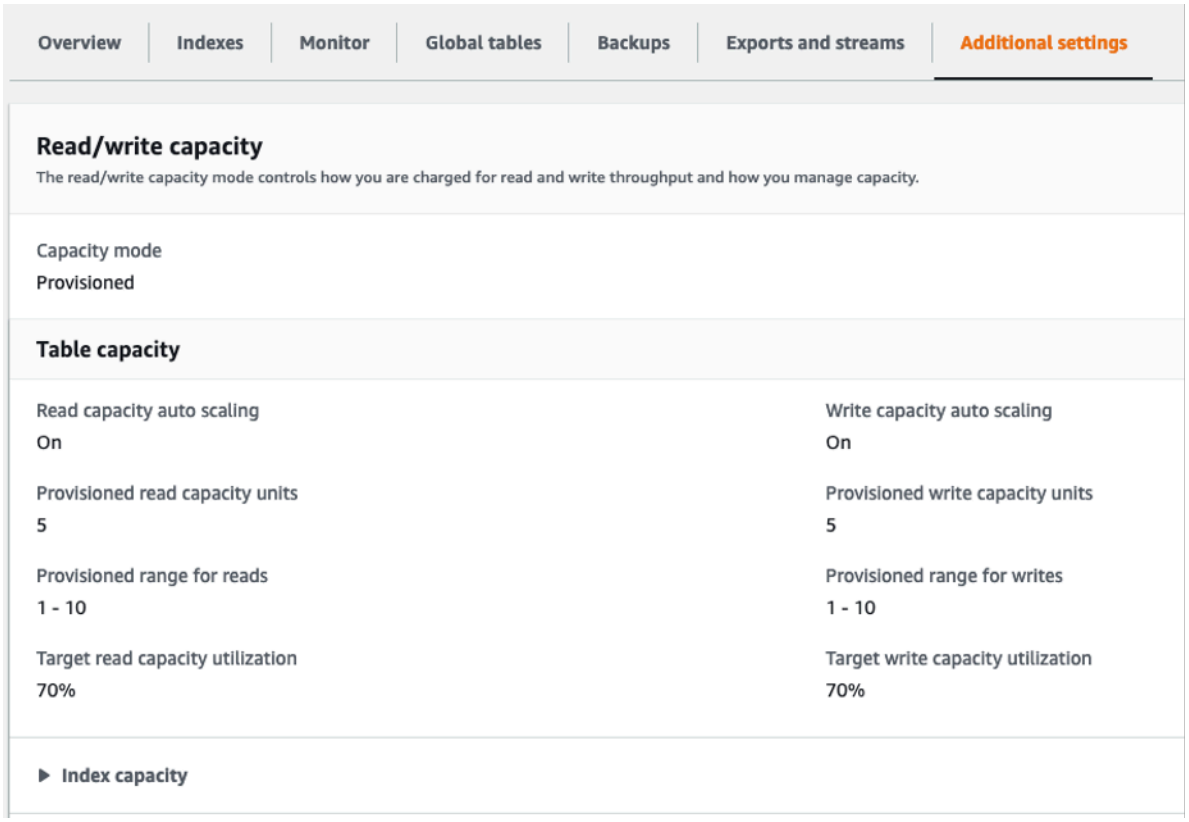
Console di gestione AWS

1. Accedi Console di gestione AWS e apri la console DynamoDB all'indirizzo. https://console.aws.amazon.com/dynamodb/

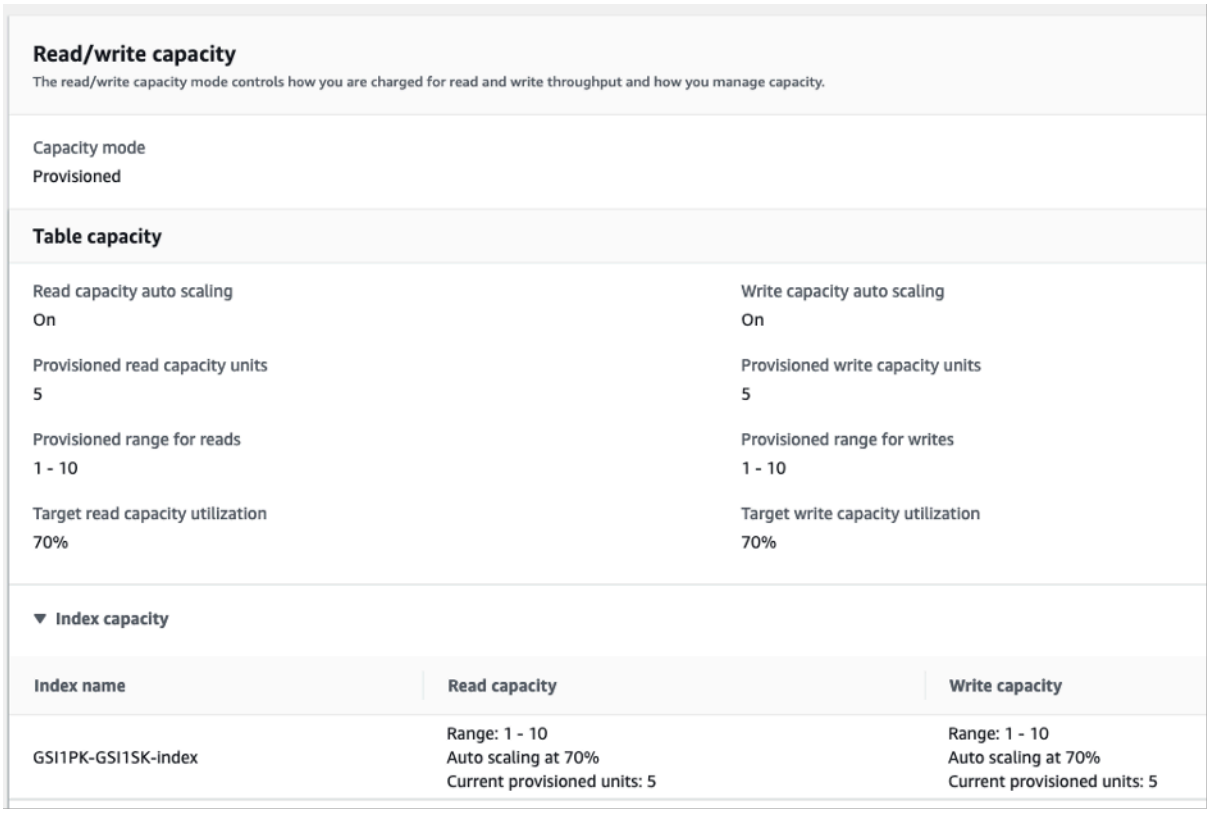
   Seleziona quella appropriata, Regione AWS se necessario.

2. Nel riquadro di navigazione sinistro selezionare Tables (Tabelle). Nella pagina Tables (Tabelle), selezionare il nome della tabella.

3. Nella pagina Dettagli tabella, seleziona Impostazioni aggiuntive e quindi controlla le impostazioni di dimensionamento automatico della tabella.



Per gli indici, espandi la scheda Capacità di indice per controllare le impostazioni di dimensionamento automatico dell'indice.



Se i valori di utilizzo di destinazione sono inferiori o uguali al 50%, è consigliabile esaminare le metriche di utilizzo della tabella per vedere se il provisioning è insufficiente o eccessivo.

Come gestire i carichi di lavoro con varianza stagionale

Si consideri il seguente scenario: l'applicazione funziona con un valore medio minimo per la maggior parte del tempo, ma il target di utilizzo è basso, quindi l'applicazione può reagire rapidamente agli eventi che si verificano in determinate ore del giorno e la capacità è sufficiente ed evitare limitazioni della larghezza di banda della rete. Questo scenario è comune quando un'applicazione è molto impegnata durante il normale orario di ufficio (dalle 9:00 alle 17:00) ma funziona a un livello base nelle altre ore. Poiché alcuni utenti inizieranno a connettersi prima delle 9:00, l'applicazione utilizza questa soglia bassa per aumentare rapidamente la capacità richiesta nelle ore di punta.

Lo scenario potrebbe essere simile al seguente:

- Tra le 17:00 e le 9:00, le unità `ConsumedWriteCapacity` sono comprese tra 90 e 100
- Gli utenti iniziano a connettersi all'applicazione prima delle 9:00 e le unità di capacità aumentano considerevolmente (il valore massimo che rilevato è 1500 WCU)
- In media, l'utilizzo delle applicazioni varia tra 800 e 1.200 durante l'orario di ufficio

Se lo scenario precedente è applicabile alla propria situazione, prendere in considerazione l'utilizzo del dimensionamento automatico pianificato, in cui la tabella potrebbe comunque avere una regola configurata di dimensionamento automatico dell'applicazione, ma con un utilizzo di destinazione meno aggressivo che fornisca la capacità aggiuntiva solo negli intervalli specifici necessari.

È possibile AWS CLI eseguire i seguenti passaggi per creare una regola di autoscaling pianificato che verrà eseguita in base all'ora del giorno e al giorno della settimana.

1. Registrazione della tabella DynamoDB o del GSI come target scalabile con Application Auto Scaling. Un target scalabile è una risorsa che Application Auto Scaling può aumentare ridurre orizzontalmente.

   ```
   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. Impostazione delle operazioni pianificate in base ai requisiti.

   Avremo bisogno di due regole per coprire lo scenario: una per l'aumento e una per la riduzione. La prima regola per aumentare l'operazione pianificata:

   ```
   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 seconda regola per ridurre l'operazione pianificata:

   ```
   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. Esegui il comando seguente per convalidare che entrambe le regole siano state attivate:

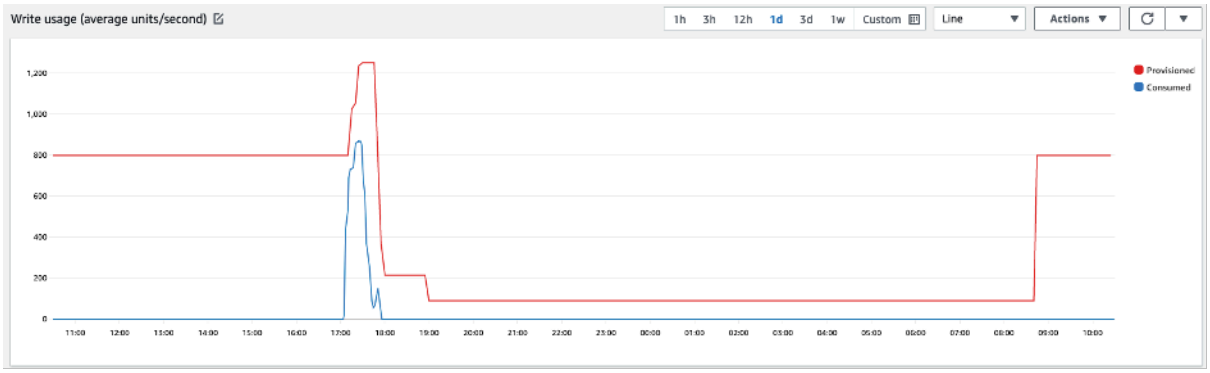
```
aws application-autoscaling describe-scheduled-actions --service-namespace dynamodb
```

Si dovrebbe ottenere un risultato simile a questo:

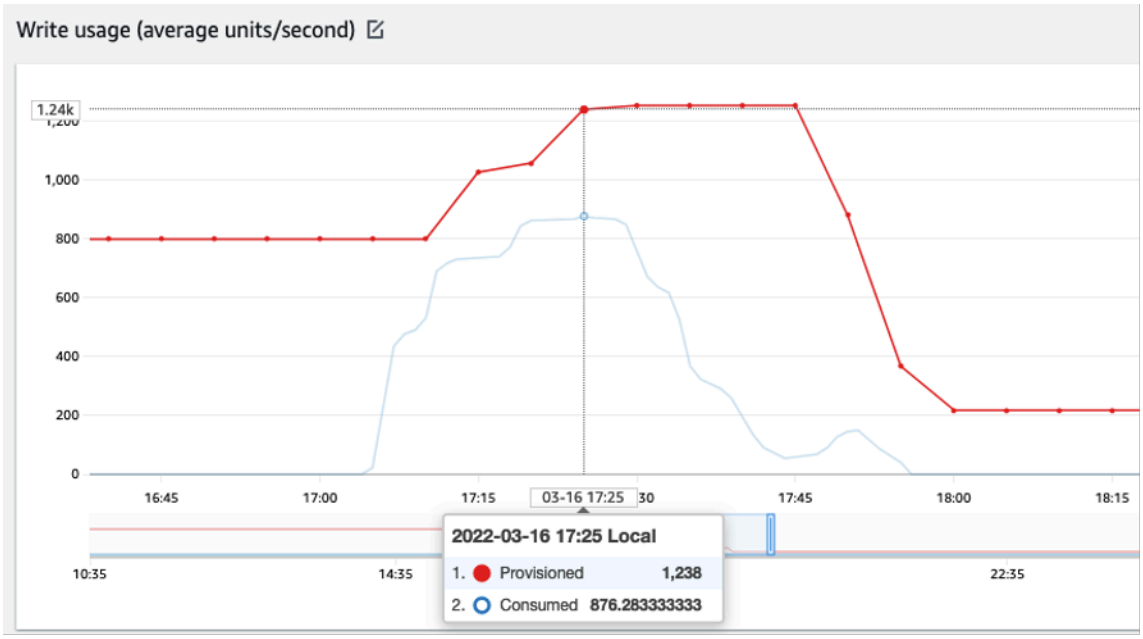
```
{
    "ScheduledActions": [
        {
            "ScheduledActionName": "my-5-8-scheduled-down-action",
            "ScheduledActionARN":
 "arn:aws:autoscaling:<region>:<account>:scheduledAction:<uuid>:resource/dynamodb/
table/<table-name>:scheduledActionName/my-5-8-scheduled-down-action",
            "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"
        }
```

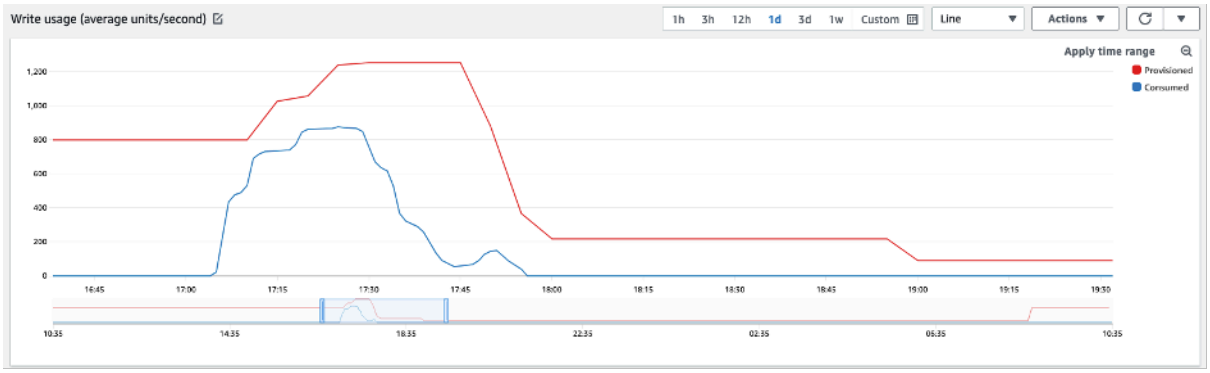
```
    ]
}
```

L'immagine seguente mostra un carico di lavoro di esempio che mantiene sempre il 70% di utilizzo di destinazione. Si noti come le regole di dimensionamento automatico siano ancora valide e la velocità di trasmissione effettiva non verrà ridotta.



Ingrandendo, è possibile notare che c'è stato un picco nell'applicazione che ha attivato la soglia di dimensionamento automatico del 70%, forzandolo ad attivarsi e a fornire la capacità aggiuntiva richiesta per la tabella. L'operazione di dimensionamento automatico pianificata influirà sui valori massimi e minimi e la configurazione deve essere effettuata dall'utente.





Come affrontare carichi di lavoro con picchi di lavoro con pattern sconosciuti

In questo scenario, l'applicazione utilizza un target di utilizzo molto basso perché non si conoscono ancora i pattern dell'applicazione e si desidera assicurarsi che il carico di lavoro non sia limitato.

Si consiglia invece di utilizzare la modalità di capacità on demand. Le tabelle on demand sono perfette per carichi di lavoro con picchi di lavoro di cui non si conoscono i pattern di traffico. Con la modalità di capacità on demand, si paga in base alla richiesta per le letture e le scritture dei dati che l'applicazione esegue sulle tabelle. Non è necessario specificare la velocità di trasmissione effettiva di lettura e scrittura che si prevede l'applicazione abbia, in quanto DynamoDB adatta istantaneamente i carichi di lavoro, man mano che aumentano o diminuiscono.

Come gestire i carichi di lavoro con applicazioni collegate

In questo scenario, l'applicazione dipende da altri sistemi, ad esempio scenari di elaborazione in batch in cui è possibile avere grandi picchi di traffico in base agli eventi nella logica dell'applicazione.

Prendere in considerazione lo sviluppo di una logica di dimensionamento automatico personalizzata che reagisca a quegli eventi in cui è possibile aumentare la capacità delle tabelle e dei `TargetValues` base alle esigenze specifiche. Potresti trarre vantaggio Amazon EventBridge e utilizzare una combinazione di AWS servizi come Lambda e Step Functions per rispondere alle esigenze specifiche delle tue applicazioni.

Valutazione della selezione della classe di tabella DynamoDB

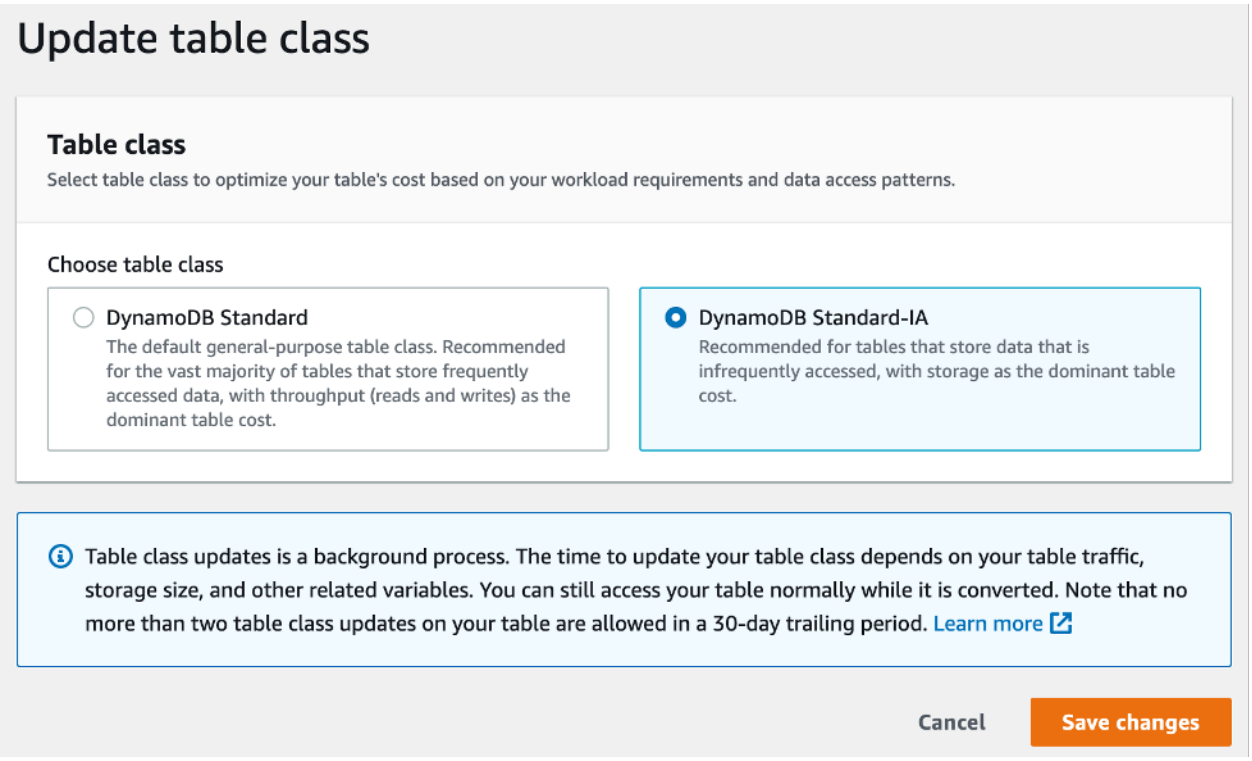
In questa sezione viene fornita una panoramica su come selezionare la classe di tabella appropriata per la tabella DynamoDB. Con il lancio della classe di tabella Standard Infrequent-Access (Standard-IA), ora è possibile ottimizzare una tabella per ridurre i costi di archiviazione o i costi di velocità di trasmissione effettiva.

Argomenti

- Classi di tabelle disponibili
- Quando selezionare la classe di tabella DynamoDB standard
- Quando selezionare la classe di tabella DynamoDB standard-IA
- Fattori aggiuntivi da valutare nella scelta di una classe di tabella

Classi di tabelle disponibili

Quando crei una tabella DynamoDB, per la classe della tabella devi selezionare DynamoDB Standard o DynamoDB Standard-IA. La classe della tabella può essere modificata due volte in un periodo di 30 giorni, pertanto è sempre possibile modificarla in futuro. La selezione di una delle classi di tabella non ha alcun effetto sulle prestazioni, sulla disponibilità, sull'affidabilità o sulla durata della tabella.



Classe di tabella standard

La classe di tabella Standard è l'opzione predefinita per le nuove tabelle. Questa opzione mantiene il bilanciamento di fatturazione originale di DynamoDB che offre un equilibrio tra velocità di trasmissione effettiva e costi di archiviazione per le tabelle con dati ad accesso frequente.

Classe di tabella Standard-IA

La classe di tabella Standard-IA offre un costo di archiviazione inferiore (inferiore circa del 60%) per i carichi di lavoro che richiedono l'archiviazione a lungo termine dei dati con aggiornamenti o letture

poco frequenti. Poiché la classe è ottimizzata per l'accesso poco frequente, le operazioni di lettura e scrittura verranno fatturate a un costo leggermente superiore (circa del 25%) rispetto alla classe di tabella Standard.

Quando selezionare la classe di tabella DynamoDB standard

La classe di tabella DynamoDB Standard è la più adatta per le tabelle il cui costo di archiviazione è circa il 50% o meno del costo mensile complessivo della tabella. Questo bilanciamento dei costi è indicativo di un carico di lavoro che accede o aggiorna regolarmente gli elementi già archiviati in DynamoDB.

Quando selezionare la classe di tabella DynamoDB standard-IA

La classe di tabella DynamoDB Standard-IA è la più adatta per le tabelle il cui costo di archiviazione è circa il 50% o più del costo mensile complessivo della tabella. Questo bilanciamento dei costi è indicativo di un carico di lavoro che crea o legge meno elementi al mese di quanti ne mantenga nell'archiviazione.

Un uso comune della classe di tabelle Standard-IA è lo spostamento dei dati a cui si accede meno frequentemente in singole tabelle Standard-IA. Per ulteriori informazioni, consulta Optimizing the storage costs of your workloads with Amazon DynamoDB Standard-IA table class (Ottimizzazione dei costi di archiviazione dei carichi di lavoro con la classe di tabella Amazon DynamoDB Standard-IA).

Fattori aggiuntivi da valutare nella scelta di una classe di tabella

Quando si decide tra le due classi di tabelle, è opportuno prendere in considerazione alcuni fattori aggiuntivi.

Capacità riservata

L'acquisto di capacità riservata per le tabelle che utilizzano la classe di tabella Standard-IA non è attualmente supportato. Quando si passa da una tabella Standard con capacità riservata a una tabella Standard-IA senza capacità riservata, è possibile che non si ottenga alcun vantaggio in termini di costi.

Identificazione di risorse inutilizzate in DynamoDB

In questa sezione viene fornita una panoramica su come valutare regolarmente le risorse inutilizzate. Man mano che i requisiti delle tue applicazioni si evolvono, dovresti assicurarti che nessuna risorsa resti inutilizzata per evitare costi Amazon DynamoDB non necessari. Le procedure descritte di

seguito utilizzeranno i CloudWatch parametri di Amazon per identificare le risorse inutilizzate e ti aiuteranno a identificare e ad agire su tali risorse per ridurre i costi.

Puoi monitorare DynamoDB CloudWatch utilizzando, che raccoglie ed elabora i dati grezzi da DynamoDB in metriche leggibili quasi in tempo reale. Queste statistiche vengono conservate per un determinato periodo di tempo, così da consentire l'accesso a informazioni cronologiche per comprendere meglio l'utilizzo. Per impostazione predefinita, i dati delle metriche DynamoDB vengono inviati automaticamente a. CloudWatch Per ulteriori informazioni, consulta What is Amazon CloudWatch? e conservazione dei parametri nella Amazon CloudWatch User Guide.

Argomenti

- Come identificare le risorse inutilizzate
- Identificazione delle risorse non utilizzate della tabella
- Pulizia delle risorse non utilizzate della tabella
- Identificazione delle risorse GSI inutilizzate
- Pulizia delle risorse GSI inutilizzate
- Pulizia delle tabelle globali inutilizzate
- Eliminazione dei backup o del point-in-time ripristino non utilizzati (PITR)

Come identificare le risorse inutilizzate

Per identificare tabelle o indici non utilizzati, esamineremo le seguenti CloudWatch metriche per un periodo di 30 giorni per capire se ci sono letture o scritture attive sulla tabella o letture sugli indici secondari globali (): GSIs

ConsumedReadCapacityUnits

Il numero di unità di capacità di lettura utilizzate nel periodo di tempo specificato, per permettere di tenere traccia di quanta capacità viene utilizzata. È possibile recuperare la capacità di lettura totale consumata per una tabella e tutti i relativi indici secondari globali o per un determinato indice secondario globale.

ConsumedWriteCapacityUnits

Il numero di unità di capacità di scrittura utilizzate nel periodo di tempo specificato, per permettere di tenere traccia di quanta capacità viene utilizzata. È possibile recuperare la capacità di scrittura totale consumata per una tabella e tutti i relativi indici secondari globali o per un determinato indice secondario globale.

Identificazione delle risorse non utilizzate della tabella

Amazon CloudWatch è un servizio di monitoraggio e osservabilità che fornisce le metriche delle tabelle DynamoDB che utilizzerai per identificare le risorse non utilizzate. CloudWatch le metriche possono essere visualizzate sia tramite. Console di gestione AWS AWS Command Line Interface

AWS Command Line Interface

Per visualizzare le metriche delle tabelle tramite AWS Command Line Interface, puoi utilizzare i seguenti comandi.

1. Per prima cosa, valuta le letture della tabella:

```
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>
```

Per evitare di identificare erroneamente una tabella come non utilizzata, valutare i parametri per un periodo più lungo. Scegli un intervallo appropriato di inizio e fine, ad esempio 30 giorni, e un periodo appropriato, ad esempio 86400.

Nei dati restituiti, qualsiasi somma superiore a 0 indica che la tabella che stai valutando ha ricevuto traffico di lettura durante quel periodo.

Il risultato seguente mostra una tabella che riceve traffico di lettura nel periodo valutato:

```
        {
            "Timestamp": "2022-08-25T19:40:00Z",
            "Sum": 36023355.0,
            "Unit": "Count"
        },
        {
            "Timestamp": "2022-08-12T19:40:00Z",
            "Sum": 38025777.5,
            "Unit": "Count"
        },
```

Il risultato seguente mostra una tabella che non riceve traffico di lettura nel periodo valutato:

```
        {
```

```
            "Timestamp": "2022-08-01T19:50:00Z",
            "Sum": 0.0,
            "Unit": "Count"
        },
        {
            "Timestamp": "2022-08-20T19:50:00Z",
            "Sum": 0.0,
            "Unit": "Count"
        },
```

2. Quindi, valuta le scritture della tabella:

```
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>
```

Per evitare di identificare erroneamente una tabella come non utilizzata, consigliamo di valutare i parametri per un periodo più lungo. Scegli un intervallo appropriato di inizio e fine, ad esempio 30 giorni, e un periodo appropriato, ad esempio 86400.

Nei dati restituiti, qualsiasi somma superiore a 0 indica che la tabella che stai valutando ha ricevuto traffico di scrittura durante quel periodo.

Il risultato seguente mostra una tabella che riceve traffico di scrittura nel periodo valutato:

```
        {
            "Timestamp": "2022-08-19T20:15:00Z",
            "Sum": 41014457.0,
            "Unit": "Count"
        },
        {
            "Timestamp": "2022-08-18T20:15:00Z",
            "Sum": 40048531.0,
            "Unit": "Count"
        },
```

Il risultato seguente mostra una tabella che non riceve traffico di scrittura nel periodo valutato:

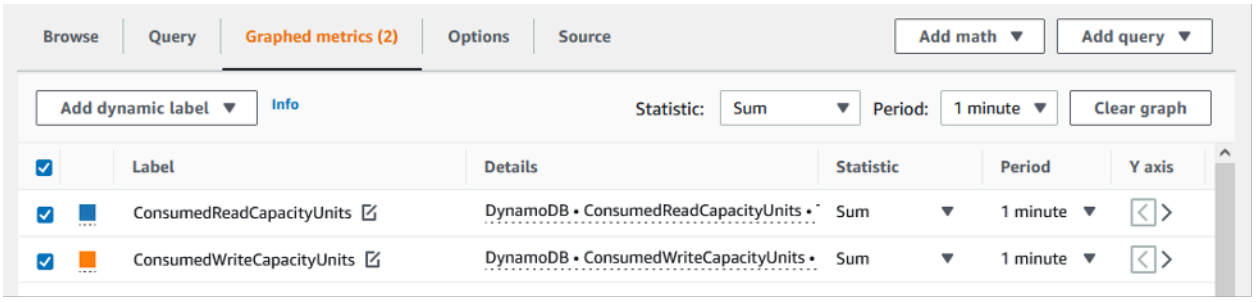
```
        {
            "Timestamp": "2022-07-31T20:15:00Z",
            "Sum": 0.0,
```

```
            "Unit": "Count"
        },
        {
            "Timestamp": "2022-08-19T20:15:00Z",
            "Sum": 0.0,
            "Unit": "Count"
        },
```

Console di gestione AWS

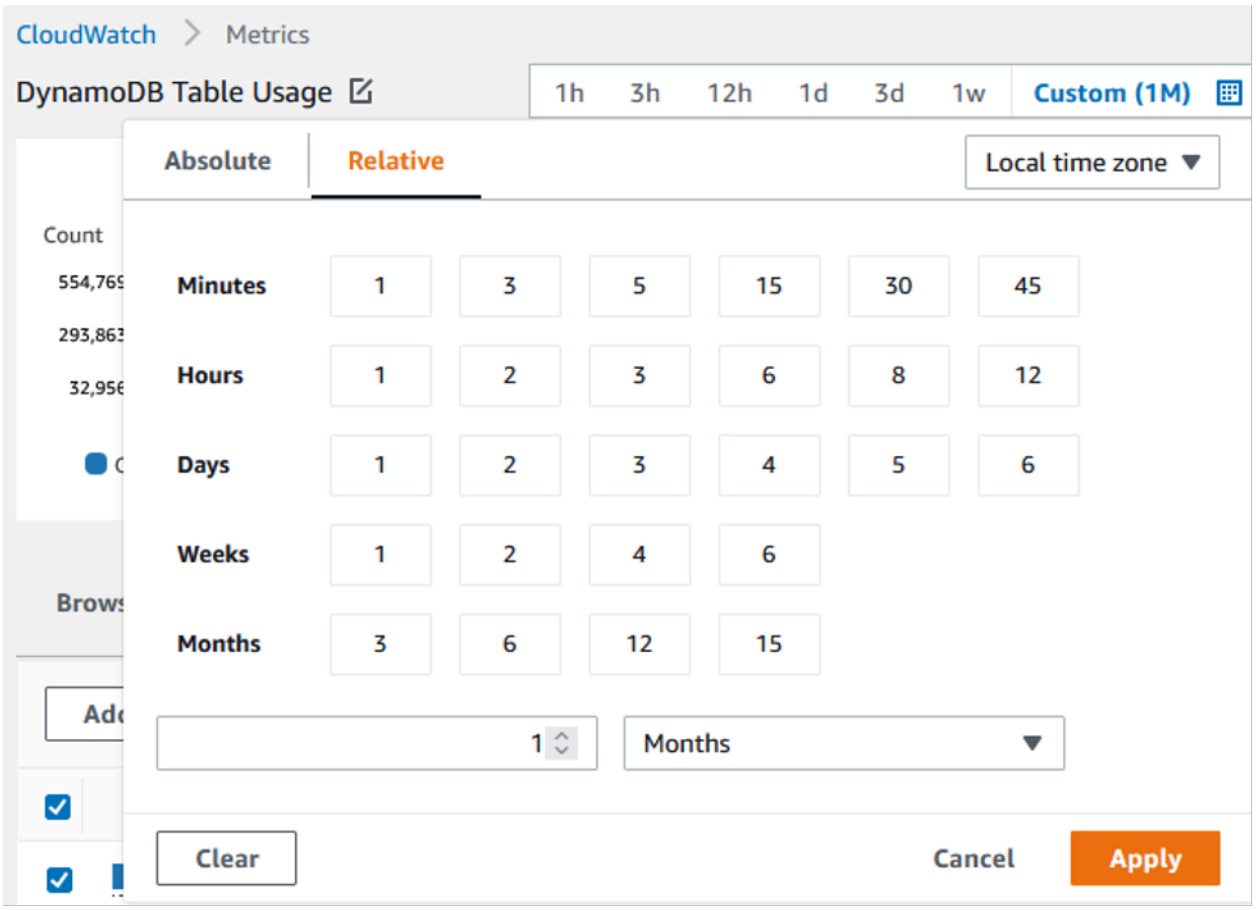
Le seguenti fasi consentiranno di valutare l'utilizzo delle risorse tramite la Console di gestione AWS.

1. Accedi alla AWS console e vai alla pagina del CloudWatch servizio all'indirizzo https://console.aws.amazon.com/cloudwatch/. Se necessario, seleziona la AWS regione appropriata in alto a destra della console.
2. Sulla barra di navigazione a sinistra, individua la sezione Metrics (Parametri) quindi seleziona All metrics (Tutti i parametri).
3. L'operazione precedente aprirà un pannello di controllo con due pannelli. Nel pannello superiore vedrai i parametri attualmente rappresentati graficamente. In basso selezionerai i parametri disponibili per il grafico. Seleziona DynamoDB nel pannello inferiore.
4. Nel pannello di selezione dei parametri di DynamoDB, seleziona la categoria Table Metrics (Parametri della tabella) per visualizzare i parametri delle tabelle nella regione corrente.
5. Identifica il nome della tabella scorrendo il menu verso il basso, quindi seleziona i parametri `ConsumedReadCapacityUnits` e `ConsumedWriteCapacityUnits` della tabella.
6. Seleziona la scheda Graphed metrics (2) (Parametri nel grafico (2) e imposta la colonna Statistic (Statistiche) su Sum (Somma).



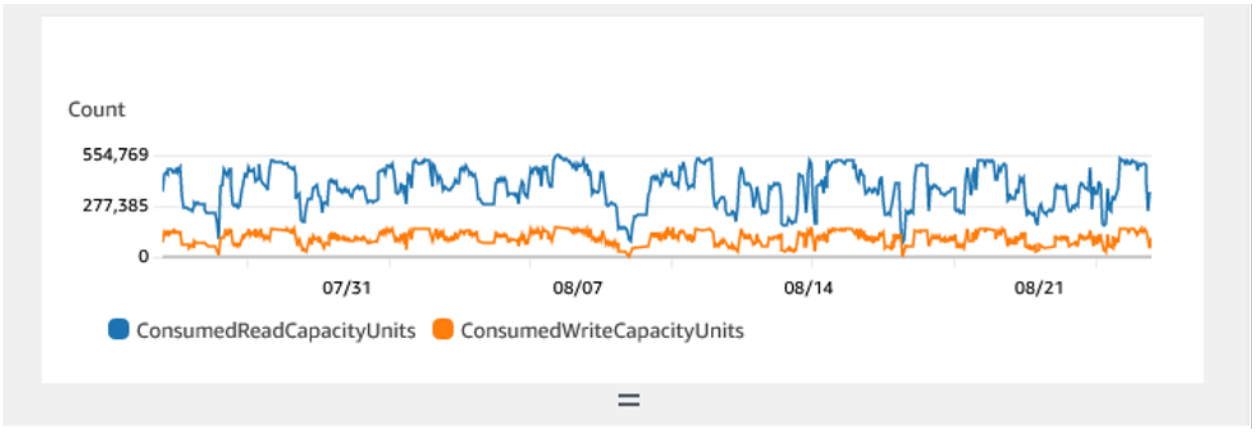
7. Per evitare di identificare erroneamente una tabella come non utilizzata, consigliamo di valutare i parametri per un periodo più lungo. Nella parte superiore del pannello del grafico, scegli un intervallo di tempo appropriato, ad esempio un mese, per valutare la tabella.

Seleziona Custom (Personalizzato), seleziona 1 Months (Un mese) nel menu a discesa e scegli Apply (Applica).

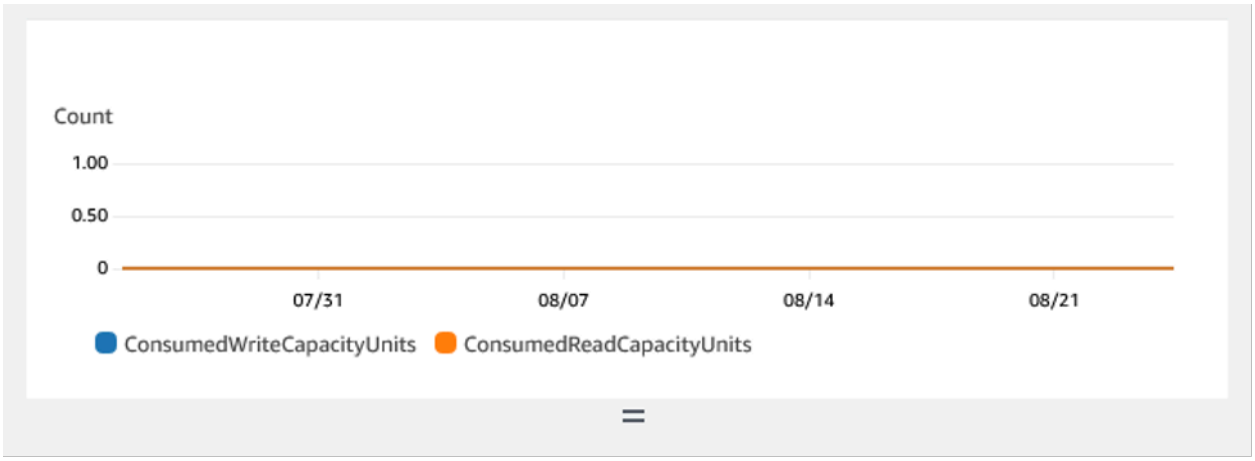


8. Valuta i parametri nel grafico della tabella per determinare se viene utilizzata. Parametri superiori a 0 indicano che durante il periodo di tempo preso in considerazione la tabella è stata utilizzata. Un grafico fermo sullo 0 per letture e scritture indica che la tabella non è stata utilizzata.

   L'immagine seguente mostra una tabella con traffico di lettura:



L'immagine seguente mostra una tabella senza traffico di lettura:



Pulizia delle risorse non utilizzate della tabella

Se sono state identificate risorse non utilizzate della tabella, è possibile ridurne i costi correnti nei seguenti modi.

> **Note**
>
> Se hai identificato una tabella non utilizzata ma desideri comunque mantenerla disponibile nel caso in cui sia necessario accedervi in futuro, valuta la possibilità di passare alla modalità on demand. Altrimenti, puoi valutare l'idea di eseguire il backup e di eliminare completamente la tabella.

Modalità di capacità

DynamoDB addebita la lettura, la scrittura e l'archiviazione dei dati nelle tabelle DynamoDB.

DynamoDB ha due modalità di capacità, che includono opzioni di fatturazione specifiche per l'elaborazione delle letture e delle scritture sulle tabelle: on demand e con assegnazione. La modalità read/write di capacità controlla il modo in cui ti viene addebitato il throughput di lettura e scrittura e come gestisci la capacità.

Per le tabelle in modalità on demand, non è necessario specificare la velocità effettiva di lettura e scrittura che si prevede l'applicazione esegua. DynamoDB addebita le letture e le scritture eseguite dall'applicazione sulle tabelle in termini di unità di richiesta di lettura e unità di richiesta di scrittura. Se

non c'è alcuna attività sul tuo computer table/index , non pagherai per la velocità effettiva, ma dovrai comunque sostenere un costo di archiviazione.

Classi di tabelle

DynamoDB offre due classi di tabelle progettate per aiutare a ottimizzare i costi. La classe di tabella DynamoDB Standard è quella predefinita ed è consigliata per la maggior parte dei carichi di lavoro. La classe di tabella DynamoDB Standard-Infrequent Access (DynamoDB Standard (accesso infrequente)) è ottimizzata per le tabelle in cui l'archiviazione è il costo principale.

Se non vi è alcuna attività sulla tabella o sull'indice, è probabile che il costo principale sia quello dell'archiviazione e cambiare classe di tabella offrirà un risparmio significativo.

Eliminazione delle tabelle

Se hai rilevato una tabella non utilizzata e desideri eliminarla, potresti voler prima effettuare un backup o un'esportazione dei dati.

I backup eseguiti tramite AWS Backup possono sfruttare lo storage a freddo su più livelli, riducendo ulteriormente i costi. Consulta la Utilizzo di AWS Backup con Dynamo DB documentazione per informazioni su come abilitare i backup tramite AWS Backup e la documentazione sulla gestione dei piani di backup per informazioni su come utilizzare il ciclo di vita per spostare il backup in cold storage.

In alternativa, puoi scegliere di esportare i dati della tabella in S3. Per fare ciò, consulta la documentazione relativa all'esportazione in Amazon S3. Una volta esportati i dati, se desideri sfruttare S3 Glacier Instant Retrieval, S3 Glacier Flexile Retrieval o S3 Glacier Deep Archive per ridurre ulteriormente i costi, consulta Gestione del ciclo di vita dell'archiviazione.

Dopo aver eseguito il backup della tabella, puoi scegliere di eliminarla tramite la Console di gestione AWS o tramite l' AWS Command Line Interface.

Identificazione delle risorse GSI inutilizzate

Le fasi per identificare un indice secondario globale non utilizzato sono simili a quelle per identificare una tabella non utilizzata. Poiché DynamoDB replica gli elementi scritti nella tabella di base nel GSI se è presente l'attributo utilizzato come chiave di partizione del GSI, è probabile che un GSI non utilizzato abbia ancora un valore `ConsumedWriteCapacityUnits` superiore a 0 se la relativa tabella di base è in uso. Di conseguenza, per determinare se il GSI non è utilizzato valuterai solo il parametro `ConsumedReadCapacityUnits`.

Per visualizzare le metriche GSI tramite AWS AWS CLI, è possibile utilizzare i seguenti comandi per valutare le letture delle tabelle:

```
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>
```

Per evitare di identificare erroneamente una tabella come non utilizzata, consigliamo di valutare i parametri per un periodo più lungo. Scegli un intervallo appropriato di inizio e fine, ad esempio 30 giorni e un periodo appropriato, ad esempio 86400.

Nei dati restituiti, qualsiasi somma superiore a 0 indica che la tabella che stai valutando ha ricevuto traffico di lettura durante quel periodo.

Il risultato seguente mostra un GSI che riceve traffico di lettura nel periodo valutato:

```
        {
          "Timestamp": "2022-08-17T21:20:00Z",
          "Sum": 36319167.0,
          "Unit": "Count"
        },
        {
          "Timestamp": "2022-08-11T21:20:00Z",
          "Sum": 1869136.0,
          "Unit": "Count"
        },
```

Il risultato seguente mostra un GSI che riceve traffico di lettura minimo nel periodo valutato:

```
        {
          "Timestamp": "2022-08-28T21:20:00Z",
          "Sum": 0.0,
          "Unit": "Count"
        },
        {
          "Timestamp": "2022-08-15T21:20:00Z",
          "Sum": 2.0,
          "Unit": "Count"
        },
```

Il risultato seguente mostra un GSI che non riceve traffico di lettura nel periodo valutato:

```
        {
          "Timestamp": "2022-08-17T21:20:00Z",
          "Sum": 0.0,
          "Unit": "Count"
        },
        {
          "Timestamp": "2022-08-11T21:20:00Z",
          "Sum": 0.0,
          "Unit": "Count"
        },
```

Pulizia delle risorse GSI inutilizzate

Se hai identificato un GSI non utilizzato, puoi scegliere di eliminarlo. Poiché tutti i dati presenti in un GSI sono presenti anche nella tabella di base, non è necessario un ulteriore backup prima di eliminare un GSI. Se in futuro il GSI sarà nuovamente necessario, può essere aggiunto nuovamente alla tabella.

Se hai identificato un GSI usato raramente, dovresti considerare delle modifiche alla progettazione dell'applicazione che consentirebbero di eliminarlo o di ridurne i costi. Ad esempio, mentre le scansioni DynamoDB devono essere utilizzate con parsimonia perché possono consumare grandi quantità di risorse di sistema, possono essere più convenienti rispetto a un GSI se il modello di accesso che supporta viene utilizzato molto raramente.

Inoltre, se è necessario un GSI per supportare un modello di accesso poco frequente, valuta la possibilità di progettare un set più limitato di attributi. Sebbene ciò possa richiedere successive query sulla tabella di base per supportare i modelli di accesso poco frequenti, può potenzialmente offrire una riduzione significativa dei costi di archiviazione e scrittura.

Pulizia delle tabelle globali inutilizzate

Le tabelle globali di Amazon DynamoDB forniscono una soluzione completamente gestita per l'implementazione di un database multi-regione e multi-attivo, senza dover sviluppare e mantenere la propria soluzione di replica.

Le tabelle globali sono ideali per fornire un accesso a bassa latenza ai dati vicini agli utenti e una regione secondaria per il ripristino di emergenza.

Se l'opzione delle tabelle globali è abilitata per una risorsa allo scopo di fornire un accesso a bassa latenza ai dati, ma non fa parte della strategia di disaster recovery, verifica che entrambe le repliche

servano attivamente il traffico di lettura valutandone le metriche. CloudWatch Se una replica non serve traffico di lettura, potrebbe trattarsi di una risorsa inutilizzata.

Se le tabelle globali fanno parte della strategia di disaster recovery, è possibile che una replica non riceva traffico di lettura secondo uno schema. active/standby

Eliminazione dei backup o del point-in-time ripristino non utilizzati (PITR)

DynamoDB offre due stili di backup. Point-in-timeil ripristino fornisce backup continui per un massimo di 35 giorni per aiutarti a proteggerti da scritture o eliminazioni accidentali, mentre il backup su richiesta consente la creazione di istantanee che possono essere salvate a lungo termine. È possibile impostare il periodo di recupero su qualsiasi valore compreso tra 1 e 35 giorni. Entrambi i tipi di backup hanno dei costi associati.

Consulta la documentazione di Backup e ripristino per DynamoDB e Backup point-in-time per DynamoDB per determinare se nelle tabelle sono abilitati i backup che potrebbero non essere più necessari.

Valutazione dei modelli di utilizzo delle tabelle DynamoDB

In questa sezione viene fornita una panoramica su come valutare se si stanno utilizzando le tabelle DynamoDB in modo efficiente. Alcuni modelli di utilizzo non sono ottimali per DynamoDB e lasciano spazio all'ottimizzazione sia dal punto di vista delle prestazioni che dei costi.

Argomenti

- Riduzione del numero di operazioni ad elevata consistenza di lettura
- Riduzione del numero di transazioni per le operazioni di lettura
- Riduzione del numero di scansioni
- Abbreviazione dei nomi di attributi
- Abilitazione del Time to Live (TTL)
- Sostituzione delle tabelle globali con backup tra regioni

Riduzione del numero di operazioni ad elevata consistenza di lettura

DynamoDB consente di configurare la consistenza di lettura in base alle richieste. Le richieste di lettura sono a consistenza finale per impostazione predefinita. Le letture a consistenza finale vengono addebitate a 0,5 RCU per un massimo di 4 KB di dati.

La maggior parte dei carichi di lavoro distribuiti sono flessibili e possono tollerare la consistenza finale. Tuttavia, possono esserci schemi di accesso che richiedono elevata consistenza di lettura. Le letture ad elevata consistenza vengono addebitate a 1 RCU per un massimo di 4 KB di dati, il che raddoppia sostanzialmente i costi di lettura. DynamoDB offre la flessibilità di utilizzare entrambi i modelli di consistenza sulla stessa tabella.

È possibile valutare il carico di lavoro e il codice dell'applicazione per verificare se vengono utilizzate letture ad elevata consistenza solo dove necessario.

Riduzione del numero di transazioni per le operazioni di lettura

DynamoDB ti consente di raggruppare determinate azioni in all-or-nothing un modo, il che significa che hai la possibilità di eseguire transazioni ACID con DynamoDB. Tuttavia, come nel caso dei database relazionali, non è necessario seguire questo approccio per ogni operazione.

Un'operazione di lettura transazionale fino a 4 KB consuma 2 RCUs a differenza dell'impostazione predefinita di 0,5 RCUs per leggere la stessa quantità di dati in modo sostanzialmente coerente. I costi delle operazioni di scrittura sono raddoppiati, il che significa che una scrittura transazionale fino a 1 KB equivale a 2. WCUs

Per determinare se tutte le operazioni sulle tabelle sono transazioni, è possibile filtrare le CloudWatch metriche relative alla tabella fino alla transazione. APIs Se APIs le transazioni sono gli unici grafici disponibili sotto le `SuccessfulRequestLatency` metriche della tabella, ciò confermerebbe che ogni operazione è una transazione per questa tabella. In alternativa, se la tendenza complessiva dell'utilizzo della capacità corrisponde alla tendenza delle API di transazione, prendi in considerazione la possibilità di rivisitare il design dell'applicazione in quanto sembra dominato da quello transazionale. APIs

Riduzione del numero di scansioni

L'uso estensivo delle operazioni `Scan` deriva generalmente dalla necessità di eseguire query analitiche sui dati DynamoDB. Eseguire frequenti operazioni `Scan` su una tabella di grandi dimensioni può essere inefficiente e costoso.

Un'alternativa migliore consiste nell'utilizzare la funzionalità Esporta in S3 e scegliere un momento temporale per esportare lo stato della tabella in S3. AWS offre servizi come Athena che possono quindi essere utilizzati per eseguire query analitiche sui dati senza consumare alcuna capacità della tabella.

La frequenza delle operazioni `Scan` può essere determinata utilizzando la statistica `SampleCount` della metrica `SuccessfulRequestLatency` per `Scan`. Se le operazioni `Scan` sono davvero molto frequenti, i modelli di accesso e il modello dei dati dovrebbero essere rivalutati.

Abbreviazione dei nomi di attributi

La dimensione totale di un elemento in DynamoDB è data dalla somma delle lunghezze dei nomi e dei valori dei relativi attributi. Avere nomi di attributi lunghi non solo contribuisce ai costi di archiviazione, ma potrebbe anche portare a un maggiore RCU/WCU consumo. È consigliabile preferire nomi di attributo brevi piuttosto che lunghi. Avere nomi di attributi più brevi può aiutare a limitare le dimensioni degli elementi entro il limite successivo di 4 KB/1 KB, dopo il quale ne consumerebbero altri per accedere ai dati. RCU/WCU

Abilitazione del Time to Live (TTL)

Il Time to Live (TTL) può identificare gli elementi più vecchi della data di scadenza impostata per un elemento e rimuoverli dalla tabella. Se i dati aumentano nel tempo e quelli più vecchi diventano irrilevanti, l'attivazione del TTL sulla tabella può aiutare a ridurre i dati e risparmiare sui costi di archiviazione.

Un altro aspetto utile del TTL è che gli elementi scaduti si trovano nei flussi di DynamoDB, quindi anziché semplicemente rimuovere i dati dai dati, è possibile utilizzare tali elementi dal flusso e archiviarli a un livello di archiviazione a costo inferiore. Inoltre, l'eliminazione di elementi tramite TTL non comporta costi aggiuntivi: non utilizza capacità e non comporta il costo di progettazione di un'applicazione di pulizia.

Sostituzione delle tabelle globali con backup tra regioni

Le tabelle globali consentono di mantenere più tabelle di replica attive in diverse regioni: possono tutte accettare operazioni di scrittura e replicare i dati l'una sull'altra. È facile configurare le repliche e la sincronizzazione viene gestita automaticamente. Le repliche convergono verso uno stato coerente facendo prevalare la versione dell'ultimo utente che ha modificato il file.

Se si utilizzano le tabelle globali esclusivamente come parte di una strategia di failover o di ripristino di emergenza (DR), è possibile sostituirle con una copia di backup tra regioni per obiettivi di punti di ripristino e requisiti relativi ai tempi di ripristino relativamente permissivi. Se non è necessario un accesso locale rapido e l'elevata disponibilità continua, mantenere una replica globale della tabella potrebbe non essere l'approccio migliore per il failover.

In alternativa, prendi in considerazione l'utilizzo di AWS Backup per gestire i backup di DynamoDB. È possibile pianificare backup regolari e copiarli in più regioni per soddisfare i requisiti di DR con un approccio più conveniente rispetto all'utilizzo delle tabelle globali.

Valutazione dell'utilizzo dei flussi DynamoDB

In questa sezione viene fornita una panoramica su come valutare l'utilizzo dei flussi DynamoDB. Alcuni modelli di utilizzo non sono ottimali per DynamoDB e lasciano spazio all'ottimizzazione sia dal punto di vista delle prestazioni che dei costi.

Sono disponibili due integrazioni di streaming native per casi d'uso basati sullo streaming e sugli eventi:

- Amazon DynamoDB Streams
- Amazon Kinesis Data Streams

Questa pagina si concentrerà solo sulle strategie di ottimizzazione dei costi per queste opzioni. Se invece vuoi scoprire come scegliere tra le due opzioni, consulta Opzioni di streaming per Change Data Capture.

Argomenti

- Ottimizzazione dei costi per i flussi DynamoDB
- Ottimizzazione dei costi per il flusso di dati Kinesis
- Strategie di ottimizzazione dei costi per entrambi i tipi di utilizzo dei flussi

Ottimizzazione dei costi per i flussi DynamoDB

Come indicato nella pagina dei prezzi dei flussi DynamoDB, indipendentemente dalla modalità di capacità effettiva di trasmissione della tabella, DynamoDB addebita il numero di richieste di lettura effettuate verso il flusso DynamoDB della tabella. Le richieste di lettura effettuate verso un flusso DynamoDB sono diverse dalle richieste di lettura effettuate verso una tabella DynamoDB.

Ogni richiesta di lettura in termini di flusso ha la forma di una chiamata API `GetRecords` che può restituire fino a 1000 record o 1 MB di record nella risposta, a seconda di quale delle due condizioni viene raggiunta per prima. Nessuno degli altri DynamoDB APIs Stream viene addebitato e DynamoDB Streams non viene addebitato per l'inattività. In altre parole, se non viene effettuata alcuna richiesta di lettura a un flusso DynamoDB, non verrà addebitato alcun costo per l'attivazione di un flusso DynamoDB su una tabella.

Ecco alcune applicazioni consumer per i flussi DynamoDB:

- AWS Lambda funzione/i
- Applicazioni basate su Amazon Kinesis Data Streams
- Applicazioni destinate ai clienti destinate ai consumatori create utilizzando un AWS SDK

Le richieste di lettura effettuate dagli utenti di AWS DynamoDB Streams basati su Lambda sono gratuite, mentre le chiamate effettuate dai consumatori di qualsiasi altro tipo sono a pagamento. Ogni mese, anche le prime 2.500.000 richieste di lettura effettuate da consumatori non Lambda sono gratuite. Questo vale per tutte le richieste di lettura effettuate a qualsiasi flusso DynamoDB in AWS un account per ogni regione. AWS

Monitoraggio dell'utilizzo dei flussi DynamoDB

I costi di DynamoDB Streams sulla console di fatturazione sono raggruppati per tutti i DynamoDB Streams in tutta la regione in un account. AWS AWS Attualmente, il tagging dei flussi DynamoDB non è supportato, quindi i tag di allocazione dei costi non possono essere utilizzati per identificare i costi granulari dei flussi DynamoDB. Il volume delle chiamate `GetRecords` può essere ottenuto a livello di flusso DynamoDB per calcolare i costi per flusso. Il volume è rappresentato dalla `SuccessfulRequestLatency` metrica di DynamoDB Stream e dalla CloudWatch sua statistica. `SampleCount` Questa metrica includerà anche `GetRecords` le chiamate effettuate da tabelle globali per eseguire la replica continua e le chiamate effettuate dai consumatori AWS Lambda, entrambe gratuite. Per informazioni su altre CloudWatch metriche pubblicate da DynamoDB Streams, vedere. Parametri e dimensioni di DynamoDB

Utilizzo di AWS Lambda come consumatore

Valuta se l'utilizzo delle funzioni AWS Lambda come consumatori per DynamoDB Streams è fattibile perché ciò può eliminare i costi associati alla lettura da DynamoDB Stream. D'altra parte le applicazioni consumer basate sull'adattatore Kinesis dei flussi DynamoDB o sull'SDK verranno addebitate in base al numero di chiamate `GetRecords` effettuate verso il flusso DynamoDB.

Le chiamate alla funzione Lambda verranno addebitate in base ai prezzi standard di Lambda, tuttavia i flussi DynamoDB non addebiteranno alcun costo. Lambda eseguirà il polling delle partizioni presenti nel flusso DynamoDB ricercando i record a una velocità di base di 4 volte al secondo. Quando sono disponibili dei record, Lambda invoca la funzione e attende il risultato. Se l'elaborazione ha esito positivo, Lambda riprende il polling fino a quando non riceve più record.

Ottimizzazione delle applicazioni consumer basate sull'adattatore Kinesis dei flussi DynamoDB

Poiché le richieste di lettura effettuate da consumatori non basati su Lambda sono a pagamento per i flussi DynamoDB, è importante trovare un equilibrio tra la necessità di un'elaborazione quasi in tempo reale e il numero di volte in cui l'applicazione consumer deve eseguire il polling del flusso DynamoDB.

La frequenza di polling dei flussi DynamoDB che utilizza un'applicazione basata sull'adattatore Kinesis dei flussi DynamoDB è determinata dal valore `idleTimeBetweenReadsInMillis` configurato. Questo parametro determina la quantità di tempo in millisecondi che il consumatore deve attendere prima di elaborare una partizione nel caso in cui la precedente chiamata `GetRecords` effettuata alla stessa partizione non abbia restituito alcun record. Per impostazione predefinita. il valore di questo parametro è 1.000 ms. Se non è richiesta un'elaborazione quasi in tempo reale, questo parametro potrebbe essere aumentato per fare in modo che l'applicazione consumer effettui meno chiamate `GetRecords` e ottimizzi le chiamate ai flussi DynamoDB.

Ottimizzazione dei costi per il flusso di dati Kinesis

Quando un flusso di dati Kinesis è impostato come destinazione per fornire eventi di acquisizione dei dati di modifica per una tabella DynamoDB, il flusso di dati Kinesis potrebbe richiedere una gestione separata del dimensionamento che influirà sui costi complessivi. DynamoDB addebita i costi in termini di Change Data capture Units CDUs (), in cui ogni unità è composta da un elemento DynamoDB della dimensione di 1 KB tentato dal servizio DynamoDB verso il Kinesis Data Stream di destinazione.

Oltre ai costi del servizio DynamoDB, verranno applicati i costi standard del flusso di dati Kinesis. Come indicato nella pagina dei prezzi, i prezzi dei servizi variano in base alla modalità di capacità, assegnata e on demand, che sono diverse dalle modalità di capacità delle tabelle DynamoDB e sono definite dall'utente. A un livello elevato, il flusso di dati Kinesis addebitano una tariffa oraria in base alla modalità di capacità e ai dati importati nel flusso dal servizio DynamoDB. Potrebbero essere previsti costi aggiuntivi, ad esempio per il recupero dei dati (per la modalità on demand), la conservazione prolungata dei dati (oltre le 24 ore predefinite) e i recuperi fan-out ottimizzati dei consumatori a seconda della configurazione utente del flusso di dati Kinesis.

Monitoraggio dell'utilizzo del flusso di dati Kinesis

Kinesis Data Streams for DynamoDB pubblica metriche di DynamoDB oltre alle metriche standard di Kinesis Data Stream. CloudWatch È possibile che un `Put` tentativo del servizio DynamoDB venga bloccato dal servizio Kinesis a causa dell'insufficiente capacità di Kinesis Data Streams o da componenti dipendenti come AWS KMS un servizio che può essere configurato per crittografare i dati di Kinesis Data Stream a riposo.

Per ulteriori informazioni sulle CloudWatch metriche pubblicate dal servizio DynamoDB per Kinesis Data Stream, consulta. Monitoraggio di Change Data Capture con Kinesis Data Streams Per evitare costi aggiuntivi per i tentativi ripetuti del servizio a causa delle limitazioni della larghezza di banda della rete, è importante dimensionare correttamente il flusso di dati Kinesis nel caso di modalità assegnata.

Scelta della modalità di capacità corretta per il flusso di dati Kinesis

Il flusso di dati Kinesis è supportato in due modalità di capacità: modalità assegnata e modalità on demand.

- Se il carico di lavoro che coinvolge il flusso di dati Kinesis presenta un traffico delle applicazioni prevedibile, un traffico coerente o che aumenta gradualmente o un traffico che può essere previsto con precisione, la modalità assegnata del flusso di dati Kinesis è quella adatta e sarà più efficiente in termini di costi
- Se il carico di lavoro è nuovo, presenta un traffico delle applicazioni imprevedibile o se preferisci non gestire la capacità, la modalità on demand del flusso di dati Kinesis è quella adatta e sarà più efficiente in termini di costi

Una best practice per ottimizzare i costi sarebbe quella di valutare se la tabella DynamoDB associata al flusso di dati Kinesis ha un modello di traffico prevedibile in grado di sfruttare la modalità assegnata del flusso di dati Kinesis. Se il carico di lavoro è nuovo, puoi utilizzare la modalità on-demand per Kinesis Data Streams per alcune settimane iniziali, rivedere le metriche per comprendere CloudWatch i modelli di traffico e quindi passare lo stesso Stream alla modalità provisioning in base alla natura del carico di lavoro. Nel caso della modalità assegnata, la stima delle partizioni può essere effettuata seguendo le considerazioni sulla gestione delle partizioni del flusso di dati Kinesis.

Valutazione delle applicazioni consumer che utilizzano il flusso di dati Kinesis per DynamoDB

Poiché il flusso di dati Kinesis non effettua addebiti in base al numero di chiamate `GetRecords` come i flussi DynamoDB, le applicazioni consumer potrebbero effettuare il maggior numero possibile di chiamate, a condizione che la frequenza sia inferiore alla limitazione della larghezza di banda della rete per `GetRecords`. In termini di modalità on demand per il flusso di dati Kinesis, le letture dei dati vengono addebitate in base ai GB. Per il flusso di dati Kinesis in modalità assegnata, le letture non vengono addebitate se i dati risalgono a meno di 7 giorni prima. Nel caso di funzioni Lambda come consumatori del flusso di dati Kinesis, Lambda esegue il polling per ogni partizione nel flusso Kinesis per i record a una velocità di base di una volta al secondo.

Strategie di ottimizzazione dei costi per entrambi i tipi di utilizzo dei flussi

Filtraggio degli eventi per i consumatori AWS Lambda

Il filtraggio degli eventi Lambda consente di eliminare gli eventi in base a un criterio di filtro rendendoli disponibili nel batch di richiamo della funzione Lambda. Ciò ottimizza i costi di Lambda per l'elaborazione o l'eliminazione di record di flusso indesiderati all'interno della logica delle funzioni consumatore. Per ulteriori informazioni sulla configurazione del filtro degli eventi e sulla scrittura dei criteri di filtro, consulta Filtro eventi Lambda.

Ottimizzazione dei consumatori AWS Lambda

I costi potrebbero essere ulteriormente ottimizzati con l'ottimizzazione dei parametri di configurazione di Lambda, ad esempio aumentando `BatchSize` per elaborare un numero maggiore di record per chiamata, abilitando `BisectBatchOnFunctionError` per impedire l'elaborazione di duplicati (che comporta costi aggiuntivi) e impostando `MaximumRetryAttempts` modo da non dover effettuare troppi tentativi. Per impostazione predefinita, le chiamate Lambda non riuscite dei consumatori vengono ritentate all'infinito fino alla scadenza del record dal flusso, che per i flussi DynamoDB è di circa 24 ore ed è configurabile da 24 ore fino a 1 anno per il flusso di dati Kinesis. Le opzioni di configurazione Lambda aggiuntive disponibili, comprese quelle indicate in precedenza per i consumatori del flusso DynamoDB, sono disponibili nella Guida per gli sviluppatori di AWS Lambda.

Valutazione della capacità allocata per un provisioning di dimensioni adeguate in una tabella DynamoDB

In questa sezione viene fornita una panoramica su come valutare se il provisioning è di dimensioni appropriate per le tabelle DynamoDB. Man mano che il carico di lavoro si evolve, è necessario modificare le procedure operative in modo appropriato, soprattutto quando la tabella DynamoDB è configurata in modalità assegnata e si corre il rischio di provisioning eccessivo o insufficiente delle tabelle.

Le procedure descritte di seguito richiedono informazioni statistiche che devono essere acquisite dalle tabelle DynamoDB che supportano l'applicazione di produzione. Per comprendere il comportamento dell'applicazione, è necessario definire un periodo di tempo sufficientemente significativo per acquisire la stagionalità dei dati dall'applicazione. Ad esempio, se l'applicazione mostra pattern settimanali, l'utilizzo di un periodo di tre settimane dovrebbe fornire spazio sufficiente per analizzare le esigenze di velocità di trasmissione effettiva dell'applicazione.

Se non sai da dove iniziare, utilizza almeno un mese di utilizzo dei dati per i calcoli seguenti.

Durante la valutazione della capacità, le tabelle DynamoDB possono configurare le unità di capacità di lettura RCUs () e le unità di capacità di scrittura (WCU) in modo indipendente. Se nelle tabelle sono configurati degli indici secondari globali (GSI), sarà necessario specificare il throughput che verrà utilizzato, che sarà anche indipendente dalla e dalla tabella di base. RCUs WCUs

> **Note**
>
> Gli indici secondari locali (LSI) utilizzano la capacità della tabella di base.

Argomenti

- Come recuperare le metriche di consumo nelle tabelle DynamoDB
- Come identificare le tabelle DynamoDB con un provisioning insufficiente
- Come identificare le tabelle DynamoDB con provisioning eccessivo

Come recuperare le metriche di consumo nelle tabelle DynamoDB

Per valutare la tabella e la capacità del GSI, monitora le seguenti CloudWatch metriche e seleziona la dimensione appropriata per recuperare le informazioni della tabella o del GSI:

unità di capacità in lettura	Unità di capacità in scrittura
`ConsumedReadCapacityUnits`	`ConsumedWriteCapacityUnits`
`ProvisionedReadCapacityUnits`	`ProvisionedWriteCapacityUnits`
`ReadThrottleEvents`	`WriteThrottleEvents`

È possibile eseguire questa operazione tramite o il. AWS CLI Console di gestione AWS

AWS CLI

Prima di recuperare le metriche di consumo della tabella, dovremo iniziare acquisendo alcuni punti dati storici utilizzando l'API. CloudWatch

Inizia creando due file: `write-calc.json` e `read-calc.json`. Questi file rappresenteranno i calcoli per una tabella o un GSI. Dovrai aggiornare alcuni campi, come indicato nella tabella seguente, in modo che corrispondano al tuo ambiente.

Nome campo	Definizione	Esempio
`<table-name>`	Il nome della tabella che verrà analizzata	SampleTable
`<period>`	Il periodo di tempo che utilizzerai per valutare il target di utilizzo, in secondi	Per un periodo di 1 ora è necessario specificare: 3600
`<start-time>`	L'inizio dell'intervallo di valutazione, specificato nel formato 01 ISO86	2022-02-21T23:00:00
`<end-time>`	La fine dell'intervallo di valutazione, specificata nel formato 01 ISO86	2022-02-22T06:00:00

Il file di calcolo delle scritture recupererà il numero di WCU assegnate e utilizzate nel periodo di tempo dell'intervallo di date specificato. Genererà inoltre una percentuale di utilizzo che verrà utilizzata per l'analisi. Il contenuto completo del file `write-calc.json` deve corrispondere a quanto segue:

```
{
  "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
}
```

Il file di calcolo delle letture utilizza un file simile. Questo file recupererà quanti ne RCUs sono stati forniti e consumati durante il periodo di tempo per l'intervallo di date specificato. Il contenuto del file `read-calc.json` deve corrispondere a quanto segue:

```
{
  "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 volta creati i file, è possibile iniziare a recuperare i dati di utilizzo.

1. Per recuperare i dati di utilizzo delle scritture, esegui il seguente comando:

   ```
   aws cloudwatch get-metric-data --cli-input-json file://write-calc.json
   ```

2. Per recuperare i dati di utilizzo delle letture, esegui il seguente comando:

   ```
   aws cloudwatch get-metric-data --cli-input-json file://read-calc.json
   ```

Il risultato di entrambe le query sarà una serie di punti dati in formato JSON che verranno utilizzati per l'analisi. Il risultato dipenderà dal numero di punti dati specificati, dal periodo e dai dati specifici del carico di lavoro. Potrebbe essere simile a quanto segue:

```
{
    "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**
>
> Se si specifica un periodo breve e un intervallo di tempo lungo, potrebbe essere necessario modificare il `MaxDatapoints` che, per impostazione predefinita, è impostato su 24 nello script. Ciò rappresenta un punto dati per ora e 24 al giorno.

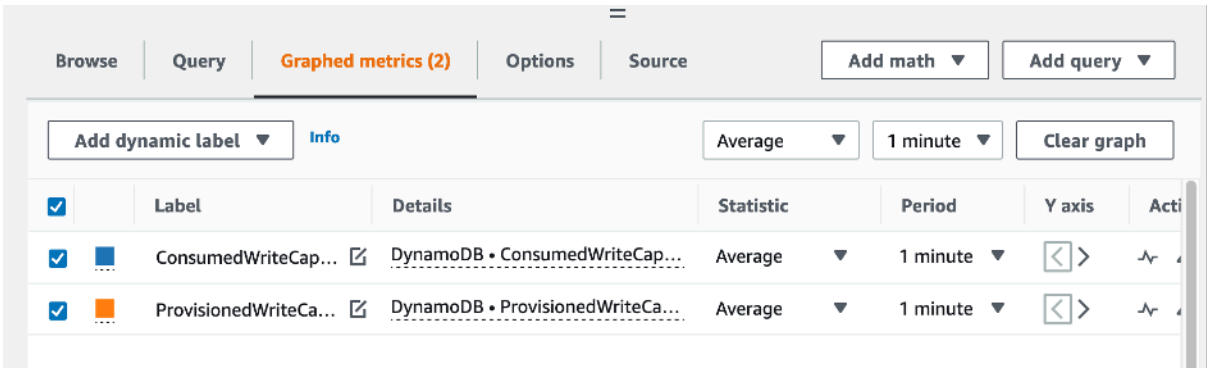
Console di gestione AWS

1. Accedi Console di gestione AWS e vai alla pagina del CloudWatch servizio. Seleziona un'opzione appropriata, Regione AWS se necessario.
2. Nella barra di navigazione a sinistra individua la sezione Metriche, quindi seleziona Tutte le metriche.
3. Verrà aperto un pannello di controllo con due pannelli. Il pannello superiore mostrerà il grafico e il pannello inferiore conterrà le metriche che si desidera visualizzare nel grafico. Seleziona DynamoDB.
4. Seleziona Metriche. Questa sezione mostrerà le tabelle relative alla regione corrente.

5. Usa la casella di ricerca per cercare il nome della tabella e seleziona le metriche dell'operazione di scrittura: `ConsumedWriteCapacityUnits` e `ProvisionedWriteCapacityUnits`

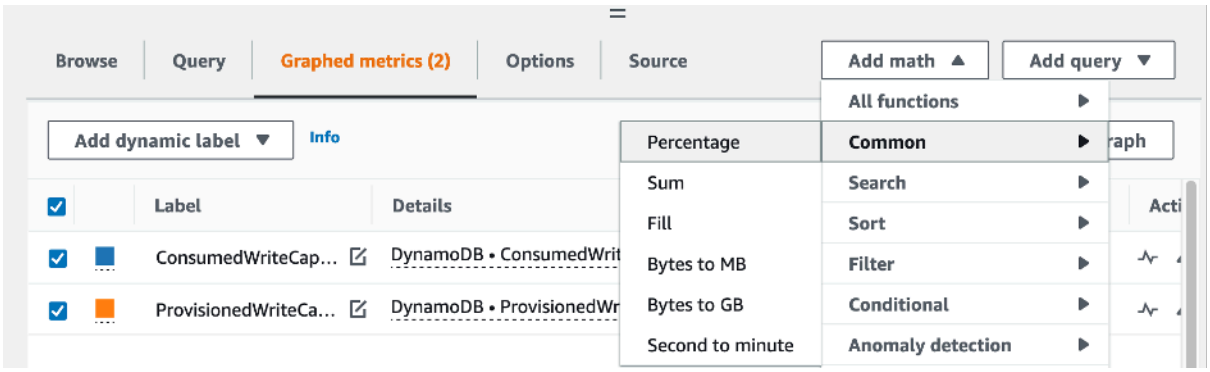
> **Note**
>
> Questo esempio illustra le metriche delle operazioni di scrittura, ma puoi anche utilizzare questi passaggi per visualizzare graficamente le metriche delle operazioni di lettura.

6. Seleziona la scheda Parametri definiti (2) per modificare le formule. Per impostazione predefinita, CloudWatch seleziona la funzione statistica Average per i grafici.

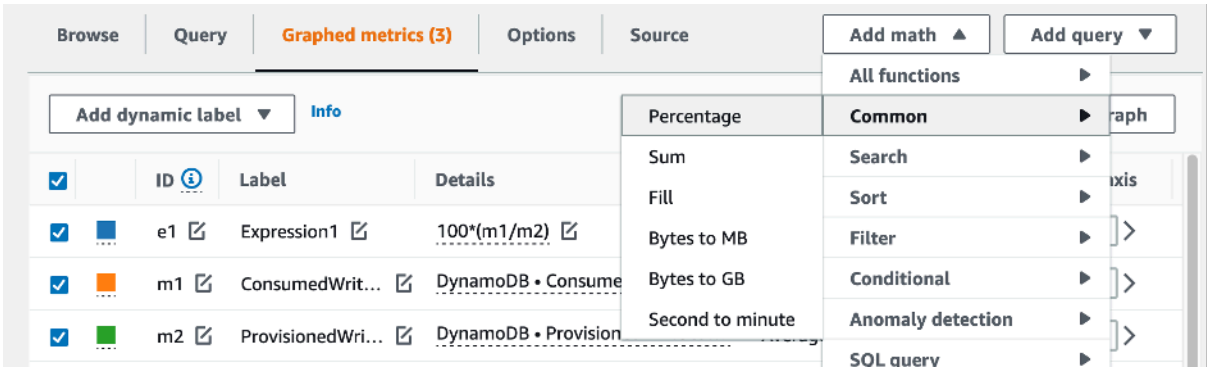


7. Dopo aver selezionato entrambe le metriche nel grafico (la casella di controllo a sinistra), selezionare il menu Add math (Aggiungi matematica), seguito da Common (Comune), quindi selezionare la funzione Percentage (Percentuale). Ripetere la procedura due volte.

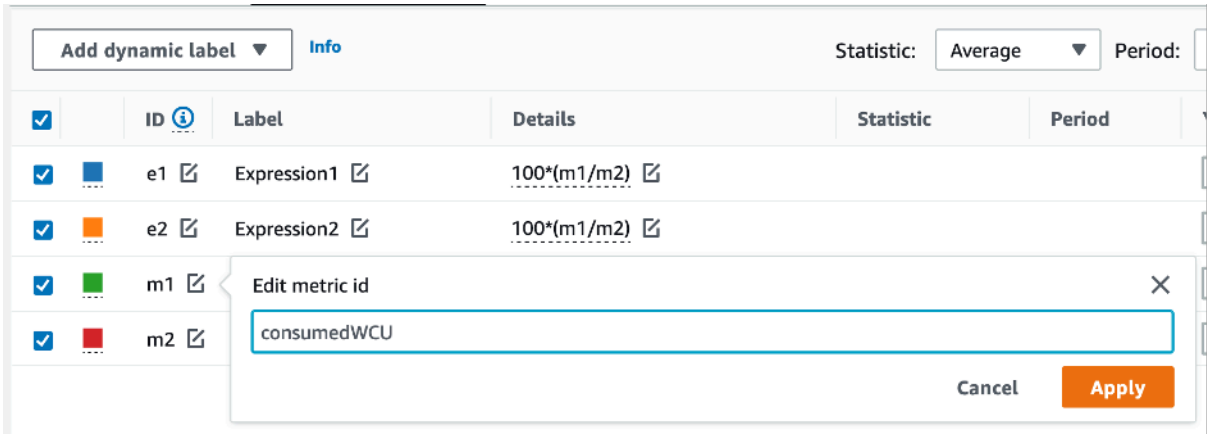
   La prima volta selezionando la funzione Percentage (Percentuale):



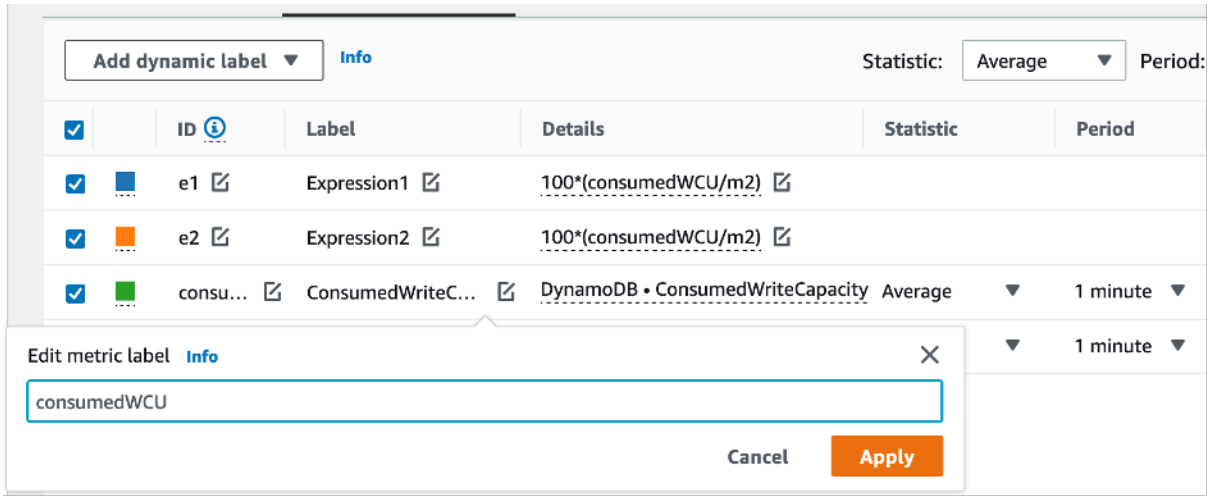
   La seconda volta selezionando la funzione Percentage (Percentuale):



8. A questo punto dovresti avere quattro metriche nel menu in basso. Lavoriamo sul calcolo `ConsumedWriteCapacityUnits`. Per essere coerenti, dobbiamo abbinare i nomi a quelli che abbiamo usato nella AWS CLI sezione. Fare clic su m1 ID e modificare questo valore in consumedWCU.

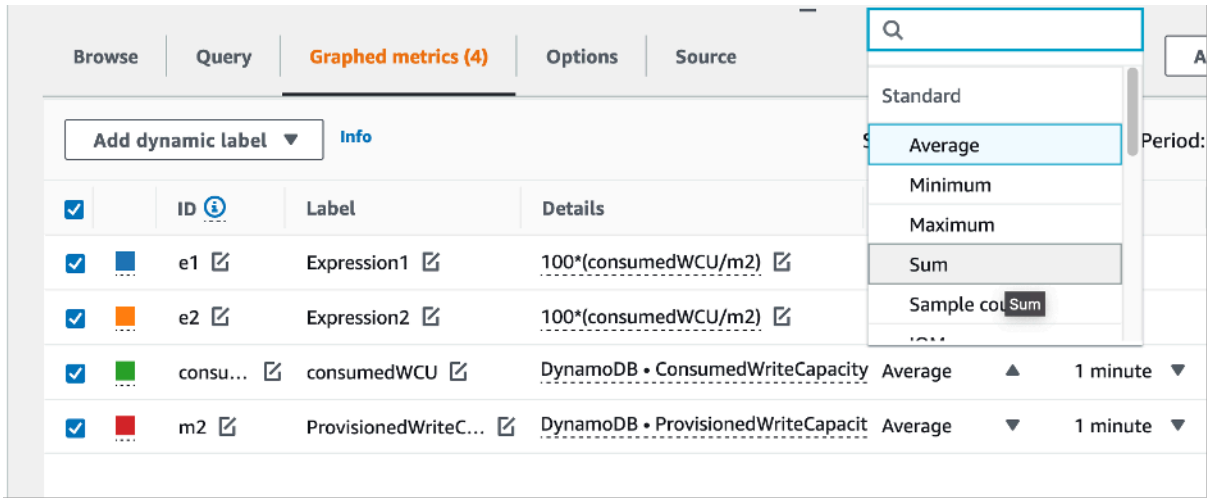


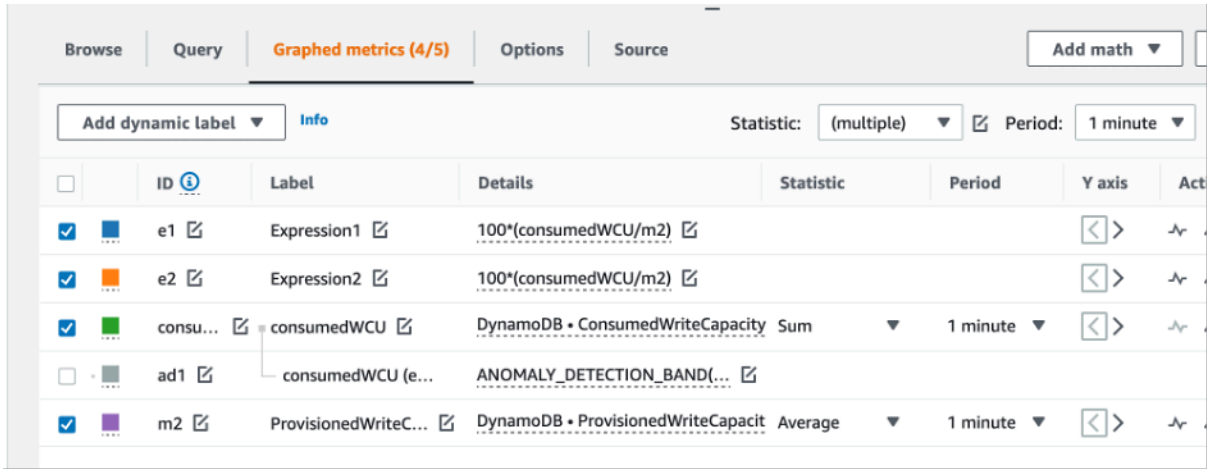
Rinomina l'ConsumedWriteCapacityUnitetichetta come**`consumedWCU`**.



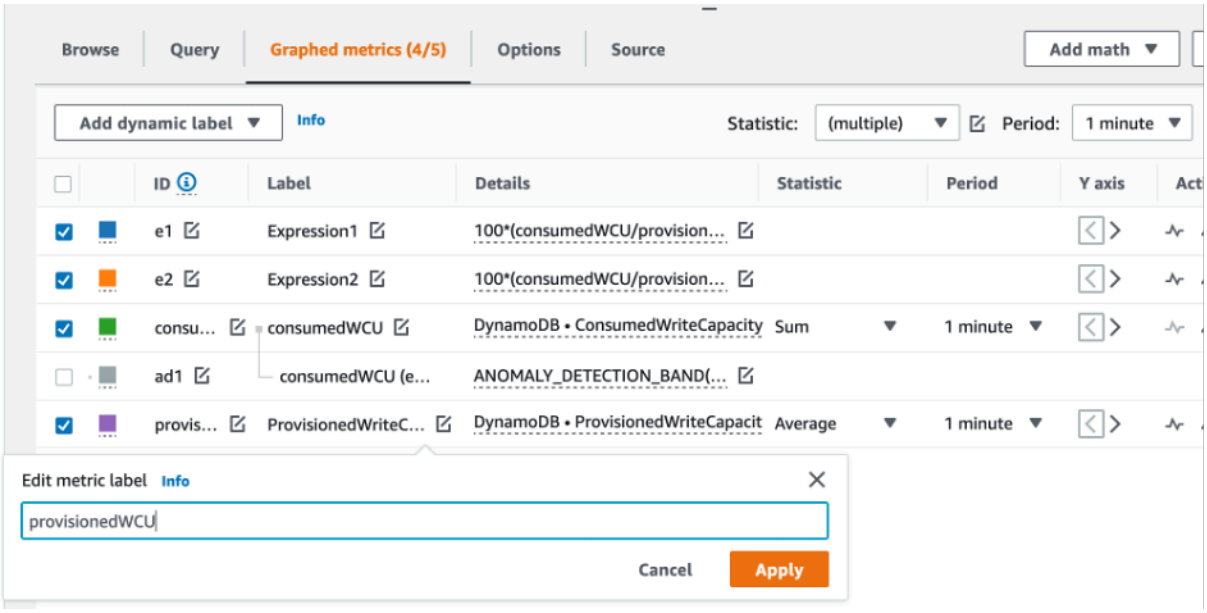
9. Modifica della statistica da Average (Media) a Sum (Somma). Questa operazione creerà automaticamente un'altra metrica denominata ANOMALY_DETECTION_BAND. Per quanto

riguarda l'ambito di questa procedura, lo si ignora rimuovendo la casella di spunta su metrica ad1 appena generata.

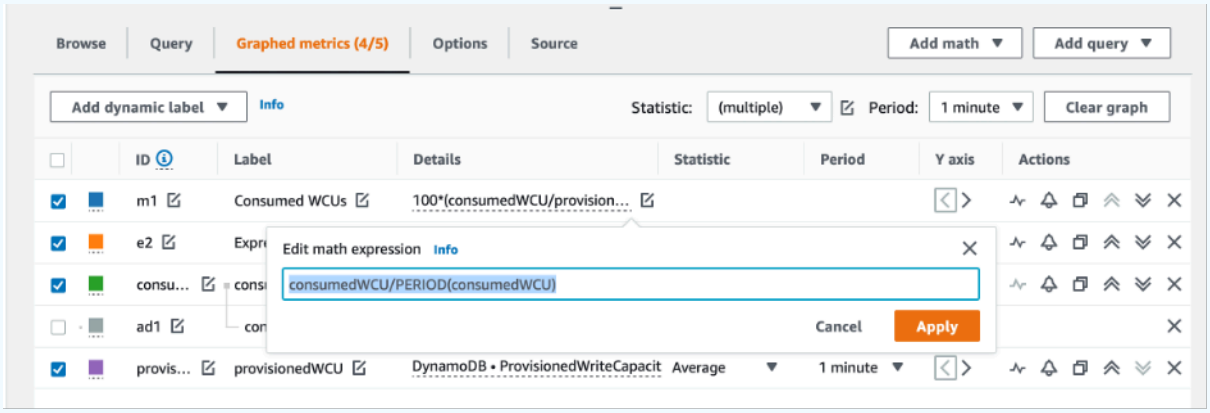




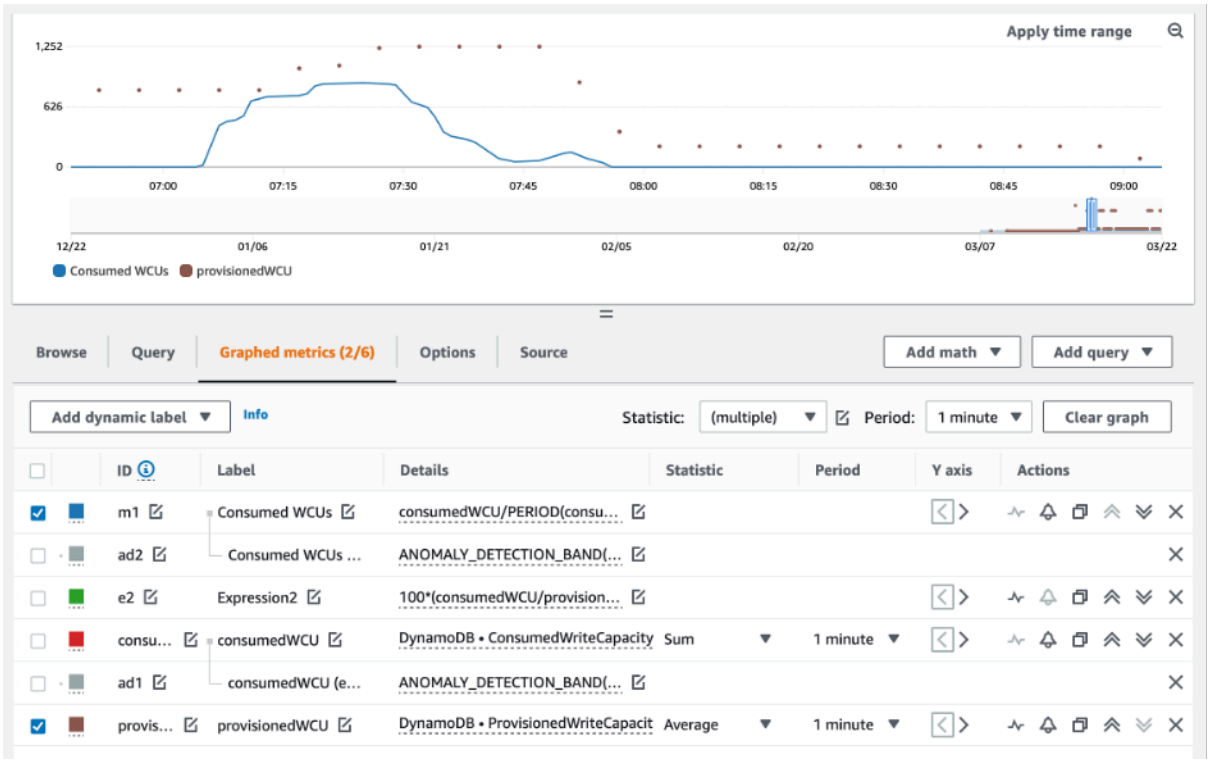
10. Ripetere il passaggio 8 per rinominare m2 ID in provisionedWCU. Lasciare la statistica impostata su Average (Media).



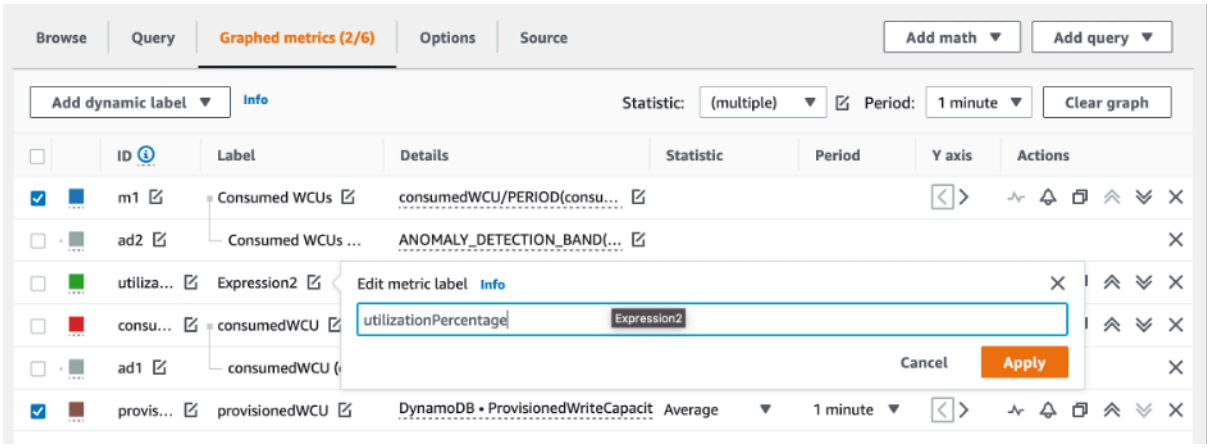
11. Seleziona l'etichetta Expression1 e aggiorna il valore su m1 e l'etichetta su Consumed. WCUs

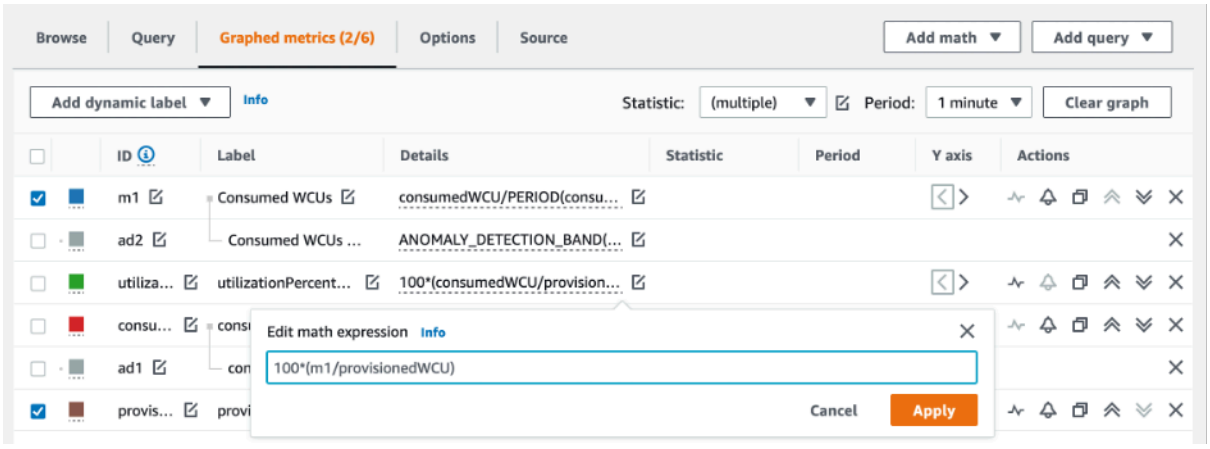
> **Note**
>
> Assicurati di aver selezionato solo m1 (casella di controllo a sinistra) e provisionedWCU per visualizzare correttamente i dati. Aggiorna la formula facendo clic su Details (Dettagli) e modificando la formula in consumedWCU/PERIOD(consumedWCU). Questo passaggio potrebbe anche generare un'altra metrica ANOMALY_DETECTION_BAND, ma per l'ambito di questa procedura possiamo ignorarla.
>
> 
>

12. Ora dovresti avere due grafici: uno che indica il tuo approvvigionamento WCUs sulla tabella e l'altro che indica il consumo. WCUs La forma del grafico potrebbe essere diversa da quella riportata di seguito, ma puoi usarla come riferimento:



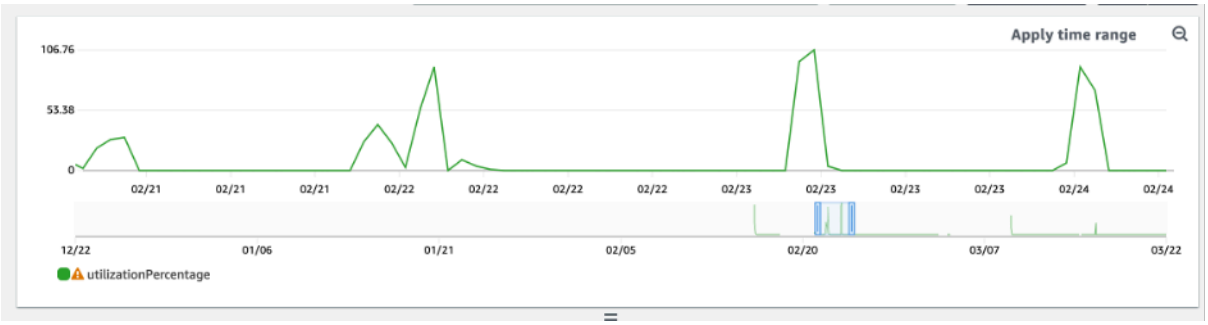
13. Aggiornare la formula percentuale selezionando il grafico Expression2 (e2). Rinomina le etichette e impostale su UtilizationPercentage. IDs Rinominare la formula in modo che corrisponda a 100*(m1/provisionedWCU).





14. Rimuovere la casella di controllo da tutte le metriche tranne utilizationPercentage per visualizzare i pattern di utilizzo. L'intervallo predefinito è impostato su 1 minuto, ma è possibile modificarlo in base alle proprie esigenze.

Ecco una vista di un periodo di tempo più lungo e di un periodo maggiore di 1 ora. Puoi vedere che ci sono alcuni intervalli in cui l'utilizzo era superiore al 100%, ma questo particolare carico di lavoro ha intervalli più lunghi con utilizzo pari a zero.



A questo punto, potresti avere risultati diversi dalle immagini di questo esempio. Tutto dipende dai dati del carico di lavoro. Gli intervalli con un utilizzo superiore al 100% sono soggetti a eventi di limitazione della larghezza di banda della rete. DynamoDB offre capacità di espansione, ma non appena viene raggiunta la capacità di espansione, qualsiasi valore superiore al 100% verrà limitato.

Come identificare le tabelle DynamoDB con un provisioning insufficiente

Per la maggior parte dei carichi di lavoro, una tabella è considerata con provisioning insufficiente quando consuma costantemente più dell'80% della capacità assegnata.

La capacità burst è una funzionalità di DynamoDB che consente ai clienti di consumare temporaneamente RCUs/WCUs più di quanto originariamente fornito (più del throughput al secondo assegnato definito nella tabella). La capacità di espansione è stata creata per assorbire improvvisi aumenti di traffico dovuti a eventi speciali o picchi di utilizzo. Questa capacità di espansione ha una durata limitata. Non appena le risorse inutilizzate RCUs WCUs sono esaurite, se si tenta di consumare più capacità di quella fornita, si rischia di subire una limitazione. Quando il traffico delle applicazioni si avvicina al tasso di utilizzo dell'80%, il rischio di limitazione della larghezza di banda della rete è notevolmente maggiore.

La regola del tasso di utilizzo dell'80% varia in base alla stagionalità dei dati e alla crescita del traffico. Considerare i seguenti scenari:

- Se il traffico è rimasto stabile a un tasso di utilizzo di circa il 90% negli ultimi 12 mesi, la tabella ha la capacità corretta
- Se il traffico delle applicazioni aumenta a un tasso dell'8% mensile in meno di 3 mesi, si arriverà al 100%
- Se il traffico delle applicazioni aumenta a un tasso dell'5% mensile in meno di 4 mesi, si arriverà al 100%

I risultati delle query precedenti forniscono un'immagine del tasso di utilizzo. Utilizzali come guida per valutare ulteriormente altre metriche che possono aiutarti a scegliere di aumentare la capacità della tabella in base alle esigenze (ad esempio: un tasso di crescita mensile o settimanale). Collabora con il tuo team operativo per definire qual è una buona percentuale per il carico di lavoro e le tabelle.

Esistono scenari speciali in cui i dati non sono affidabili quando li analizziamo su base giornaliera o settimanale. Ad esempio, con le applicazioni stagionali che presentano picchi di utilizzo durante l'orario di lavoro (ma poi scendono quasi a zero al di fuori dell'orario di lavoro), si potrebbe trarre vantaggio dalla pianificazione del dimensionamento automatico in cui si specificano le ore del giorno (e i giorni della settimana) in cui aumentare la capacità allocata e quando ridurla. Invece di puntare a una maggiore capacità per coprire le ore di punta, è possibile anche trarre vantaggio dalle configurazioni del dimensionamento automatico delle tabelle DynamoDB se la stagionalità è meno marcata.

> **Note**
>
> Quando crei una configurazione del dimensionamento automatico di DynamoDB per la tabella di base, ricordati di includere un'altra configurazione per tutti i GSI associati alla tabella.

Come identificare le tabelle DynamoDB con provisioning eccessivo

I risultati delle query ottenuti dagli script precedenti forniscono i dati necessari per eseguire alcune analisi iniziali. Se il set di dati presenta valori di utilizzo inferiori al 20% per diversi intervalli, è possibile che la tabella presenti un provisioning eccessivo. Per definire ulteriormente se è necessario ridurre il numero di WCUs e RCUS, è necessario rivedere le altre letture negli intervalli.

Quando le tabelle contengono diversi intervalli di basso utilizzo, è possibile davvero trarre vantaggio dall'utilizzo di policy di dimensionamento automatico, pianificandolo o semplicemente configurando policy di dimensionamento automatico predefinite per la tabella in base all'utilizzo.

Se avete un carico di lavoro con un basso utilizzo e un rapporto accelerazione elevato (Max (ThrottleEvents) /Min () nell'intervalloThrottleEvents), ciò potrebbe accadere quando avete un carico di lavoro molto intenso in cui il traffico aumenta notevolmente nel corso di alcuni giorni (o ore), ma in generale il traffico è costantemente basso. In questi scenari potrebbe essere utile utilizzare il dimensionamento automatico pianificato.

AWS Well-Architected Framework aiuta gli architetti del cloud a creare un'infrastruttura sicura, ad alte prestazioni, resiliente ed efficiente per una varietà di applicazioni e carichi di lavoro. Costruito attorno a sei pilastri: eccellenza operativa, sicurezza, affidabilità, efficienza delle prestazioni, ottimizzazione dei costi e sostenibilità,AWS Well-Architected offre a clienti e partner un approccio coerente per valutare le architetture e implementare progetti scalabili.

Le AWS Well-Architected Lenses estendono la guida offerta da AWS Well-Architected a specifici settori industriali e tecnologici. Amazon DynamoDB Well-Architected Lens si concentra sui carichi di lavoro DynamoDB. Fornisce le best practice, i principi di progettazione e le domande per valutare ed esaminare un carico di lavoro DynamoDB. Il completamento di un valutazione Amazon DynamoDB Well-Architected Lens ti fornirà formazione e indicazioni sui principi di progettazione consigliati in relazione a ciascuno dei pilastri di AWS Well-Architected. Questa guida si basa sulla nostra esperienza di lavoro con clienti in vari settori, segmenti, dimensioni e aree geografiche.

Come risultato diretto della valutazione di Well-Architected Lens, riceverai un riepilogo dei suggerimenti attuabili per ottimizzare e migliorare il tuo carico di lavoro DynamoDB.

Esecuzione della valutazione di Amazon DynamoDB Well-Architected Lens

La revisione di DynamoDB Well-Architected Lens viene solitamente eseguita AWS da un Solutions Architect insieme al cliente, ma può anche essere eseguita dal cliente come self-service. Sebbene consigliamo di esaminare tutti e sei i pilastri Well-Architected come parte di Amazon DynamoDB Well-Architected Lens, puoi anche decidere di concentrarti per prima cosa su uno o più pilastri.

Informazioni e istruzioni aggiuntive per condurre una revisione di Amazon DynamoDB Well-Architected Lens sono disponibili in questo video e nella pagina DynamoDB Well-Architected Lens. GitHub

I pilastri di Amazon DynamoDB Well-Architected Lens

Amazon DynamoDB Well-Architected Lens è basato su sei pilastri:

Pilastro dell'efficienza delle prestazioni

Il pilastro dell'efficienza delle prestazioni include la capacità di utilizzare in modo efficiente le risorse di calcolo per soddisfare i requisiti di sistema e di mantenere tale efficienza di fronte al cambiamento delle richieste e all'evoluzione delle tecnologie.

I principi di progettazione principali di DynamoDB per questo pilastro ruotano attorno alla modellazione dei dati, alla scelta delle chiavi di partizione e delle chiavi di ordinamento e alla definizione di indici secondari in base ai modelli di accesso alle applicazioni. Altre considerazioni includono la scelta della modalità di throughput ottimale per il carico di lavoro, l'ottimizzazione dell' AWS SDK e, se del caso, l'utilizzo di una strategia di caching ottimale. Per ulteriori informazioni su questi principi di progettazione, guarda questo video di approfondimento sul pilastro dell'efficienza delle prestazioni di DynamoDB Well-Architected Lens.

Pilastro dell'ottimizzazione dei costi

Il pilastro dell'ottimizzazione dei costi si concentra sull'evitare costi inutili.

Gli argomenti chiave includono la comprensione e il controllo di dove vengono spesi i soldi, la selezione del numero più appropriato e corretto di tipi di risorse, l'analisi della spesa nel corso del tempo, la progettazione di modelli di dati per ottimizzare i costi dei modelli di accesso specifici delle applicazioni e la scalabilità per soddisfare le esigenze aziendali senza spese eccessive.

I principi chiave di progettazione dell'ottimizzazione dei costi per DynamoDB ruotano attorno alla scelta della modalità di capacità e della classe di tabella più appropriate per le tabelle ed evitare l'over-provisioning di capacità utilizzando la modalità di capacità on demand o la modalità di capacità assegnata con dimensionamento automatico. Altre considerazioni includono la modellazione efficiente dei dati e query per ridurre la quantità di capacità utilizzata, la prenotazione di porzioni della capacità utilizzata a prezzi scontati, la riduzione al minimo delle dimensioni degli elementi, l'identificazione e la rimozione delle risorse inutilizzate e l'utilizzo del TTL per eliminare automaticamente i dati obsoleti senza alcun costo. Per ulteriori informazioni su questi principi di progettazione, guarda questo video di approfondimento sul pilastro dell'ottimizzazione dei costi di DynamoDB Well-Architected Lens.

Consulta Ottimizzazione dei costi per ulteriori informazioni sulle best practice di ottimizzazione dei costi per DynamoDB.

Il pilastro dell'eccellenza operativa

Il pilastro dell'eccellenza operativa si concentra sull'esecuzione e sul monitoraggio dei sistemi per fornire valore aziendale e migliorare continuamente processi e procedure. Gli argomenti chiave includono l'automazione delle modifiche, la risposta agli eventi e la definizione degli standard per gestire le operazioni quotidiane.

I principali principi di progettazione di eccellenza operativa per DynamoDB includono il monitoraggio delle metriche di DynamoDB tramite CloudWatch AWS Config Amazon e avvisi e rimedi automatici quando vengono violate soglie predefinite o vengono rilevate regole non conformi. Ulteriori considerazioni riguardano la definizione delle risorse DynamoDB tramite l'infrastruttura come codice e l'utilizzo dei tag per una migliore organizzazione, identificazione e contabilità dei costi delle risorse DynamoDB. Per ulteriori informazioni su questi principi di progettazione, guarda questo video di approfondimento sul pilastro dell'eccellenza operativa di DynamoDB Well-Architected Lens.

Pilastro dell'affidabilità

Il pilastro dell'affidabilità si concentra sull'assicurare che un carico di lavoro svolga la funzione prevista in modo corretto e coerente quando previsto. Un carico di lavoro resiliente recupera rapidamente dai guasti per soddisfare le esigenze aziendali e dei clienti. Gli argomenti principali includono la progettazione di sistemi distribuiti, la pianificazione del ripristino e la gestione delle modifiche.

I principi essenziali di progettazione dell'affidabilità per DynamoDB ruotano attorno alla scelta della strategia di backup e della conservazione in base ai requisiti RPO e RTO, all'utilizzo di tabelle

globali DynamoDB per carichi di lavoro multiregionali o scenari di disaster recovery tra regioni con RTO basso, all'implementazione della logica di riprova con backoff esponenziale nell'applicazione configurando e utilizzando queste funzionalità nell'SDK AWS e al monitoraggio delle metriche DynamoDB tramite Amazon e automaticamente avvisare e porre rimedio in caso di superamento di soglie predefinite. CloudWatch Per ulteriori informazioni su questi principi di progettazione, guarda questo video di approfondimento sul pilastro dell'affidabilità di DynamoDB Well-Architected Lens.

Pilastro della sicurezza

Il pilastro della sicurezza si concentra sulla protezione delle informazioni e dei sistemi. Gli argomenti chiave includono la riservatezza e l'integrità dei dati, l'identificazione e la gestione di chi può fare cosa mediante la gestione dei privilegi, la protezione dei sistemi e l'istituzione di controlli per rilevare gli eventi di sicurezza.

I principi di progettazione principali della sicurezza per DynamoDB sono la crittografia dei dati in transito con HTTPS, la scelta del tipo di chiavi per la crittografia dei dati a riposo e la definizione dei ruoli e delle policy IAM per autenticare, autorizzare e fornire un accesso granulare alle risorse DynamoDB. Altre considerazioni includono il controllo delle operazioni del piano di controllo e del piano dati di DynamoDB tramite. AWS CloudTrail Per ulteriori informazioni su questi principi di progettazione, guarda questo video di approfondimento sul pilastro della sicurezza di DynamoDB Well-Architected Lens.

Per ulteriori informazioni sulla sicurezza di DynamoDB, consulta Security (Sicurezza).

Pilastro della sostenibilità

Il pilastro della sostenibilità si concentra sulla riduzione al minimo degli impatti ambientali dell'esecuzione dei carichi di lavoro nel cloud. Gli argomenti chiave includono un modello di responsabilità condivisa per la sostenibilità, la comprensione dell'impatto e la massimizzazione dell'utilizzo per ridurre al minimo le risorse necessarie e ridurre gli impatti a valle.

I principi di progettazione principali della sostenibilità per DynamoDB includono l'identificazione e la rimozione delle risorse DynamoDB inutilizzate, l'evitare l'over-provisioning tramite l'uso della modalità di capacità on demand o della modalità di capacità assegnata con il dimensionamento automatico, le query efficienti per ridurre la quantità di capacità consumata e la riduzione del footprint di archiviazione comprimendo i dati ed eliminando i dati obsoleti tramite l'uso del TTL. Per ulteriori informazioni su questi principi di progettazione, guarda questo video di approfondimento sul pilastro della sostenibilità di DynamoDB Well-Architected Lens.

Best practice per la progettazione delle chiavi di partizione e relativo uso in DynamoDB

La chiave primaria che identifica in modo univoco ciascun elemento in una tabella Amazon DynamoDB può essere semplice (solo una chiave di partizione) o composita (una chiave di partizione combinata con una chiave di ordinamento).

Sarebbe opportuno progettare la propria applicazione affinché svolga attività uniformi tra tutte le chiavi di partizione nella tabella e nei suoi indici secondari. Puoi determinare i modelli di accesso necessari alla tua applicazione, nonché le unità di capacità in lettura (RCU) e scrittura (WCU) necessarie a ogni tabella e indice secondario.

> **Note**
>
> La capacità adattiva si applica alla modalità on demand e alla capacità allocata.

Ogni partizione in una tabella DynamoDB è progettata per fornire una capacità massima di 3.000 unità di lettura al secondo e 1.000 unità di scrittura al secondo. Un'unità di capacità di lettura rappresenta un'operazione a elevata consistenza di lettura al secondo o due operazioni di lettura a coerenza finale al secondo, per un elemento di dimensioni fino a 4 KB. Un'unità di scrittura rappresenta un'operazione di scrittura per un elemento di dimensioni fino a 1 KB.

È necessario tenere conto della dimensione dell'elemento quando si valutano i limiti di throughput delle partizioni per la tabella. Ad esempio, se la tabella ha una dimensione dell'elemento di 20 KB, una singola operazione a consistenza di lettura consumerà 5 unità di lettura. Ciò significa che è possibile eseguire contemporaneamente 600 operazioni a consistenza di lettura al secondo su quel singolo elemento prima di raggiungere i limiti di partizione. Il throughput totale su tutte le partizioni nella tabella può essere limitato dal throughput allocato in modalità con provisioning o dal limite del throughput a livello di tabella in modalità on demand. Per ulteriori informazioni, consulta Service Quotas.

Argomenti

- Progettazione delle chiavi di partizione per distribuire il carico di lavoro in DynamoDB
- Utilizzo dello sharding di scrittura per distribuire i carichi di lavoro in modo uniforme in una tabella DynamoDB
- Distribuzione efficiente dell'attività di scrittura durante il caricamento dei dati in DynamoDB

Progettazione delle chiavi di partizione per distribuire il carico di lavoro in DynamoDB

La porzione della chiave di partizione della chiave primaria di una tabella determina le partizioni logiche in cui sono archiviati i dati di una tabella. Questo a sua volta influisce sulle partizioni fisiche sottostanti. Una progettazione di chiavi di partizione che non distribuisce le I/O richieste in modo efficace può creare partizioni «calde» che comportano limitazioni e utilizzano la capacità assegnata in modo inefficiente. I/O

L'uso ottimale del throughput assegnato a una tabella dipende non solo dai modelli di carico di lavoro dei singoli item, ma anche dal modo in cui vengono progettate le chiavi di partizione. Ciò non significa che è necessario accedere a tutti i valori delle chiavi di partizione per raggiungere un livello efficiente di throughput, né significa che la percentuale dei valori delle chiavi di partizione accessibili debba essere elevata. Significa che più sono i valori distinti delle chiavi di partizione a cui può accedere il carico di lavoro e più tali richieste verranno distribuite nello spazio partizionato. In generale, si utilizzerà il throughput allocato in modo più efficiente in quanto il rapporto tra i valori delle chiavi di partizione a cui accede il numero totale di valori delle chiavi di partizione in una tabella aumenta.

Di seguito è riportato un confronto dell'efficienza del throughput assegnato di alcuni schemi di chiavi di partizione comuni.

Valore della chiave di partizione	Uniformità
ID utente, se l'applicazione ha molti utenti.	Buona
Codice di stato, se ci sono solo alcuni codici di stato possibili.	Non buona
Data di creazione dell'item, arrotondata al periodo di tempo più vicino (ad esempio giorno, ora o minuto).	Non buona
ID dispositivo, se ogni dispositivo accede ai dati a intervalli relativamente simili.	Buona
ID dispositivo, se anche quando ci sono molti dispositivi tracciati, uno è di gran lunga più popolare di tutti gli altri.	Non buona

Se una singola tabella ha solo un numero ridotto di valori delle chiavi di partizione, considera la possibilità di distribuire le operazioni di scrittura su più valori delle chiavi di partizione distinti. In altre parole, struttura gli elementi chiave primari per evitare un valore della chiave di partizione "hot" (fortemente richiesto) che rallenti le prestazioni generali.

Ad esempio, considera una tabella con una chiave primaria composita. La chiave di partizione rappresenta la data di creazione dell'item, arrotondata al giorno più vicino. La chiave di ordinamento è l'identificatore di un item. In un dato giorno, ad esempio il `2014-07-09`, tutti i nuovi item vengono scritti nel valore singolo della chiave di partizione (e nella partizione fisica corrispondente).

Se la tabella si adatta interamente a una singola partizione (prendendo in considerazione la crescita dei dati nel tempo) e se i requisiti di throughput in lettura e scrittura dell'applicazione non superano le capacità di lettura e scrittura di una singola partizione, l'applicazione non rileverà alcun throttling imprevisto a causa del partizionamento.

Per utilizzare NoSQL Workbench per DynamoDB per aiutarti a visualizzare la progettazione delle chiavi di partizione, consulta Creazione di modelli di dati con NoSQL Workbench.

Utilizzo dello sharding di scrittura per distribuire i carichi di lavoro in modo uniforme in una tabella DynamoDB

Un modo per distribuire scritture al meglio nello spazio della chiave di partizione in Amazon DynamoDB consiste nell'espandere lo spazio. Questa operazione può essere eseguita in modi diversi. Puoi aggiungere un numero casuale ai valori della chiave di partizione per distribuire gli elementi tra le partizioni. In alternativa, puoi usare un numero calcolato in base al risultato di una query.

Partizionamento utilizzando suffissi casuali

Una strategia per distribuire i carichi in maniera più uniforme nello spazio della chiave di partizione è di aggiungere un numero casuale alla fine dei valori delle chiavi di partizioni. In seguito scegli in modo casuale le scritture nello spazio di dimensioni maggiori.

Ad esempio, per una chiave di partizioni che rappresenta la data di oggi, puoi scegliere un numero casuale tra `1` e `200` e concatenarlo alla data come suffisso. Ciò produce valori delle chiavi di partizione come `2014-07-09.1`, `2014-07-09.2` e così via fino a `2014-07-09.200`. In quanto stai scegliendo in modo casuale la chiave di partizione, le scritture alla tabella ogni giorno vengono distribuite in modo uniforme tra le partizioni multiple. Ciò risulta in parallelismo migliore e throughput complessivo maggiore.

Comunque, per leggere tutte le voci per un determinato giorno, dovresti eseguire una query su tutti i suffissi delle voci e in seguito unire i risultati. Ad esempio, per prima cosa devi inviare una richiesta `Query` per il valore della chiave di partizione `2014-07-09.1`. Quindi emettere un'altra `Query` per `2014-07-09.2` e così via fino a `2014-07-09.200`. Infine, la tua applicazione dovrà unire i risultati di tutte quelle richieste `Query`.

Partizionamento utilizzando suffissi calcolati

Una strategia di scelta casuale può migliorare molto il throughput di scrittura. Tuttavia è difficile leggere una voce specifica perché non si sa quale valore del suffisso è stato utilizzato durante la scrittura della voce. Per facilitare la lettura delle voci individuali, puoi utilizzare una strategia diversa. Invece di utilizzare un numero casuale per distribuire le voci tra le partizioni, utilizza un numero che puoi calcolare in base a qualcosa sul quale desideri effettuare una query.

Guarda l'esempio precedente, dove una tabella utilizza la giornata di oggi nella chiave di partizione. Presumi che ogni voce abbia un attributo `OrderId` accessibile e che molto spesso devi trovare voci per ID dell'ordine oltre alla data. Prima che l'applicazione scriva l'item nella tabella, potrebbe calcolare il suffisso hash in base all'ID dell'ordine e aggiungerlo alla data della chiave di partizione. Il calcolo potrebbe dare come risultato un numero compreso tra 1 e 200 distribuito in modo abbastanza uniforme, in maniera simile a cosa viene prodotto dalla strategia casuale.

Sarebbe sufficiente un semplice calcolo, ad esempio il prodotto dei valori del punto di codice UTF-8 per i caratteri nell'ID ordine, modulo 200 + 1. Il valore della chiave di partizione sarebbe quindi la data concatenata con il risultato del calcolo.

Con questa strategia, le scritture vengono distribuite in modo uniforme tra i valori delle chiavi di partizione e quindi tra le partizioni fisiche. Puoi eseguire facilmente un'operazione `GetItem` su una voce e data particolari, in quanto puoi calcolare il valore della chiave di partizione per un valore `OrderId` specifico.

Per leggere tutti gli elementi per un determinato giorno, è comunque necessario eseguire una `Query`per ciascuna delle chiavi `2014-07-09.N` (dove `N` è 1-200) e l'applicazione deve quindi unire tutti i risultati. Il beneficio è che eviterai che un singolo valore di chiave di partizione "hot" prenda tutto il carico di lavoro.

> **Note**
>
> Per una strategia più efficace progettata in maniera specifica per gestire volumi elevati di dati di serie temporali, consulta Dati di serie temporali.

Distribuzione efficiente dell'attività di scrittura durante il caricamento dei dati in DynamoDB

In genere, quando vengono caricati i dati da altre origini dati, Amazon DynamoDB partiziona i dati della tabella su più server. È possibile ottenere prestazioni migliori se si caricano i dati su tutti i server allocati contemporaneamente.

Ad esempio, si supponga di voler caricare i messaggi utente in una tabella DynamoDB che utilizza una chiave primaria composita con `UserID` come chiave di partizione e `MessageID` come chiave di ordinamento.

Quando vengono caricati i dati, un approccio da adottare è quello di caricare tutti gli elementi del messaggio per ogni utente, un utente dopo l'altro:

UserID	MessageID
U1	1
U1	2
U1	...
U1	... fino a 100
U2	1
U2	2
U2	...
U2	... fino a 200

Il problema in questo caso è che non vengono distribuite le richieste di scrittura a DynamoDB tra i valori della chiave di partizione. Si prende un valore di chiave di partizione alla volta e se ne caricano tutti gli elementi prima di passare al valore della chiave di partizione successivo e fare lo stesso.

Dietro le quinte, DynamoDB esegue il partizionamento dei dati nella tabella su più server. Per utilizzare completamente tutta la capacità di velocità effettiva assegnata per la tabella, è necessario distribuire il carico di lavoro tra i valori della chiave di partizione. Dirigendo una quantità irregolare di

lavoro di caricamento verso elementi che hanno tutti lo stesso valore della chiave di partizione, non si utilizzano completamente tutte le risorse assegnate da DynamoDB alla tabella.

È possibile distribuire il lavoro di caricamento utilizzando la chiave di ordinamento per caricare un elemento da ogni valore di chiave di partizione, quindi un altro elemento da ogni valore di chiave di partizione e così via:

UserID	MessageID
U1	1
U2	1
U3	1
...	...
U1	2
U2	2
U3	2
...	...

Ogni caricamento in questa sequenza utilizza un valore di chiave di partizione diverso, mantenendo contemporaneamente più server DynamoDB e migliorando le prestazioni di velocità effettiva.

Best practice per l’uso delle chiavi di ordinamento per organizzare i dati in DynamoDB

In una tabella Amazon DynamoDB la chiave primaria che identifica in modo univoco ciascun elemento della tabella può essere composta da una chiave di partizione e da una chiave di ordinamento.

Le chiavi di ordinamento progettate adeguatamente offrono due vantaggi principali:

- Raccolgono le informazioni correlate in un'unica posizione in cui possono essere interrogate in modo efficiente. Una progettazione attenta della chiave di ordinamento ti permette di recuperare i

gruppi di item correlati comunemente necessari utilizzando le query in un intervallo con operatori come `begins_with`, `between`, >, < e così via.

- Le chiavi di ordinamento composite consentono di definire relazioni gerarchiche (one-to-many) nei dati su cui è possibile interrogare a qualsiasi livello della gerarchia.

  Ad esempio, in una tabella che elenca delle posizioni geografiche, potresti strutturare la chiave di ordinamento come segue.

  ```
  [country]#[region]#[state]#[county]#[city]#[neighborhood]
  ```

  In questo modo potresti eseguire query di intervallo efficienti per un elenco di posizioni a uno qualsiasi di questi livelli di aggregazione, da `country` fino a `neighborhood`, interrogando qualsiasi item tra questi valori.

Uso delle chiavi di ordinamento per il controllo di versione

Molte applicazioni devono mantenere una cronologia delle revisioni a livello di item a scopi di analisi o conformità e devono essere in grado di recuperare facilmente la versione più recente. A tale scopo, esiste un modello di progettazione efficace che utilizza i prefissi delle chiavi di ordinamento:

- Per ciascun nuovo item, crea due copie dell'item: una copia deve avere un prefisso del numero di versione pari a zero (come `v0_`) all'inizio della chiave di ordinamento, l'altra copia deve avere un prefisso del numero di versione pari a uno (come `v1_`).
- Ogni volta che l'item viene aggiornato, utilizza il successivo prefisso della versione più alto nella chiave di ordinamento della versione aggiornata e copia i contenuti aggiornati nell'item con il prefisso zero. Ciò significa che la versione più aggiornata di un item può essere individuata facilmente tramite il prefisso zero.

Ad esempio, un produttore di parti potrebbe usare uno schema simile a quello illustrato sotto.

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_Tukwilla *(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.

L'item `Equipment_1` è sottoposto a una serie di controlli da parte di vari revisori. I risultati di ogni nuovo controllo vengono catturati in un nuovo item nella tabella, a partire dalla versione numero uno e aumentando il numero per ogni revisione successiva.

Quando viene aggiunta ogni nuova revisione, il livello dell'applicazione sostituisce i contenuti dell'item della versione zero (con chiave di ordinamento pari a `v0_Audit`) con i contenuti della nuova revisione.

Ogni volta che l'applicazione deve recuperare lo stato del controllo più recente, può eseguire una query per il prefisso `v0_` della chiave di ordinamento.

Se l'applicazione deve recuperare l'intera cronologia delle revisioni, può eseguire una query per tutti gli elementi della chiave di partizione dell'item e filtrare l'item `v0_`.

Questo design funziona anche per i controlli su più parti di un'apparecchiatura, se si include la singola parte IDs nella chiave di ordinamento dopo il prefisso della chiave di ordinamento.

Best practice per l'uso di indici secondari in DynamoDB.

Gli indici secondari spesso sono essenziali per supportare i modelli di query che la tua applicazione richiede. Allo stesso tempo, l'uso eccessivo di indici secondari o l'utilizzo non efficace può aggiungere costi e ridurre le prestazioni in maniera non necessaria.

Indice

- Linee guida generali per gli indici secondari in DynamoDB
  - Uso degli indici in modo efficiente
  - Scelta accurata delle proiezioni

  - Ottimizzazione delle query frequenti per evitare recuperi
  - Fai attenzione ai limiti delle dimensioni della raccolta di elementi quando crei indici secondari locali
- Trai vantaggio degli indici di tipo sparse
  - Esempi di indici di tipo sparse in DynamoDB
- Utilizzo di indici secondari globali per query di aggregazione materializzate in DynamoDB
- Overload degli indici secondari globali in DynamoDB
- Utilizzo dello sharding di scrittura negli indici secondari globali per query di tabelle selettive in DynamoDB
  - Progettazione dei modelli
  - Strategia di sharding
  - Esecuzione di query sul GSI sottoposto a sharding
  - Considerazioni sull'esecuzione di query parallele
  - esempio di codice
- Utilizzo degli indici secondari globali per creare una replica a coerenza finale in DynamoDB

Linee guida generali per gli indici secondari in DynamoDB

Amazon DynamoDB supporta due tipi di indici secondari:

- Indice secondario globale (GSI): un indice con una chiave di partizione e una chiave di ordinamento che possono essere differenti da quelle presenti sulla tabella di base. Un indice secondario viene considerato globale perché le query possono riferirsi a tutti i dati di una tabella di base, in tutte le partizioni. Un indice secondario globale non ha limiti di dimensioni e ha le proprie impostazioni di throughput assegnate per le attività di lettura e scrittura che sono separate da quelle della tabella.
- Indice secondario locale (LSI): un indice con la stessa chiave di partizione della tabella di base ma con una chiave di ordinamento diversa. Un indice secondario è locale nel senso che l'ambito di ogni partizione di un indice secondario locale è rappresentato da una partizione di tabella di base con lo stesso valore di chiave di partizione. Quindi, la dimensione totale degli elementi indicizzati per ogni valore di chiave di partizione non può eccedere 10 GB. Inoltre, un indice secondario locale condivide le impostazioni di throughput assegnate per le attività di lettura e scrittura con la tabella che sta indicizzando.

Ogni tabella in DynamoDB può avere fino a 20 indici secondari globali (quota predefinita) e 5 indici secondari locali.

Gli indici secondari globali sono spesso più utili degli indici secondari locali. La determinazione del tipo di indice da utilizzare dipenderà anche dai requisiti dell'applicazione. Per un confronto tra gli indici secondari globali e gli indici secondari locali e per ulteriori informazioni su come scegliere tra questi, consulta the section called “Utilizzo degli indici”.

Di seguito sono riportati alcuni principi e modelli di progettazione generici da tenere in considerazione quando si creano gli indici in DynamoDB:

Argomenti

- Uso degli indici in modo efficiente
- Scelta accurata delle proiezioni
- Ottimizzazione delle query frequenti per evitare recuperi
- Fai attenzione ai limiti delle dimensioni della raccolta di elementi quando crei indici secondari locali

Uso degli indici in modo efficiente

Limita il numero di indici al minimo necessario. Non creare indici secondari su attributi per i quali non esegui spesso query. Gli indici utilizzati raramente contribuiscono ad aumentare lo storage e i I/O costi senza migliorare le prestazioni delle applicazioni.

Scelta accurata delle proiezioni

In quanto gli indici secondari consumano storage e throughput assegnato, mantieni le dimensioni dell'indice al minimo possibile. Inoltre, più piccolo è l'indice, maggiore è il vantaggio in termini di prestazioni rispetto all'esecuzione di query sull'intera tabella. Se le tue query restituiscono generalmente solo un piccolo sottoinsieme di attributi e la dimensione totale di tali attributi è considerevolmente più piccola dell'intera voce, progetta solo gli attributi che richiedi regolarmente.

Se prevedi molte attività di scrittura in una tabella, rispetto a quelle di lettura, segui queste best practice:

- Considera la proiezione di un numero minore di attributi per ridurre la dimensione delle voci scritte nell'indice. Tuttavia, ciò si applica se le dimensioni degli attributi proiettati sono di dimensioni più grandi rispetto a una singola unità di capacità di scrittura (1 KB). Ad esempio, se la dimensione di

una voce dell'indice è solo di 200 byte, DynamoDB la arrotonda a 1 KB. In altre parole, nella misura in cui gli elementi degli indici sono di piccole dimensioni, puoi progettare più attributi senza costi aggiuntivi;

- Evita di proiettare attributi che sai saranno necessari raramente nelle query. Ogni volta che aggiorni un attributo che è assegnato in un indice, devi sostenere anche il costo aggiuntivo di aggiornamento dell'indice. Puoi comunque recuperare gli attributi non assegnati in una `Query` a un costo di throughput assegnato più elevato, ma il costo della query può essere molto più basso rispetto al costo di aggiornare l'indice frequentemente.
- Specifica `ALL` solo se desideri che le tue query restituiscano tutte le voci della tabella, ordinata da una chiave di ordinamento differente. La proiezione di tutti gli attributi elimina l'esigenza di recuperare le tabelle, ma nella maggior parte dei casi raddoppia i costi di storage e delle attività di scrittura.

Mantieni un equilibrio tra la necessità di mantenere piccoli gli indici e i recuperi al minimo, come illustrato nella prossima sezione.

Ottimizzazione delle query frequenti per evitare recuperi

Per ottenere le prestazioni più veloci in termini di query con la latenza più bassa possibile, proietta tutti gli attributi che prevedi vengano restituiti da tali query. In particolare, se si esegue una query su un indice secondario locale per gli attributi che non sono proiettati, DynamoDB recupera automaticamente quegli attributi dalla tabella, il che richiede la lettura di un elemento intero dalla tabella. Ciò introduce latenza e I/O operazioni aggiuntive che è possibile evitare.

Tieni a mente che le query occasionali possono spesso tramutarsi in query essenziali. Se ci sono attributi che non intendi proiettare perché pensi di eseguire query solo occasionalmente, considera se le circostanze possono cambiare e potresti pentirti di non aver proiettato quegli attributi.

Per ulteriori informazioni sul recupero delle tabelle, consulta Considerazioni sulla velocità di trasmissione effettiva assegnata per indici secondari locali.

Fai attenzione ai limiti delle dimensioni della raccolta di elementi quando crei indici secondari locali

Una raccolta di voci è un gruppo di voci in una tabella e i suoi indici locali secondari, che hanno la stessa chiave di partizione. Nessuna raccolta di elementi può superare i 10 GB pertanto è possibile che lo spazio per un determinato valore della chiave di partizione venga esaurito.

Quando si aggiunge o si aggiorna un elemento di una tabella, DynamoDB aggiornerà ogni indice secondario locale interessato. Se gli attributi indicizzati sono definiti nella tabella, anche gli indici secondari locali crescono.

Quando crei un indice secondario locale, pensa alla quantità di dati che dovrà contenere e quante di quelle voci di quei dati avranno lo stesso valore della chiave di partizione. Se si ritiene che la somma della tabella e degli elementi dell'indice per un determinato valore della chiave di partizione possa superare 10 GB, considerare se evitare la creazione dell'indice.

Se non puoi evitare la creazione dell'indice secondario locale, devi capire il limite delle dimensioni della raccolta delle voci e agire a riguardo prima di eccederlo. Una best practice è utilizzare il parametro `ReturnItemCollectionMetrics` durante la scrittura degli elementi per monitorare e ricevere avvisi sulle dimensioni delle raccolte di elementi che si avvicinano al limite di 10 GB. Il superamento della dimensione massima per le raccolte degli elementi comporterà tentativi di scrittura non riusciti. È possibile mitigare i problemi relativi alle dimensioni delle raccolte degli elementi monitorando e inviando avvisi sulle dimensioni delle raccolte degli elementi prima che generino un impatto sull'applicazione.

> **Note**
>
> Dopo la creazione non è possibile eliminare un indice secondario locale.

Per le strategie da attuare, al fine di non eccedere il limite e per eseguire delle operazioni correttive, consulta Limite delle dimensioni delle raccolte di elementi.

Trai vantaggio degli indici di tipo sparse

Per qualsiasi elemento in una tabella, DynamoDB scrive una voce di indice corrispondente solo se nell'elemento è presente il valore della chiave di ordinamento dell'indice. Se la chiave di ordinamento non appare in tutti gli elementi della tabella o se la chiave di partizione dell'indice non è presente nell'elemento, si dice che l'indice sia sparso.

Gli indici sparse sono utili per query su una sottosezione piccola della tabella. Ad esempio immagina di avere una tabella dove archivi tutti gli ordini dei clienti, con i seguenti attributi di chiave:

- Chiave di partizione: `CustomerId`
- Chiave di ordinamento: `OrderId`

Per tracciare gli ordini aperti, è possibile inserire un attributo denominato `isOpen` negli articoli degli ordini che non sono ancora stati spediti. Poi quando viene spedito l'ordine, puoi eliminare l'attributo. Se poi crei un indice su `CustomerId` (chiave di partizione) e `isOpen` (chiave di ordinamento), solo quegli ordini con `isOpen` definito vi appaiono. Quando hai migliaia di ordini dei quali solo un numero piccolo sono aperti, è più facile e meno costoso eseguire query sull'indice sugli ordini aperti che scannerizzare la tabella intera.

Invece di utilizzare un tipo di attributo come `isOpen`, è possibile utilizzare un attributo con un valore che risulta in un ordine di ordinamento utile nell'indice. Ad esempio, puoi utilizzare un set di attributi `OrderOpenDate` per la data in cui è stato effettuato l'ordine e poi eliminarlo dopo che l'ordine è stato consegnato. In quel modo, quando esegui una query sull'indice sparso, le voci vengono restituite ordinate per la data in cui l'ordine è stato effettuato.

Esempi di indici di tipo sparse in DynamoDB

Gli indici secondari globali sono sparse come impostazione predefinita. Quando crei un indice secondario globale, specifica una chiave di partizione e opzionalmente una chiave di ordinamento. Solo voci nella tabella base che contengono quegli attributi appaiono nell'indice.

Progettando che un indice secondario globale sia sparse, puoi assegnargli throughput di scrittura più basso rispetto a quello della tabella base, ottenendo comunque prestazioni eccellenti.

Ad esempio, un'applicazione di gaming può registrare tutti i punteggi di ogni utente, ma in genere deve eseguire delle query solo su alcuni punteggi bassi. La seguente progettazione gestisce questo scenario in modo efficiente:

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)*

Rick ha giocato tre partite e ottenuto lo stato `Champ` in una. Padma ha giocato quattro partite e ottenuto lo stato `Champ` in due. Nota che l'attributo `Award` è presente solo nelle voci dove l'utente ha ottenuto un premio. L'indice secondario globale associato ha il seguente aspetto:

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

L'indice secondario globale contiene solo i punteggi alti sui quali vengono spesso eseguite delle query, che sono un sottoinsieme piccolo delle voci nella tabella base.

Utilizzo di indici secondari globali per query di aggregazione materializzate in DynamoDB

Il mantenimento di aggregazioni quasi in tempo reale e parametri chiave oltre a dati in rapida evoluzione sta acquisendo sempre più valore per permettere alle aziende di prendere decisioni in

maniera veloce. Ad esempio, una libreria musicale potrebbe voler presentare le canzoni più scaricate in tempo reale.

Considera il seguente layout di tabella per libreria musicale:

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 tabella in questo esempio contiene canzoni con `songID` come chiave di partizione. È possibile abilitare Amazon DynamoDB Streams su questa tabella e collegare una funzione Lambda ai flussi in modo che ogni volta che viene scaricata una canzone, viene aggiunta una voce alla tabella con `Partition-Key=SongID` e `Sort-Key=DownloadID`. Quando effettuati, questi aggiornamenti attivano una funzione Lambda in DynamoDB Streams. La funzione di Lambda può aggregare e raggruppare i download per `songID` e aggiungere la voce di primo livello, `Partition-Key=songID` e `Sort-Key=Month`. Tieni presente che se un'esecuzione lambda fallisce subito dopo aver scritto il nuovo valore aggregato, potrebbe essere effettuato un nuovo tentativo e aggregato il valore più di una volta, lasciandoti un valore approssimativo.

Per leggere gli aggiornamenti quasi in tempo reale, con una latenza di pochi millisecondi, utilizza l'indice secondario globale con le condizioni di query `Month=2018-01`, `ScanIndexForward=False`, `Limit=1`.

Un'altra chiave di ottimizzazione utilizzata qui è che l'indice secondario globale è un indice di tipo sparse ed è disponibile solo su voce sulle quali bisogna eseguire una query per recuperare i dati in tempo reale. L'indice secondario globale può assistere i flussi di lavoro addizionali che hanno bisogno di informazioni sulle 10 canzoni più popolari o qualsiasi canzone scaricata in quel mese.

Overload degli indici secondari globali in DynamoDB

Sebbene Amazon DynamoDB abbia una quota predefinita di 20 indici secondari globali per tabella, in pratica, puoi indicizzare su più di 20 campi di dati. A differenza di una tabella in un sistema di gestione del database relazionale (RDBMS), in cui lo schema è uniforme, una tabella in può contenere molti tipi di elementi di dati diversi allo stesso tempo. Inoltre, lo stesso attributo in elementi diversi può contenere tipi di informazioni completamente diversi.

Si consideri il seguente esempio di layout di tabella DynamoDB che salva una varietà di diversi tipi di dati.

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.

L'attributo `Data`, che è comune per tutte le voci ha un contenuto diverso in base alla voce padre. Se crei un indice secondario globale per la tabella che utilizza la chiave di ordinamento della tabella come chiave di partizione e l'attributo `Data` come chiave di ordinamento, puoi eseguire query diverse utilizzando quel singolo indice secondario globale. Queste query possono includere quanto segue:

- Cercare un dipendente in base al nome nell'indice secondario globale, utilizzando `Employee_Name` come valore della chiave di partizione e il nome del dipendente (ad esempio `Murphy, John`) come valore della chiave di ordinamento.
- Utilizza l'indice secondario globale per trovare tutti i dipendenti che lavorano in un magazzino specifico cercando un ID magazzino (come `Warehouse_01`).
- Ottieni una lista dei neoassunti eseguendo una query sull'indice secondario globale su `HR_confidential` come valore di chiave di partizione e utilizzando un intervallo di date come valore della chiave di ordinamento.

Utilizzo dello sharding di scrittura negli indici secondari globali per query di tabelle selettive in DynamoDB

Quando è necessario eseguire query su dati recenti entro una finestra temporale specifica, il requisito di DynamoDB di fornire una chiave di partizione per la maggior parte delle operazioni di lettura può rappresentare una sfida. Per risolvere questo scenario, è possibile implementare un modello di query efficace utilizzando una combinazione di sharding di scrittura e un indice secondario globale (GSI).

Questo approccio consente di recuperare e analizzare in modo efficiente i dati sensibili al fattore tempo senza eseguire scansioni complete delle tabelle, che possono richiedere molte risorse e costi. Progettando strategicamente la struttura e l'indicizzazione delle tabelle è possibile creare una soluzione flessibile che supporti il recupero dei dati basato sul tempo, mantenendo al contempo prestazioni ottimali.

Argomenti

- Progettazione dei modelli
- Strategia di sharding
- Esecuzione di query sul GSI sottoposto a sharding
- Considerazioni sull'esecuzione di query parallele
- esempio di codice

Progettazione dei modelli

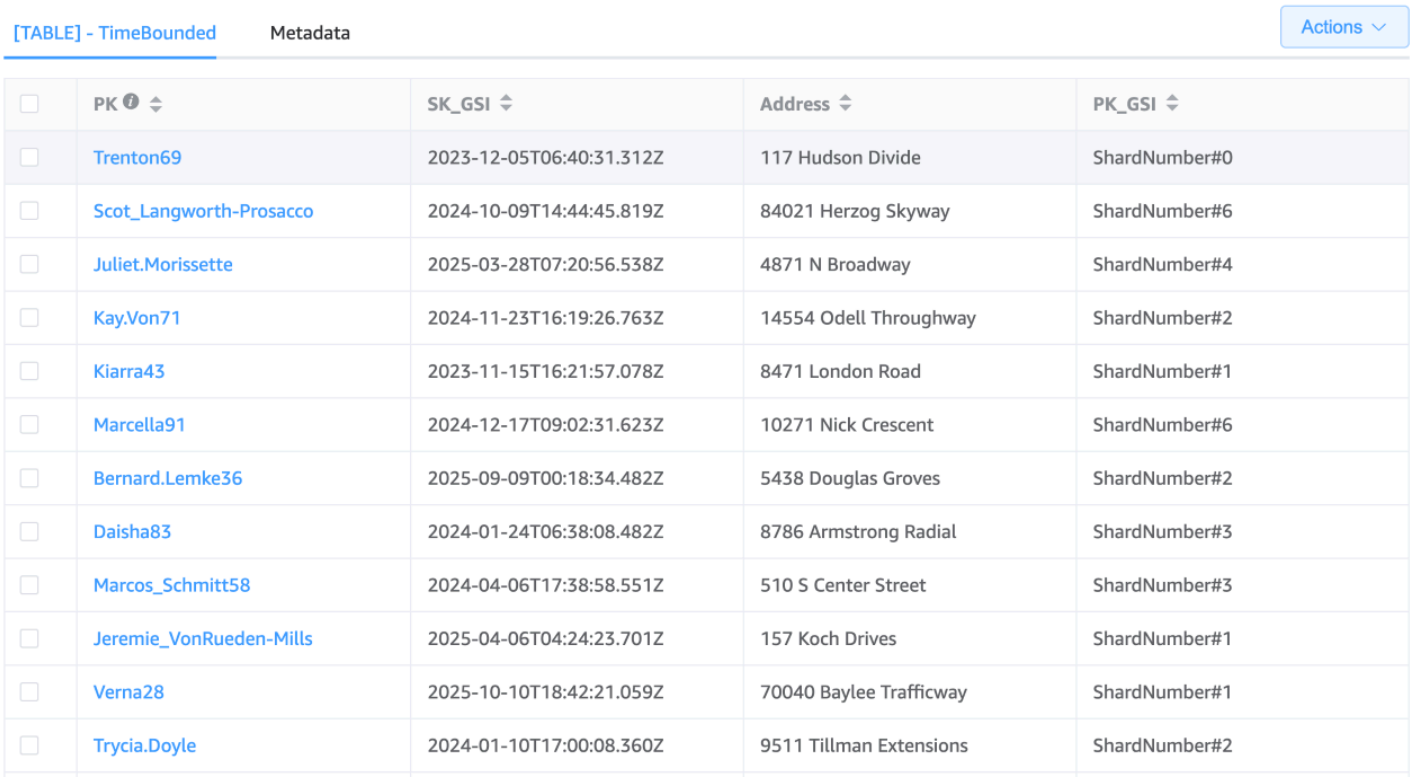
Durante il lavoro con DynamoDB, è possibile superare le sfide derivanti dal recupero dei dati basato sul tempo implementando un modello sofisticato che combina lo sharding di scrittura e gli indici secondari globali per consentire query flessibili ed efficienti su finestre di dati recenti.

Struttura della tabella

- Chiave di partizione (PK): “Username”

Struttura del GSI

- Chiave di partizione GSI (PK_GSI): "#» ShardNumber
- Chiave di ordinamento del GSI (SK_GSI): ISO 8601 timestamp (ad esempio, “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

Strategia di sharding

Supponendo che si decida di utilizzare 10 shard, i numeri degli shard potrebbero variare da 0 a 9. Quando si registrano i log di un’attività, è necessario calcolare il numero di shard (ad esempio, utilizzando una funzione hash sull’ID utente e quindi prendendo il modulo del numero di shard) e anteponendolo alla chiave di partizione del GSI. Questo metodo distribuisce le voci su diversi shard, mitigando il rischio di partizioni calde.

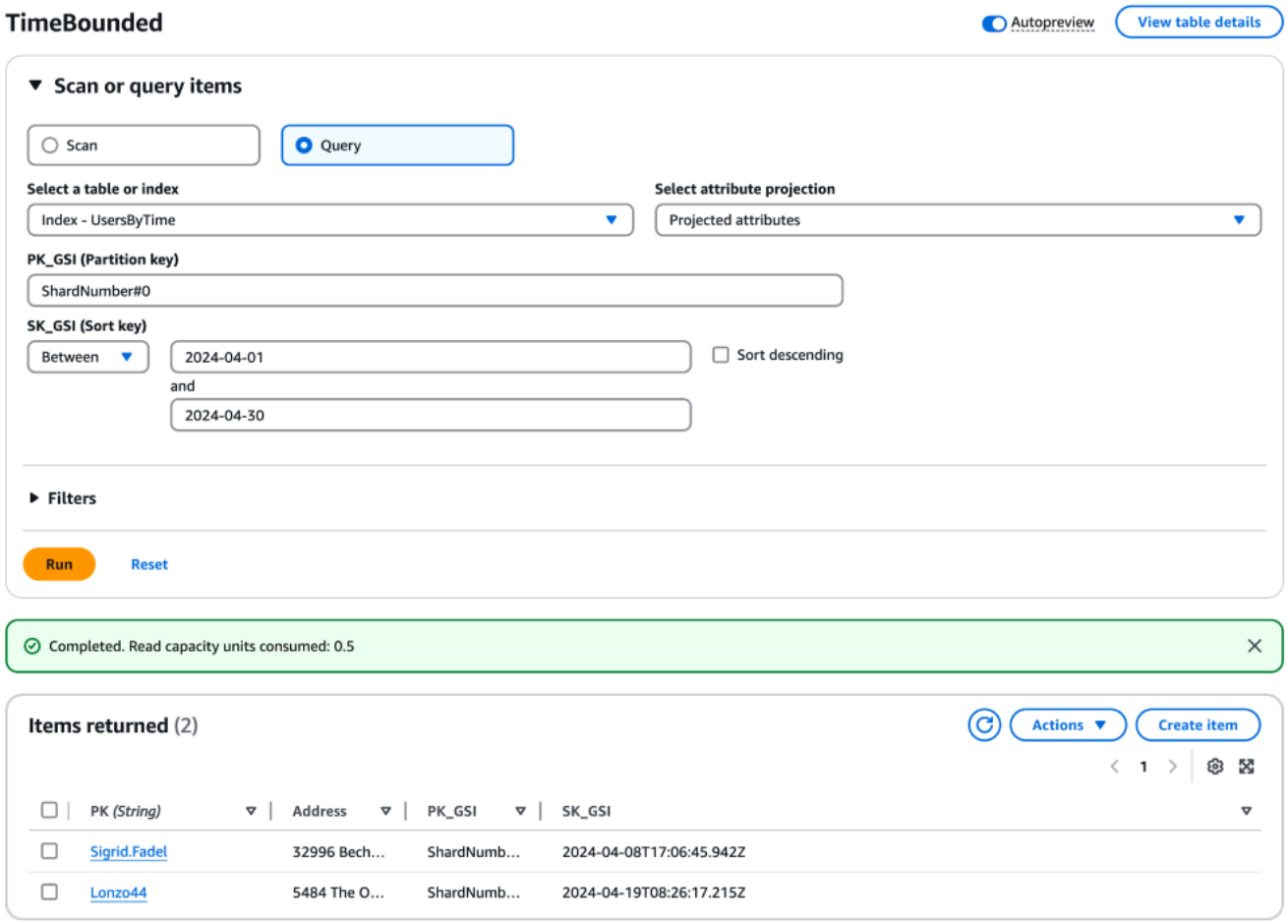
Esecuzione di query sul GSI sottoposto a sharding

L'esecuzione di query su tutti gli shard per gli elementi all'interno di un determinato intervallo di tempo in una tabella DynamoDB, in cui i dati sono sottoposti a sharding su più chiavi di partizione, richiede un approccio diverso rispetto all'esecuzione di query su una singola partizione. Poiché le query DynamoDB sono limitate a una singola chiave di partizione alla volta, non è possibile eseguire query direttamente su più shard con una singola operazione di query. Tuttavia, è possibile ottenere il risultato desiderato tramite la logica a livello di applicazione eseguendo più query, ciascuna mirata a uno shard specifico, e quindi aggregando i risultati. Di seguito viene descritto come procedere.

Come eseguire query sugli shard e aggregarli

1. Identifica l'intervallo di numeri di shard utilizzati nella strategia di sharding. Ad esempio, con 10 frammenti, i numeri degli shard andrebbero da 0 a 9.
2. Per ogni shard, costruisci ed esegui una query per recuperare gli elementi entro l'intervallo di tempo desiderato. Queste query possono essere eseguite in parallelo per migliorare l'efficienza. Usa la chiave di partizione con il numero di shard e la chiave di ordinamento con l'intervallo di tempo per queste query. Ecco un esempio di query per un singolo shard:

```
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"}
    }'
```



Dovreste replicare questa query per ogni shard, regolando la chiave di partizione di conseguenza (ad esempio, "ShardNumber#1 «," ShardNumber #2 «,...," #9 «). ShardNumber

3. Aggrega i risultati di ogni query dopo che tutte le query sono state completate. Esegui questa aggregazione nel codice dell'applicazione, combinando i risultati in un unico set di dati che rappresenta gli elementi di tutti gli shard entro l'intervallo di tempo specificato.

Considerazioni sull'esecuzione di query parallele

Ogni query consuma la capacità di lettura della tabella o dell'indice. In caso di utilizzo di un throughput allocato, assicurati che la tabella disponga di una capacità sufficiente per gestire l'espansione di query parallele. In caso di utilizzo di capacità on demand, tieni presente le potenziali implicazioni in termini di costi.

esempio di codice

Per eseguire query parallele su shard in DynamoDB usando Python, è possibile usare la libreria boto3, che è l'SDK Amazon Web Services per Python. Questo esempio presuppone che boto3 sia installato e configurato con le credenziali appropriate. AWS

Il seguente codice Python dimostra come eseguire query parallele su più shard per un determinato intervallo di tempo. Vengono utilizzati oggetti futuri simultanei per eseguire query in parallelo, riducendo il tempo di esecuzione complessivo rispetto all'esecuzione sequenziale.

```
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)}")
```

Prima di eseguire questo codice, assicurati di sostituire `YourTableName` e `YourGSIName` con la tabella e i nomi dei GSI effettivi della configurazione di DynamoDB. Inoltre, regola le variabili `total_shards`, `time_start` e `time_end` in base alle esigenze specifiche.

Questo script esegue query su ogni shard alla ricerca di elementi entro l'intervallo di tempo specificato e aggrega i risultati.

Utilizzo degli indici secondari globali per creare una replica a coerenza finale in DynamoDB

È possibile utilizzare un indice secondario globale per creare una replica di consistenza finale di una tabella. La creazione di una replica può consentire il completamento delle seguenti operazioni:

- Configurare una capacità di lettura con provisioning differente per lettori diversi. Si supponga, ad esempio, di disporre di due applicazioni: un'applicazione gestisce le query con priorità elevata e richiede i massimi livelli di prestazioni di lettura, mentre l'altra gestisce le query con priorità bassa in grado di tollerare la limitazione dell'attività di lettura.

  Se entrambe queste applicazioni leggono dalla stessa tabella, un carico di lettura pesante dall'applicazione con priorità bassa potrebbe consumare tutta la capacità di lettura disponibile per la tabella. Ciò limiterebbe l'attività di lettura dell'applicazione con priorità alta.

È invece possibile creare una replica tramite un indice secondario globale la cui capacità di lettura può essere impostata separatamente da quella della tabella stessa. È quindi possibile fare in modo che l'app a priorità bassa interroghi la replica anziché la tabella.

- Eliminare completamente le letture da una tabella. Ad esempio, è possibile avere un'applicazione che acquisisce un volume elevato di attività di clickstream da un sito Web e non si vuole rischiare che le letture interferiscano. È possibile isolare questa tabella e impedire la lettura da parte di altre applicazioni (consulta Utilizzo di condizioni di policy IAM per il controllo granulare degli accessi), consentendo ad altre applicazioni di leggere una replica creata utilizzando un indice secondario globale.

Per creare una replica, impostare un indice secondario globale con lo stesso schema chiave della tabella padre, con alcuni o tutti gli attributi non chiave proiettati. Nelle applicazioni, è possibile indirizzare alcune o tutte le attività di lettura a questo indice secondario globale anziché alla tabella padre. È quindi possibile regolare la capacità di lettura con provisioning dell'indice secondario globale per gestire tali letture senza modificare la capacità di lettura con provisioning della tabella padre.

C'è sempre un breve ritardo di propagazione tra una scrittura nella tabella padre e il momento in cui i dati scritti vengono visualizzati nell'indice. In altre parole, le applicazioni dovrebbero tenere conto del fatto che la replica dell'indice secondario globale è solo a consistenza finale con la tabella padre.

È possibile creare più repliche di indice secondario globale per supportare diversi modelli di lettura. Quando si creano le repliche, proiettare solo gli attributi effettivamente richiesti da ciascun modello di lettura. Un'applicazione può quindi consumare meno capacità di lettura con provisioning per ottenere solo i dati necessari anziché dover leggere l'elemento dalla tabella padre. Questa ottimizzazione può comportare significativi risparmi sui costi nel tempo.

Best practice per l’archiviazione di elementi e attributi di grandi dimensioni in DynamoDB

Amazon DynamoDB limita le dimensioni di ogni elemento che viene archiviato in una tabella a 400 KB (consulta Quote in Amazon DynamoDB). Se l'applicazione ha bisogno di archiviare più dati in un elemento rispetto al limite di dimensione consentito da DynamoDB, è possibile provare a comprimere uno o più attributi large o suddividere l'elemento in più elementi (indicizzati in modo efficiente mediante chiavi di ordinamento). È possibile archiviare l'elemento anche come un oggetto in Amazon Simple Storage Service (Amazon S3) e archiviare l'identificatore dell'oggetto Amazon S3 nell'elemento DynamoDB.

Come best practice, è consigliabile utilizzare il parametro `ReturnConsumedCapacity` durante la scrittura degli elementi per monitorare e avvisare sulle dimensioni degli elementi che si avvicinano al limite massimo di 400 KB. Il superamento della dimensione massima degli elementi comporterà tentativi di scrittura non riusciti. DynamoDB restituirà un errore. ValidationException Il monitoraggio e l'invio di avvisi sulle dimensioni degli elementi consente di mitigare i problemi relativi alle dimensioni degli elementi prima che generino un impatto sull'applicazione.

Compressione di valori di attributi di grandi dimensioni

La compressione di valori di attributi large permette la loro archiviazione nei limiti degli elementi in DynamoDB e ridurre così i costi di archiviazione. Gli algoritmi di compressione come GZIP o LZO producono un output binario che è possibile archiviare in un tipo di attributo `Binary` all'interno dell'elemento.

Ad esempio, si consideri una tabella che archivia i messaggi scritti dagli utenti di un forum. Tali messaggi contengono spesso lunghe stringhe di testo, che possono essere compresse. Sebbene la compressione possa ridurre le dimensioni degli elementi, lo svantaggio è che i valori degli attributi compressi non sono utili per il filtraggio.

Per il codice di esempio che illustra come comprimere tali messaggi in DynamoDB, consulta i seguenti argomenti:

- Esempio: gestione degli attributi di tipo binario utilizzando l'API del documento AWS SDK per Java
- Esempio: gestione degli attributi di tipo binario utilizzando l'API di basso livello AWS SDK per .NET

Partizionamento verticale

Una soluzione alternativa alla gestione di elementi di grandi dimensioni consiste nel suddividerli in blocchi di dati più piccoli e nell'associare tutti gli elementi pertinenti in base al valore della chiave di partizione. È quindi possibile utilizzare una stringa di chiavi di ordinamento per identificare le informazioni associate archiviate insieme a essa. In questo modo e raggruppando più elementi in base allo stesso valore della chiave di partizione, si crea una raccolta di elementi.

Per ulteriori informazioni su questo approccio, consulta:

- Use vertical partitioning to scale data efficiently in Amazon DynamoDB
- Implementa il partizionamento verticale in Amazon DynamoDB utilizzando AWS Glue

Archiviazione dei valori di attributi di grandi dimensioni in Amazon S3

Come indicato in precedenza, è possibile utilizzare Amazon S3 anche per archiviare valori di attributi large che non rientrano in un elemento DynamoDB. Puoi archiviarli come oggetto in Amazon S3, quindi archiviare l'identificatore di oggetto nell'elemento DynamoDB.

È possibile anche utilizzare il supporto per i metadati dell'oggetto in Amazon S3 per fornire un link all'elemento padre in DynamoDB. Archiviare il valore della chiave primaria dell'elemento come metadati Amazon S3 dell'oggetto in Amazon S3. Ciò spesso aiuta con la manutenzione degli oggetti di Amazon S3.

Ad esempio, considera la tabella `ProductCatalog`. gli elementi in questa tabella archiviano informazioni su prezzo dell'item, descrizione, autori dei libri e dimensioni degli altri prodotti. Se si desidera archiviare un'immagine per ogni prodotto che è troppo grande per un elemento, è possibile puoi archiviarla in Amazon S3 anziché in DynamoDB.

Quando implementi questa strategia, tieni a mente quanto segue:

- DynamoDB non supporta le transazioni che passano in Amazon S3 e DynamoDB. Pertanto, l'applicazione deve gestire eventuali fallimenti, che potrebbero includere la pulizia di oggetti Amazon S3 orfani.
- Amazon S3 limita la lunghezza degli identificatori dell'oggetto. Quindi è necessario organizzare i dati in una maniera che non generi identificatori d'oggetto troppo lunghi o che violi altri vincoli di Amazon S3.

Per maggiori informazioni su come utilizzare Amazon S3, consulta la Amazon Simple Storage Service User Guide.

Best practice per la gestione dei dati di serie temporali in DynamoDB

I principi di progettazione generali di Amazon DynamoDB consigliano di mantenere al minimo il numero di tabelle utilizzate. Per la maggior parte delle applicazioni, una singola tabella è sufficiente. Tuttavia, spesso è meglio gestire i dati delle serie temporali usando una tabella per applicazione per periodo.

Modello di progetto per i dati di serie temporali

Prendi in considerazione uno scenario tipico relativo alle serie temporali, in cui vuoi tenere traccia di un elevato volume di eventi. Il tuo modello di accesso in scrittura equivale a quello di tutti gli eventi registrati con la data odierna. Il tuo modello di accesso in lettura potrebbe prevedere la lettura degli eventi odierni con più frequenza, degli eventi di ieri con meno frequenza e quindi degli eventi più vecchi con una frequenza ridotta. Una soluzione di gestione prevede l'aggiunta della data e dell'ora correnti alla chiave primaria.

Il seguente modello di progetto spesso è in grado di gestire questo tipo di scenario in modo efficace:

- Crea una tabella per periodo che abbia la capacità in lettura e scrittura e gli indici richiesti.
- Prima della fine di ogni periodo, prepara la tabella per il periodo successivo. Non appena termina il periodo corrente, dirigi il traffico degli eventi alla nuova tabella. A queste tabelle puoi assegnare dei nomi che specifichino i periodi registrati.
- Non appena si interrompe la scrittura su una tabella, imposta la capacità in scrittura assegnata su un valore inferiore, ad esempio 1 WCU, e assegna la capacità in lettura appropriata. Riduci la capacità in lettura assegnata delle tabelle precedenti man mano che passa il tempo. Puoi scegliere di archiviare o eliminare le tabelle di cui utilizzi raramente o mai i contenuti.

L'idea è allocare le risorse necessarie per il periodo corrente con il volume di traffico più alto e di ridurre il provisioning per le tabelle precedenti che non vengono utilizzate attivamente, riducendo così i costi. A seconda delle esigenze aziendali, può essere utile partizionare la scrittura per distribuire equamente il traffico alla chiave di partizione logica. Per ulteriori informazioni, consulta Utilizzo dello sharding di scrittura per distribuire i carichi di lavoro in modo uniforme in una tabella DynamoDB.

Esempi di tabella di serie temporali

Di seguito è riportato un esempio di dati di serie temporali in cui la tabella corrente viene fornita con una read/write capacità maggiore e le tabelle precedenti vengono ridimensionate perché vi si accede di rado.

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	...
• • •	• • •	• • •	• • •	...

Le migliori pratiche per la gestione delle many-to-many relazioni nelle tabelle DynamoDB

Gli elenchi di adiacenza sono un modello di progettazione utile per modellare many-to-many le relazioni in Amazon DynamoDB. Più in generale, forniscono un modo per rappresentare i dati grafici (nodi ed edge) in DynamoDB.

Modello di progettazione elenchi di adiacenza

Quando diverse entità di un'applicazione hanno una many-to-many relazione tra loro, la relazione può essere modellata come un elenco di adiacenze. In questo modello, tutte le entità di primo livello (sinonimi di nodi nel modello grafico) vengono rappresentate utilizzando la chiave di partizione. Qualsiasi relazione con altre entità (edge in un grafico) viene rappresentata come voce in una partizione impostando il valore della chiave di ordinamento sull'ID dell'entità target (nodo target).

I vantaggi di questo modello comprendono una duplicazione minima dei dati e modelli di query semplificati per trovare tutte le entità (nodi) legate a un'entità target (avendo un edge in un nodo target).

Un esempio reale dove questo modello è stato utile è un sistema di fatturazione dove le fatture contengono ricevute multiple. Una ricevuta può appartenere a fatture multiple. La chiave di partizione in questo esempio è un `InvoiceID` o un `BillID`. Le partizioni `BillID` hanno tutti gli attributi specifici delle fatture. Le partizioni `InvoiceID` hanno una voce che contiene attributi specifici delle fatture e una voce per ogni `BillID` che si trova nella fattura.

Lo schema ha il seguente aspetto.

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...

Utilizzando lo schema precedente, puoi vedere che per tutte le ricevute per una fattura può essere eseguita una query utilizzando la chiave primaria nella tabella. Per controllare tutte le fatture che

contengono una parte di ricevuta, crea un indice secondario globale nella chiave di ordinamento della tabella.

Le previsioni per l'indice secondario globale hanno il seguente aspetto.

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...

Modello grafico materializzato

Tante applicazioni sono costruite sulla base della comprensione delle classificazioni tra peer, relazioni comuni tra entità, stato entità vicina e altri tipi di flussi di lavoro in stile grafico. Per questi tipi di applicazioni, considera il seguente modello di progettazione dello schema.

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	

GSI 2	Primary Key		Attributes		
	GSI PK	GSI 2 SK (TypeTarget)			
	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	

Lo schema precedente mostra una struttura grafica di dati definita da un set di partizioni di dati contenenti le voci che definiscono gli edge e i nodi del grafico. Le voci edge contengono un attributo `Target` e `Type`. Questi attributi vengono utilizzati come parte di un nome di chiave composito "TypeTarget" per identificare l'elemento in una partizione nella tabella primaria o in un secondo indice secondario globale.

Il primo indice secondario globale è costruito sull'attributo `Data`. Questo attributo utilizza l'overloading dell'indice secondario globale come descritto in precedenza per indicizzare diversi tipi di attributo, ovvero `Dates`, `Names`, `Places` e `Skills`. Qui, un indice secondario globale indicizza quattro diversi attributi in modo efficace.

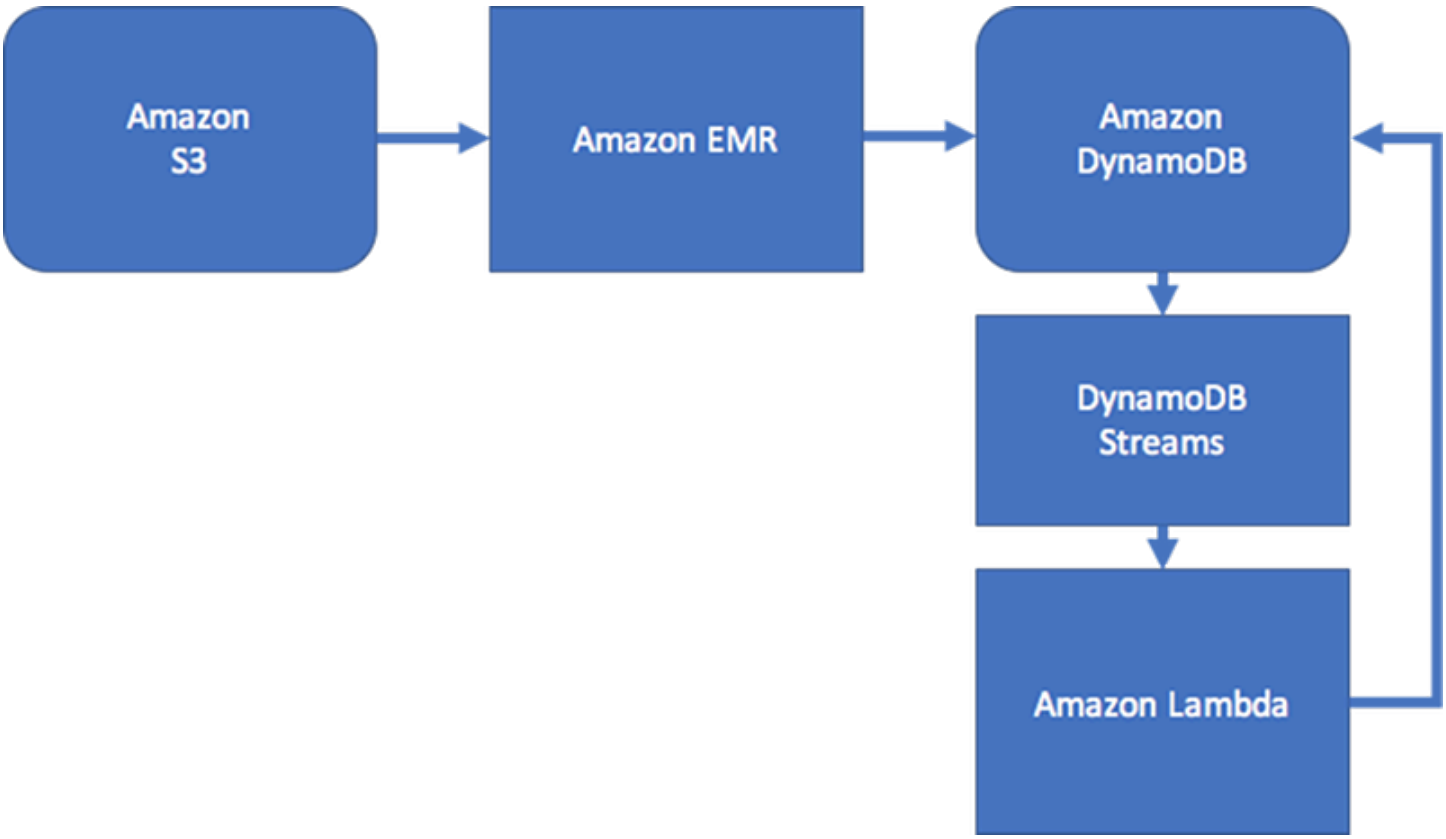
Mentre inserisci voci nella tabella, puoi utilizzare una strategia di partizionamento intelligente per distribuire i set di voci con aggregazioni large (data di nascita, competenza) nel maggior numero necessario di partizioni logiche negli indici secondari globali per evitare problemi hot di lettura/scrittura.

Il risultato di questa combinazione di modelli di design è un datastore solido per flussi di lavoro grafici in tempo reale altamente efficaci. Questi flussi di lavoro possono fornire query di prestazioni alte

sullo stato dell'entità vicina e sull'aggregazione degli edge per i motori delle raccomandazioni, le applicazioni di social network, le classifiche dei nodi, le aggregazioni di sottostruttura e altri casi d'uso comuni dei grafici.

Se il tuo caso d'uso non è sensibile alla coerenza dei dati in tempo reale, è possibile utilizzare un processo Amazon EMR programmato per popolare gli edge con aggregazioni di riepilogo grafico rilevanti per i flussi di lavoro. Se la tua applicazione non deve sapere immediatamente quando un edge è aggiunto al grafico, puoi utilizzare un processo programmato per aggregare i risultati.

Per mantenere un certo divello di coerenza, la progettazione può includere Amazon DynamoDB Streams e AWS Lambda per elaborare gli aggiornamenti edge. Può anche utilizzare un processo Amazon EMR per convalidare i risultati a un intervallo regolare. Questo approccio viene illustrato nel seguente diagramma. Viene comunemente utilizzato nelle applicazioni di social network dove il costo di una query in tempo reale è alta e la necessità di ottenere aggiornamenti all'utente individuale è bassa.



Le applicazioni di gestione dei servizi IT (ITSM) e di sicurezza in genere devono rispondere in tempo reale alle modifiche all'entità stato composte da aggregazioni degli edge complesse. Tali applicazioni necessitano di un sistema che può supportare aggregazioni dei nodi multiple in tempo reale di relazioni di secondo e terzo livello o elementi trasversali di edge complessi. Se il tuo caso d'uso

richiede questi tipi di flusso di lavoro grafici di query in tempo reale, consigliamo l'utilizzo di Amazon Neptune per gestire questi flussi di lavoro.

> **Note**
>
> Se devi eseguire query sui set di dati altamente connessi o eseguire query che devono attraversare più nodi (note anche come query multi-hop) con una latenza di millisecondi, è opportuno prendere in considerazione l'utilizzo di Amazon Neptune. Amazon Neptune è un motore di database a grafo dedicato e a prestazioni elevate, ottimizzato per conservare miliardi di relazioni e inviare query al grafo con una latenza di millisecondi.

Best practice per eseguire query e scansioni dei dati in DynamoDB

In questa sezione vengono illustrate alcune best practice per l'utilizzo delle operazioni `Query` e `Scan` in Amazon DynamoDB.

Considerazioni sulle prestazioni per le scansioni

In generale, le operazioni `Scan` sono meno efficienti rispetto ad altre operazioni in DynamoDB. Un'operazione `Scan` esegue sempre la scansione dell'intera tabella o dell'indice secondario. Quindi filtra i valori per fornire il risultato desiderato, aggiungendo essenzialmente il passaggio aggiuntivo di rimozione dei dati dal set di risultati.

Se possibile, è preferibile evitare di utilizzare un'operazione `Scan` su una tabella o un indice di grandi dimensioni con un filtro che rimuove molti risultati. Inoltre, man mano che una tabella o un indice cresce, l'operazione `Scan` rallenta. L'operazione `Scan` esamina ogni elemento per i valori richiesti e può utilizzare la velocità effettiva assegnata per una tabella o un indice di grandi dimensioni in un'unica operazione. Per tempi di risposta più rapidi, progettare tabelle e indici in modo che le applicazioni possano utilizzare `Query`invece di `Scan`. (Per le tabelle, puoi anche prendere in considerazione l'utilizzo di `GetItem` and `BatchGetItem` APIs.)

In alternativa, puoi progettare l'applicazione in modo da utilizzare operazioni `Scan` e ridurre al minimo l'impatto sulla percentuale di frequenza delle richieste. Ciò può includere la modellazione quando potrebbe essere più efficiente utilizzare un indice secondario globale anziché un'operazione `Scan`. Ulteriori informazioni su questo processo sono disponibili nel seguente video.

Modellazione di modelli di accesso a bassa velocità

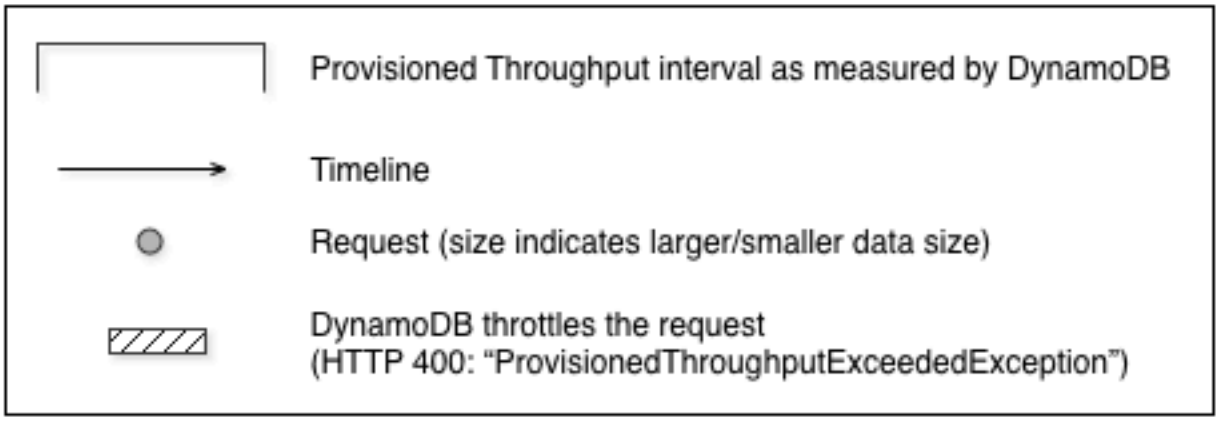
Come evitare picchi improvvisi nell'attività di lettura

Quando si crea una tabella, si impostano i requisiti di unità di capacità di lettura e scrittura. Per le letture, le unità di capacità sono espresse come il numero di richieste di lettura dati fortemente coerenti da 4 KB al secondo. Per letture a consistenza finale, un'unità di capacità di lettura è due richieste di lettura da 4 KB al secondo. Un'operazione `Scan` esegue letture a consistenza finale per impostazione predefinita e può restituire fino a 1 MB (una pagina) di dati. Pertanto, una singola richiesta `Scan` può consumare (1 MB di dimensione della pagina/4 KB dimensione elemento)/2 (letture a consistenza finale) = 128 operazioni di lettura. Se invece si richiedono letture fortemente consistenti, il metodo `Scan` consumerebbe il doppio della velocità effettiva assegnata, ovvero 256 operazioni di lettura.

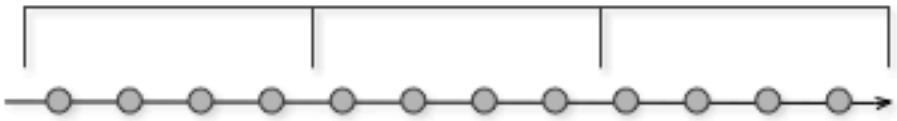
Questo rappresenta un picco improvviso di utilizzo rispetto alla capacità di lettura configurata per la tabella. Questo utilizzo di unità di capacità da parte di una scansione impedisce ad altre richieste potenzialmente più importanti per la stessa tabella di utilizzare le unità di capacità disponibili. Di conseguenza, è probabile che si ottenga una eccezione `ProvisionedThroughputExceeded` per tali richieste.

Il problema non è solo l'improvviso aumento delle unità di capacità utilizzate da `Scan`. È inoltre probabile che la scansione utilizzi tutte le unità di capacità dalla stessa partizione, poiché le richieste della scansione leggono gli elementi che si trovano uno accanto all'altro nella partizione. Ciò significa che la richiesta considera la stessa partizione, causando l'utilizzo di tutte le unità di capacità e la limitazione di altre richieste a tale partizione. Se la richiesta di lettura dei dati è distribuita su più partizioni, l'operazione non riduce una partizione specifica.

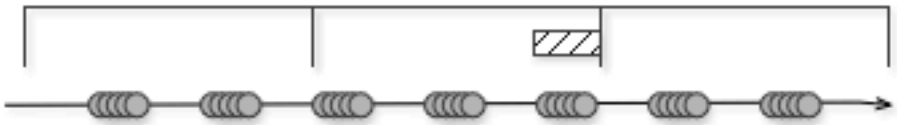
Il diagramma seguente illustra l'impatto di un improvviso picco di utilizzo delle unità di capacità da parte delle operazioni `Query` e `Scan` e il suo impatto sulle altre richieste sulla stessa tabella.



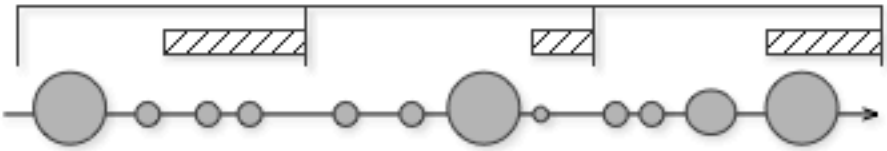
1. Good: Even distribution of requests and size



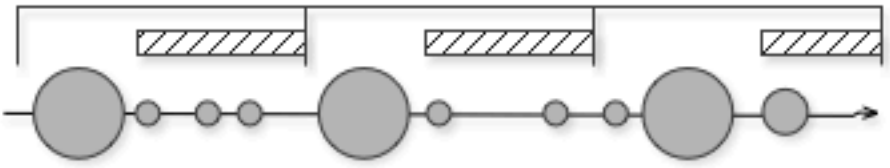
2. Not as Good: Frequent requests in bursts



3. Bad: A few random large requests



4. Bad: Large scan operations



Come illustrato di seguito, il picco di utilizzo può influire sulla velocità effettiva assegnata della tabella in diversi modi:

1. Buono: distribuzione uniforme delle richieste e delle dimensioni

2. Non tanto buono: richieste frequenti a raffica
3. Cattivo: alcune richieste casuali di grandi dimensioni
4. Cattivo: operazioni di scansione di grandi dimensioni

Invece di usare un'operazione `Scan` di grandi dimensioni, è possibile utilizzare le seguenti tecniche per ridurre al minimo l'impatto di una scansione sulla velocità effettiva assegnata di una tabella.

- Riduzione delle dimensioni delle pagine

  Poiché un'operazione di scansione legge un'intera pagina (per impostazione predefinita, 1 MB), è possibile ridurre l'impatto dell'operazione di scansione impostando una dimensione di pagina più piccola. L'operazione `Scan` fornisce un parametro Limit che può essere utilizzato per impostare le dimensioni della pagina per la richiesta. Ogni richiesta `Query` o `Scan` che ha una dimensione di pagina più piccola utilizza meno operazioni di lettura e crea una "pausa" tra ogni richiesta. Si supponga, ad esempio, che ogni elemento sia di 4 KB e di impostare la dimensione della pagina su 40 elementi. Una richiesta `Query` utilizzerebbe quindi solo 20 operazioni di lettura a consistenza finale o 40 operazioni di lettura fortemente consistenti. Un numero maggiore di operazioni `Query` o `Scan` più piccole permetterebbe alle altre richieste critiche di avere successo senza limitazioni.
- Isolamento delle operazioni di scansione

  DynamoDB è progettato per una facile scalabilità. Di conseguenza, un'applicazione può creare tabelle per scopi distinti, possibilmente anche duplicando il contenuto su più tabelle. Si desidera eseguire scansioni su una tabella che non utilizza traffico "mission-critical". Alcune applicazioni gestiscono questo carico ruotando il traffico ogni ora tra due tabelle, una per il traffico critico e una per la contabilità. Altre applicazioni possono farlo eseguendo ogni scrittura su due tabelle: una tabella "mission-critical" e una tabella "shadow".

Configurare l'applicazione in modo da riprovare qualsiasi richiesta che riceve un codice di risposta che indica che è stato superata la velocità effettiva assegnata. In alternativa, aumentare la velocità effettiva assegnata per la tabella utilizzando l'operazione `UpdateTable`. Se nel carico di lavoro sono presenti picchi temporanei che causano il superamento della velocità effettiva, occasionalmente, oltre il livello assegnato, riprovare la richiesta con backoff esponenziale. Per ulteriori informazioni sull'implementazione del backoff esponenziale, consulta Ripetizione dei tentativi in caso di errore e backoff esponenziale.

Vantaggi delle scansioni parallele

Molte applicazioni possono trarre vantaggio dall'utilizzo di operazioni `Scan` anziché delle scansioni sequenziali. Ad esempio, un'applicazione che elabora una tabella di dati cronologici di grosse dimensioni può eseguire una scansione parallela molto più velocemente di una scansione sequenziale. Più thread di lavoro in un processo "sweeper" in background possono eseguire la scansione di una tabella con priorità bassa senza influire sul traffico di produzione. In ciascuno di questi esempi, viene utilizzata una `Scan`parallela in modo tale da non privare altre applicazioni delle risorse di velocità effettiva assegnata.

Sebbene le scansioni parallele possano essere utili, possono porre una forte domanda sulla velocità effettiva assegnata. Con una scansione parallela, l'applicazione dispone di più worker che eseguono tutti simultaneamente le operazioni `Scan`. In questo modo, possono consumare rapidamente tutta la capacità di lettura assegnata della tabella. In tal caso, le applicazioni che devono poter accedere alla tabella potrebbero essere limitate.

Una scansione parallela può essere la scelta giusta se sono soddisfatte le seguenti condizioni:

- La dimensione della tabella è pari o superiore a 20 GB.
- La velocità effettiva di lettura assegnata della tabella non viene utilizzata completamente.
- Le operazioni `Scan` sequenziali sono troppo lente.

Scelta TotalSegments

La migliore impostazione per `TotalSegments` dipende dai dati specifici, dalle impostazioni della velocità effettiva assegnata della tabella e dai requisiti di prestazioni. Per trovare la migliore impostazione potrebbe essere necessario sperimentare un po'. Si consiglia di iniziare con un rapporto semplice, ad esempio un segmento per 2 GB di dati. Ad esempio, per una tabella da 30 GB, è possibile impostare `TotalSegments` su 15 (30 GB/2 GB). L'applicazione utilizzerebbe quindi 15 worker, con ogni worker che esegue la scansione di un segmento diverso.

È possibile anche selezionare un valore per `TotalSegments` basato sulle risorse client. È possibile impostare `TotalSegments` su qualsiasi numero compreso tra 1 e 1.000.000 e DynamoDB consentirà la scansione di quel numero di segmenti. Ad esempio, se il client limita il numero di thread che possono essere eseguiti contemporaneamente, è possibile aumentare gradualmente `TotalSegments` fino a ottenere le migliori prestazioni `Scan` con l'applicazione.

Monitorare le scansioni parallele per ottimizzare l'utilizzo della velocità effettiva assegnata, assicurandosi al contempo che le altre applicazioni non siano private delle risorse. Aumentare il valore per `TotalSegments` se non si consuma tutta la velocità effettiva assegnata, ma si verifica comunque una limitazione nelle richieste `Scan`. Riduci il valore per `TotalSegments` se le richieste `Scan` consumano una velocità effettiva assegnata superiore a quella che desideri utilizzare.

Best practice per la progettazione di tabelle DynamoDB

I principi di progettazione generali di Amazon DynamoDB consigliano di mantenere al minimo il numero di tabelle utilizzate. Nella maggior parte dei casi, si consiglia di utilizzare un'unica tabella. Tuttavia, se non è possibile utilizzare un'unica tabella o un numero ridotto di tabelle, queste linee guida possono essere utili.

- Il limite per account non può superare le 10.000 tabelle per account. Se l'applicazione richiede più tabelle, pianificarne la distribuzione su più account. Per ulteriori informazioni, consulta Service Quotas, quote di servizio, account e tabelle in Amazon DynamoDB.
- Valutare i limiti del piano di controllo (control-plane) per le operazioni simultanee su tale piano, che potrebbero influire sulla gestione delle tabelle.
- Collabora con gli architetti delle AWS soluzioni per convalidare i tuoi modelli di progettazione per progetti multi-tenant.

Utilizzo delle tabelle globali DynamoDB

Le tabelle globali sono progettate in base all’impatto globale di Amazon DynamoDB per offrire un database completamente gestito, multi-Regione e multi-attivo in grado di garantire prestazioni di lettura e scrittura rapide e locali per applicazioni globali dimensionate su larga scala. Le tabelle globali replicano automaticamente le tabelle DynamoDB tra quelle di tua scelta. Regioni AWS Non sono necessarie modifiche all'applicazione perché le tabelle globali utilizzano DynamoDB APIs esistente. Non sono previsti costi anticipati o impegni per l'utilizzo delle tabelle globali; vengono infatti addebitati solo i costi per le risorse utilizzate.

In questa guida viene illustrato come utilizzare efficacemente le tabelle globali di DynamoDB. Vengono fornite informazioni chiave sulle tabelle globali, spiegati i casi d’uso principali della funzionalità, descritte le due modalità di coerenza, introdotta una tassonomia di tre diversi modelli di scrittura da prendere in considerazione, vengono illustrate le quattro principali scelte di instradamento delle richieste che è possibile implementare, illustrati i modi per evacuare una Regione attiva o una

Regione offline, viene spiegato come pensare alla pianificazione della capacità di throughput e viene fornito un elenco di aspetti da considerare per l'implementazione delle tabelle globali.

Questa guida si inserisce in un contesto più ampio di implementazioni in più AWS regioni, come illustrato nel white paper AWS Multi-Region Fundamentals e nei modelli di progettazione della resilienza dei dati con video. AWS

Argomenti

- Aspetti chiave sulla progettazione di tabelle DynamoDB globali
- Aspetti chiave della MREC
- Aspetti chiave della MRSC
- Casi d'uso delle tabelle globali MREC DynamoDB
- Modalità di scrittura con le tabelle globali DynamoDB
- Strategie di instradamento in DynamoDB
- Processi di evacuazione
- Pianificazione della capacità di throughput per le tabelle globali DynamoDB
- Elenco di controllo di preparazione per tabelle globali DynamoDB
- Conclusioni e risorse

Aspetti chiave sulla progettazione di tabelle DynamoDB globali

- Esistono due versioni delle tabelle globali: la versione corrente Tabelle globali versione 2019.11.21 (Corrente) (a volte detta "V2") e Tabelle globali versione 2017.11.29 (Legacy) (a volte detta "V1"). Questa guida fa riferimento esclusivamente alla versione corrente.
- DynamoDB (senza tabelle globali) è un servizio Regionale, il che significa che è altamente disponibile e intrinsecamente resiliente ai guasti dell'infrastruttura, inclusi i guasti in un'intera zona di disponibilità. Una tabella DynamoDB a Regione singola è progettata per garantire una disponibilità del 99,99%. Per ulteriori informazioni, consulta l'Accordo sul livello di servizio (SLA).
- Una tabella globale DynamoDB replica i propri dati tra due o più Regioni. Una tabella DynamoDB multi-Regione è progettata per garantire una disponibilità del 99,999%. Con una pianificazione adeguata, le tabelle globali possono contribuire a creare un'architettura resiliente ai guasti a livello Regionale.
- DynamoDB non ha un endpoint globale. Tutte le richieste vengono effettuate a un endpoint Regionale, che accede all'istanza della tabella globale locale per tale Regione.

- Le chiamate a DynamoDB non devono passare tra Regioni. La best practice prevede che un'applicazione che risiede in una Regione acceda direttamente solo all'endpoint DynamoDB locale per tale Regione. Se vengono rilevati problemi all'interno di una Regione, nel livello DynamoDB o nello stack circostante, il traffico degli utenti finali deve essere instradato a un endpoint dell'applicazione diverso ospitato in una Regione diversa. Le tabelle globali assicurano che l'applicazione ospitata in ogni Regione abbia accesso agli stessi dati.

Modalità di coerenza

Quando viene creata una tabella globale, viene configurata la modalità di coerenza. Le tabelle globali supportano due modalità di coerenza: coerenza finale multi-Regione (MREC, Multi-Region Eventual Consistency) ed elevata consistenza multi-Regione (MRSC, Multi-Region Strong Consistency), introdotta nel giugno 2025.

Se non si specifica una modalità di coerenza durante la creazione di una tabella globale, la tabella globale per impostazione predefinita è MREC. Una tabella globale non può contenere repliche configurate con modalità di coerenza diverse. Non è possibile modificare la modalità di coerenza di una tabella globale dopo la sua creazione.

Aspetti chiave della MREC

- Anche le tabelle globali che utilizzano la MRSC impiegano un modello di replica attivo-attivo. Dal punto di vista di DynamoDB, la tabella in ogni regione può accettare richieste di lettura e scrittura. Dopo aver ricevuto una richiesta di scrittura, la tabella di replica locale replicherà l'operazione di scrittura in altre Regioni remote partecipanti in background.
- Gli elementi vengono replicati singolarmente. Gli elementi aggiornati all'interno di una singola transazione potrebbero non venire replicati congiuntamente.
- Ogni partizione di tabella nella Regione di origine replica le proprie operazioni di scrittura in parallelo con ogni altra partizione. La sequenza delle operazioni di scrittura all'interno della Regione remota potrebbe non corrispondere alla sequenza delle operazioni di scrittura eseguite all'interno della Regione di origine. Per ulteriori informazioni sulle partizioni delle tabelle, consulta il post del blog relativo al dimensionamento di DynamoDB e all'impatto sulle prestazioni di partizioni, tasti di scelta rapida e isolamento.
- Un elemento appena scritto viene in genere propagato a tutte le tabelle di replica entro un secondo. La propagazione nelle regioni vicine è in genere più veloce.
- Amazon CloudWatch fornisce una `ReplicationLatency` metrica per ogni coppia di regioni. Viene calcolata in base all'analisi degli elementi in arrivo e al confronto tra la relativa ora di arrivo

e il tempo di scrittura iniziale. Viene infine calcolata una media. Le tempistiche sono memorizzate CloudWatch nella regione di origine. L'analisi dei tempi medi e massimi può essere utile a determinare il ritardo di replica medio e peggiore. Non esiste alcun Accordo sul livello di servizio (SLA) per questa latenza.

- Se un singolo elemento viene aggiornato all'incirca nello stesso momento (all'interno della finestra `ReplicationLatency`) in due Regioni diverse e la seconda operazione di scrittura avviene prima della replica della prima operazione di scrittura, è possibile che si verifichino conflitti di scrittura. Le tabelle globali che utilizzano la MREC risolvono tali conflitti utilizzando un meccanismo basato sulla priorità dell'ultima istanza di scrittura, ovvero sul timestamp delle operazione di scrittura. La prima operazione "perde" rispetto alla seconda operazione. Questi conflitti non vengono registrati in CloudWatch o AWS CloudTrail.
- Il timestamp dell'ultima scrittura viene conservato come proprietà di sistema privata di ciascun elemento. L'approccio basato sulla priorità dell'ultima istanza viene implementato utilizzando un'operazione di scrittura condizionale che richiede che il timestamp dell'elemento in arrivo sia maggiore (cronologicamente successivo) del timestamp dell'elemento esistente.
- Una tabella globale replica tutti gli elementi in tutte le Regioni partecipanti. Per avere ambiti di replica diversi, è possibile creare più tabelle globali e assegnare a ciascuna di esse diverse Regioni partecipanti.
- La Regione locale accetta operazioni di scrittura anche se la Regione di replica è offline o se `ReplicationLatency` aumenta. La tabella locale continua a tentare di replicare gli elementi nella tabella remota finché la replica di ogni elemento non ha esito positivo.
- Nell'improbabile eventualità che una Regione risulti offline, quando in seguito torna online per tutte le repliche in uscita e in entrata in sospeso verranno effettuate ripetizioni di tentativi. Non è richiesta alcuna azione speciale per ripristinare la sincronizzazione delle tabelle. Il meccanismo basato sulla priorità dell'ultima istanza di scrittura garantisce la coerenza finale dei dati.
- È possibile aggiungere una nuova Regione a una tabella MREC DynamoDB in qualsiasi momento. DynamoDB gestisce la sincronizzazione iniziale e la replica continua. È possibile anche rimuovere una Regione (anche la Regione originale), e questo eliminerà la tabella locale in quella Regione.

Aspetti chiave della MRSC

- Anche le tabelle globali che utilizzano la MRSC impiegano un modello di replica attivo-attivo. Dal punto di vista di DynamoDB, la tabella in ogni regione può accettare richieste di lettura e scrittura. Le modifiche agli elementi in una replica di tabella globale MRSC vengono replicate in modo

sincrono in almeno un'altra Regione prima che l'operazione di scrittura restituisca una risposta con esito positivo.

- Le operazioni a elevata consistenza di lettura su qualsiasi replica MRSC restituiscono sempre la versione più recente di un elemento. Le operazioni di scrittura condizionale valutano sempre l'espressione della condizione rispetto alla versione più recente di un elemento. Gli aggiornamenti funzionano sempre sulla base della versione più recente di un elemento.
- Le operazioni di lettura a coerenza finale su una replica MRSC potrebbero non includere le modifiche apportate di recente in un'altra Regione e potrebbero non includere nemmeno le modifiche avvenute di recente nella stessa Regione.
- Un'operazione di scrittura ha esito negativo con un'eccezione `ReplicatedWriteConflictException` quando tenta di modificare un elemento che è già stato modificato in un'altra Regione. Per le operazioni di scrittura che hanno esito negativo con l'eccezione `ReplicatedWriteConflictException` è possibile effettuare ripetizioni di tentativi. Queste avranno esito positivo se l'elemento non viene più modificato in un'altra Regione.
- Con MRSC, le latenze sono più elevate per le operazioni di scrittura e per le operazioni a elevata consistenza di lettura. Queste operazioni richiedono una comunicazione multi-Regione. Questa comunicazione può aggiungere una latenza che aumenta in base alla latenza del round trip tra la Regione a cui si accede e la Regione più vicina che partecipa alla tabella globale. Per ulteriori informazioni, consulta la presentazione di AWS re:Invent 2024, Forte coerenza multiregionale con le tabelle globali DynamoDB. Le operazioni di lettura a coerenza finale non comportano alcuna latenza aggiuntiva. Esiste uno strumento di test open source che consente di calcolare sperimentalmente queste latenze con le proprie Regioni.
- Gli elementi vengono replicati singolarmente. Le tabelle globali che utilizzano MRSC non supportano la transazione. APIs
- Una tabella globale MRSC deve essere implementata esattamente in tre Regioni. È possibile configurare una tabella globale MRSC con tre repliche o con due repliche e un testimone. Un testimone è un componente di una tabella globale MRSC che contiene dati recenti scritti su repliche di tabelle globali. Un testimone fornisce un'alternativa opzionale a una replica completa, supportando al contempo l'architettura di disponibilità di MRSC. Non è possibile eseguire operazioni di lettura o scrittura su un testimone. Un testimone non sostiene costi di archiviazione o scrittura. Un testimone si trova in una Regione diversa dalle due repliche.
- Per creare una tabella globale MRSC, si aggiungono una replica e un testimone oppure si aggiungono due repliche a una tabella DynamoDB esistente che non contiene dati. Non è possibile aggiungere repliche aggiuntive a una tabella globale MRSC esistente. Non è possibile eliminare una singola replica o un testimone da una tabella globale MRSC. È possibile eliminare due repliche

o eliminare una replica e un testimone da una tabella globale MRSC. Il secondo scenario converte la replica rimanente in una tabella DynamoDB a Regione singola.

- È possibile determinare se una tabella globale MRSC ha un testimone configurato e in quale regione è configurato, dall'output dell' DescribeTable API. Il testimone è di proprietà e gestito da DynamoDB e non appare nella regione in cui è Account AWS configurato.
- Le tabelle globali MRSC sono disponibili nei seguenti set di Regioni:
  - Set della Regione degli Stati Uniti: Stati Uniti orientali (Virginia settentrionale), Stati Uniti orientali (Ohio), Stati Uniti occidentali (Oregon)
  - Set della Regione Europa: Europa (Irlanda), Europa (Londra), Europa (Parigi), Europa (Francoforte)
  - Set della Regione Asia Pacifico: Asia Pacifico (Tokyo), Asia Pacifico (Seoul) e Asia Pacifico (Osaka)
- Le tabelle globali MRSC non possono estendersi su set di Regioni. Ad esempio, una tabella globale MRSC non può contenere repliche di set di Regioni degli Stati Uniti e dell’UE.
- Il valore Time to Live (TTL) non è supportato per le tabelle globali MRSC.
- Gli indici secondari locali (LSIs) non sono supportati per le tabelle globali MRSC.
- CloudWatch Le informazioni di Contributor Insights vengono riportate solo per la regione in cui si è verificata un'operazione.
- La Regione locale accetta tutte le operazioni di lettura e scrittura purché esista una seconda Regione che ospita una replica o un testimone per stabilire il quorum. Se una seconda Regione non è disponibile, la Regione locale può fornire solo letture a coerenza finale.
- Nel caso improbabile che una Regione vada completamente offline, quando tornerà online si aggiornerà automaticamente. Fino a quando non verranno recuperate, le operazioni di scrittura e le operazioni di lettura fortemente coerenti solo nella regione che sta recuperando restituiranno errori, mentre le richieste verso le altre regioni continueranno a funzionare normalmente. Alla fine, operazioni di lettura coerenti nella regione di recupero restituiranno i dati che finora sono stati propagati nella regione, con il consueto comportamento di coerenza locale tra il nodo leader e le repliche locali. Non è richiesta alcuna azione speciale per ripristinare la sincronizzazione delle tabelle.

Casi d’uso delle tabelle globali MREC DynamoDB

Le tabelle globali MREC sono caratterizzate dai seguenti vantaggi comuni:

- Operazioni di lettura a bassa latenza. Posiziona una copia dei dati più vicino all'utente finale per ridurre la latenza di rete durante le operazioni di lettura. L'aggiornamento dei dati è in linea con l'aggiornamento del valore `ReplicationLatency`.
- Operazioni di scrittura a bassa latenza. È possibile scrivere in una regione vicina per ridurre la latenza di rete e il tempo impiegato per eseguire la scrittura. Il traffico di scrittura deve essere instradato con attenzione per garantire l'assenza di conflitti. Le tecniche di routing sono descritte in dettaglio in Strategie di instradamento in DynamoDB.
- Migrazione regionale senza interruzioni. È possibile aggiungere una nuova Regione ed eliminare quella vecchia per eseguire la migrazione di un'implementazione da una Regione all'altra senza tempi di inattività a livello di dati.

Le tabelle globali MREC e MRSC offrono entrambe questo vantaggio:

- Resilienza e ripristino di emergenza migliorati. Se una Regione presenta prestazioni ridotte o un'interruzione totale, è possibile evacuarla. Evacuare significa spostare alcune o tutte le richieste destinate a quella Regione. L'utilizzo delle tabelle globali aumenta la disponibilità dello SLA di DynamoDB per il tempo di attività mensile dal 99,99% al 99,999%. L'utilizzo della MREC supporta un Obiettivo del punto di ripristino (RPO) e un Obiettivo del tempo di ripristino (RTO) misurati in secondi. L'utilizzo della MRSC supporta un RPO pari a zero.

  Ad esempio, Fidelity Investments ha presentato a re:Invent 2022 la modalità di utilizzo delle tabelle globali DynamoDB per il proprio sistema di gestione degli ordini. L'obiettivo era ottenere un'elaborazione affidabile a bassa latenza su una scala che non sarebbe possibile raggiungere con l'elaborazione on-premises, mantenendo al contempo la resilienza ai guasti nelle zone di disponibilità e nelle aree di disponibilità.

Se l'obiettivo è la resilienza e il disaster recovery, le tabelle MRSC hanno latenze di scrittura superiori e latenze per le operazioni a elevata consistenza di lettura più elevate, ma supportano un RPO pari a zero. Le tabelle globali MREC supportano un RPO pari al ritardo di replica tra le repliche, in genere di pochi secondi a seconda delle Regioni di replica.

Modalità di scrittura con le tabelle globali DynamoDB

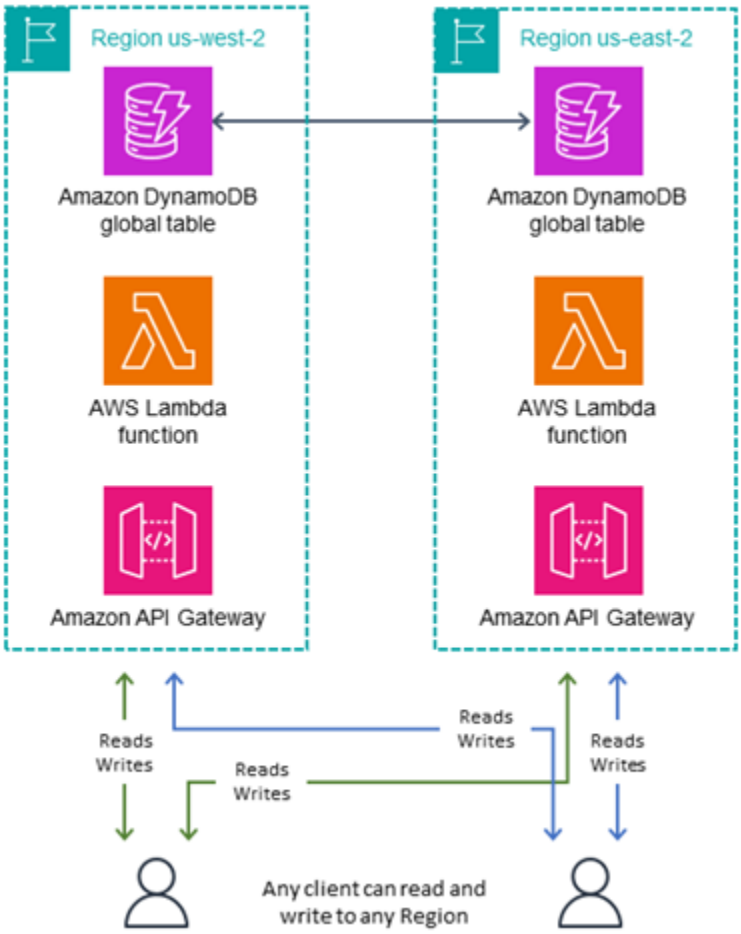
Le tabelle globali utilizzano sempre un modello di replica attivo-attivo a livello di tabella. Tuttavia, in particolare per le tabelle MREC, è possibile considerarle basate su un modello attivo-passivo mediante il controllo della modalità di instradamento delle richieste di scrittura. Ad esempio, è

possibile decidere di instradare le richieste di scrittura a un’unica Regione per evitare potenziali conflitti di scrittura che possono verificarsi con le tabelle MREC.

Esistono tre modelli principali di scrittura gestita, come spiegato nelle tre sezioni successive. È consigliabile valutare il modello di scrittura più adatto a uno specifico caso d'uso. Questa scelta influisce sulle modalità di instradamento delle richieste, evacuazione di una regione e gestione del ripristino di emergenza. Le indicazioni nelle sezioni successive dipendono dalla modalità di scrittura dell’applicazione.

Modalità di scrittura in qualsiasi regione (non primaria)

La modalità di scrittura in qualsiasi Regione, illustrata nel diagramma seguente, si basa su un modello di replica attivo-attivo e non impone restrizioni su dove può avvenire la scrittura. Qualsiasi regione può accettare un'operazione di scrittura in qualsiasi momento. Questa è la modalità più semplice, ma può essere utilizzata solo con alcuni tipi di applicazioni. Questa modalità è adatta a tutte le tabelle MRSC. È adatta anche per le tabelle MREC quando tutte le istanze di scrittura sono idempotenti e quindi ripetibili in modo sicuro in modo che le operazioni di scrittura simultanee o ripetute tra Regioni non siano in conflitto. Ad esempio, quando un utente aggiorna i propri dati di contatto. Questa modalità è adatta anche per un caso speciale di idempotenza, ovvero un set di dati di tipo "solo accodamento", in cui tutte le scritture sono inserimenti univoci sotto una chiave primaria deterministica. Infine, questa modalità è adatta per la MREC nei casi in cui il rischio di scritture in conflitto sia accettabile.



La modalità scrittura in qualsiasi regione rappresenta l'architettura più semplice da implementare. L'instradamento è più semplice perché qualsiasi regione può essere la destinazione delle operazioni di scrittura in qualsiasi momento. Il failover è più semplice, perché con le tabelle MRSC gli elementi sono sempre sincronizzati e con le tabelle MRSC, qualsiasi scrittura recente può essere riprodotta un numero qualsiasi di volte in qualsiasi Regione secondaria. Laddove possibile, è consigliabile usare questa modalità di scrittura nella fase di progettazione.

Ad esempio, vari servizi di streaming video utilizzano tabelle globali per tenere traccia di segnalibri, recensioni, flag relativi allo stato di visualizzazione e così via. Queste implementazioni utilizzano tabelle MREC perché necessitano di repliche di tipo sparse in tutto il mondo, con ogni replica che fornisce operazioni di lettura e scrittura a bassa latenza. Queste implementazioni possono utilizzare la modalità di scrittura in qualsiasi Regione purché assicurino che ogni operazione di scrittura sia idempotente. Questo si verifica se ogni aggiornamento, ad esempio l'impostazione di un nuovo timecode più aggiornato, l'assegnazione di una nuova recensione o l'impostazione di un nuovo stato di visualizzazione, assegna direttamente il nuovo stato dell'utente e il successivo valore corretto per un elemento non dipende dal valore corrente. Se, per caso, le richieste di scrittura dell'utente

vengono instradate a Regioni diverse, l'ultima operazione di scrittura risulterà persistente e lo stato globale verrà adeguato in base all'ultima assegnazione. Le operazioni di lettura in questa modalità risulteranno a coerenza finale, con un ritardo in base al valore `ReplicationLatency` più recente.

In un altro esempio, una società di servizi finanziari utilizza tabelle globali come parte di un sistema per il conteggio continuo degli acquisti con carta di debito per ogni cliente, per calcolare il valore del cashback per il cliente specifico. L'intenzione è conservare un elemento `RunningBalance` per cliente. Questa modalità di scrittura non è per natura idempotente perché, man mano che le transazioni arrivano, modificano il saldo attraverso un'espressione `ADD` in cui il nuovo valore corretto dipende dal valore corrente. Utilizzando le tabelle MRSC è comunque possibile scrivere in qualsiasi Regione, poiché ogni chiamata `ADD` opera sempre in base al valore più recente dell'elemento.

Un terzo esempio riguarda un'azienda che fornisce servizi di inserimento di annunci online. Questa azienda ha deciso che è accettabile un basso rischio di perdita di dati per ottenere le semplificazioni di progettazione derivanti dalla modalità di scrittura in qualsiasi Regione. Alla pubblicazione degli annunci, sono disponibili solo pochi millisecondi per recuperare metadati sufficienti per determinare l'annuncio da mostrare e quindi registrare l'impression di tale annuncio in modo da non ripeterlo nel giro di poco tempo. L'azienda impiega le tabelle globali per ottenere sia operazioni di lettura a bassa latenza per gli utenti finali di tutto il mondo che operazioni di scrittura a bassa latenza. L'azienda registra tutte le impression degli annunci di un utente all'interno di un singolo elemento, che è rappresentato come un elenco crescente. Utilizza un solo elemento anziché aggiungerlo a una raccolta di elementi, così che le impression meno recenti possano essere rimosse come parte di ogni operazione di scrittura senza dover pagare tale operazione. Questa operazione di scrittura NON è idempotente; se lo stesso utente finale vede annunci pubblicati in più Regioni all'incirca nello stesso momento, è possibile che l'operazione di scrittura di un'impression possa sovrascrivere l'altra. Il rischio è che un utente veda un annuncio ripetuto di tanto in tanto, ma l'azienda ha deciso che l'opzione è accettabile.

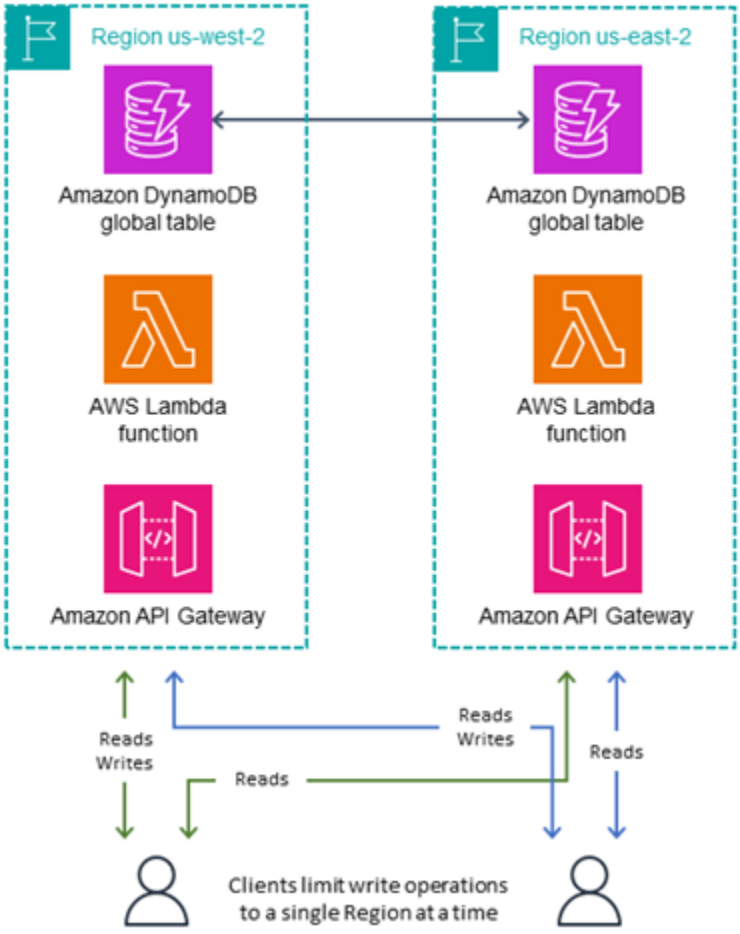
Scrittura in una Regione (unica primaria)

La modalità di scrittura in una Regione, illustrata nel diagramma seguente, si basa su un modello di replica attivo-passivo e instrada tutte le scritture a livello di tabelle in un'unica Regione attiva. Si noti che DynamoDB non ha "percezione" del concetto di regione attiva; l'instradamento delle applicazioni esternamente a DynamoDB gestisce questo scenario. La modalità di scrittura in una Regione è molto utile per tabelle MREC che devono evitare i conflitti di scrittura assicurando che le operazioni di scrittura vengano instradate verso solo una Regione alla volta. Questa modalità di scrittura è utile quando si desidera utilizzare espressioni condizionali e non è possibile utilizzare la MRSC per qualche motivo, o quando è necessario eseguire transazioni. Queste espressioni non sono possibili

a meno di non essere consapevoli di stare agendo sulla base dei dati più recenti, quindi richiedono l'invio di tutte le richieste di scrittura a un'unica Regione con i dati più recenti.

Quando si utilizza una tabella MRSC, è possibile scegliere di scrivere in genere in un'unica Regione per comodità. Ad esempio, questo può aiutare a ridurre al minimo lo sviluppo dell'infrastruttura oltre a DynamoDB. La modalità di scrittura sarebbe comunque quella di scrivere su qualsiasi Regione, perché con la MRSC è possibile scrivere in tutta sicurezza in qualsiasi Regione in qualsiasi momento senza preoccuparsi della risoluzione dei conflitti che farebbe sì che le tabelle MREC decidano di scrivere in una Regione.

Le letture a consistenza finale possono essere eseguite in qualsiasi regione di replica per ottenere latenze più basse. Le operazioni a elevata consistenza di lettura devono essere instradate all'unica Regione primaria.



A volte è necessario modificare la Regione attiva in risposta a un guasto a livello Regionale. Alcuni utenti modificano la regione attualmente attiva secondo una pianificazione regolare, ad esempio implementando una distribuzione. follow-the-sun In questo modo la Regione attiva si colloca vicino

all'area geografica con la maggiore attività (di solito dove è giorno, da qui il nome), il che si traduce nelle operazioni di lettura e scrittura con la latenza più bassa. L'approccio ha anche il vantaggio di chiamare quotidianamente il codice di modifica della Regione, assicurandosi che sia ben testato prima di eventuali disaster recovery.

Le Regioni passive possono mantenere un set sottodimensionato dell'infrastruttura DynamoDB, che potrà essere potenziato solo quando la Regione dovesse diventare attiva. Questa guida non copre i modelli pilot light e warm standby. Per ulteriori informazioni, vedere Disaster Recovery (DR) Architecture on AWS, Parte III: Pilot Light and Warm Standby.

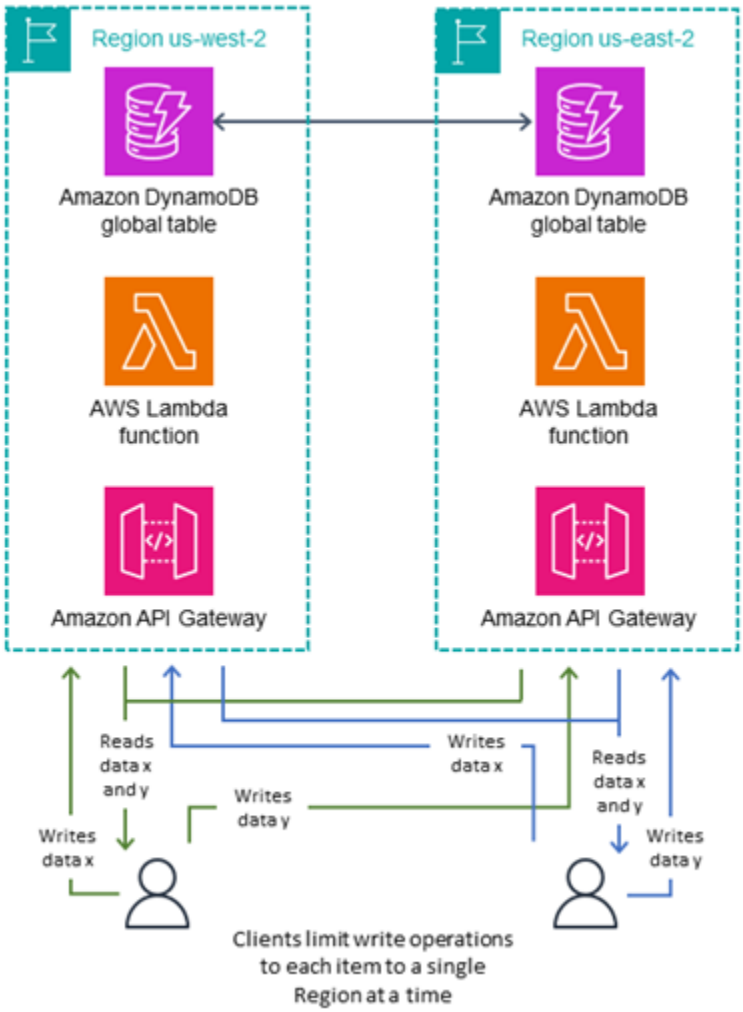
L'utilizzo della modalità di scrittura in una Regione funziona bene quando si utilizzano tabelle globali per operazioni di lettura distribuite a livello globale a bassa latenza. Un esempio è rappresentato da una grande società di social media che deve disporre degli stessi dati di riferimento in tutte le Regioni del mondo. Non aggiorna spesso i dati, ma quando lo fa, scrive in una sola Regione per evitare potenziali conflitti di scrittura. Le operazioni di lettura sono sempre consentite da qualsiasi Regione.

Come altro esempio, si consideri l'azienda di servizi finanziari discussa in precedenza che ha implementato un calcolo del cashback giornaliero. Usa la modalità di scrittura in qualsiasi Regione per calcolare il saldo, ma scrive in una Regione per tenere traccia dei pagamenti. Questo lavoro richiede transazioni che non sono supportate nelle tabelle MRSC, pertanto funziona meglio con una tabella MREC separata e la modalità di scrittura in una Regione.

Scrittura nella propria Regione (primaria mista)

La modalità di scrittura nella propria Regione, illustrata nel diagramma seguente, funziona con le tabelle MREC. Assegna diversi sottoinsiemi di dati a diverse Regioni di origine e consente operazioni di scrittura su un elemento solo tramite la relativa Regione di origine. Questa modalità è attiva-passiva ma assegna la Regione attiva in base all'elemento. Ogni Regione è primaria per il proprio set di dati non sovrapposto e le operazioni di scrittura devono essere protette per garantire la corretta localizzazione.

Questa modalità è simile alla modalità di scrittura in una Regione, tranne per il fatto che consente operazioni di scrittura a latenza inferiore, poiché i dati associati a ciascun utente possono essere collocati in prossimità della rete più vicina a quell'utente. Inoltre, distribuisce l'infrastruttura circostante in modo più uniforme tra le Regioni e richiede meno lavoro per ricostruire l'infrastruttura durante uno scenario di failover, poiché tutte le Regioni hanno una parte della propria infrastruttura già attiva.



È possibile determinare la Regione di origine degli elementi in diversi modi:

- Modalità intrinseca: alcuni aspetti dei dati, ad esempio un attributo speciale o un valore incorporato nella relativa chiave di partizione, rendono chiara la Regione di origine. Questa tecnica è descritta nel post del blog relativo all'uso del pinning della Regione per impostare una Regione di origine per gli elementi in una tabella globale di Amazon DynamoDB.
- Modalità negoziata: l'area di origine di ogni set di dati viene negoziata esternamente, ad esempio con un servizio globale distinto che gestisce le assegnazioni. L'assegnazione può avere una durata limitata, trascorsa la quale è soggetta a rinegoziazione.
- Modalità orientata alla tabella: anziché creare una singola tabella globale di replica, si crea lo stesso numero di tabelle globali quante sono le Regioni di replica. Il nome di ogni tabella fa riferimento alla regione di origine. Nelle operazioni standard, tutti i dati vengono scritti nella regione di origine, mentre le altre regioni ne conservano una copia di sola lettura. Durante un failover, un'altra Regione adotta temporaneamente le funzioni di scrittura per tale tabella.

Ad esempio, si supponga di lavorare per una società di videogiochi. Deve essere disponibile una bassa latenza per le operazioni di lettura e scrittura per tutti i giocatori di tutto il mondo. A ciascun giocatore viene assegnata la Regione più vicina. Quella regione esegue tutte le operazioni di lettura e scrittura, garantendo una forte read-after-write coerenza. Tuttavia, quando il giocatore viaggia o la sua Regione di origine subisce un'interruzione dei servizi, una copia completa dei suoi dati sarà disponibile in altre Regioni e per il giocatore si verificherà l'assegnazione di una Regione diversa.

Per fare un altro esempio, si supponga di lavorare in un'azienda di servizi di videoconferenza. I metadati di ogni chiamata di conferenza vengono assegnati a una Regione specifica. I partecipanti (chiamanti) possono utilizzare la Regione più vicina a loro in modo da avere la latenza più bassa. In caso di interruzione dei servizi in tale Regione, l'utilizzo delle tabelle globali consente un ripristino rapido poiché il sistema può spostare l'elaborazione della chiamata in un'altra Regione in cui è già presente una copia replicata dei dati.

Riepilogo

- La modalità di scrittura in qualsiasi Regione è adatta per le tabelle MRSC e le chiamate idempotenti alle tabelle MREC.
- La modalità di scrittura in una Regione è adatta per le chiamate non idempotenti alle tabelle MREC.
- La modalità di scrittura nella propria Regione è adatta per chiamate non idempotenti alle tabelle MREC, in cui è importante che i client scrivano in una Regione vicina a loro.

Strategie di instradamento in DynamoDB

Forse la parte più complessa di una implementazione di tabelle globali è la gestione dell'instradamento delle richieste. Le richieste devono prima essere inviate da un utente finale a una regione scelta e destinataria dell'instradamento. La richiesta incontra alcuni stack di servizi in quella regione, tra cui un livello di calcolo che forse consiste in un sistema di bilanciamento del carico supportato da una AWS Lambda funzione, un contenitore o un nodo Amazon Elastic Compute Cloud (Amazon EC2) e possibilmente altri servizi tra cui un altro database. Questo livello di calcolo comunica con DynamoDB mediante l'endpoint locale per tale regione. I dati nella tabella globale vengono replicati in tutte le altre regioni partecipanti e ogni regione ha uno stack di servizi simile attorno alla propria tabella DynamoDB.

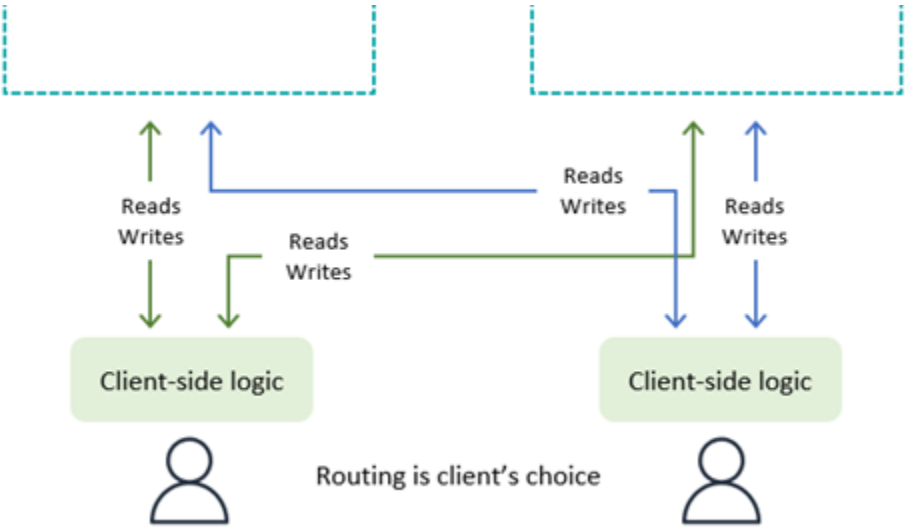
A ogni stack nelle varie regioni la tabella globale fornisce una copia locale degli stessi dati. È possibile considerare l'ipotesi di progettare un unico stack in un'unica regione e prevedere di effettuare chiamate remote all'endpoint DynamoDB di una regione secondaria in caso di problemi

con la tabella DynamoDB locale. Questa non è una best practice. Se c'è un problema in una Regione causato da DynamoDB (o, più probabilmente, causato da qualcos'altro nello stack o da un altro servizio che dipende da DynamoDB), è meglio indirizzare l'utente finale verso un'altra Regione per l'elaborazione e utilizzare il livello di calcolo dell'altra Regione, che comunicherà con l'endpoint DynamoDB locale. Questo approccio riguarda interamente la Regione problematica. Per garantire la resilienza, è necessario eseguire la replica su più Regioni, con la replica dei livelli di calcolo e dati.

Esistono numerose tecniche alternative per instradare una richiesta dell'utente finale a una regione per l'elaborazione. La scelta ottimale dipende dalla modalità di scrittura e dalle considerazioni relative al failover. Questa sezione illustra quattro opzioni di instradamento: basato sul client, livello di calcolo, Route 53 e Global Accelerator.

Instradamento delle richieste basato sul client

Con il routing delle richieste basato sul client, illustrato nel diagramma seguente, il client dell'utente finale (un'applicazione, una pagina Web con o un altro client) tiene traccia degli endpoint applicativi validi (ad esempio JavaScript, un endpoint Amazon API Gateway anziché un endpoint DynamoDB letterale) e utilizza la propria logica incorporata per scegliere la regione con cui comunicare. Potrebbe scegliere in base alla selezione casuale, alle latenze più basse rilevate, alle misurazioni della larghezza di banda più elevata rilevate o ai controlli di integrità eseguiti localmente.



Il vantaggio dell'instradamento delle richieste basato sul client è che può adattarsi relativamente a fattori come le condizioni reali del traffico Internet pubblico e che pertanto può cambiare Regione in caso di peggioramento delle prestazioni. Il client deve conoscere tutti i potenziali endpoint, ma il lancio di un nuovo endpoint regionale non è un evento frequente.

Con la modalità di scrittura in qualsiasi Regione, un client può selezionare unilateralmente il suo endpoint preferito. Se il suo accesso a una regione viene compromesso, il client può reindirizzare le richieste a un altro endpoint.

Con la modalità scrittura in una regione, il client avrà bisogno di un meccanismo per instradare le sue operazioni di scrittura alla regione attualmente attiva. Questa operazione potrebbe essere semplice, come verificare empiricamente quale Regione accetta le scritture rilevando eventuali errori di scrittura e ricorrendo eventualmente a un'alternativa, oppure complessa, come chiamare un coordinatore globale per richiedere lo stato corrente dell'applicazione (basato sul controllo di instradamento del Sistema di controllo Amazon per il ripristino delle applicazioni (ARC) che fornisce un sistema basato su quorum a 5 Regioni per mantenere lo stato globale per esigenze di questo tipo). Il client può decidere se le letture possono essere instradate a una regione qualsiasi per ottenere un'eventuale coerenza o se devono essere indirizzate alla regione attiva per una maggiore coerenza. Per ulteriori informazioni, consulta l'argomento relativo al funzionamento di Route 53.

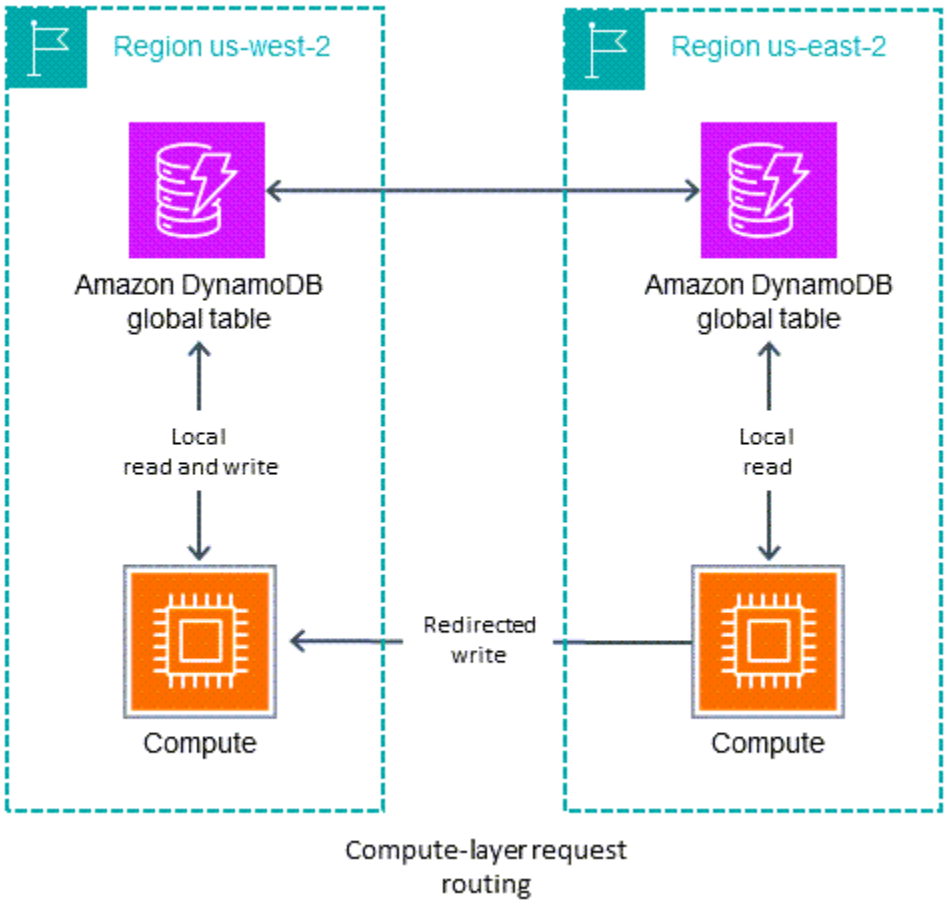
Con la modalità di scrittura nella propria regione, il client deve determinare la regione di origine del set di dati che sta usando. Ad esempio, se il client corrisponde a un account utente e ogni account utente si trova in una regione specifica, il client può richiedere l'endpoint appropriato da un sistema di accesso globale.

Ad esempio, una società di servizi finanziari che aiuta gli utenti a gestire le proprie finanze aziendali tramite il Web potrebbe utilizzare tabelle globali con la modalità di scrittura nella propria regione. Ogni utente deve accedere a un servizio centrale. Tale servizio restituisce le credenziali e l'endpoint per la regione in cui tali credenziali funzioneranno. Le credenziali sono valide per un breve periodo. Successivamente, la pagina web negozia automaticamente un nuovo accesso, che offre l'opportunità di reindirizzare potenzialmente l'attività dell'utente verso una nuova regione.

Instradamento delle richieste al livello di calcolo

Con l'instradamento delle richieste al livello di calcolo, illustrato nel seguente diagramma, il codice in esecuzione nel livello di calcolo stabilisce se elaborare la richiesta localmente o passarla a una copia di se stesso in esecuzione in un'altra Regione. Quando si utilizza la modalità di scrittura in una Regione, il livello di calcolo può rilevare che la Regione non è la Regione attiva e consentire operazioni di lettura locali mentre inoltra tutte le operazioni di scrittura a un'altra Regione. Questo codice al livello di calcolo deve conoscere la topologia dei dati e le regole di instradamento e applicarle in modo affidabile in base alle impostazioni più recenti che specificano le Regioni attive e i dati da utilizzare. Lo stack software esterno all'interno della Regione non deve conoscere il modo in cui il microservizio instrada le richieste di lettura e scrittura. In una progettazione affidabile, la regione

ricevente verifica se è la regione primaria corrente per l'operazione di scrittura. In caso contrario, genera un errore che indica che lo stato globale deve essere corretto. La regione ricevente potrebbe anche memorizzare l'operazione di scrittura nel buffer per un breve intervallo, se la regione primaria è in fase di modifica. In ogni caso, lo stack di calcolo in una regione effettua la scrittura solo sul proprio endpoint DynamoDB locale, ma gli stack di calcolo potrebbero comunicare tra loro.

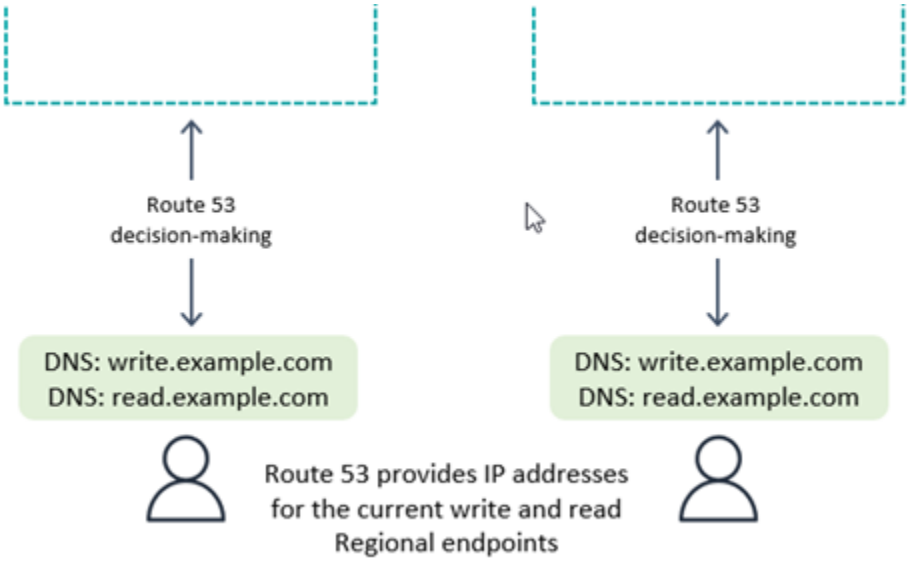


Il Vanguard Group utilizza un sistema chiamato Global Orchestration and Status Tool (GOaST) e una libreria chiamata Global Multi-Region library (GMRlib) per questo processo di routing, presentato a re:Invent 2022. follow-the-sunUsano un unico modello primario. GOaST mantiene lo stato globale, in modo simile al controllo di routing ARC discusso nella sezione precedente. Viene utilizzata una tabella globale per monitorare la Regione primaria e controllare quando è programmato il passaggio alla Regione primaria successiva. Tutte le operazioni di lettura e scrittura vengono eseguite GMRlib, il che si coordina con GOa ST. GMRlib consente di eseguire le operazioni di lettura localmente, a bassa latenza. Per le operazioni di scrittura, GMRlib verifica se la regione locale è la regione principale corrente. In caso affermativo, l'operazione di scrittura viene completata direttamente. In caso contrario, GMRlib inoltra l'attività di scrittura GMRlib alla regione principale. La libreria ricevente conferma di essere la regione primaria e genera un errore in caso contrario, il che genera un ritardo

di propagazione con lo stato globale. Questo approccio offre un vantaggio in termini di convalida in quanto non viene effettuata una scrittura diretta su un endpoint DynamoDB remoto.

Instradamento delle richieste Route 53

Il sistema ARC è una tecnologia DNS (Domain Name Service). Con Route 53, il client richiede il proprio endpoint mediante la ricerca di un nome di dominio DNS noto; Route 53 restituisce l'indirizzo IP corrispondente agli endpoint regionali ritenuti più appropriati. Il diagramma seguente ne è l'illustrazione. Route 53 dispone di un lungo elenco di policy di instradamento che utilizza per determinare la Regione appropriata. Può anche eseguire il routing di failover per deviare il traffico dalle Regioni in cui non vengono superati i controlli dell'integrità.



Con la modalità di scrittura in qualsiasi regione o combinato con l'instradamento delle richieste al livello di calcolo sul backend, Route 53 può avere l'accesso completo per restituire la regione in base a regole interne complesse come la regione più vicina alla rete o la prossimità geografica più vicina o qualsiasi altra scelta.

Con la modalità di scrittura in una Regione, Route 53 può essere configurato in modo da restituire la Regione attualmente attiva (utilizzando il sistema ARC Route 53). Se il client desidera connettersi a una Regione passiva (ad esempio, per operazioni di lettura), potrebbe cercare un nome DNS diverso.

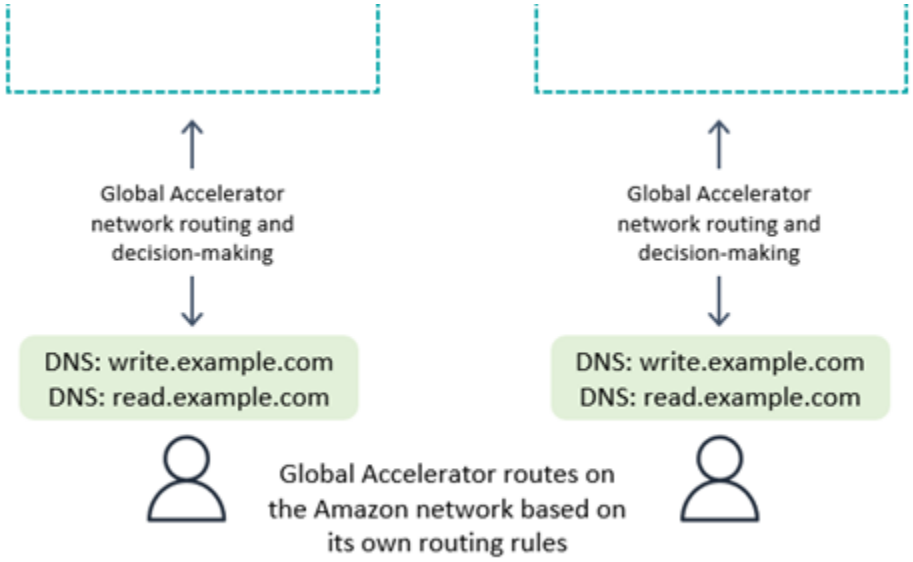
> **Note**
>
> I client memorizzano nella cache gli indirizzi IP nella risposta di Route 53 per il periodo di tempo specificato nell'impostazione dell'opzione Time to Live (TTL) sul nome di dominio. Un TTL più lungo estende l'obiettivo del tempo di ripristino (RTO) affinché tutti i client

riconoscano il nuovo endpoint. Un valore di 60 secondi è tipico per l'utilizzo del failover. Non tutti i software rispettano perfettamente la scadenza del TTL del DNS e potrebbero esserci più livelli di caching DNS, ad esempio a livello di sistema operativo, macchina virtuale e applicazione.

Con la modalità di scrittura nella propria regione, è consigliabile evitare l'uso di Route 53 a meno che non si utilizzi anche l'instradamento delle richieste al livello di calcolo.

Instradamento delle richieste Global Accelerator

Con AWS Global Accelerator, illustrato nel diagramma seguente, un client cerca il nome di dominio noto in Route 53. Tuttavia, invece di recuperare un indirizzo IP corrispondente a un endpoint regionale, il client riceve un indirizzo IP statico anycast che indirizza verso la edge location più vicina AWS . A partire da tale posizione edge, tutto il traffico viene instradato sulla rete privata AWS e verso alcuni endpoint (come un sistema di bilanciamento del carico o un Gateway API) in una regione scelta in base alle regole di instradamento gestite in Global Accelerator. Rispetto all'instradamento basato sulle regole Route 53, l'instradamento delle richieste Global Accelerator presenta latenze inferiori perché riduce la quantità di traffico sulla rete Internet pubblica. Inoltre, poiché Global Accelerator non dipende dalla scadenza del TTL del DNS per modificare le regole di instradamento, può modificare l'instradamento più rapidamente.



Con la modalità di scrittura in qualsiasi regione o combinato con l'instradamento delle richieste al livello di calcolo sul backend, Global Accelerator funziona senza problemi. Il client si connette alla posizione edge più vicina e non deve preoccuparsi di quale regione riceve la richiesta.

Con la modalità di scrittura in una regione, le regole di instradamento di Global Accelerator devono inviare le richieste alla regione attualmente attiva. È possibile utilizzare i controlli dell'integrità per segnalare artificialmente un guasto in qualsiasi regione non considerata dal sistema globale come regione attiva. Come con il DNS, è possibile utilizzare un nome di dominio DNS alternativo per instradare le richieste di lettura se le richieste possono provenire da qualsiasi regione.

Con la modalità di scrittura nella propria regione, è consigliabile evitare l'uso di Global Accelerator a meno che non si utilizzi anche l'instradamento delle richieste al livello di calcolo.

Processi di evacuazione

L’evacuazione di una Regione è il processo di migrazione delle attività, solitamente delle attività di lettura e scrittura, o delle attività di lettura, dalla Regione.

Evacuazione di una regione in tempo reale

Potresti decidere di evacuare una regione attiva per diversi motivi: come parte della normale attività aziendale (ad esempio, se utilizzi una modalità di scrittura in una regione) follow-the-sun, a causa della decisione aziendale di modificare la regione attualmente attiva, in risposta a guasti nello stack software esterno a DynamoDB o perché stai riscontrando problemi generali come latenze più elevate del solito all'interno della regione.

Con la modalità di scrittura in qualsiasi regione, l'evacuazione di una regione in tempo reale è semplice. È possibile instradare il traffico verso le Regioni alternative tramite qualsiasi sistema di instradamento e lasciare che le operazioni di scrittura della Regione evacuata si replichino come al solito.

Le modalità di scrittura in una Regione e quella di scrittura nella propria Regione vengono generalmente utilizzate con le tabelle MREC. Pertanto, è necessario assicurarsi che tutte le operazioni di scrittura nella nuova Regione attiva siano state completamente registrate, elaborate in streaming e propagate globalmente prima di iniziare le operazioni di scrittura nella nuova Regione attiva, per garantire che le future operazioni di scrittura vengano elaborate sulla versione più recente dei dati.

Si supponga che la regione A sia attiva e la regione B sia passiva (per la tabella completa o per gli elementi che si trovano nella regione A). Il meccanismo tipico per eseguire un'evacuazione consiste nel sospendere le operazioni di scrittura nella Regione A, attendere il tempo necessario affinché tali operazioni si siano propagate completamente nella Regione B, aggiornare lo stack dell'architettura per riconoscere la Regione B come attiva e quindi riprendere le operazioni di scrittura

nella Regione B. Non esiste una metrica che indichi con assoluta certezza che la Regione A ha replicato completamente i propri dati nella Regione B. Se la Regione A è integra, la sospensione delle operazioni di scrittura nella regione A e l'attesa di 10 volte il valore massimo recente della metrica `ReplicationLatency` sarebbero in genere sufficienti per determinare che la replica è completa. Se la Regione A non è integra e mostra altre aree con latenze aumentate, per definire il tempo di attesa scegliere un valore multiplo più grande.

Evacuazione di una regione offline

Esiste un caso speciale da considerare: cosa succede se la Regione A risulta inaspettatamente offline? Ciò è estremamente improbabile, ma va comunque considerato.

Evacuazione di una tabella MRSC offline

Se ciò accade con una tabella MRSC, non è necessario fare nulla di particolare. Le tabelle MRSC supportano un obiettivo del punto di ripristino (RPO) pari a zero. Tutte le operazioni di scrittura eseguite correttamente sulla tabella MRSC nella Regione offline saranno disponibili in tutte le altre tabelle delle Regioni, quindi non vi è alcuna potenziale lacuna nei dati anche se la Regione passa completamente offline senza preavviso. Le aziende possono continuare a utilizzare le repliche situate nelle altre Regioni.

Evacuazione di una tabella MREC offline

In questo caso, tutte le operazioni di scrittura nella Regione A non ancora propagate vengono conservate e propagate dopo che la Regione A torna online. Le operazioni di scrittura non vengono perse, ma la loro propagazione viene ritardata a tempo indeterminato.

Come procedere in questo caso dipende dall'applicazione. Per la continuità aziendale, potrebbe essere necessario procedere alle operazioni di scrittura nella nuova Regione B. Tuttavia, se un elemento nella Regione B riceve un aggiornamento mentre è in corso la propagazione di un'operazione di scrittura per tale elemento dalla Regione A, la propagazione viene soppressa in base al modello basato sulla priorità dell'ultima istanza di scrittura. Qualsiasi aggiornamento nella Regione B potrebbe eliminare una richiesta di scrittura in arrivo.

Con la modalità di scrittura in qualsiasi Regione, le operazioni di lettura e scrittura possono continuare nella Regione B, con la certezza che gli elementi nella Regione A alla fine si propagheranno alla Regione B e con la possibilità di elementi mancanti fino a quando la Regione A non tornerà online. Quando possibile, come con le operazioni di scrittura idempotenti, è consigliabile valutare la possibilità di rieseguire il traffico di scrittura recente (ad esempio,

utilizzando una fonte a monte di eventi) per colmare la lacuna di eventuali operazioni di scrittura potenzialmente mancanti e lasciare che la risoluzione dei conflitti relativi alla priorità dell'ultima istanza di scrittura annulli la propagazione finale dell'operazione di scrittura in entrata.

Con le altre modalità di scrittura, è necessario considerare il grado in cui il lavoro può continuare con una out-of-date visione leggermente del mondo. Alcune operazioni di scrittura di breve durata, tracciate da `ReplicationLatency`, risulteranno mancanti fino a quando la Regione A non tornerà online. L'attività aziendale può andare avanti? In alcuni casi d'uso ciò è possibile, ma in altri casi potrebbe non essere possibile senza meccanismi di mitigazione aggiuntivi.

Ad esempio, si supponga di dover mantenere disponibile il saldo del credito senza interruzioni anche dopo un'interruzione totale nella Regione. È possibile dividere il saldo in due elementi diversi, uno situato nella Regione A e uno nella Regione B, ciascuno riferito a metà del saldo disponibile. In questo caso si utilizzerebbe la modalità di scrittura nella propria regione. Gli aggiornamenti transazionali elaborati in ciascuna regione verrebbero contabilizzati sulla copia locale del saldo. Se la Regione A passa alla modalità offline, l'elaborazione delle transazioni nella Regione B potrebbe continuare e le operazioni di scrittura sarebbero limitate alla parte del saldo conservata nella Regione B. Suddividere il saldo in questo modo comporta difficoltà quando il saldo si esaurisce o il credito deve essere ribilanciato, ma fornisce un esempio di ripristino aziendale sicuro anche con operazioni di scrittura in sospeso incerte.

Per fare un altro esempio, si supponga di acquisire i dati dei moduli web. È possibile utilizzare la funzionalità di controllo della simultaneità ottimistica (OCC, Optimistic Concurrency Control) per assegnare versioni agli elementi di dati e incorporare la versione più recente nel modulo web come campo nascosto. Ad ogni invio, l'operazione di scrittura ha esito positivo solo se la versione nel database corrisponde alla versione con cui è stato creato il modulo. Se le versioni non corrispondono, il modulo web può essere aggiornato (o unito) in base alla versione corrente nel database e l'utente può procedere. Il modello OCC di solito protegge dalla sovrascrittura e dalla produzione di una nuova versione dei dati da parte di un altro client, ma può anche essere utile durante il failover, situazione in cui un client potrebbe riscontrare versioni precedenti dei dati. Si supponga di utilizzare il timestamp come versione. Si immagini inoltre che il modulo sia stato creato per la prima volta nella Regione A alle 12:00 ma (dopo il failover) tenti di eseguire un'operazione di scrittura nella Regione B e si accorga che l'ultima versione nel database risale alle 11:59. In questo scenario, il client può attendere che la versione delle 12:00 si propaghi nella Regione B e quindi sovrascrivere su quella versione oppure eseguire una compilazione alle 11:59 e creare una nuova versione alle 12:01 (che, dopo la scrittura, eliminerebbe la versione in arrivo dopo il ripristino della Regione A).

Come terzo esempio, una società di servizi finanziari conserva i dati sugli account dei clienti e sulle relative transazioni finanziarie in un database DynamoDB. In caso di interruzione completa dei servizi nella Regione A, la società vuole essere sicura che qualsiasi attività di scrittura relativa ai propri account sia disponibile nella Regione B oppure che i propri account vengano messi in quarantena come account parzialmente noti fino a quando la Regione A non torna online. Invece di sospendere tutte le attività, la società decide di sospendere l'attività solo per la piccola parte di account con transazioni ritenute non propagate. A tal fine, la società utilizza una terza regione, definita Regione C. Prima di elaborare qualsiasi operazione di scrittura nella Regione A, nella Regione C la società inserisce un breve riepilogo delle operazioni in sospeso (ad esempio, un nuovo numero di transazioni per un conto). Questo riepilogo è sufficiente per consentire alla Regione B di determinare se la visualizzazione è completamente aggiornata. Questa azione effettivamente blocca l'account dal momento della scrittura nella Regione C fino a quando la Regione A accetta le operazioni di scrittura e la Regione B le riceve. I dati nella Regione C non vengono utilizzati se non come parte di un processo di failover, dopodiché la Regione B controlla i dati con quelli della Regione C per verificare se alcuni dei suoi account non sono aggiornati. Tali account vengono contrassegnati come messi in quarantena fino a quando il ripristino della Regione A non propaga i dati parziali alla Regione C, potrebbe invece essere utilizzata una nuova Regione D. I dati nella regione C erano molto transitori e dopo pochi minuti la regione D avrebbe avuto una up-to-date registrazione sufficiente delle operazioni di scrittura in volo per essere pienamente utili. Se si verifica un guasto nella Regione B, la Regione A potrebbe continuare ad accettare richieste di scrittura in collaborazione con la Regione C. La società è disposta ad accettare scritture a latenza più elevata (verso due regioni, ovvero C e poi A) ed è fortunata a disporre di un modello di dati in cui lo stato di un account può essere riassunto in modo sintetico.

Pianificazione della capacità di throughput per le tabelle globali DynamoDB

In relazione alla capacità, la migrazione del traffico da una regione all'altra richiede un'attenta valutazione delle impostazioni delle tabelle DynamoDB.

Alcune considerazioni sulla gestione della capacità di scrittura:

- Una tabella globale deve essere in modalità on demand o per essa deve essere effettuato il provisioning con il dimensionamento automatico abilitato.
- Se viene effettuato il provisioning del dimensionamento automatico, le impostazioni di scrittura (utilizzo minimo, massimo e obiettivo) vengono replicate tra le regioni. Anche se le impostazioni del dimensionamento automatico sono sincronizzate, la capacità di scrittura effettiva allocata potrebbe variare in modo indipendente tra le Regioni.

- Uno dei motivi per cui potrebbe essere possibile riscontrare una diversa capacità di scrittura allocata è dovuto alla funzionalità TTL. Quando si abilita il TTL in DynamoDB, è possibile specificare un nome di attributo il cui valore indica l'ora di scadenza dell'elemento, espresso in secondi, nel formato epoch Unix. Alla fine di tale periodo, DynamoDB può eliminare l'elemento senza incorrere in costi di scrittura. Con le tabelle globali è possibile configurare il TTL in una regione. Tale impostazione viene replicata automaticamente nelle altre regioni associate alla tabella globale. Quando un elemento è idoneo per l'eliminazione tramite una regola TTL, questa operazione può essere eseguita in qualsiasi regione. L'operazione di eliminazione viene eseguita senza consumare unità di scrittura sulla tabella di origine, ma le tabelle di replica riceveranno una scrittura replicata di tale operazione di eliminazione e comporteranno costi unitari di scrittura replicati. Il TTL non è supportato nelle tabelle MRSC.
- Se si utilizza il dimensionamento automatico, assicurarsi che l'impostazione della capacità di scrittura massima con provisioning sia sufficientemente alta per gestire tutte le operazioni di scrittura e tutte le potenziali operazioni di eliminazione TTL. Il dimensionamento automatico adatta ogni regione in base al consumo delle operazioni di scrittura. Le tabelle on demand non hanno un'impostazione di capacità di scrittura massima con provisioning, ma il limite massimo di velocità di trasmissione effettiva di scrittura a livello di tabella specifica la capacità massima di scrittura sostenuta consentita dalla tabella on demand. Il limite predefinito è 40.000, ma questo valore è modificabile. È consigliabile impostarlo su un valore sufficientemente alto da gestire tutte le operazioni di scrittura (incluse le operazioni di scrittura TTL) che la tabella on demand potrebbe richiedere. Questo valore deve essere lo stesso in tutte le regioni partecipanti quando vengono configurate le tabelle globali.

Alcune considerazioni sulla gestione della capacità di lettura:

- Le impostazioni relative alla gestione della capacità di lettura possono differire tra regioni perché si presume che regioni diverse possano avere modelli di lettura indipendenti. Quando si aggiunge una replica globale a una tabella, la capacità della regione di origine viene propagata. Dopo la creazione, è possibile adattare la capacità di lettura di una replica; questa nuova impostazione non viene trasferita all'altra regione.
- Quando si usa il dimensionamento automatico di DynamoDB, assicurarsi che le impostazioni relative alla capacità massima di lettura con provisioning siano sufficientemente elevate da gestire tutte le operazioni di lettura in tutte le regioni. Durante le operazioni standard, è possibile che la capacità di lettura venga distribuita tra le regioni, ma durante il failover la tabella dovrebbe essere in grado di adattarsi automaticamente all'aumento del carico di lavoro di lettura. Le tabelle on demand non hanno un'impostazione di capacità di lettura massima con provisioning, ma il limite

massimo di velocità di trasmissione effettiva di lettura a livello di tabella specifica la capacità massima di lettura sostenuta consentita dalla tabella on demand. Il limite predefinito è 40.000, ma questo valore è modificabile. È consigliabile impostarlo su un livello sufficientemente alto da gestire tutte le operazioni di lettura necessarie alla tabella se tutte le operazioni di lettura dovessero essere instradate a questa regione.

- Se una tabella in una Regione in genere non riceve traffico di lettura ma potrebbe dover assorbire una grande quantità di traffico di lettura dopo un failover, è possibile pre-riscaldare la capacità di lettura per accettare un livello più elevato di traffico di lettura.

Il sistema ARC dispone di controlli di idoneità che possono essere utili per controllare se le Regioni DynamoDB hanno impostazioni di tabella e quote di account simili, indipendentemente dal fatto che si utilizzi Route 53 per instradare le richieste. Questi controlli di idoneità possono anche aiutare ad adattare le quote a livello di account per assicurarsi che corrispondano.

Elenco di controllo di preparazione per tabelle globali DynamoDB

Utilizza il seguente elenco di controllo per decisioni e attività relative all'implementazione delle tabelle globali.

- Determina se il caso d'uso trae maggiori vantaggi da una modalità di coerenza MRSC o MREC. È necessaria un'elevata consistenza, anche con una latenza più elevata e altri compromessi?
- Determina quante e quali Regioni devono essere incluse nella tabella globale. Se intendi utilizzare MRSC, decidi se desideri che la terza Regione sia una replica o un testimone.
- Determina la modalità di scrittura dell'applicazione. Questa non è la stessa della modalità di coerenza. Per ulteriori informazioni, consulta Modalità di scrittura con le tabelle globali DynamoDB.
- Pianifica la strategia Strategie di instradamento in DynamoDB in base alla modalità di scrittura scelta.
- Definisci i

L'evacuazione di una Regione è il processo di migrazione delle attività, solitamente delle attività di lettura e scrittura, o delle attività di lettura, dalla Regione.

Evacuazione di una regione in tempo reale

Potresti decidere di evacuare una regione attiva per diversi motivi: come parte della normale attività aziendale (ad esempio, se utilizzi una modalità di scrittura in una regione) follow-the-sun, a causa della decisione aziendale di modificare la regione attualmente attiva, in risposta a guasti

nello stack software esterno a DynamoDB o perché stai riscontrando problemi generali come latenze più elevate del solito all'interno della regione.

Con la modalità di scrittura in qualsiasi regione, l'evacuazione di una regione in tempo reale è semplice. È possibile instradare il traffico verso le Regioni alternative tramite qualsiasi sistema di instradamento e lasciare che le operazioni di scrittura della Regione evacuata si replichino come al solito.

Le modalità di scrittura in una Regione e quella di scrittura nella propria Regione vengono generalmente utilizzate con le tabelle MREC. Pertanto, è necessario assicurarsi che tutte le operazioni di scrittura nella nuova Regione attiva siano state completamente registrate, elaborate in streaming e propagate globalmente prima di iniziare le operazioni di scrittura nella nuova Regione attiva, per garantire che le future operazioni di scrittura vengano elaborate sulla versione più recente dei dati.

Si supponga che la regione A sia attiva e la regione B sia passiva (per la tabella completa o per gli elementi che si trovano nella regione A). Il meccanismo tipico per eseguire un'evacuazione consiste nel sospendere le operazioni di scrittura nella Regione A, attendere il tempo necessario affinché tali operazioni si siano propagate completamente nella Regione B, aggiornare lo stack dell'architettura per riconoscere la Regione B come attiva e quindi riprendere le operazioni di scrittura nella Regione B. Non esiste una metrica che indichi con assoluta certezza che la Regione A ha replicato completamente i propri dati nella Regione B. Se la Regione A è integra, la sospensione delle operazioni di scrittura nella regione A e l'attesa di 10 volte il valore massimo recente della metrica `ReplicationLatency` sarebbero in genere sufficienti per determinare che la replica è completa. Se la Regione A non è integra e mostra altre aree con latenze aumentate, per definire il tempo di attesa scegliere un valore multiplo più grande.

Evacuazione di una regione offline

Esiste un caso speciale da considerare: cosa succede se la Regione A risulta inaspettatamente offline? Ciò è estremamente improbabile, ma va comunque considerato.

Evacuazione di una tabella MRSC offline

Se ciò accade con una tabella MRSC, non è necessario fare nulla di particolare. Le tabelle MRSC supportano un obiettivo del punto di ripristino (RPO) pari a zero. Tutte le operazioni di scrittura eseguite correttamente sulla tabella MRSC nella Regione offline saranno disponibili in tutte le altre tabelle delle Regioni, quindi non vi è alcuna potenziale lacuna nei dati anche

se la Regione passa completamente offline senza preavviso. Le aziende possono continuare a utilizzare le repliche situate nelle altre Regioni.

Evacuazione di una tabella MREC offline

In questo caso, tutte le operazioni di scrittura nella Regione A non ancora propagate vengono conservate e propagate dopo che la Regione A torna online. Le operazioni di scrittura non vengono perse, ma la loro propagazione viene ritardata a tempo indeterminato.

Come procedere in questo caso dipende dall'applicazione. Per la continuità aziendale, potrebbe essere necessario procedere alle operazioni di scrittura nella nuova Regione B. Tuttavia, se un elemento nella Regione B riceve un aggiornamento mentre è in corso la propagazione di un'operazione di scrittura per tale elemento dalla Regione A, la propagazione viene soppressa in base al modello basato sulla priorità dell'ultima istanza di scrittura. Qualsiasi aggiornamento nella Regione B potrebbe eliminare una richiesta di scrittura in arrivo.

Con la modalità di scrittura in qualsiasi Regione, le operazioni di lettura e scrittura possono continuare nella Regione B, con la certezza che gli elementi nella Regione A alla fine si propagheranno alla Regione B e con la possibilità di elementi mancanti fino a quando la Regione A non tornerà online. Quando possibile, come con le operazioni di scrittura idempotenti, è consigliabile valutare la possibilità di rieseguire il traffico di scrittura recente (ad esempio, utilizzando una fonte a monte di eventi) per colmare la lacuna di eventuali operazioni di scrittura potenzialmente mancanti e lasciare che la risoluzione dei conflitti relativi alla priorità dell’ultima istanza di scrittura annulli la propagazione finale dell’operazione di scrittura in entrata.

Con le altre modalità di scrittura, è necessario considerare il grado in cui il lavoro può continuare con una out-of-date visione leggermente del mondo. Alcune operazioni di scrittura di breve durata, tracciate da `ReplicationLatency`, risulteranno mancanti fino a quando la Regione A non tornerà online. L'attività aziendale può andare avanti? In alcuni casi d'uso ciò è possibile, ma in altri casi potrebbe non essere possibile senza meccanismi di mitigazione aggiuntivi.

Ad esempio, si supponga di dover mantenere disponibile il saldo del credito senza interruzioni anche dopo un’interruzione totale nella Regione. È possibile dividere il saldo in due elementi diversi, uno situato nella Regione A e uno nella Regione B, ciascuno riferito a metà del saldo disponibile. In questo caso si utilizzerebbe la modalità di scrittura nella propria regione. Gli aggiornamenti transazionali elaborati in ciascuna regione verrebbero contabilizzati sulla copia

locale del saldo. Se la Regione A passa alla modalità offline, l'elaborazione delle transazioni nella Regione B potrebbe continuare e le operazioni di scrittura sarebbero limitate alla parte del saldo conservata nella Regione B. Suddividere il saldo in questo modo comporta difficoltà quando il saldo si esaurisce o il credito deve essere ribilanciato, ma fornisce un esempio di ripristino aziendale sicuro anche con operazioni di scrittura in sospeso incerte.

Per fare un altro esempio, si supponga di acquisire i dati dei moduli web. È possibile utilizzare la funzionalità di controllo della simultaneità ottimistica (OCC, Optimistic Concurrency Control) per assegnare versioni agli elementi di dati e incorporare la versione più recente nel modulo web come campo nascosto. Ad ogni invio, l'operazione di scrittura ha esito positivo solo se la versione nel database corrisponde alla versione con cui è stato creato il modulo. Se le versioni non corrispondono, il modulo web può essere aggiornato (o unito) in base alla versione corrente nel database e l'utente può procedere. Il modello OCC di solito protegge dalla sovrascrittura e dalla produzione di una nuova versione dei dati da parte di un altro client, ma può anche essere utile durante il failover, situazione in cui un client potrebbe riscontrare versioni precedenti dei dati. Si supponga di utilizzare il timestamp come versione. Si immagini inoltre che il modulo sia stato creato per la prima volta nella Regione A alle 12:00 ma (dopo il failover) tenti di eseguire un'operazione di scrittura nella Regione B e si accorga che l'ultima versione nel database risale alle 11:59. In questo scenario, il client può attendere che la versione delle 12:00 si propaghi nella Regione B e quindi sovrascrivere su quella versione oppure eseguire una compilazione alle 11:59 e creare una nuova versione alle 12:01 (che, dopo la scrittura, eliminerebbe la versione in arrivo dopo il ripristino della Regione A).

Come terzo esempio, una società di servizi finanziari conserva i dati sugli account dei clienti e sulle relative transazioni finanziarie in un database DynamoDB. In caso di interruzione completa dei servizi nella Regione A, la società vuole essere sicura che qualsiasi attività di scrittura relativa ai propri account sia disponibile nella Regione B oppure che i propri account vengano messi in quarantena come account parzialmente noti fino a quando la Regione A non torna online. Invece di sospendere tutte le attività, la società decide di sospendere l'attività solo per la piccola parte di account con transazioni ritenute non propagate. A tal fine, la società utilizza una terza regione, definita Regione C. Prima di elaborare qualsiasi operazione di scrittura nella Regione A, nella Regione C la società inserisce un breve riepilogo delle operazioni in sospeso (ad esempio, un nuovo numero di transazioni per un conto). Questo riepilogo è sufficiente per consentire alla Regione B di determinare se la visualizzazione è completamente aggiornata. Questa azione effettivamente blocca l'account dal momento della scrittura nella Regione C fino a quando la Regione A accetta le operazioni

di scrittura e la Regione B le riceve. I dati nella Regione C non vengono utilizzati se non come parte di un processo di failover, dopodiché la Regione B controlla i dati con quelli della Regione C per verificare se alcuni dei suoi account non sono aggiornati. Tali account vengono contrassegnati come messi in quarantena fino a quando il ripristino della Regione A non propaga i dati parziali alla Regione C, potrebbe invece essere utilizzata una nuova Regione D. I dati nella regione C erano molto transitori e dopo pochi minuti la regione D avrebbe avuto una up-to-date registrazione sufficiente delle operazioni di scrittura in volo per essere pienamente utili. Se si verifica un guasto nella Regione B, la Regione A potrebbe continuare ad accettare richieste di scrittura in collaborazione con la Regione C. La società è disposta ad accettare scritture a latenza più elevata (verso due regioni, ovvero C e poi A) ed è fortunata a disporre di un modello di dati in cui lo stato di un account può essere riassunto in modo sintetico.

the section called “Processi di evacuazione” in base alla modalità di coerenza, alla modalità di scrittura e alla strategia di instradamento scelti.

- Acquisisci metriche su integrità, latenza ed errori in ogni Regione. Per un elenco dei parametri di DynamoDB, consulta AWS il post del blog Monitoring Amazon DynamoDB for Operational Awareness per un elenco di metriche da osservare. È inoltre consigliabile utilizzare canary sintetici (richieste artificiali progettate per rilevare i guasti), nonché l'osservazione in tempo reale del traffico dei clienti. Non tutti i problemi verranno visualizzati nelle metriche di DynamoDB.
- In caso di utilizzo di MREC, imposta allarmi per qualsiasi aumento prolungato di `ReplicationLatency`. Un incremento potrebbe indicare una configurazione errata accidentale in cui la tabella globale ha impostazioni di scrittura diverse in varie regioni, il che porta a richieste replicate non riuscite e a un aumento della latenza. Potrebbe anche indicare che si è verificata un'interruzione dei servizi a livello regionale. Un buon esempio è generare un avviso se il valore medio recente supera i 180.000 millisecondi. È anche possibile controllare se `ReplicationLatency` scende a 0; ciò indica una replica bloccata.
- Assegna impostazioni massime di lettura e scrittura sufficienti per ogni tabella globale.
- Individuare in anticipo i motivi dell'evacuazione di una regione. Se la decisione implica un giudizio umano, documentare tutte le considerazioni. Questa fase deve essere svolta con attenzione in anticipo, non in situazioni di stress.
- Gestire un runbook per ogni operazione da eseguire in caso di evacuazione di una regione. Di solito è richiesto pochissimo lavoro per le tabelle globali, ma il trasferimento del resto dello stack potrebbe essere una procedura complessa.

> **Note**
>
> Con le procedure di failover, è consigliabile fare affidamento solo sulle operazioni del piano dati e non sulle operazioni del piano di controllo (control-plane) perché alcune operazioni del piano di controllo (control-plane) potrebbero degradarsi durante i guasti a livello di Regione.

Per ulteriori informazioni, consulta il post AWS sul blog Crea applicazioni resilienti con le tabelle globali di Amazon DynamoDB: parte 4.

- Testa periodicamente tutti gli aspetti del runbook, comprese le evacuazioni della Regione. Un runbook non testato è un runbook inaffidabile.
- Considera la possibilità di utilizzare AWS Resilience Hub per valutare la resilienza dell'intera applicazione (comprese le tabelle globali). Fornisce una visione completa dello stato di resilienza complessivo del portafoglio di applicazioni tramite il relativo pannello di controllo.
- Considera la possibilità di utilizzare i controlli di idoneità del sistema ARC per valutare la configurazione corrente dell'applicazione e tenere traccia di eventuali anomalie rispetto alle best practice.
- In caso di scrittura di controlli di integrità da utilizzare con Route 53 o Global Accelerator, esegui una serie di chiamate che coprano l'intero flusso del database. Se limiti il controllo alla sola conferma che l'endpoint DynamoDB è attivo, non sarai in grado di coprire molte modalità di errore AWS Identity and Access Management come errori di configurazione (IAM), problemi di distribuzione del codice, errori nello stack esterno a DynamoDB, latenze di lettura o scrittura superiori alla media e così via.

Domande frequenti relative alla distribuzione delle tabelle globali

Quali sono i prezzi delle tabelle globali?

- Il prezzo di un'operazione di scrittura in una tabella DynamoDB tradizionale è espresso in unità di capacità di scrittura WCUs (per le tabelle fornite) o in unità di richiesta di scrittura () per le WRUs tabelle su richiesta. In caso di scrittura di un elemento da 5 KB, viene addebitato un costo pari a 5 unità. Il prezzo di un'operazione di scrittura su una tabella globale è espresso in unità di capacità di scrittura replicate (rWCUs, per le tabelle fornite) o in unità di richiesta di scrittura replicate

(rWRUs, per le tabelle su richiesta). r e r hanno lo stesso prezzo di and. WCUs WRUs WGUs WRUs

- I costi di rWCU e rWRU vengono applicati in tutte le Regioni in cui l'elemento viene scritto direttamente o tramite una replica. Si applicano tariffe per il trasferimento di dati tra regioni.
- La scrittura su un indice secondario globale (GSI) è considerata una scrittura locale e utilizza le normali unità di scrittura.
- Al momento non è disponibile alcuna capacità riservata per r o r. WCUs WRUs L'acquisto di capacità riservata per WCUs può essere utile per le tabelle in cui si GSIs consumano unità di scrittura.
- Quando aggiungi una nuova Regione a una tabella globale, DynamoDB avvia automaticamente la nuova Regione e ti addebita come se si trattasse di un ripristino della tabella, in base alla sua dimensione in GB. Addebita inoltre tariffe aggiuntive per il trasferimento di dati tra regioni.

Quali sono regioni supportate dalle tabelle globali?

La versione 2019.11.21 (corrente) di Global Tables supporta tutte le tabelle MREC e i seguenti set di regioni Regioni AWS per le tabelle MRSC:

- Set della Regione degli Stati Uniti: Stati Uniti orientali (Virginia settentrionale), Stati Uniti orientali (Ohio), Stati Uniti occidentali (Oregon)
- Set della Regione Europa: Europa (Irlanda), Europa (Londra), Europa (Parigi), Europa (Francoforte)
- Set della Regione Asia Pacifico: Asia Pacifico (Tokyo), Asia Pacifico (Seoul) e Asia Pacifico (Osaka)

Come vengono gestite le tabelle globali? GSIs

Nelle Tabelle globali versione 2019.11.21 (Corrente), quando viene creato un GSI in una Regione, tale indice viene automaticamente creato e compilato nelle altre Regioni partecipanti.

Come posso interrompere la replica di una tabella globale?

- È possibile eliminare una tabella di replica con la stessa procedura usata per eliminare qualsiasi altra tabella. Ciò interromperà la replica ed eliminerà la copia della tabella conservata in tale regione. Tuttavia, non è possibile interrompere la replica e conservare le copie della tabella come entità indipendenti, né è possibile sospendere la replica.

- Una tabella MRSC deve essere distribuita esattamente in tre Regioni. Per eliminare le repliche è necessario eliminare tutte le repliche e il testimone in modo che la tabella MRSC diventi una tabella locale.

In che modo i flussi DynamoDB interagiscono con le tabelle globali?

- Ogni tabella globale produce un flusso indipendente basato su tutte le relative operazioni di scrittura, a prescindere dal punto di partenza. È possibile scegliere di utilizzare il flusso DynamoDB in una regione o in tutte le regioni in modo indipendente. Per elaborare operazioni di scrittura locali non replicate, è possibile aggiungere l'attributo relativo alla regione a ciascun elemento. È quindi possibile utilizzare un filtro di eventi Lambda per richiamare solo la funzione Lambda per le operazioni di scrittura nella regione locale. Questa procedura serve per le operazioni di inserimento e aggiornamento, ma non per le operazioni di eliminazione.
- Le tabelle globali configurate per la coerenza finale tra più Regioni (tabelle MREC) replicano le modifiche leggendo tali modifiche da un flusso DynamoDB su una tabella di replica e applicandole a tutte le altre tabelle di replica. Pertanto, DynamoDB è abilitato per impostazione predefinita su tutte le repliche in una tabella globale MREC e non può essere disabilitato su tali repliche. Il processo di replica MREC può combinare più modifiche in un breve periodo di tempo in un'unica operazione di scrittura replicata. Di conseguenza, il flusso di ogni replica potrebbe contenere record leggermente diversi. I record dei flussi DynamoDB sulle repliche MREC vengono sempre ordinati per elemento, ma l'ordinamento tra gli elementi potrebbe differire tra le repliche.
- Le tabelle globali configurate per un'elevata consistenza multi-Regione (tabelle MRSC) non utilizzano i flussi DynamoDB per la replica, quindi questa funzionalità non è abilitata per impostazione predefinita sulle repliche MRSC. È possibile abilitare i flussi DynamoDB su una replica MRSC. I record dei flussi DynamoDB sulle repliche MRSC sono identici per ogni replica e vengono sempre ordinati per elemento, ma l'ordinamento tra gli elementi potrebbe differire tra le repliche.

In che modo le tabelle globali gestiscono le transazioni?

- Le operazioni transazionali sulle tabelle MRSC genereranno errori.
- Le operazioni transazionali sulle tabelle MREC forniscono garanzie di atomicità, consistenza, isolamento e durabilità (ACID) solo all'interno della Regione in cui si è originariamente verificata l'operazione di scrittura. Le transazioni non sono supportate tra le regioni nelle tabelle globali. Ad esempio, se è presente una tabella globale MREC con repliche nelle Regioni Stati Uniti orientali (Ohio) e Stati Uniti occidentali (Oregon) e si esegue un'operazione `TransactWriteItems` nella

Regione Stati Uniti orientali (Ohio), è possibile osservare transazioni parzialmente completate nella Regione Stati Uniti occidentali (Oregon) man mano che le modifiche vengono replicate. Le modifiche vengono replicate in altre Regioni solo dopo essere state confermate nella Regione di origine.

In che modo le tabelle globali interagiscono con la cache DynamoDB Accelerator (DAX)?

Le tabelle globali ignorano DAX aggiornando direttamente DynamoDB, quindi DAX non è consapevole della presenza di dati obsoleti. La cache DAX verrà aggiornata solo alla scadenza del TTL della cache.

I tag presenti nelle tabelle vengono propagati?

No, i tag non vengono propagati automaticamente.

Devo eseguire il backup delle tabelle in tutte le regioni o solo in una?

La risposta dipende dallo scopo del backup.

- Se è necessario garantire la durabilità dei dati, DynamoDB fornisce già questa protezione. Il servizio garantisce la durabilità dei dati.
- Se si desidera conservare uno snapshot dei record storici (ad esempio per soddisfare i requisiti normativi), il backup in una regione dovrebbe essere sufficiente. È possibile copiare il backup in altre regioni utilizzando AWS Backup.
- Se desideri recuperare dati cancellati o modificati erroneamente, utilizza DynamoDB point-in-time recovery (PITR) in una regione.

Come posso distribuire tabelle globali utilizzando? CloudFormation

- CloudFormation rappresenta una tabella DynamoDB e una tabella globale come due risorse separate: e. `AWS::DynamoDB::Table` `AWS::DynamoDB::GlobalTable` Un approccio consiste nel creare tutte le tabelle che possono essere potenzialmente globali utilizzando il costrutto `GlobalTable` per mantenerle inizialmente come tabelle autonome e aggiungere Regioni in un secondo momento, se necessario.
- In CloudFormation, ogni tabella globale è controllata da un singolo stack, in una singola regione, indipendentemente dal numero di repliche. Quando distribuisci il modello, CloudFormation crea e aggiorna tutte le repliche come parte di un'unica operazione di stack. Non è consigliabile distribuire la stessa risorsa AWS::DynamoDB::GlobalTable in più regioni. Questo metodo non è supportato

e genererà errori. Se si distribuisce il modello di applicazione in più regioni, è possibile utilizzare le condizioni per creare la risorsa `AWS::DynamoDB::GlobalTable` solo in una regione. In alternativa, è possibile scegliere di definire le risorse `AWS::DynamoDB::GlobalTable` in uno stack separato dallo stack dell'applicazione e verificare che sia distribuito solo in un'unica regione.

- Se disponi di una tabella normale e desideri convertirla in una tabella globale mantenendola gestita, imposta la politica di eliminazione su`Retain`, rimuovi la tabella dallo stack, converti la tabella in una tabella globale nella console e quindi importa la tabella globale come nuova risorsa nello stack. CloudFormation Per ulteriori informazioni, consulta il AWS GitHub repository.
- La replica su più account non è al momento supportata.

Conclusioni e risorse

Le tabelle globali DynamoDB hanno pochissimi controlli ma richiedono comunque un'attenta considerazione. È necessario determinare la modalità di scrittura, il modello di instradamento e i processi di evacuazione. È necessario dotare l'applicazione della strumentazione necessaria in ogni regione ed essere pronti a modificare l'instradamento o eseguire un'evacuazione per salvaguardare l'integrità globale. Il vantaggio è avere un set di dati distribuito a livello globale con operazioni di lettura e scrittura a bassa latenza progettato per una disponibilità del 99,999%.

Per ulteriori informazioni sulle tabelle globali DynamoDB, consulta le seguenti risorse:

- Documentazione di DynamoDB
- Amazon Application Recovery Controller
- Controllo di idoneità in ARC (documentazione)AWS
- Route 53 routing policies
- AWS Acceleratore globale
- Accordo sul livello di servizio di DynamoDB
- AWS Nozioni di base su più regioni (white paper)AWS
- Modelli di progettazione della resilienza dei dati con (presentazione re:Invent 2022) AWSAWS
- Come Fidelity Investments e Reltio si sono modernizzati con Amazon DynamoDB (presentazione re:Invent 2022)AWS
- Modelli di progettazione e best practice multiregionali (presentazione re:Invent 2022)AWS
- Architettura di disaster recovery (DR) attiva AWS, parte III: Pilot Light and Warm Standby (post sul blog)AWS

- Usa Region Pinning per impostare una regione di origine per gli elementi in una tabella globale AWS di Amazon DynamoDB (post del blog)
- Monitoraggio di Amazon DynamoDB per la consapevolezza AWS operativa (post sul blog)
- Scalabilità di DynamoDB: come le partizioni, i tasti di scelta rapida e la suddivisione per il riscaldamento influiscono sulle prestazioni (post sul blog)AWS
- Forte coerenza multiregionale con le tabelle AWS globali di DynamoDB (presentazione re:Invent 2024)

Best practice per la gestione del piano di controllo (control plane) in DynamoDB

> **Note**
>
> DynamoDB sta introducendo una limitazione della larghezza della banda della rete del piano di controllo (control plane) di 2.500 richieste al secondo con l'opzione di un nuovo tentativo. Vedere qui di seguito per ulteriori dettagli.

Le operazioni del piano di controllo (control plane) di DynamoDB consentono di gestire le tabelle DynamoDB e gli oggetti che dipendono dalle tabelle, come gli indici. Per ulteriori informazioni su queste operazioni, consulta Piano di controllo.

In alcune circostanze, potrebbe essere necessario intraprendere azioni e utilizzare i dati restituiti dalle chiamate al piano di controllo (control plane) come parte della logica aziendale. Ad esempio, potrebbe essere necessario conoscere il valore di `ProvisionedThroughput` restituito da `DescribeTable`. In queste circostanze, segui queste best practice:

- Non eseguire query eccessive sul piano di controllo (control plane) di DynamoDB.
- Non mescolare chiamate al piano di controllo (control plane) e chiamate al piano dati all'interno dello stesso codice.
- Gestisci le limitazioni della larghezza di banda della rete sulle richieste del piano di controllo (control plane) e riprova con un backoff.
- Richiama e monitora le modifiche a una particolare risorsa da un singolo client.
- Invece di recuperare i dati per la stessa tabella più volte a brevi intervalli, memorizza nella cache i dati per l'elaborazione.

Best practice per l'utilizzo di operazioni sui dati in blocco in DynamoDB

DynamoDB supporta operazioni batch, ad esempio `BatchWriteItem`, con le quali è possibile eseguire fino a 25 richieste `PutItem` e `DeleteItem` contemporaneamente. Tuttavia, `BatchWriteItem` non supporta le operazioni `UpdateItem`. Per gli aggiornamenti in blocco, la differenza sta nei requisiti e nella natura dell'aggiornamento. È possibile utilizzare altri DynamoDB APIs , ad esempio per batch di dimensioni `TransactWriteItems` fino a 100. Quando sono coinvolti più articoli, puoi utilizzare servizi come AWS Glue Amazon EMR AWS Step Functions o utilizzare script e strumenti personalizzati come DynamoDB-Shell per aggiornamenti di massa.

Argomenti

- Aggiornamento batch condizionale
- Operazioni in blocco efficienti

Aggiornamento batch condizionale

DynamoDB supporta operazioni batch, ad esempio `BatchWriteItem`, con le quali è possibile eseguire fino a 25 richieste `PutItem` e `DeleteItem` in un unico batch. Tuttavia, `BatchWriteItem` non supporta le operazioni `UpdateItem` e non supporta le espressioni condizionali. Come soluzione alternativa, puoi utilizzare altri APIs DynamoDB, ad esempio per batch di dimensioni `TransactWriteItems` fino a 100.

Quando sono coinvolti più elementi ed è necessario modificare una parte importante dei dati, è possibile utilizzare servizi come Amazon EMR AWS Step Functions o utilizzare script e strumenti personalizzati come AWS Glue DynamoDB-Shell per aggiornamenti di massa efficienti.

Quando usare questo modello

- DynamoDB-Shell non è un caso d'uso supportato per la produzione.
- `TransactWriteItems`: fino a 100 aggiornamenti singoli con o senza condizioni, eseguiti come un pacchetto ACID "tutto o niente". Le chiamate `TransactWriteItems` possono essere fornite anche con un `ClientRequestToken` se l'applicazione richiede idempotenza, il che significa che più chiamate identiche hanno lo stesso effetto di una singola chiamata. In questo modo non si esegue la stessa transazione più volte e si ottiene uno stato dei dati errato.

Compromesso: viene consumata una velocità di trasmissione aggiuntiva. 2 WCUs per 1 KB di scrittura anziché 1 WGU standard per 1 KB di scrittura.

- PartiQL`BatchExecuteStatement`: fino a 25 aggiornamenti con o senza condizioni. `BatchExecuteStatement` restituisce sempre una risposta corretta alla richiesta complessiva e restituisce anche un elenco di risposte operative individuali che preserva l'ordine.

  Compromesso: per batch più grandi, è necessaria una logica aggiuntiva lato client per distribuire le richieste in batch da 25. È necessario prendere in considerazione le risposte di errore individuali per determinare la strategia dei nuovi tentativi.

Esempi di codice

Questi esempi di codice utilizzano la libreria boto3, che è l' AWS SDK per Python. Gli esempi presuppongono che boto3 sia installato e configurato con le credenziali AWS appropriate.

Si supponga un database di inventario per un fornitore di elettrodomestici che ha più magazzini in diverse città europee. Poiché è la fine dell'estate, il venditore vorrebbe eliminare i ventilatori da tavolo per fare spazio ad altre scorte. Il venditore desidera offrire uno sconto sul prezzo per tutti i ventilatori da tavolo forniti dai magazzini in Italia, ma solo se hanno una scorta di riserva di 20 unità. La tabella DynamoDB è denominata inventory, ha uno schema di chiave di partizione sku che è un identificatore univoco per ogni prodotto e una chiave di ordinamento warehouse che è un identificatore per un magazzino.

Il seguente codice Python dimostra come eseguire questo aggiornamento batch condizionale utilizzando una chiamata 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'])
```

L'esecuzione produce l'output seguente su dati di esempio:

```
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'}}]
```

Poiché si tratta di un'operazione limitata per un sistema interno, i requisiti di idempotenza non sono stati presi in considerazione. Per rendere gli aggiornamenti più solidi, è possibile inserire guardrail aggiuntivi, come ad esempio l'aggiornamento del prezzo, solo se il prezzo è superiore a 35 e inferiore a 40.

In alternativa, è possibile eseguire la stessa operazione di aggiornamento batch utilizzando `TransactWriteItems` in caso di requisiti di idempotenza e ACID più rigorosi. Tuttavia, è importante ricordare che o tutte le operazioni del pacchetto di transazioni vengono eseguite o l'intero pacchetto genera un errore.

Ipotizziamo un caso in cui si verifichi un'ondata di caldo in Italia e la domanda di ventilatori da tavolo aumenti notevolmente. Il fornitore vuole aumentare di 20 euro il costo dei ventilatori da tavolo in uscita da tutti i magazzini in Italia, ma l'organismo di regolamentazione consente questo aumento solo se il costo attuale è inferiore a 70 euro per l'intero inventario. È essenziale che il prezzo venga aggiornato in tutto l'inventario contemporaneamente e solo una volta e solo se il costo è inferiore a 70 euro in ogni magazzino.

Il seguente codice Python dimostra come eseguire questo aggiornamento batch utilizzando una chiamata 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'])
```

L’esecuzione produce l’output seguente su dati di esempio:

```
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'}}]
```

Esistono diversi approcci per eseguire aggiornamenti batch in DynamoDB. L'approccio adatto dipende da fattori quali i requisiti di and/or idempotenza ACID, il numero di elementi da aggiornare e la familiarità con cui. APIs

Operazioni in blocco efficienti

Quando usare questo modello

Questi modelli sono utili per eseguire in modo efficiente aggiornamenti in blocco sugli elementi di DynamoDB.

- DynamoDB-Shell non è un caso d’uso supportato per la produzione.
- `TransactWriteItems`: fino a 100 aggiornamenti singoli con o senza condizioni, eseguiti come un pacchetto ACID “tutto o niente”

  Compromesso: viene consumata una velocità effettiva aggiuntiva, 2 per 1 KB di scrittura. WCUs
- PartiQL`BatchExecuteStatement`: fino a 25 aggiornamenti con o senza condizioni

  Compromesso: è necessaria una logica aggiuntiva per distribuire le richieste in batch da 25.
- AWS Step Functions — operazioni di massa a velocità limitata per gli sviluppatori che hanno dimestichezza con. AWS Lambda

Compromesso: il tempo di esecuzione è inversamente proporzionale al limite di velocità. Limitazione data dal timeout massimo della funzione Lambda. La funzionalità prevede che le modifiche ai dati che si verificano tra la lettura e la scrittura possano essere sovrascritte. Per ulteriori informazioni, consulta Backfilling an Amazon DynamoDB Time to Live attribute using Amazon EMR: Part 2.

- AWS Glue e Amazon EMR: operazioni di massa a velocità limitata con parallelismo gestito. Per le applicazioni o gli aggiornamenti non sensibili al fattore temporale, queste opzioni possono essere eseguite in background consumando solo una piccola percentuale del throughput. Entrambi i servizi utilizzano il emr-dynamodb-connector per eseguire operazioni DynamoDB. Questi servizi eseguono una lettura approfondita seguita da una scrittura approfondita degli elementi aggiornati con un'opzione per limitare la velocità.

  Compromesso: il tempo di esecuzione è inversamente proporzionale al limite di velocità. La funzionalità prevede che le modifiche ai dati che si verificano tra la lettura e la scrittura possano essere sovrascritte. Non puoi leggere da Global Secondary Indexes (). GSIs Consulta Backfilling an Amazon DynamoDB Time to Live attribute using Amazon EMR: Part 2.
- DynamoDB Shell: operazioni in blocco a velocità limitata attraverso query di tipo SQL. Puoi leggere da GSIs per una migliore efficienza.

  Compromesso: il tempo di esecuzione è inversamente proporzionale al limite di velocità. Consulta Rate limited bulk operations in DynamoDB Shell.

Utilizzo del modello

Gli aggiornamenti in blocco possono avere implicazioni significative in termini di costi, soprattutto se si utilizza la modalità di throughput on demand. Esiste un compromesso tra velocità e costi se si utilizza la modalità di throughput allocato. L'impostazione molto rigorosa del parametro del limite di velocità può portare a tempi di elaborazione molto lunghi. È possibile determinare approssimativamente la velocità di aggiornamento utilizzando la dimensione media dell'elemento e il limite di velocità.

In alternativa, è possibile determinare la quantità di throughput necessaria per il processo in base alla durata prevista del processo di aggiornamento e alla dimensione media degli elementi. I riferimenti del blog condivisi con ogni modello forniscono dettagli sulla strategia, l'implementazione e i limiti dell'utilizzo del modello. Per ulteriori informazioni, consulta Cost-effective bulk processing with Amazon DynamoDB.

Esistono diversi approcci per eseguire aggiornamenti in blocco su una tabella DynamoDB attiva. L'approccio adatto dipende da fattori quali i requisiti di and/or idempotenza ACID, il numero di elementi da aggiornare e la familiarità con cui si è in possesso. APIs È importante considerare il rapporto tra costi e tempi: la maggior parte degli approcci sopra descritti prevede un'opzione per limitare il throughput utilizzato dal processo di aggiornamento in blocco.

Best practice per l'implementazione del controllo delle versioni in DynamoDB

Nei sistemi distribuiti come DynamoDB, il controllo delle versioni degli elementi mediante il blocco ottimistico previene gli aggiornamenti in conflitto. Monitorando le versioni degli elementi e utilizzando scritture condizionali, le applicazioni possono gestire le modifiche simultanee, garantendo l'integrità dei dati in ambienti ad alta simultaneità.

Il blocco ottimistico è una strategia utilizzata per garantire che le modifiche ai dati vengano applicate correttamente senza conflitti. Invece di bloccare i dati quando vengono letti (come nel caso del blocco pessimistico), il blocco ottimistico verifica se i dati sono cambiati prima di riscriverli. In DynamoDB, ciò si ottiene tramite una forma di controllo delle versioni, in cui ogni elemento include un identificatore che aumenta con ogni aggiornamento. Quando si aggiorna un elemento, l'operazione avrà successo solo se l'identificatore corrisponde a quello previsto dall'applicazione.

Quando usare questo modello

Questo modello è utile nei seguenti scenari:

- Più utenti o processi possono tentare di aggiornare lo stesso elemento contemporaneamente.
- Garantire l'integrità e la coerenza dei dati è fondamentale.
- È necessario evitare il sovraccarico e la complessità della gestione dei blocchi distribuiti.

Gli esempi includono:

- Applicazioni di e-commerce in cui i livelli di inventario vengono aggiornati frequentemente.
- Piattaforme collaborative in cui più utenti modificano gli stessi dati.
- Sistemi finanziari in cui i record delle transazioni devono rimanere coerenti.

Compromessi

Sebbene il blocco ottimistico e i controlli condizionali forniscano una solida integrità dei dati, presentano i seguenti compromessi:

Conflitti di simultaneità

In ambienti caratterizzati da un elevato tasso di simultaneità, aumenta la probabilità che si verifichino conflitti, con conseguenti potenziali ripetizioni dei tentativi e costi di scrittura.

Complessità dell'implementazione

L'aggiunta del controllo delle versioni agli elementi e la gestione dei controlli condizionali possono aggiungere complessità alla logica dell'applicazione.

Sovraccarico di archiviazione aggiuntivo

L'archiviazione dei numeri di versione per ogni elemento aumenta leggermente i requisiti di archiviazione.

Progettazione dei modelli

Per implementare questo modello, lo schema DynamoDB deve includere un attributo version per ogni elemento. Ecco un semplice schema di progettazione:

- Chiave di partizione: un identificatore univoco per ogni elemento (es. `ItemId`).
- Attributi:
  - `ItemId`: l'identificatore univoco per l'elemento.
  - `Version`: un numero intero che rappresenta il numero di versione dell'elemento.
  - `QuantityLeft`: l'inventario rimanente dell'elemento.

Quando un elemento viene creato per la prima volta, l'attributo `Version` è impostato su 1. A ogni aggiornamento, il numero di versione aumenta di 1.

VersionControl

Primary key Partition key: ItemID	Attributes	
Bananas	Version	QuantityLeft
	1	10
Apples	Version	QuantityLeft
	1	5
Oranges	Version	QuantityLeft
	1	7

Utilizzo del modello

Per implementare questo modello, segui questa procedura nel flusso dell'applicazione:

1. La versione corrente dell'elemento.

   Recupera l'elemento corrente da DynamoDB e leggi il numero di versione.

   ```
   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. Incrementa il numero di versione nella logica dell'applicazione. Questa sarà la versione prevista per l'aggiornamento.

   ```
   new_version = current_version + 1
   ```

3. Tentativo di aggiornare l'elemento utilizzando un'espressione condizionale per garantire che il numero di versione corrisponda.

   ```
   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)
```

Se l'aggiornamento ha esito positivo, il prezzo QuantityLeft per l'articolo verrà ridotto di 2.

VersionControl

Primary key Partition key: ItemID	Attributes	
Bananas	Version	QuantityLeft
	2	8
Apples	Version	QuantityLeft
	1	5
Oranges	Version	QuantityLeft
	1	7

4. Gestisci i conflitti se si verificano.

   Se si verifica un conflitto (ad esempio, un altro processo ha aggiornato l'elemento dall'ultima lettura), gestiscilo in modo appropriato, ad esempio effettuando una ripetizione del tentativo per l'operazione o avvisando l'utente.

Ciò richiederà una lettura aggiuntiva dell'elemento per ogni ripetizione di tentativo, pertanto limita il numero totale di ripetizioni di tentativi consentite prima di interrompere definitivamente il ciclo di richieste.

```
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)
```

L'implementazione del controllo delle versioni degli elementi utilizzando DynamoDB con blocco ottimistico e controlli condizionali è un modello efficace per garantire l'integrità dei dati nelle applicazioni distribuite. Sebbene introduca una certa complessità e potenziali compromessi in termini di prestazioni, è inestimabile in scenari che richiedono un solido controllo della simultaneità. Progettando attentamente lo schema e implementando i controlli necessari nella logica dell'applicazione, è possibile gestire efficacemente gli aggiornamenti simultanei e mantenere la coerenza dei dati.

Ulteriori linee guida e strategie su come implementare il controllo delle versioni dei dati DynamoDB sono disponibili sul blog sui database AWS.

Le migliori pratiche per comprendere i report di AWS fatturazione e utilizzo in DynamoDB

Questo documento illustra i codici di fatturazione `UsageType` per gli addebiti relativi a DynamoDB.

AWS fornisce report sui costi e sull'utilizzo (CUR) che contengono dati relativi ai servizi utilizzati. Puoi utilizzarlo AWS Cost and Usage Report per pubblicare report di fatturazione su Amazon S3 in formato CSV. Durante la configurazione del report CUR è possibile scegliere di suddividere i periodi di tempo per ora, giorno o mese ed è possibile scegliere se suddividere l'utilizzo in base all'ID della risorsa o meno. Per maggiori dettagli sulla generazione del report CUR, consulta Creating Cost and Usage Reports.

Nell'esportazione in formato CSV sono disponibili gli attributi pertinenti elencati per ogni riga. Di seguito sono riportati esempi di attributi che è possibile includere:

- lineitem/UsageStartDate: la data e l'ora di inizio dell'articolo in UTC, incluso.
- lineitem/UsageEndDate: la data e l'ora di fine della riga corrispondente in UTC, esclusive.
- lineitem/: ProductCode Per DynamoDB questo sarà «DB» AmazonDynamo
- lineitem/UsageType: un codice descrittivo specifico per il tipo di utilizzo, come enumerato in questo documento
- lineitem/Operation: un nome che fornisce un contesto all'addebito, ad esempio il nome dell'operazione che ha comportato l'addebito (opzionale).
- lineitem/ResourceId: l'identificatore della risorsa che ha comportato l'utilizzo. Disponibile se il report CUR include una suddivisione per ID di risorsa.
- lineitem/UsageAmount: la quantità di utilizzo effettuata durante il periodo di tempo specificato.
- lineitem/ UnblendedCost: il costo di questo utilizzo.
- lineitem/LineItemDescription: descrizione testuale dell'elemento della riga.

Per ulteriori informazioni sul dizionario dei dati CUR, consulta Cost and Usage Report (CUR) 2.0. I nomi esatti variano a seconda del contesto.

Un `UsageType` è una stringa con un valore come `ReadCapacityUnit-Hrs`, `USW2-ReadRequestUnits`, `EU-WriteCapacityUnit-Hrs` o`USE1-TimedPITRStorage-ByteHrs`. Ogni tipo di utilizzo inizia con un prefisso della Regione opzionale. Se assente, indica la Regione Stati Uniti orientali us-east-1. Se presente, la tabella seguente associa il codice della Regione di fatturazione breve al codice della Regione e al nome convenzionali.

Ad esempio, l'utilizzo denominato `USW2-ReadRequestUnits` indica le unità di richiesta di lettura utilizzate nella Regione us-west-2.

Codice Regionale di fatturazione	Codice regione	Nome della regione
AFS1	af-south-1	Africa (Città del Capo)
APE1	ap-east-1	Asia Pacifico (Hong Kong)
APN1	ap-northeast-1	Asia Pacifico (Tokyo)
APN2	ap-northeast-2	Asia Pacifico (Seul)
APN3	ap-northeast-3	Asia Pacifico (Osaka-Locale)
APS1	ap-southeast-1	Asia Pacifico (Singapore)
APS2	ap-southeast-2	Asia Pacifico (Sydney)
APS3	ap-south-1	Asia Pacifico (Mumbai)
APS4	ap-southeast-3	Asia Pacifico (Giacarta)
APS5	ap-south-2	Asia Pacific (Hyderabad)
APS6	ap-southeast-4	Asia Pacifico (Melbourne)
CAN1	ca-central-1	Canada (Centrale)
UE	eu-west-1	Europa (Irlanda)
EUC1	eu-central-1	Europa (Francoforte)
EUC2	eu-central-2	Europa (Zurigo)
EUN1	eu-north-1	Europa (Stoccolma)
EUS1	eu-south-1	Europa (Milano)
EUS2	eu-south-2	Europa (Spagna)
EUW1	eu-west-1	Europa (Irlanda)
EUW2	eu-west-2	Europa (Londra)

Codice Regionale di fatturazione	Codice regione	Nome della regione
EUW3	eu-west-3	Europa (Parigi)
ILC1	Il-central-1	Israele (Tel Aviv)
MEC1	me-central-1	Medio Oriente (Emirati Arabi Uniti)
MES1	me-south-1	Medio Oriente (Bahrein)
SAE1	sa-east-1	Sud America (San Paolo)
USE1 (impostazione predefinita)	us-east-1	Stati Uniti orientali (Virginia settentrionale)
USE2	us-east-2	Stati Uniti orientali (Ohio)
UGE1	us-gov-east-1	Governo degli Stati Uniti orientali
UGW1	us-gov-west-1	Governo degli Stati Uniti occidentali
USW1	us-west-1	Stati Uniti occidentali (California settentrionale)
USW2	us-west-2	Stati Uniti occidentali (Oregon)

Nelle sezioni seguenti, si utilizzano modelli `REG-UsageType` per esaminare gli addebiti per DynamoDB, dove REG specifica la Regione in cui si è verificato l'utilizzo e usageType è il codice per il tipo di addebito. Ad esempio, se nel file CSV è presente una voce per `USW1-ReadCapacityUnit-Hrs`, significa che l'utilizzo è avvenuto nella Regione US-West-1 per la capacità di lettura allocata. In tal caso, l'elenco indicherebbe `REG-ReadCapacityUnit-Hrs`.

Argomenti

- Capacità di throughput
- Streams

- Storage
- Backup e ripristino
- Trasferimento dati
- CloudWatch Approfondimenti per i contributori
- DynamoDB Accelerator (DAX)

Capacità di throughput

Capacità allocata per letture e scritture

Quando si crea una tabella DynamoDB in modalità con capacità allocata, si specifica la capacità di lettura e scrittura che si prevede che l'applicazione richieda. Il tipo di utilizzo dipende dalla classe di tabella (Standard o Standard-Accesso infrequente). Il servizio di lettura e scrittura viene fornito in base al tasso di consumo al secondo, ma gli addebiti sono calcolati su base oraria secondo la capacità allocata.

UsageType	Unità	Granularità	Description
REG- -Ore ReadCapacityUnit	RCU/ora	Ora	Addebita le letture in modalità con capacità allocata utilizzando la classe di tabella Standard.
REG-IA-ReadCapacityUnit-Hrs	RCU/ora	Ora	Addebita le letture in modalità con capacità allocata utilizzando la classe di tabella Standard-IA.
REG- -Ore WriteCapacityUnit	WCU/ora	Ora	Addebita le scritture in modalità con capacità allocata utilizzando la classe di tabella Standard.

UsageType	Unità	Granularità	Description
REG-IA-WriteCapacityUnit-Hrs	WCU/ora	Ora	Addebita le scritture in modalità con capacità allocata utilizzando la classe di tabella Standard-IA.

Letture e scritture con capacità riservata

Con la capacità riservata sono previsti il pagamento di una commissione anticipata una tantum e l'impegno a un livello minimo di utilizzo allocato per un periodo di tempo. La capacità riservata viene fatturata a una tariffa oraria scontata. Qualsiasi capacità allocata in eccesso rispetto alla capacità riservata viene fatturata alle tariffe standard della capacità allocata. La capacità riservata è disponibile per unità di capacità di lettura e scrittura (RCU e WCU) allocate a Regione singola su tabelle DynamoDB che utilizzano la classe di tabella standard. La capacità riservata per 1 anno e per 3 anni viene fatturata utilizzando lo stesso metodo. SKUs

UsageType	Unità	Granularità	Description
REG HeavyUsage -:dynamodb.read	RCU/ora	In anticipo e poi mensilmente	I costi per le letture con capacità riservata sono i seguenti: un addebito anticipato una tantum e un addebito mensile all'inizio di ogni mese, che copre tutte le RCU/ora impegnate scontate durante il mese. Avrà voci corrispondenti a costo zero. REG-ReadCapacityUnit-Hrs
REG-:dynamodb.write HeavyUsage	WCU/ora	In anticipo e poi mensilmente	I costi per le scritture con capacità riservata

UsageType	Unità	Granularità	Description
			sono i seguenti: un addebito anticipat o una tantum e un addebito mensile all'inizio di ogni mese, che copre tutte le WCU/ora impegnate scontate durante il mese. Avrà voci corrispondenti a costo zero. REG-Write CapacityUnit-Hrs

Letture e scritture con capacità on demand

Quando si crea una tabella DynamoDB in modalità con capacità on demand, il pagamento avviene solo per le letture e le scritture eseguite dall'applicazione. I prezzi per le richieste di lettura e scrittura dipendono dalla classe di tabella.

UsageType	Unità	Granularità	Description
REG- ReadReque stUnits	RRUs	Unità	Addebita le letture in modalità con capacità on demand con classe di tabella Standard.
REGIA- ReadReque stUnits	RRUs	Unità	Addebita le letture in modalità con capacità on demand con classe di tabella Standard-IA.
REG- WriteRequ estUnits	WRUs	Unità	Addebita le scritture in modalità con capacità

UsageType	Unità	Granularità	Description
			on demand con la classe di tabella Standard.
REGIA- WriteRequestUnits	WRUs	Unità	Addebita le scritture in modalità con capacità on demand con la classe di tabella Standard-IA.

Letture e scritture per tabelle globali

DynamoDB addebita l'utilizzo delle tabelle globali in base alle risorse utilizzate in ciascuna tabella di replica. Per le tabelle globali fornite, le richieste di scrittura per le tabelle globali vengono misurate in replica WCUs (RWCu) anziché in modalità standard WCUs e le scritture su indici secondari globali nelle tabelle globali vengono misurate in. WCUs Per le tabelle globali su richiesta, le richieste di scrittura vengono misurate in modalità replicata (RWRu) anziché in modalità standard WRUs . WRUs Il numero di r WCUs o r WRUs utilizzati per la replica dipende dalla versione delle tabelle globali in uso. Il prezzo dipende dalla classe di tabella.

Le scritture sugli indici secondari globali (GSIs) vengono fatturate utilizzando unità di scrittura standard (and). WCUs WRUs Le richieste di lettura e l'archiviazione dei dati vengono fatturate in modo identico alle tabelle a Regione singola.

Se si aggiunge una replica di tabella per creare o estendere una tabella globale in nuove Regioni, DynamoDB addebita il ripristino della tabella nelle Regioni aggiunte per gigabyte di dati ripristinati. I dati ripristinati vengono addebitati come. REG-RestoreDataSize-Bytes Per ulteriori dettagli, consulta Backup e ripristino per DynamoDB. La replica tra Regioni e l'aggiunta di repliche alle tabelle che contengono dati comportano inoltre costi per il trasferimento dei dati in uscita.

Quando si seleziona la modalità con capacità on demand per le tabelle globali DynamoDB, il pagamento avviene solo per le risorse utilizzate dall'applicazione su ciascuna tabella di replica.

UsageType	Unità	Granularità	Description
REG- -Hrs ReplWrite CapacityUnit	rWCU/ora	Ora	Tabella globale, con provisioning, classe di tabella Standard.
REG-IA- -Ore ReplWriteCapacityU nit	rWCU/ora	Ora	Tabella globale, con provisioning, classe di tabella Standard-IA.
REG- ReplWrite RequestUnits	rWRU	Unità	Tabella globale, on demand, classe di tabella Standard.
REGIA- ReplWrite RequestUnits	rWRU	Unità	Tabella globale, on demand, classe di tabella Standard-IA.

Streams

DynamoDB dispone di due tecnologie di flussi, flussi DynamoDB e Kinesis. Ciascuna ha prezzi separati.

Per i flussi DynamoDB vengono addebitati i costi per la lettura dei dati nelle unità di richiesta di lettura. Ogni chiamata API `GetRecords` viene fatturata come richiesta di lettura dei flussi. Non ti vengono addebitati costi per le chiamate `GetRecords` API richiamate AWS Lambda come parte dei trigger di DynamoDB o dalle tabelle globali di DynamoDB come parte della replica.

UsageType	Unità	Granularità	Description
Reg-stream-RequestsCount	Conteggio	Unità	Unità di richiesta di lettura per i flussi DynamoDB.

Per il flusso di dati Amazon Kinesis viene addebitato il relativo utilizzo nelle unità di Change Data Capture. DynamoDB addebita un'unità di Change Data Capture per ogni scrittura (fino a 1 KB).

Per elementi di dimensioni superiori a 1 KB sono richieste ulteriori unità di Change Data Capture. Il pagamento avviene solo per le scritture eseguite dall'applicazione senza dover gestire la capacità di throughput sulla tabella.

UsageType	Unità	Granularità	Description
REG- -Kinesis ChangeDataCaptureUnits	Unità CDC	Unità	Unità di Change Data Capture per il flusso di dati Kinesis.

Storage

DynamoDB misura la dimensione dei dati fatturabili aggiungendo la dimensione in byte dei dati grezzi e un sovraccarico di archiviazione per elemento che dipende dalle funzionalità abilitate.

> **Note**
>
> I valori di utilizzo dell'archiviazione nel report CUR saranno più alti rispetto ai valori dell'archiviazione durante l'utilizzo di `DescribeTable`, perché `DescribeTable` non include il sovraccarico di archiviazione per elemento.

L'archiviazione viene calcolata su base oraria ma il prezzo mensile è calcolato sulla base di una media delle tariffe orarie.

Sebbene l'archiviazione `UsageType` utilizzi `ByteHrs` come suffisso, l'utilizzo dell'archiviazione nel report CUR viene misurato in GB e il prezzo è espresso in GB al mese.

UsageType	Unità	Granularità	Description
REG- - TimedStorage ByteHrs	GB	Mese	Quantità di archiviazione utilizzata dalle tabelle e dagli indici DynamoDB, per tabelle con la classe di tabella Standard.

UsageType	Unità	Granularità	Description
REGIA- - TimedStorage ByteHrs	GB	Mese	Quantità di archiviazione utilizzata dalle tabelle e dagli indici DynamoDB, per tabelle con la classe di tabella Standard-IA.

Backup e ripristino

DynamoDB offre due tipi di backup: backup con recupero point-in-time (PITR) e backup on demand. Gli utenti possono anche eseguire il ripristino da tali backup nelle tabelle DynamoDB. I costi riportati di seguito si riferiscono sia ai backup che ai ripristini.

I costi per l'archiviazione dei backup vengono addebitati il primo giorno del mese, e gli aggiustamenti applicati nel corso del mese man mano che i backup vengono aggiunti o rimossi. Per ulteriori informazioni, consulta il blog Understanding Amazon DynamoDB On-demand Backups and Billing

UsageType	Unità	Granularità	Description
REG- - TimedBackupStorage ByteHrs	GB	Mese	L'archiviazione utilizzata dai backup on demand delle tabelle e degli indici secondari locali DynamoDB.
Cronometrato - PITRStorage ByteHrs	GB	Mese	Lo spazio di archiviazione utilizzato dai backup point-in-time di ripristino (PITR). DynamoDB monitora continuamente le dimensioni delle tabelle con PITR abilitato durante tutto

UsageType	Unità	Granularità	Description
			il mese per determina re i costi di backup e le fatture per l'archivi azione fintantoché PITR è abilitato.
REG- -Byte RestoreDataSize	GB	Dimensione	La dimensione totale dei dati ripristinati (inclusi dati di tabella, indici secondari locali e indici secondari globali) misurata in GB dai backup DynamoDB.

AWS Backup

AWS Backup è un servizio di backup completamente gestito che semplifica la centralizzazione e l'automazione del backup dei dati tra i AWS servizi nel cloud e in locale. AWS Backup viene addebitato per l'archiviazione (archiviazione a caldo o freddo), le attività di ripristino e il trasferimento di dati tra regioni. I seguenti `UsageType` addebiti vengono visualizzati nella sezione «AWS Backup» ProductCode anziché in «AmazonDynamoDB».

UsageType	Unità	Granularità	Description
REG- WarmStorage -ByteHrs -DynamoDB	GB	Mese	L'archiviazione utilizzata dai backup DynamoDB gestiti da AWS Backup ne l corso del mese, misurata in GB al mese.

UsageType	Unità	Granularità	Description
REG- CrossRegion - WarmBytes - DynamoDB	GB	Dimensione	I dati trasferiti in una AWS regione diversa all'interno dello stesso account o su un account diverso. AWS Quando si copiano i backup da una Regione a un’altra, vengono addebitati i costi per i trasferimenti tra Regioni. L’addebito avviene sempre sull’account da cui vengono trasferiti i dati.
Reg-Restore- -DynamoDB WarmBytes	GB	Dimensione	La dimensione totale dei dati ripristinati dall’archiviazione a caldo, misurata in GB.
REG- ColdStorage - ByteHrs -DynamoDB	GB	Mese	L’archiviazione a freddo utilizzata dai backup di DynamoDB gestiti da AWS Backup durante il mese, misurata in GB al mese.
Reg-Restore- - DynamoDB ColdBytes	GB	Mese	La dimensione totale dei dati ripristinati dall’archiviazione a freddo, misurata in GB.

Importazione ed esportazione

È possibile esportare dati da DynamoDB in Amazon S3 o importare dati da Amazon S3 in una nuova tabella DynamoDB.

Sebbene `UsageType` utilizzi `Bytes` come suffisso, l'utilizzo delle esportazioni e delle importazioni nel report CUR viene misurato e valutato in GB.

UsageType	Unità	Granularità	Description
REG- ExportDataSize -Byte	GB	Dimensione	Il costo per l'esportazione dei dati in S3. Per DynamoDB l'addebito avviene per i dati esportati in base alle dimensioni della tabella di base DynamoDB (dati della tabella e indici secondari locali) nel punto temporale specificato in cui viene effettuata l'esportazione.
REG- -Byte ImportDataSize	GB	Dimensione	Il costo per l'importazione dei dati da S3. La dimensione viene calcolata in base alla dimensione dell'oggetto non compresso dei dati all'interno di Amazon S3. Non sono previsti costi aggiuntivi per l'importazione nelle tabelle con. GSIs

UsageType	Unità	Granularità	Description
REG- -Byte IncrementalExportDataSize	GB	Dimensione	L'addebito per la dimensione dei dati elaborati dal backup continuo per produrre esportazioni incrementali.

Trasferimento dati

L'attività di trasferimento dei dati può apparire associata al servizio DynamoDB. DynamoDB non addebita alcun costo per il trasferimento di dati in entrata e non addebita alcun costo per i dati trasferiti tra DynamoDB e AWS altri servizi all'interno della AWS stessa regione (in altre parole, 0,00 USD per GB). I dati trasferiti tra AWS regioni (ad esempio tra DynamoDB nella regione Stati Uniti orientali [Virginia settentrionale] e EC2 Amazon nella regione UE [Irlanda]) vengono addebitati su entrambi i lati del trasferimento.

UsageType	Unità	Granularità	Description
REG- In byte DataTransfer	GB	Unità	Dati trasferiti in entrata a DynamoDB da Internet.
REG- -Byte in uscita DataTransfer	GB	Unità	Dati trasferiti in uscita da DynamoDB a Internet.

CloudWatch Approfondimenti per i contributori

CloudWatch Contributor Insights for DynamoDB è uno strumento diagnostico per identificare le chiavi a cui si accede più di frequente e quelle con limitazioni nella tabella DynamoDB. I seguenti `UsageType` addebiti appaiono sotto «» anziché «DB»AmazonCloudWatch. ProductCode AmazonDynamo

UsageType	Unità	Granularità	Description
REG-CW: ContributorEventsManaged	Eventi elaborati	Unità	La quantità di eventi DynamoDB elaborati. Ad esempio, per una tabella con CloudWatch Contributor Insights abilitato, ogni volta che un elemento viene letto o scritto, viene conteggiato come un evento. Se la tabella ha una chiave di ordinamento, vengono addebitati due eventi.
REG-CW: ContributorRulesManaged	Numero di regole	Mese	DynamoDB crea regole per identificare gli elementi con maggiore accesso e le chiavi con maggiore limitazione (della larghezza di banda della rete) quando Contributor Insights di CloudWatch è abilitato. Questo costo viene addebitato per le regole aggiunte per ogni entità (tabelle e GSIs) configurata per la registrazione delle informazioni sui contributori. CloudWatch

DynamoDB Accelerator (DAX)

DynamoDB Accelerator (DAX) viene fatturato su base oraria in base al tipo di istanza selezionato per il servizio. I costi riportati di seguito si riferiscono alle istanze di DynamoDB Accelerator con provisioning. I seguenti `UsageType` addebiti vengono visualizzati sotto «AmazonDAX» ProductCode anziché «AmazonDynamoDB».

UsageType	Unità	Granularità	Description
REG-:dax-NodeUsage <INSTANCETYPE>	Nodi/ora	Ora	L'utilizzo orario di un particolare tipo di istanza. Il prezzo si riferisce ai nodi/ora utilizzati, dal momento in cui un nodo viene avviato fino alla sua chiusura. Ogni nodo/ora parziale utilizzato verrà fatturato come un'ora completa. I costi di DAX sono calcolati per ogni nodo di un cluster DAX. Se si dispone di un cluster con più nodi, nel report di fatturazione verranno visualizzate più voci.

Il tipo di istanza sarà uno dei valori dell'elenco seguente. Per informazioni dettagliate sui tipi di nodo, consulta Nodi.

- 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

Migrazione di una tabella DynamoDB da un account a un altro

È possibile migrare una tabella Amazon DynamoDB da un account a un altro per implementare una strategia multi-account o una strategia di backup. È possibile farlo anche per motivi di test, debug o conformità. Un caso d'uso comune è la copia di tabelle DynamoDB in ambienti di produzione, staging, test e sviluppo in cui ogni ambiente utilizza un account diverso. AWS

DynamoDB offre due opzioni per la migrazione delle tabelle da AWS un account all'altro:

- AWS Backup per il backup e il ripristino tra account: AWS Backup è un servizio di backup completamente gestito che consente di gestire centralmente i backup su più AWS servizi. Con la sua funzionalità di backup e ripristino tra account, è possibile eseguire il backup di una tabella DynamoDB in un account e ripristinare il backup su un altro account della stessa Organizzazione AWS .
- Esportazione e importazione di DynamoDB su Amazon S3: l'utilizzo delle funzionalità di esportazione e importazione di DynamoDB su Amazon S3 consente di eseguire un'esportazione completa in un bucket Amazon S3 e quindi importare tali dati in una nuova tabella in un altro account. AWS Questo approccio è adatto quando è necessario migrare tra account che non fanno parte della stessa AWS organizzazione o se non si desidera utilizzarlo. AWS Backup

> **Note**
>
> L'importazione da Amazon S3 non supporta tabelle con indici secondari locali (LSIs), ma supporta indici secondari globali (). GSIs Per ulteriori informazioni su and, consulta. LSIs GSIs Miglioramento dell’accesso ai dati con gli indici secondari in DynamoDB

Argomenti

- Esegui la migrazione di una tabella utilizzando AWS Backup per il backup e il ripristino tra account

- Migrazione di una tabella utilizzando l'esportazione in S3 e l'importazione da S3

Esegui la migrazione di una tabella utilizzando AWS Backup per il backup e il ripristino tra account

Prerequisiti

- AWS Gli account di origine e di destinazione devono appartenere alla stessa organizzazione nel servizio AWS Organizations
- Autorizzazioni valide di AWS Identity and Access Management (IAM) per creare e utilizzare casseforti AWS Backup

Per ulteriori informazioni sulla configurazione dei backup tra account, consulta Creazione di copie di backup tra account. AWS

Informazioni sui prezzi

AWS costi per il backup (in base alla dimensione della tabella), qualsiasi copia dei dati tra AWS regioni (in base alla quantità di dati), per il ripristino (in base alla quantità di dati) e per eventuali costi di archiviazione correnti. Per evitare addebiti continui, è possibile eliminare il backup se non serve dopo il ripristino.

Per ulteriori informazioni sui prezzi, consulta Prezzi di AWS Backup.

Fase 1: abilitare le funzionalità avanzate per DynamoDB e il backup tra account

1. Sia nell' AWS account di origine che in quello di destinazione, accedi alla console di AWS gestione e apri la console AWS di Backup.
2. Seleziona l'opzione Impostazioni.
3. In Funzionalità avanzate per i backup di Amazon DynamoDB, verifica che le Funzionalità avanzate siano abilitate. In caso contrario, seleziona Abilita.
4. In Gestione di più account, per Backup tra account, seleziona Abilita.

Fase 2: creare un vault di backup nell'account di origine e nell'account di destinazione

1. Negli AWS account di origine, apri la console AWS di Backup.

2. Scegliere Vault di Backup.
3. Scegliere Crea vault di Backup.
4. Copia e salva l'Amazon Resource Name (ARN) degli archivi di backup creati e l'account di destinazione. AWS
5. Avrai bisogno degli archivi ARNs di backup di origine e di destinazione per copiare il backup della tabella DynamoDB tra account.

Fase 3: creare un backup della tabella DynamoDB nell’account di origine

1. Dalla pagina del pannello di controllo di AWS Backup seleziona Crea backup on demand.
2. Nella sezione Impostazioni, seleziona DynamoDB come Tipo di risorsa, quindi seleziona il nome della tabella.
3. Nell’elenco a discesa Vault di backup, seleziona il vault di backup creato nell’account di origine.
4. Seleziona il Periodo di conservazione desiderato.
5. Seleziona Crea backup on demand.
6. Monitora lo stato del processo nella scheda Lavori di Backup della pagina Processi di AWS Backup.

Fase 4: copiare il backup della tabella DynamoDB dall’account di origine all’account di destinazione

1. Al termine del processo di backup, apri la AWS Backup console nell'account di origine e scegli Backup vault.
2. In Backup, seleziona il backup della tabella DynamoDB. Seleziona Operazioni, quindi seleziona Copia.
3. Inserisci la AWS regione dell'account di destinazione.
4. Per ARN di vault esterno, inserisci l’ARN del vault di backup creato nell’account di destinazione.
5. Nel vault di backup dell’account di destinazione, abilita l’accesso da un account di origine per consentire la copia dei backup.

Fase 5: ripristinare il backup della tabella DynamoDB nell’account di destinazione

1. Nell' AWS account di destinazione, apri la AWS Backup console e scegli Backup vault

2. In Backup, seleziona il backup copiato dall'account di origine. Seleziona Operazioni, quindi Ripristina.
3. Inserisci il nome della nuova tabella DynamoDB, la crittografia di questa nuova tabella, la chiave con cui vuoi criptare il ripristino ed eventuali altre opzioni.
4. Una volta completato il ripristino, lo stato della tabella verrà visualizzato come Attivo.

Migrazione di una tabella utilizzando l'esportazione in S3 e l'importazione da S3

Prerequisiti

- È necessario abilitare Point-in-Time Recovery (PITR) per la tabella per eseguire l'esportazione in S3. Per ulteriori informazioni, consulta Abilitazione del recupero point-in-time in DynamoDB.
- Autorizzazioni IAM valide per eseguire l'esportazione. Per ulteriori informazioni, consulta Richiesta di esportazione di una tabella in DynamoDB.
- Autorizzazioni IAM valide sufficienti per eseguire l'importazione. Per ulteriori informazioni, consulta Richiesta di importazione di una tabella in DynamoDB.

Informazioni sui prezzi

AWS addebiti per PITR (in base alle dimensioni della tabella e per quanto tempo PITR è abilitato). Se non si ha bisogno di PITR tranne che per l'esportazione, è possibile disattivarlo al termine dell'operazione. AWS inoltre effettua addebiti per le richieste effettuate nei confronti di S3, per l'archiviazione dei dati esportati in S3 e per l'importazione (in base alla dimensione non compressa dei dati importati).

Per ulteriori informazioni sui prezzi di DynamoDB, consulta Prezzi di Amazon DynamoDB.

> **Note**
>
> Esistono limiti alla dimensione e al numero di oggetti consentiti durante l'importazione da S3 a DynamoDB. Per ulteriori informazioni, consulta Quote di importazione.

Richiesta di esportazione di una tabella ad Amazon S3

1. Accedi alla console di AWS gestione e apri la console DynamoDB.

2. Nel riquadro di navigazione sul lato sinistro della console, scegli Exports to S3 (Esportazioni su S3).
3. Scegli una tabella di origine e un bucket S3 di destinazione. Immetti l'URL del bucket dell'account di destinazione utilizzando il formato `s3://bucketname/prefix`. `/prefix` è una cartella facoltativa che consente di mantenere organizzato il bucket di destinazione.
4. Seleziona Esportazione completa. Un'esportazione completa restituisce lo snapshot dell'intera tabella nel punto temporale specificato.
   a. Seleziona Ora corrente per esportare l'ultimo snapshot completo della tabella.
   b. Per Formato di file esportato, seleziona tra DynamoDB JSON e Amazon Ion. L'opzione predefinita è DynamoDB JSON.
5. Fai clic sul pulsante Esporta per iniziare l'esportazione.
6. Le esportazioni di tabelle di piccole dimensioni dovrebbero essere completate in pochi minuti, ma le tabelle con dimensioni nell'intervallo dei terabyte possono richiedere più di un'ora.

Richiesta di importazione di una tabella da Amazon S3

1. Accedi alla console di AWS gestione e apri la console DynamoDB.
2. Nel pannello di navigazione sul lato sinistro della console, scegliere Exports to S3 (Esportazioni su S3).
3. Nella pagina visualizzata, selezionare Import from S3 (Importazione da S3).
4. Immettere l'URL di origine di Amazon S3. È possibile trovarla anche usando il pulsante Sfoglia S3. Il formato del percorso previsto è `s3://bucket/prefix/AWSDynamoDB/<XXXXXXXX-XXXXXX>/data/`.
5. Specificare se si è il proprietario nella casella S3 bucket owner (Proprietario bucket S3).
6. In Compressione dei file di importazione, seleziona GZIP in modo che corrisponda all'esportazione.
7. In Formato file di importazione, seleziona JSON DynamoDB in modo che corrisponda all'esportazione.
8. Seleziona Avanti. In Specifica i dettagli della tabella, seleziona le opzioni per la nuova tabella che verrà creata per archiviare i dati.
9. Seleziona Avanti. Per Configura le impostazioni della tabella, personalizza eventuali impostazioni aggiuntive della tabella, se applicabile.

10. Selezionare di nuovo Next (Successivo) per rivedere le opzioni di importazione, quindi fare clic su Import (Importa) per avviare l'attività di importazione. Vedrai la nuova tabella elencata in Importazioni da S3 con lo stato Importazione in corso. Non è possibile accedere alla tabella durante questo periodo. Le importazioni di piccole dimensioni dovrebbero essere completate in pochi minuti, ma le tabelle con dimensioni nell'intervallo dei terabyte possono richiedere più di un'ora.
11. Una volta completata l'importazione, verrà visualizzato lo stato Attivo. A questo punto è possibile iniziare a utilizzare la tabella.

Mantenere le tabelle sincronizzate durante la migrazione

Sospendendo le operazioni di scrittura sulla tabella di origine per tutta la durata della migrazione, l'origine e l'output dovrebbero corrispondere esattamente dopo la migrazione. Se non è possibile sospendere le operazioni di scrittura, la tabella di destinazione normalmente risulterà leggermente in ritardo rispetto all'origine dopo la migrazione. Per riallineare la tabella di origine, è possibile utilizzare i flussi (flussi DynamoDB o flusso di dati Kinesis per DynamoDB) per riprodurre le scritture avvenute nella tabella di origine dopo il backup o l'esportazione.

Sarebbe necessario iniziare a leggere i record del flusso prima del timestamp in cui la tabella di origine è stata esportata in S3. Ad esempio, se l'esportazione in S3 è avvenuta alle 14:00 e l'importazione nella tabella di destinazione è stata conclusa alle 23:00, la lettura del flusso DynamoDB deve essere avviata alle 13:58. La tabella delle opzioni di streaming per il Change Data Capture riassume le funzionalità di ciascun modello di streaming.

L'uso dei flussi DynamoDB con Lambda offre un approccio semplificato per la sincronizzazione dei dati tra le tabelle DynamoDB di origine e di destinazione. È possibile utilizzare una funzione Lambda per riprodurre ogni scrittura nella tabella di destinazione.

> **Note**
>
> Gli elementi vengono conservati nei flussi DynamoDB per 24 ore, quindi è necessario pianificare il completamento del backup e del ripristino o l'esportazione e l'importazione all'interno di tale finestra.

Prontuario per l'integrazione di DAX con le applicazioni DynamoDB

DynamoDB Accelerator (DAX) è un servizio di caching compatibile con DynamoDB che offre rapide prestazioni in memoria per le applicazioni con requisiti impegnativi, come le applicazioni con elevati requisiti di lettura. Con DAX è possibile ottenere tempi di risposta di pochi microsecondi per accedere ai dati richiesti di frequente. Questo prontuario di DynamoDB Accelerator fornisce approfondimenti completi e best practice per l'integrazione di DAX con le applicazioni DynamoDB.

Questa guida offre le conoscenze di base per chi è alle prime armi con DAX o desidera ottimizzare le configurazioni esistenti. Questa guida tratta vari argomenti, ad esempio, quando utilizzare DAX e creare un cluster DAX. Sono inclusi anche esempi pratici e spiegazioni dettagliate per aiutare a implementare efficacemente DAX nei propri progetti. Infine, vengono illustrate strategie avanzate da implementare per massimizzare le capacità di caching di DAX e garantire applicazioni veloci e scalabili.

Argomenti

- Valutazione dell'idoneità di DAX per i vari casi d'uso
- Configurazione del client DAX
- Configurazione del cluster DAX
- Dimensionamento del cluster DAX
- Implementazione di un cluster
- Gestione delle operazioni sui cluster
- Monitoraggio di DAX

Valutazione dell'idoneità di DAX per i vari casi d'uso

Questa sezione illustra quando e perché usare DAX. L'utilizzo di questa guida consente di determinare se l'integrazione di DAX con DynamoDB è vantaggiosa per i modelli di carico di lavoro, i requisiti di prestazioni e le esigenze di coerenza dei dati dell'applicazione. Vengono inoltre illustrati gli scenari in cui DAX potrebbe non essere adatto, ad esempio, carichi di lavoro con elevati requisiti di scrittura e dati con accesso infrequente.

In questa sezione

- Quando e perché scegliere DAX
- Quando non utilizzare DAX

Quando e perché scegliere DAX

È possibile prendere in considerazione l'aggiunta di DAX allo stack di applicazioni in diversi scenari. Ad esempio, DAX è utile per ridurre la latenza complessiva delle richieste di lettura su DynamoDB o per ridurre al minimo le letture ripetute degli stessi dati da una tabella. L'elenco seguente presenta esempi di scenari in cui è possibile trarre vantaggio dall'integrazione di DAX con DynamoDB:

- Requisito di alte prestazioni
  - Letture a bassa latenza: è consigliabile prendere in considerazione l'utilizzo di DAX se l'applicazione richiede tempi di risposta di pochi microsecondi per letture a coerenza finale. DAX può anche ridurre drasticamente i tempi di risposta per l'accesso ai dati con letture frequenti.
- Carichi di lavoro a elevati requisiti di lettura
  - Applicazioni ad alta intensità di lettura: per le applicazioni con un read-to-write rapporto elevato, ad esempio 10:1 o più, DAX genera più accessi alla cache e meno dati obsoleti. In questo modo si riducono le letture su una tabella. Per evitare di leggere dati obsoleti dalla cache se l'applicazione ha elevati requisiti di scrittura, assicurati di impostare un valore Utilizzo del Time to Live in DynamoDB inferiore per la cache.
  - Caching delle query più comuni: se l'applicazione legge spesso gli stessi dati, ad esempio i prodotti più diffusi su una piattaforma di e-commerce, DAX può gestire queste richieste direttamente dalla sua cache.
- Schemi di traffico con momenti di maggiore carico
  - Dimensionamento più fluido delle tabelle: DAX aiuta a mitigare gli impatti dei picchi di traffico improvvisi. DAX fornisce un buffer per aumentare verticalmente la capacità delle tabelle DynamoDB in modo corretto, riducendo il rischio di limitazione (della larghezza di banda della rete) della lettura.
  - Maggiore throughput di lettura per ogni elemento: DynamoDB alloca partizioni individuali per ogni elemento. Tuttavia, una partizione inizia a sottoporre a limitazione (della larghezza di banda della rete) le letture di un elemento quando raggiunge le 3.000 unità di capacità di lettura (RCU). DAX consente di scalare le letture di un singolo elemento oltre 3.000 RCU.
- Ottimizzazione dei costi
  - Riduzione dei costi di DynamoDB: la lettura con DAX può ridurre le letture inviate a una tabella DynamoDB, con un impatto diretto sui costi. Con un'elevata percentuale di riscontri nella cache, il costo ridotto di lettura delle tabelle può superare il costo di un cluster DAX, il che comporta una riduzione dei costi netti.
- Requisiti di coerenza dei dati

- Coerenza finale: DAX supporta letture a coerenza finale. Ciò rende DAX adatto a casi d'uso in cui la coerenza immediata non è fondamentale.
- Caching di tipo write-through: le scritture eseguite su DAX sono di tipo write-through. Una volta che DAX conferma di aver scritto un elemento in DynamoDB, mantiene la versione dell'elemento nella cache degli elementi. Questo meccanismo di write-through aiuta a mantenere una maggiore coerenza dei dati tra cache e database, ma utilizza risorse aggiuntive del cluster DAX.

Quando non utilizzare DAX

Sebbene DAX sia potente, non è adatto a tutti gli scenari. L'elenco seguente presenta esempi di scenari in cui l'integrazione di DAX con DynamoDB non è adatta:

- Carichi di lavoro con elevati requisiti di scrittura: il vantaggio principale di DAX è la velocizzazione delle letture, ma le scritture utilizzano più risorse DAX rispetto alle letture. Se l'applicazione è prevalentemente caratterizzata da elevati requisiti di scrittura, i vantaggi di DAX potrebbero essere limitati.
- Dati con letture infrequenti: se l'applicazione accede ai dati raramente o se l'applicazione accede a un'ampia gamma di dati riutilizzati raramente (dati inutilizzati), è probabile che si verifichi un cache hit ratio basso. In questo caso, il sovraccarico legato alla manutenzione della cache potrebbe non giustificare l'aumento delle prestazioni.
- Letture o scritture in blocco: se l'applicazione esegue più scritture in blocco rispetto alle singole scritture, è consigliabile utilizzare il write-around su DAX. Inoltre, per le letture collettive, è necessario eseguire scansioni complete della tabella direttamente su una tabella DynamoDB.
- Requisiti di coerenza o transazione elevati: DAX trasmette letture e TransactGetItemschiamate estremamente coerenti a una tabella DynamoDB. È necessario effettuare queste letture sul cluster DAX per evitare di utilizzare le risorse del cluster. Gli elementi letti in questo modo non verranno archiviati nella cache; pertanto, l'instradamento di tali elementi tramite DAX non ha utilità.
- Applicazioni semplici con requisiti prestazionali modesti: per le applicazioni con requisiti prestazionali modesti e tolleranza per la latenza diretta di DynamoDB, la complessità e il costo dell'aggiunta di DAX potrebbero non essere necessari. Da solo, DynamoDB gestisce un throughput elevato e fornisce prestazioni a una cifra in pochi millisecondi.
- Esigenze di query complesse che vanno oltre l'accesso chiave-valore: DAX è ottimizzato per modelli di accesso chiave-valore. Se l'applicazione richiede funzionalità di query complesse con

filtri complessi, come le operazioni Query e Scan, i vantaggi del caching con DAX potrebbero essere limitati.

In queste situazioni, usa Amazon ElastiCache (Redis OSS) come alternativa. ElastiCache (Redis OSS) supporta strutture di dati avanzate, come elenchi, set e hash. Offre anche funzionalità come pub/sub, indici geospaziali e scripting.

- Requisiti di conformità: attualmente DAX non offre gli stessi accreditamenti di conformità di DynamoDB. Ad esempio, DAX non ha ancora ottenuto l'accreditamento SOC.

Configurazione del client DAX

Il cluster DAX è un cluster basato su istanze a cui è possibile accedere utilizzando vari DAX. SDKs Ogni SDK offre agli sviluppatori opzioni configurabili, come requestTimeout e connessioni, per soddisfare requisiti applicativi specifici.

Durante la configurazione di un client DAX, una considerazione fondamentale è il dimensionamento dell'applicazione client, in particolare il rapporto tra le istanze client e le istanze del server DAX (che dispone di un massimo di 11 istanze). Parchi di istanze client di grandi dimensioni possono generare numerose connessioni alle istanze del server DAX, con il rischio di sovraccaricarle. Questa guida descrive le best practice per la configurazione del client DAX.

Best practice

1. Istanze client: implementazione delle istanze client singleton per garantire il riutilizzo delle istanze tra le richieste. Per informazioni sull'implementazione, consulta the section called "Fase 4: esecuzione di una applicazione di esempio".
2. Timeout delle richieste: sebbene le applicazioni richiedano spesso timeout di richiesta bassi per garantire una latenza minima per i sistemi a monte, impostare timeout troppo bassi può causare problemi. Timeout bassi possono causare frequenti riconnessioni alle istanze del server quando i server DAX presentano picchi di latenza temporanei. Quando si verifica un timeout, il client DAX interrompe la connessione esistente al nodo del server e ne stabilisce una nuova. Poiché la creazione della connessione richiede molte risorse, numerose connessioni consecutive possono sovraccaricare i server DAX. Consigliamo quanto segue:
   - Mantieni le impostazioni predefinite di timeout delle richieste.
   - Se sono necessari timeout inferiori, implementa thread dell'applicazione separati con valori di timeout inferiori e includi meccanismi di ripetizione dei tentativi con backoff esponenziale.

3. Timeout di connessione: per la maggior parte delle applicazioni, si consiglia di mantenere le impostazioni di timeout di connessione predefinite.
4. Connessioni simultanee: alcune SDKs, come JavaV2, consentono la regolazione delle connessioni simultanee al server DAX. Considerazioni chiave:
   - Le istanze del server DAX possono gestire fino a 40.000 connessioni simultanee.
   - Le impostazioni predefinite sono adatte alla maggior parte dei casi d'uso.
   - Le istanze client di grandi dimensioni combinate con connessioni simultanee elevate possono sovraccaricare i server.
   - Valori di connessione simultanei inferiori riducono il rischio di sovraccarico del server.
   - Esempio di calcolo delle prestazioni:
     - Supponendo una latenza di richiesta di 1 ms, ogni connessione può teoricamente gestire 1.000 richieste al secondo.
     - Per un cluster a 3 nodi, una singola istanza client connessa a tutti i nodi può elaborare 3.000 richieste al secondo.
     - Con 10 connessioni, il client può gestire circa 30.000 richieste al secondo.

     Raccomandazione: inizia con l'impostazione di un numero inferiore di connessioni simultanee e convalida mediante test delle prestazioni con i modelli di carico di lavoro di produzione previsti.

Configurazione del cluster DAX

Il cluster DAX è un cluster gestito, ma è possibile modificarne le configurazioni in base ai requisiti delle applicazioni. A causa della stretta integrazione con le operazioni API DynamoDB, è necessario considerare i seguenti aspetti quando si integra l'applicazione con DAX.

In questa sezione

- Prezzi di DAX
- Cache degli elementi e cache delle query
- Selezione dell'impostazione del TTL per le cache
- Caching di più tabelle con un cluster DAX
- Replica dei dati nelle tabelle globali DAX e DynamoDB
- Disponibilità di DAX nelle Regioni
- Comportamento del caching di DAX

Prezzi di DAX

Il costo di un cluster dipende dal numero e dalla dimensione dei nodi allocati. Ogni nodo viene fatturato per ogni ora di esecuzione nel cluster. Per ulteriori informazioni, consulta Prezzi di Amazon DynamoDB.

I riscontri nella cache non comportano costi per DynamoDB, ma influiscono sulle risorse del cluster DAX. I mancati riscontri nella cache comportano costi di lettura di DynamoDB e richiedono risorse DAX. Le scritture comportano costi di scrittura di DynamoDB e influiscono sulle risorse del cluster DAX per delegare la scrittura.

Cache degli elementi e cache delle query

DAX gestisce una cache degli elementi e una cache delle query. Comprendere le differenze tra queste cache può aiutare a determinare le caratteristiche di prestazioni e coerenza offerte all'applicazione.

Caratteristiche delle cache	Cache degli elementi	Cache delle query
Scopo	Memorizza i risultati e le operazioni API. GetItemBatchGetItem	Archivia i risultati delle operazioni API Query e Scan. Queste operazioni possono restituire più elementi in base alle condizioni di query anziché a chiavi di elemento specifiche.
Tipo di accesso	Utilizza accessi basati su chiavi. Quando un'applicazione richiede dati attraverso `GetItem` o `BatchGetItem`, DAX controlla innanzitutto la cache degli elementi utilizzando la chiave primaria degli elementi richiesti. Se l'elemento è archiviato nella cache e non è scaduto, DAX	Utilizza l'accesso basato su parametri. DAX archivia nella cache il set di operazioni API `Query` e `Scan`. DAX gestisce le richieste successive con gli stessi parametri che includono le stesse condizioni di query, tabella e indice, dalla cache. Ciò riduce significativamente i tempi di risposta e l'utilizz

Caratteristiche delle cache	Cache degli elementi	Cache delle query
	lo restituisce immediatamente senza accedere alla tabella DynamoDB.	o di throughput di lettura di DynamoDB.
Invalidamento della cache	DAX replica automaticamente gli elementi aggiornati nella cache degli elementi dei nodi del cluster DAX nei seguenti scenari: • Scrittura di un aggiornamento di un elemento tramite la cache. • Lettura di una versione aggiornata dell'elemento dalla tabella.	La cache delle query è più difficile da invalidare rispetto alla cache degli elementi. Gli aggiornamenti degli elementi potrebbero non corrispondere direttamente alle query o alle scansioni archiviate nella cache. È necessario ottimizzare attentamente il TTL della cache delle query per mantenere la coerenza dei dati. Le scritture tramite DAX o la tabella di base non si riflettono nella cache delle query finché il TTL non fa scadere la risposta precedentemente archiviata nella cache e DAX non esegue una nuova query su DynamoDB.
Indice secondario globale	Poiché l'operazione API `GetItem` non è supportata sugli indici secondari locali o sugli indici secondari globali, la cache degli elementi archivia nella cache solo le letture dalla tabella di base.	La cache delle query archivia nella cache le query relative sia alle tabelle che agli indici.

Selezione dell'impostazione del TTL per le cache

Il TTL determina il periodo per il quale i dati vengono archiviati nella cache prima che diventino obsoleti. Dopo questo periodo, i dati vengono aggiornati automaticamente alla richiesta successiva. La scelta dell'impostazione del TTL corretta per le cache DAX implica un equilibrio tra l'ottimizzazione delle prestazioni delle applicazioni e la coerenza dei dati. Poiché non esiste un'impostazione del TTL universale che funzioni per tutte le applicazioni, l'impostazione del TTL ottimale varia in base alle caratteristiche e ai requisiti specifici dell'applicazione. Si consiglia di iniziare con un'impostazione del TTL conservativa utilizzando questo prontuario. Quindi sarà possibile modificare in modo iterativo l'impostazione del TTL in base ai dati e alle informazioni sulle prestazioni dell'applicazione.

DAX conserva un elenco degli elementi utilizzati meno di recente (LRU) per la cache di elementi. L'elenco LRU tiene traccia della prima scrittura o dell'ultima lettura degli elementi dalla cache. Quando la memoria del nodo DAX è piena, DAX elimina gli elementi più vecchi, anche se non sono ancora scaduti, per fare spazio a quelli nuovi. L'algoritmo LRU è sempre abilitato e non è configurabile dall'utente.

Per impostare una durata del TTL adatta alle applicazioni, è necessario considerare i seguenti punti:

Funzionamento dei modelli di accesso ai dati

- Carichi di lavoro con elevati requisiti di lettura: per le applicazioni con carichi di lavoro di questo tipo e aggiornamenti dei dati poco frequenti, imposta una durata del TTL più lunga per ridurre il numero di mancati riscontri nella cache. Una durata del TTL più lunga riduce anche la necessità di accedere alla tabella DynamoDB sottostante.
- Carichi di lavoro con elevati requisiti di scrittura: per le applicazioni con aggiornamenti frequenti che non vengono scritti tramite DAX, imposta una durata del TTL più breve per garantire che la cache rimanga coerente con il database. Una durata del TTL più breve riduce anche il rischio di fornire dati obsoleti.

Valutazione dei requisiti prestazionali delle applicazioni

- Sensibilità alla latenza: se l'applicazione richiede una bassa latenza rispetto all'aggiornamento dei dati, utilizza una durata del TTL più lunga. Una durata del TTL più lunga massimizza i riscontri nella cache, riducendo la latenza media di lettura.
- Throughput e scalabilità: una durata del TTL più lunga riduce il carico sulle tabelle DynamoDB e migliora il throughput e la scalabilità. Tuttavia, è necessario bilanciare questo aspetto con la necessità di up-to-date dati.

Analisi dell'espulsione della cache e dell'utilizzo della memoria

- Limiti della memoria cache: monitora l'utilizzo della memoria del cluster DAX. Una durata del TTL più lunga può archiviare più dati nella cache, il che potrebbe portare a raggiungere i limiti di memoria e a espulsioni basate in base all'algoritmo LRU.

Utilizzo di metriche e monitoraggio per regolare il TTL

Esamina regolarmente le metriche, ad esempio i riscontri e i mancati riscontri nella cache e l'utilizzo della CPU e della memoria. Modifica l'impostazione del TTL in base a queste metriche per trovare un equilibrio ottimale tra prestazioni e aggiornamento dei dati. Se i mancati riscontri nella cache sono elevati e l'utilizzo della memoria è basso, aumenta la durata del TTL per aumentare la percentuale di riscontri nella cache.

Considerazione dei requisiti aziendali e della conformità

Le policy di conservazione dei dati potrebbero dettare la durata massima del TTL che è possibile impostare per le informazioni sensibili o personali.

Comportamento della cache in caso di impostazione del TTL su zero

Se si imposta il TTL su 0, la cache degli elementi e la cache delle query presentano i seguenti comportamenti:

- Cache degli elementi: gli elementi nella cache vengono aggiornati solo quando si verifica un'espulsione in base all'algoritmo LRU o un'operazione write-through.
- Cache delle query: le risposte alle query non vengono archiviate nella cache.

Caching di più tabelle con un cluster DAX

Per i carichi di lavoro con più tabelle DynamoDB di piccole dimensioni che non richiedono cache individuali, un singolo cluster DAX archivia nella cache le richieste per queste tabelle. Ciò consente un uso più flessibile ed efficiente di DAX, in particolare per le applicazioni che accedono a più tabelle e richiedono letture ad alte prestazioni.

Analogamente al APIs piano dati DynamoDB, le richieste DAX richiedono un nome di tabella. Se si utilizzano più tabelle nello stesso cluster DAX, non è necessaria alcuna configurazione specifica. Tuttavia, è necessario verificare che le autorizzazioni di sicurezza del cluster consentano l'accesso a tutte le tabelle archiviate nella cache.

Considerazioni sull'utilizzo di DAX con più tabelle

Durante l'utilizzo di DAX con più tabelle DynamoDB, è necessario considerare i seguenti punti:

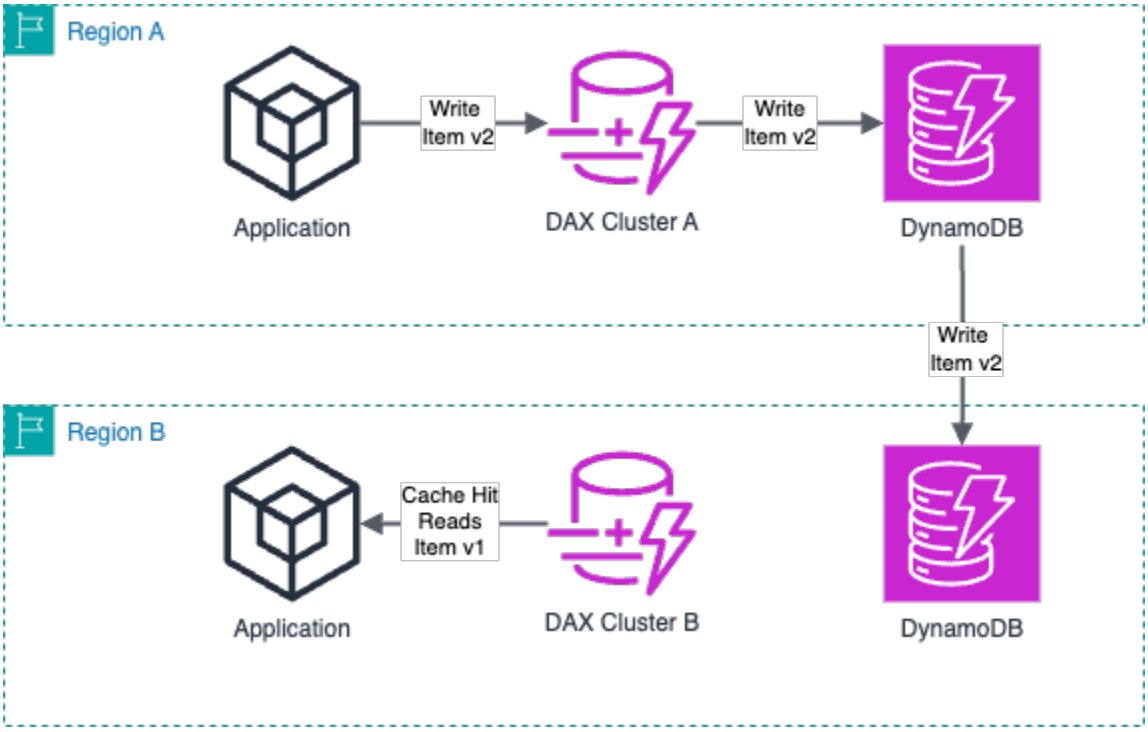
- Gestione della memoria: quando si utilizza DAX con più tabelle, è necessario considerare la dimensione totale del set di dati di lavoro. Tutte le tabelle del set di dati condivideranno lo stesso spazio di memoria del tipo di nodo selezionato.
- Allocazione delle risorse: le risorse del cluster DAX sono condivise tra tutte le tabelle archiviate nella cache. Tuttavia, una tabella a traffico elevato può causare l'espulsione dei dati dalle tabelle più piccole adiacenti.
- Economie di scala: raggruppa le risorse più piccole in un cluster DAX più grande per calcolare la media del traffico secondo uno schema più stabile. Per il numero totale di risorse di lettura richieste dal cluster DAX, è anche conveniente disporre di tre o più nodi. Ciò aumenta anche la disponibilità di tutte le tabelle archiviate nella cache del cluster.

Replica dei dati nelle tabelle globali DAX e DynamoDB

DAX è un servizio basato sulla Regione, quindi un cluster è a conoscenza solo del traffico all'interno della sua Regione AWS. Le tabelle globali scrivono nella cache in modalità write-around quando replicano i dati da un'altra Regione.

Una durata del TTL più lunga può far sì che i dati non aggiornati rimangano nella Regione secondaria più a lungo rispetto alla Regione principale. Ciò può causare mancati riscontri nella cache locale della Regione secondaria.

Il diagramma seguente mostra la replica dei dati che si verifica a livello di tabella globale nella Regione di origine A. Il cluster DAX nella Regione B non è immediatamente a conoscenza dei nuovi dati replicati dalla Regione di origine A.



Disponibilità di DAX nelle Regioni

Non tutte le Regioni che supportano le tabelle DynamoDB supportano l'implementazione di cluster DAX. Se l'applicazione richiede una bassa latenza di lettura tramite DAX, consulta innanzitutto l'elenco delle Regioni che supportano DAX. Quindi, seleziona una Regione per la tabella DynamoDB.

Comportamento del caching di DAX

DAX esegue caching su metadati e caching negativo. La comprensione di questi comportamenti di caching aiuta a gestire in modo efficace i metadati degli attributi degli elementi archiviati nella cache e delle voci della cache negativa.

- Caching dei metadati: i cluster DAX mantengono indefinitamente i metadati sui nomi degli attributi degli elementi archiviati nella cache. Questi metadati persistono anche dopo la scadenza dell'elemento o dopo l'espulsione dalla cache.

  Col tempo, le applicazioni che utilizzano un numero illimitato di nomi di attributi possono causare il riempimento totale della memoria nel cluster DAX. Questo limite si applica solo ai nomi di attributi di primo livello, ma non ai nomi di attributi annidati. Esempi di nomi di attributi illimitati includono timestamp e session. UUIDs IDs Sebbene sia possibile utilizzare timestamp e session IDs come valori di attributo, si consiglia di utilizzare nomi di attributi più brevi e prevedibili.

- Caching negativo: se si verifica un errore nella cache e la lettura da una tabella DynamoDB non produce elementi corrispondenti, DAX aggiunge una voce di cache negativa nella rispettiva cache degli elementi o delle query. Questa voce rimane valida fino alla scadenza della durata del TTL della cache o fino a quando non si verifica una procedura write-through. DAX continua a restituire questa voce della cache negativa per le richieste future.

  Se il comportamento di caching negativo non si adatta al modello dell'applicazione, leggi direttamente la tabella DynamoDB quando DAX restituisce un risultato vuoto. Si consiglia inoltre di impostare una durata della cache del TTL inferiore per evitare risultati vuoti a lungo termine nella cache e migliorare la coerenza con la tabella.

Dimensionamento del cluster DAX

La capacità e la disponibilità totali di un cluster DAX dipendono dal tipo e dal numero di nodi. Un numero maggiore di nodi nel cluster ne aumenta la capacità di lettura, ma non la capacità di scrittura. I tipi di nodi più grandi (fino a r5.8xlarge) possono gestire più scritture, ma un numero troppo basso di nodi può influire sulla disponibilità in caso di errore del nodo. Per ulteriori informazioni sul dimensionamento del cluster DAX, consulta Guida alle dimensioni del cluster DAX.

Nelle sezioni seguenti vengono descritti i diversi aspetti di dimensionamento da prendere in considerazione per bilanciare il tipo di nodo e il numero di nodi necessari per la creazione di un cluster scalabile e conveniente.

In questa sezione

- Disponibilità di pianificazione
- Pianificazione del throughput di scrittura
- Pianificazione del throughput di lettura
- Pianificazione delle dimensioni del set di dati
- Calcolo dei requisiti approssimativi di capacità del cluster
- Stima della capacità di throughput del cluster per tipo di nodo
- Dimensionamento della capacità di scrittura nei cluster DAX

Disponibilità di pianificazione

Quando si dimensiona un cluster DAX, è necessario innanzitutto concentrarsi sulla sua disponibilità mirata. La disponibilità di un servizio in cluster, ad esempio DAX, è una dimensione del numero

totale di nodi del cluster. Poiché un cluster a nodo singolo non ha alcuna tolleranza agli errori, la sua disponibilità è pari a un nodo. In un cluster a 10 nodi, la perdita di un singolo nodo ha un impatto minimo sulla capacità complessiva del cluster. Questa perdita non ha un impatto diretto sulla disponibilità perché i nodi rimanenti possono ancora soddisfare le richieste di lettura. Per riprendere le scritture, DAX nomina rapidamente un nuovo nodo primario.

DAX è basato su VPC: utilizza un gruppo di sottoreti per determinare in quali zone di disponibilità può eseguire i nodi e quali indirizzi IP utilizzare dalle sottoreti. Per i carichi di lavoro di produzione, si consiglia vivamente di utilizzare DAX con almeno tre nodi in diverse zone di disponibilità. Ciò garantisce che al cluster rimanga più di un nodo per gestire le richieste anche in caso di errore di un singolo nodo o di una zona di disponibilità. Un cluster può avere fino a 11 nodi, di cui uno è un nodo primario e 10 sono repliche di lettura.

Pianificazione del throughput di scrittura

Tutti i cluster DAX dispongono di un nodo primario per le richieste di write-through. La dimensione del tipo di nodo per il cluster determina la sua capacità di scrittura. L'aggiunta di repliche di lettura aggiuntive non aumenta la capacità di scrittura del cluster. Pertanto, è necessario considerare la capacità di scrittura durante la creazione del cluster perché non è possibile modificare il tipo di nodo in un secondo momento.

Se l'applicazione deve eseguire una scrittura di tipo write-though tramite DAX per aggiornare la cache degli elementi, prendi in considerazione un maggiore utilizzo delle risorse del cluster per facilitare la scrittura. Le scritture su DAX consumano circa 25 volte più risorse rispetto alle letture con riscontro nella cache. Ciò potrebbe richiedere un tipo di nodo più grande rispetto ai cluster di sola lettura.

Per ulteriori indicazioni su come determinare se la scrittura di tipo write-thorugh o quella di tipo write-around sia la soluzione migliore per l'applicazione in uso, consulta Strategie di scrittura.

Pianificazione del throughput di lettura

La capacità di lettura di un cluster DAX dipende dal rapporto di riscontri nella cache del carico di lavoro. Poiché DAX legge i dati da DynamoDB quando si verifica un mancato riscontro nella cache, consuma circa 10 volte più risorse del cluster rispetto a un riscontro nella cache. Per aumentare i riscontri nella cache, aumenta l'impostazione del TTL della cache per definire il periodo per il quale un elemento viene archiviato nella cache. Una durata del TTL più elevata, tuttavia, aumenta la possibilità di leggere versioni precedenti degli elementi, a meno che gli aggiornamenti non vengano effettuati tramite write-through con DAX.

Per assicurarti che il cluster abbia una capacità di lettura sufficiente, dimensiona il cluster orizzontalmente come indicato in Dimensionamento orizzontale di un cluster. L'aggiunta di altri nodi aggiunge repliche di lettura al pool di risorse, mentre la rimozione dei nodi riduce la capacità di lettura. Nella selezione del numero di nodi e delle relative dimensioni per un cluster, considera sia la quantità minima che quella massima di capacità di lettura necessaria. Se non riesci a dimensionare orizzontalmente un cluster con tipi di nodi più piccoli per soddisfare i requisiti di lettura, utilizza un tipo di nodo più grande.

Pianificazione delle dimensioni del set di dati

Ogni tipo di nodo disponibile ha una dimensione di memoria impostata per consentire a DAX di archiviare i dati nella cache. Se un tipo di nodo è troppo piccolo, il set di dati di lavoro richiesto da un'applicazione non entrerà in memoria e si verificheranno mancati riscontri nella cache. Poiché i nodi più grandi supportano cache più grandi, utilizza un tipo di nodo più grande del set di dati stimato da inserire nella cache. Una cache più grande migliora anche il tasso di riscontri nella cache.

Potrebbe essere possibile ottenere rendimenti decrescenti per il caching di elementi con poche letture ripetute. Calcola la dimensione della memoria per gli elementi con accesso frequente e assicurati che la cache sia sufficientemente grande per l'archiviazione di quel set di dati.

Calcolo dei requisiti approssimativi di capacità del cluster

È possibile stimare le esigenze di capacità totale di un carico di lavoro per aiutare a selezionare la dimensione e la quantità appropriate di nodi del cluster. Per eseguire questa stima, calcola la variabile richiesta al secondo normalizzata (RPS normalizzata). Questa variabile rappresenta le unità di lavoro totali che l'applicazione richiede al cluster DAX per supportare, inclusi riscontri nella cache, mancati riscontri nella cache e scritture. Per calcolare l'RPS normalizzata, utilizza i seguenti input:

- `ReadRPS_CacheHit`: specifica il numero di letture al secondo che generano un riscontro nella cache.
- `ReadRPS_CacheMiss`: specifica il numero di letture al secondo che generano un mancato riscontro nella cache.
- `WriteRPS`: specifica il numero di scritture al secondo che passano attraverso DAX.
- `DaxNodeCount`: specifica il numero di nodi nel cluster DAX.
- `Size`: specifica la dimensione dell'elemento da scrivere o leggere in KB, arrotondata per eccesso al KB più vicino.
- `10x_ReadMissFactor`: rappresenta un valore di 10. Quando si verifica un mancato riscontro nella cache, DAX utilizza circa 10 volte più risorse rispetto ai riscontri nella cache.

- `25x_WriteFactor`: rappresenta un valore pari a 25 perché una procedura di write-through DAX utilizza circa 25 volte più risorse rispetto ai riscontri nella cache.

Con la seguente formula è possibile calcolare l'RPS normalizzata.

```
Normalized RPS = (ReadRPS_CacheHit * Size) + (ReadRPS_CacheMiss * Size *
 10x_ReadMissFactor) + (WriteRequestRate * 25x_WriteFactor * Size * DaxNodeCount)
```

Si considerino, ad esempio, i seguenti valori di input:

- `ReadRPS_CacheHit` = 50.000
- `ReadRPS_CacheMiss` = 1.000
- `ReadMissFactor` = 1
- `Size` = 2 KB
- `WriteRPS` = 100
- `WriteFactor` = 1
- `DaxNodeCount` = 3

Sostituendo questi valori nella formula, è possibile calcolare l'RPS normalizzata come segue.

```
Normalized RPS = (50,000 Cache Hits/Sec * 2KB) + (1,000 Cache Misses/Sec * 2KB * 10) +
 (100 Writes/Sec * 25 * 2KB * 3)
```

In questo esempio, il valore calcolato dell'RPS normalizzata è 135.000. Tuttavia, questo valore dell'RPS normalizzata non tiene conto del mantenimento dell'utilizzo del cluster al di sotto del 100% o della perdita di nodi. È consigliabile considerare la capacità aggiuntiva. A tale scopo, determina il maggiore dei due fattori di moltiplicazione: l'utilizzo di destinazione o la tolleranza alla perdita dei nodi. Quindi, moltiplica l'RPS normalizzata per il fattore maggiore per ottenere una richiesta di destinazione al secondo (RPS di destinazione).

- Utilizzo dell'obiettivo

  Poiché gli impatti sulle prestazioni aumentano i mancati riscontri nella cache, non è consigliabile eseguire il cluster DAX al 100% di utilizzo. Idealmente, si dovrebbe mantenere l'utilizzo del cluster pari o inferiore al 70%. A tale scopo, moltiplica l'RPS normalizzata per 1,43.
- Tolleranza alla perdita dei nodi

Se un nodo va in errore, l'applicazione deve essere in grado di distribuire le richieste tra i nodi rimanenti. Per assicurarti che i nodi rimangano al di sotto del 100% di utilizzo, seleziona un tipo di nodo sufficientemente grande da assorbire traffico aggiuntivo fino a quando il nodo con l'errore non torna online. Per un cluster con meno nodi, ogni nodo deve tollerare aumenti di traffico maggiori in caso di errore di un nodo.

Nel caso di un errore del nodo primario, DAX esegue automaticamente il failover su un nodo di replica di lettura e lo designa come nuovo nodo primario. Se un nodo di replica va in errore, altri nodi nel cluster DAX possono ancora essere in grado di assolvere le richieste finché il nodo con l'errore non viene ripristinato.

Ad esempio, un cluster DAX a 3 nodi con un errore in un nodo richiede una capacità aggiuntiva del 50% sui due nodi rimanenti. Ciò richiede un fattore di moltiplicazione pari a 1,5. Al contrario, un cluster a 11 nodi con un errore in un nodo richiede una capacità aggiuntiva del 10% sui nodi rimanenti o un fattore di moltiplicazione di 1,1.

Con la seguente formula è possibile calcolare l'RPS di destinazione.

```
Target RPS = Normalized RPS * CEILING(TargetUtilization, NodeLossTolerance)
```

Ad esempio, per calcolare l'RPS di destinazione, si considerino i seguenti valori:

- `Normalized RPS` = 135.000
- `TargetUtilization` = 1,43

  Con un obiettivo di un utilizzo massimo del cluster del 70%, si imposta `TargetUtilization` su 1,43.
- `NodeLossTolerance` = 1,5

  Si supponga di utilizzare un cluster a 3 nodi, pertanto si imposta `NodeLossTolerance` su una capacità del 50%.

Sostituendo questi valori nella formula, è possibile calcolare l'RPS di destinazione come segue.

```
Target RPS = 135,000 * CEILING(1.43, 1.5)
```

In questo esempio, poiché il valore di `NodeLossTolerance` è maggiore di `TargetUtilization`, il valore di RPS di destinazione viene calcolato con `NodeLossTolerance`. Questo restituisce una RPS di destinazione di 202.500, che è la quantità totale di capacità che il cluster DAX deve supportare. Per determinare il numero di nodi necessari in un cluster, mappa l'RPS di destinazione su una colonna appropriata nella tabella seguente. Per questo esempio di RPS di destinazione di 202.500, è necessario il tipo di nodo dax.r5.large con tre nodi.

Stima della capacità di throughput del cluster per tipo di nodo

Utilizzando Target RPS formula è possibile stimare la capacità del cluster per diversi tipi di nodi. La tabella seguente mostra le capacità approssimative per i cluster con 1, 3, 5 e 11 tipi di nodi. Queste capacità non sostituiscono la necessità di eseguire test di carico di DAX con modelli di dati e richieste personalizzati. Inoltre, queste capacità non includono le istanze di tipo t a causa della mancanza di capacità della CPU fissa. L'unità per tutti i valori nella tabella seguente è RPS normalizzata.

Tipo di nodo (memoria)	1 nodo	3 nodi	5 nodi	11 nodi
dax.r5.24xlarge (768 GB)	1 milione	3 milioni	5 milioni	11 milioni
dax.r5.16xlarge (512 GB)	1 milione	3 milioni	5 milioni	11 milioni
dax.r5.12xlarge (384 GB)	1 milione	3 milioni	5 milioni	11 milioni
dax.r5.8xlarge (256 GB)	1 milione	3 milioni	5 milioni	11 milioni
dax.r5.4xlarge (128 GB)	600.000	1,8 milioni	3 milioni	6,6 milioni
dax.r5.2xlarge (64 GB)	300.000	900.000	1,5 milioni	3,3 milioni
dax.r5.xlarge (32 GB)	150.000	450.000	750.000	1,65 milioni

Tipo di nodo (memoria)	1 nodo	3 nodi	5 nodi	11 nodi
dax.r5.large (16 GB)	75.000	225.000	375.000	825.000

A causa del limite massimo di 1 milione di NPS (operazioni di rete al secondo) per ogni nodo, i nodi di tipo dax.r5.8xlarge o superiore non forniscono capacità aggiuntiva al cluster. I tipi di nodi più grandi di 8xlarge potrebbero non contribuire alla capacità di throughput totale del cluster. Tuttavia, questi tipi di nodi possono essere utili per archiviare in memoria un set di dati di lavoro più ampio.

Dimensionamento della capacità di scrittura nei cluster DAX

Ogni scrittura su DAX consuma 25 richieste normalizzate su ogni nodo. Poiché esiste un limite di 1 milione di RPS per ogni nodo, un cluster DAX è limitato a 40.000 scritture al secondo, senza contare l'utilizzo di lettura.

Se il caso d'uso richiede più di 40.000 scritture al secondo nella cache, è necessario utilizzare cluster DAX separati e sottoporre a sharding le scritture tra di essi. Analogamente a DynamoDB, è possibile eseguire l'hash della chiave di partizione per i dati che si stanno scrivendo nella cache. Quindi, viene utilizzato il modulo per determinare su quale shard scrivere i dati.

L'esempio seguente calcola l'hash di una stringa di input, quindi calcola il modulo del valore 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}.")
```

Implementazione di un cluster

La creazione di un nuovo cluster DAX richiede configurazioni che vanno oltre quelle necessarie per DynamoDB. Queste configurazioni sono destinate in particolare al networking perché DAX è basato su Amazon VPC. Questo offre il controllo completo sull'ambiente di rete virtuale, compresi il posizionamento delle risorse, la connettività e la sicurezza. Questa sezione presenta le best practice per le impostazioni necessarie per la creazione del cluster.

Per informazioni sulla scelta dei nodi dei cluster, consulta Dimensionamento del cluster DAX.

In questa sezione

- Configurazione di reti
- Configurazione della sicurezza
- Gruppo di parametri
- Finestra di manutenzione

Configurazione di reti

DAX utilizza un gruppo di sottoreti per determinare in quali zone di disponibilità può eseguire i nodi e quali indirizzi IP utilizzare dalle sottoreti. Per ridurre al minimo la latenza tra l'applicazione e DAX, le sottoreti e le zone di disponibilità per i server delle applicazioni e il cluster DAX devono essere le stesse.

Si consiglia di distribuire i nodi DAX in più zone di disponibilità. L'opzione predefinita di Allocazione automatica esegue questa operazione automaticamente.

Per le best practice sulla configurazione del VPC, consulta la sezione Nozioni di base su Amazon VPC nella Guida per l'utente di Amazon VPC.

Configurazione della sicurezza

Questa sezione descrive le misure di sicurezza da implementare per le applicazioni che utilizzano DAX. Questa sezione descrive inoltre brevemente il supporto che DAX include per la crittografia dei dati.

IAM

DAX e DynamoDB hanno meccanismi di controllo degli accessi separati. DAX richiede un ruolo IAM per accedere alle tabelle DynamoDB. Questo ruolo dovrebbe seguire il principio del privilegio minimo

e concedere l'accesso solo a tabelle e operazioni DynamoDB specifiche, come e. GetItemPutItem Per ulteriori informazioni sui meccanismi di controllo dell'accesso forniti da DAX, consulta Controllo degli accessi DAX.

Encryption (Crittografia)

Durante la creazione di un cluster DAX viene configura la crittografia dei dati a riposo e la crittografia dei dati in transito. Queste sono attivate per impostazione predefinita. Si consiglia di mantenere le impostazioni di crittografia predefinite a meno che i requisiti aziendali non lo impediscano. Per ulteriori informazioni, consultare Crittografia DAX dei dati inattivi e Crittografia DAX in transito.

Gruppo di parametri

DAX applica un set di configurazioni su ogni nodo di un cluster denominato gruppo di parametri. È possibile modificare questa configurazione dopo aver creato il cluster.

Il gruppo di parametri DAX contiene le impostazioni del TTL per la cache degli elementi e la cache delle query. Per impostazione predefinita, la durata del TTL è 5 minuti. È possibile sostituire la durata del TTL con qualsiasi valore intero maggiore o uguale a 1 millisecondo.

Non è possibile modificare i gruppi di parametri quando vengono utilizzati da un'istanza DAX in esecuzione. È possibile modificare i valori del gruppo di parametri durante il tempo di inattività di un cluster DAX.

Finestra di manutenzione

Per consentire aggiornamenti software e patch occasionali ai nodi, viene configurata una finestra di manutenzione settimanale per il cluster DAX. Durante questa finestra, DAX esegue aggiornamenti continui ai nodi. I cluster con più di un nodo non perdono la disponibilità del cluster durante questi aggiornamenti, ma hanno una capacità ridotta del cluster fino al ripristino del nodo. Se la tua organizzazione ha un periodo prevedibile di basso utilizzo, valuta la possibilità di impostare manualmente la finestra di manutenzione su questo periodo.

Gestione delle operazioni sui cluster

DAX si occupa della gestione della manutenzione e dell'integrità del cluster. Tuttavia, è necessario fornire un input operativo per dimensionare il cluster orizzontalmente o verticalmente in base ai modelli di utilizzo. Questa sezione descrive la procedura consigliata per scalare i cluster DAX.

In questa sezione

- Dimensionamento orizzontale di un cluster
- Dimensionamento verticale di un cluster

Dimensionamento orizzontale di un cluster

Il dimensionamento di un cluster DAX implica la regolazione della sua capacità per soddisfare le esigenze di throughput. Questa regolazione viene effettuata aumentando o diminuendo il numero di nodi (repliche) nel cluster durante l'esecuzione. Questo processo, noto come dimensionamento orizzontale, aiuta a distribuire il carico di lavoro su più nodi o a combinarlo su un numero inferiore di nodi quando la domanda è bassa.

È possibile dimensionare orizzontalmente il cluster DAX, riducendolo o aumentandolo, attraverso i comandi `decrease-replication-factor` o `increase-replication-factor` nella AWS CLI.

Aumento del fattore di replica (aumento orizzontale)

L'aumento del fattore di replica di un cluster DAX aggiunge più nodi al cluster. L'esempio seguente mostra l'utilizzo del comando `increase-replication-factor`.

```
aws dax increase-replication-factor \
    --cluster-name yourClusterName  \
    --new-replication-factor desiredReplicationFactor
```

- In questo comando, l'argomento `cluster-name` specifica il nome del cluster. Ad esempio, *`yourClusterName`*.
- L'argomento `new-replication-factor` specifica il numero totale di nodi da aggiungere nel cluster dopo il dimensionamento. Ciò include il nodo primario e i nodi di replica. Ad esempio, se il cluster ha attualmente 3 nodi e si desidera aggiungere altri 2 nodi, imposta il valore di `new-replication-factor` su 5.

Riduzione del fattore di replica (riduzione orizzontale)

La riduzione del fattore di replica di un cluster DAX rimuove i nodi dal cluster. La rimozione dei nodi può aiutare a ridurre i costi nei periodi di bassa domanda. L'esempio seguente mostra l'utilizzo del comando `decrease-replication-factor`.

```
aws dax decrease-replication-factor \
    --cluster-name yourClusterName  \
```

```
    --new-replication-factor desiredReplicationFactor
```

- In questo comando, l'argomento `cluster-name` specifica il nome del cluster. Ad esempio, *yourClusterName*.
- L'argomento `new-replication-factor` specifica il numero ridotto di nodi del cluster dopo il dimensionamento. Questo numero deve essere inferiore al fattore di replica corrente e deve includere il nodo primario. Ad esempio, se il cluster ha 5 nodi e si desidera rimuovere 2 nodi, imposta il valore di `new-replication-factor` su 3.

Considerazioni sul dimensionamento orizzontale

Durante la pianificazione del dimensionamento orizzontale, prendi in considerazione quanto segue:

- Nodo primario: il cluster DAX include un nodo primario. Il fattore di replica include questo nodo primario. Ad esempio, un fattore di replica pari a 3 indica un nodo primario e due nodi di replica.
- Disponibilità: l'aggiunta o la rimozione di nodi DAX modifica la disponibilità e la tolleranza ai guasti del cluster. Un numero maggiore di nodi può migliorare la disponibilità, ma anche aumentare i costi.
- Migrazione dei dati: quando si aumenta il fattore di replica, DAX gestisce automaticamente la distribuzione dei dati nel nuovo set di nodi. Quando un nuovo nodo inizia a servire il traffico, la sua cache è già riscaldata. Tuttavia, durante questo processo, potrebbe verificarsi un impatto temporaneo sulle prestazioni durante la migrazione dei dati.

Monitora attentamente i cluster DAX durante e dopo il processo di dimensionamento per assicurarti che funzionino come previsto e, se necessario, apporta ulteriori modifiche.

Dimensionamento verticale di un cluster

Per dimensionare verticalmente le dimensioni dei nodi di un cluster esistente, è necessario creare un nuovo cluster e migrare il traffico dell'applicazione verso il nuovo cluster. La migrazione a un nuovo cluster con nodi diversi prevede numerose fasi per garantire una transizione fluida con un impatto minimo sulle prestazioni e sulla disponibilità dell'applicazione.

Per creare un nuovo cluster per scalare verticalmente le dimensioni dei nodi, prendi in considerazione i seguenti punti:

- Accedi alla configurazione attuale: esamina le metriche dell'attuale cluster DAX per determinare la dimensione e la quantità dei nuovi nodi necessari. Utilizza queste informazioni come input per definire le dimensioni del cluster. Per informazioni, consulta Dimensionamento del cluster DAX.

- Configura un nuovo cluster DAX: crea un nuovo cluster DAX con il tipo di nodo e la quantità determinati. È possibile utilizzare le impostazioni di configurazione esistenti del gruppo di parametri, a meno che non sia necessario apportare modifiche.
- Sincronizza i dati: poiché DAX è un livello di caching per DynamoDB, non è necessario migrare direttamente i dati. Tuttavia, il nuovo cluster DAX non avrà alcun set di dati di lavoro in memoria finché non gli verrà inviato traffico.
- Aggiorna la configurazione dell'applicazione: aggiorna la configurazione dell'applicazione in modo che punti all'endpoint del nuovo cluster DAX. Potrebbe essere necessario modificare il codice o aggiornare le variabili dell'ambiente, a seconda della configurazione dell'applicazione.

  Per ridurre l'impatto durante il passaggio a un nuovo cluster, invia traffico canary al nuovo cluster da una piccola parte del parco applicazioni. È possibile farlo implementando lentamente gli aggiornamenti delle applicazioni o utilizzando una voce DNS di instradamento basata sul peso davanti all'endpoint DAX.
- Monitora e ottimizza: dopo il passaggio al nuovo cluster DAX, monitora attentamente le metriche e i log delle prestazioni per individuare eventuali problemi. Preparati a modificare il numero di nodi in base a modelli di carico di lavoro aggiornati.

  Fino a quando il nuovo cluster non archivierà correttamente nella cache il set di dati di lavoro, vedrai mancati riscontri nella cache e latenze più elevati.
- Disattiva il vecchio cluster: una volta che avrai la certezza che il nuovo cluster funziona come previsto, rimuovi in sicurezza il vecchio cluster DAX per evitare costi inutili.

Monitoraggio di DAX

È possibile monitorare le metriche chiave, ad esempio il rapporto di riscontri nella cache, per garantire prestazioni ottimali del cluster DAX, diagnosticare i problemi e determinare quando è necessario scalare il cluster. Il controllo regolare delle metriche chiave aiuta a mantenere prestazioni, stabilità ed efficienza in termini di costi scalando il cluster in base ai requisiti del carico di lavoro. Per ulteriori informazioni sul monitoraggio di DAX, consulta Monitoraggio della produzione.

L'elenco seguente presenta alcune delle metriche chiave da monitorare:

- Rapporti di riscontri nella cache: mostra l'efficacia con cui DAX serve i dati archiviati nella cache, riducendo la necessità di accedere alle tabelle DynamoDB sottostanti. Un numero basso di mancati riscontri nella cache per il cluster indica una buona efficienza di caching. Tuttavia, un

numero basso di riscontri nella cache suggerisce che potrebbe essere necessario riesaminare l'impostazione del TTL di caching, oppure che il carico di lavoro non è adatto al caching.

Usa Amazon CloudWatch per calcolare il rapporto di accessi alla cache del tuo cluster DAX. Confronta le metriche `ItemCacheHits`, `ItemCacheMisses`, `QueryCacheHits` e `QueryCacheMisses` e per ottenere questo rapporto. La formula seguente mostra come viene calcolato il rapporto di riscontri nella cache. Per calcolare il rapporto utilizzando questa formula, dividi i riscontri nella cache per la somma dei riscontri nella cache e dei mancati riscontri.

```
Cache hit ratio = Cache hits / (Cache hits + Cache misses)
```

Il rapporto di riscontri nella cache è un numero compreso tra 0 e 1, rappresentato in percentuale. Una percentuale più alta indica un migliore utilizzo complessivo della cache.

- ErrorRequestCount— Numero di richieste che hanno provocato errori per gli utenti segnalate dal nodo o dal cluster. `ErrorRequestCount`include le richieste che sono state limitate dal nodo o dal cluster. Il monitoraggio degli errori degli utenti può aiutarvi a identificare le configurazioni errate di scalabilità o gli hot item/partition pattern presenti nell'applicazione.
- Latenze delle operazioni: il monitoraggio della latenza delle operazioni di lettura e scrittura da e verso il cluster DAX può aiutare a identificare i punti deboli delle prestazioni. L'aumento delle latenze potrebbe indicare problemi relativi alla configurazione del cluster DAX, alla rete o la necessità di scalare.
- Utilizzo di rete: monitora le metriche `NetworkBytesIn` e `NetworkBytesOut` e per controllare il traffico di rete del cluster DAX. Un aumento imprevisto del throughput di rete potrebbe significare un maggior numero di richieste da parte dei client o modelli di query inefficienti che causano il trasferimento di una maggiore quantità di dati.

  Il monitoraggio dell'utilizzo di rete consente di gestire i costi del cluster DAX e garantisce che la rete non diventi un ostacolo alle prestazioni del cluster.
- Tasso di espulsione: mostra la frequenza con cui gli elementi vengono rimossi dalla cache per fare spazio a nuovi elementi. Se il tasso di espulsione aumenta nel tempo, è possibile che la cache sia troppo piccola o che la strategia di caching non sia efficace.

  Monitora la `EvictedSize` metrica CloudWatch per determinare se la dimensione della cache è adeguata al carico di lavoro. Se la dimensione totale degli elementi espulsi continua a crescere, potrebbe essere necessario aumentare verticalmente il cluster DAX per ospitare una cache più grande.

- Utilizzo della CPU: si riferisce alla percentuale di utilizzo della CPU del nodo o del cluster. Si tratta di una metrica di fondamentale monitoraggio per qualsiasi database o sistema di caching. L'elevato utilizzo della CPU può comportare un sovraccarico del cluster DAX e la necessità di dimensionamento per gestire l'aumento della domanda.

  Monitora la metrica `CPUUtilization` per un cluster DAX. Se l'utilizzo della CPU si avvicina o supera costantemente il 70 – 80%, è consigliabile aumentare verticalmente il cluster DAX come descritto nella sezione seguente.

  Se il numero di richieste inviate a DAX supera la capacità di un nodo, DAX limita il tasso con cui accetta le richieste aggiuntive Lo fa restituendo un. ThrottlingException DAX valuta continuamente l'utilizzo della CPU da parte del cluster per determinare il volume di richieste che può elaborare mantenendo allo stesso tempo uno stato del cluster integro.

  È possibile monitorare la `ThrottledRequestCount` metrica su cui DAX pubblica. CloudWatch Se queste eccezioni vengono visualizzate regolarmente, è consigliabile prendere in considerazione la possibilità di aumentare la dimensione del cluster.

Dimensionamento del cluster DAX attraverso i dati di monitoraggio

È possibile determinare se è necessario aumentare o diminuire verticalmente il cluster DAX monitorandone le metriche delle prestazioni.

- Aumento verticale o orizzontale: se il cluster DAX presenta un elevato utilizzo della CPU, pochi riscontri nella cache (dopo l'ottimizzazione della strategia di caching) o latenze delle operazioni elevate, è necessario aumentare il cluster verticalmente. L'aggiunta di più nodi, detta anche aumento orizzontale, può aiutare a distribuire il carico in modo più uniforme. Per carichi di lavoro con un aumento delle scritture al secondo, potrebbe essere necessario scegliere nodi più potenti (aumento verticale).
- Riduzione verticale: se si riscontrano costantemente bassi livelli di utilizzo della CPU e latenze delle operazioni inferiori alle soglie stabilite, è possibile che le risorse siano sovradimensionate. In questi casi, è opportuno ridurre verticalmente i nodi per abbassare i costi. È possibile ridurre il numero di nodi a 1 durante i periodi di utilizzo ridotto, ma non è possibile chiudere completamente il cluster.

Utilizzo di DynamoDB con altri servizi AWS

Amazon DynamoDB è integrato con AWS altri servizi e consente di automatizzare attività ripetute o creare applicazioni che si estendono su più servizi.

Argomenti

- Configurazione delle AWS credenziali con Amazon Cognito per DynamoDB
- Integrazione con Amazon Redshift
- Elaborazione dei dati di DynamoDB con Apache Hive su Amazon EMR
- Integrazione di DynamoDB con Amazon S3
- Integrazione Zero-ETL di DynamoDB con Amazon SageMaker Lakehouse
- Integrazione Zero-ETL di Amazon DynamoDB con il Servizio OpenSearch di Amazon
- Integrazione di DynamoDB con Amazon EventBridge
- Integrazione di DynamoDB con Streaming gestito da Amazon per Apache Kafka
- Best practice per l'integrazione con DynamoDB

Configurazione delle AWS credenziali con Amazon Cognito per DynamoDB

Il modo consigliato per ottenere AWS le credenziali per le tue applicazioni web e mobili è usare Amazon Cognito. Amazon Cognito ti aiuta a evitare di inserire le tue AWS credenziali nei tuoi file. Utilizza i ruoli AWS Identity and Access Management (IAM) per generare credenziali temporanee per gli utenti autenticati e non autenticati dell'applicazione.

Ad esempio, per configurare JavaScript i tuoi file per utilizzare un ruolo non autenticato di Amazon Cognito per accedere al servizio web Amazon DynamoDB, procedi come segue.

Come configurare le credenziali da integrare con Amazon Cognito

1. Crea un pool di identità di Amazon Cognito che supporti le identità non autenticate.

```
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. Copia la policy seguente in un file denominato `myCognitoPolicy.json`. Sostituisci l'ID del pool di identità (*`us-west-2:12345678-1ab2-123a-1234-a12345ab12`*) con il tuo `IdentityPoolId` ottenuto nel passaggio precedente.

   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. Crea un ruolo IAM che assuma la policy precedente. In questo modo Amazon Cognito diviene un'entità affidabile che può assumere il ruolo `Cognito_DynamoPoolUnauth`.

   ```
   aws iam create-role --role-name Cognito_DynamoPoolUnauth \
   --assume-role-policy-document file://PathToFile/myCognitoPolicy.json --output json
   ```

4. Concedere al ruolo `Cognito_DynamoPoolUnauth` l'accesso completo a DynamoDB collegando una policy gestita (`AmazonDynamoDBFullAccess`).

```
aws iam attach-role-policy --policy-arn arn:aws:iam::aws:policy/
AmazonDynamoDBFullAccess \
--role-name Cognito_DynamoPoolUnauth
```

> **Note**
>
> In alternativa, è possibile concedere un accesso fine-grained a DynamoDB. Per ulteriori informazioni, consulta Utilizzo di condizioni di policy IAM per il controllo granulare degli accessi.

5. Ottenere e copiare l'Amazon Resource Name (ARN) del ruolo IAM.

```
aws iam get-role --role-name Cognito_DynamoPoolUnauth --output json
```

6. Aggiungi il ruolo `Cognito_DynamoPoolUnauth` al pool di identità `DynamoPool`. Il formato da specificare è `KeyName=string`, in cui `KeyName` è `unauthenticated` e la stringa è l'ARN del ruolo ottenuto nella fase precedente.

```
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. Specifica le credenziali di Amazon Cognito nei tuoi file. Modifica i valori di `IdentityPoolId` e `RoleArn` di conseguenza.

```
AWS.config.credentials = new AWS.CognitoIdentityCredentials({
IdentityPoolId: "us-west-2:12345678-1ab2-123a-1234-a12345ab12",
RoleArn: "arn:aws:iam::123456789012:role/Cognito_DynamoPoolUnauth"
});
```

Ora puoi eseguire i tuoi JavaScript programmi sul servizio web DynamoDB utilizzando le credenziali di Amazon Cognito. Per ulteriori informazioni, consulta Impostazione delle credenziali in un browser Web nella Guida alle operazioni di base di AWS SDK per JavaScript .

Integrazione con Amazon Redshift

Amazon Redshift è un servizio di data warehouse rapido, gestito e scalabile a livello di petabyte in grado analizzare correttamente i dati in modo semplice e conveniente utilizzando tutti gli strumenti di business intelligence già presenti.

DynamoDB e Amazon Redshift possono essere usati insieme per soddisfare diverse esigenze di archiviazione ed elaborazione dei dati all'interno di un'applicazione o di un ecosistema di dati.

Consulta gli argomenti seguenti per informazioni più dettagliate su come integrare DynamoDB con Amazon Redshift.

Argomenti

- Considerazioni sull'integrazione multi-account con CMK
- Integrazione Zero-ETL di DynamoDB con Amazon Redshift
- Caricamento di dati da DynamoDB in Amazon Redshift con il comando COPY

Considerazioni sull'integrazione multi-account con CMK

Quando si effettua l'integrazione da DynamoDB ad Amazon Redshift, l'azione iniziale viene avviata da Amazon Redshift. Senza le autorizzazioni appropriate, questa azione potrebbe causare un errore non segnalato. Le sezioni seguenti descrivono in dettaglio le autorizzazioni necessarie per l'integrazione multi-account.

Politiche e autorizzazioni richieste AWS KMS

Negli esempi, sostituisci i segnaposto seguenti:

- `111122223333`: l' Account AWS ID in cui è ospitato Amazon Redshift
- `444455556666`: L' Account AWS ID in cui è ospitato DynamoDB
- `REDSHIFT_ROLE_NAME`: il nome del ruolo IAM utilizzato da Amazon Redshift
- `REGION`: Il Regione AWS luogo in cui si trovano le tue risorse
- `TABLE_NAME`: il nome della tabella DynamoDB
- `KMS_KEY_ID`: l'ID della chiave KMS

Policy della chiave KMS nell'account DynamoDB

La seguente policy AWS KMS chiave consente l'accesso tra account diversi tra i servizi DynamoDB e Amazon Redshift. In questo esempio, l'account 444455556666 contiene la tabella e la AWS KMS chiave DynamoDB, mentre l'account 111122223333 contiene il cluster Amazon Redshift che necessita dell'accesso per decrittografare i dati.

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": "*"
        }
    ]
}
```

Policy IAM per il ruolo Amazon Redshift (in un account Amazon Redshift)

La seguente policy IAM consente a un servizio Amazon Redshift di accedere alle tabelle DynamoDB e alle relative chiavi di AWS KMS crittografia associate in uno scenario tra più account. In questo esempio, l'account 444455556666 contiene le risorse e le chiavi DynamoDB a cui AWS KMS il servizio Amazon Redshift deve accedere.

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"
```

```
        }
    ]
}
```

Relazione di attendibilità per il ruolo Amazon Redshift

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "redshift.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

Policy per le tabelle DynamoDB (in caso di utilizzo di policy basate su risorse)

La seguente policy basata su risorse consente a un servizio Amazon Redshift nell'account 111122223333 di accedere alle tabelle DynamoDB e ai flussi nell'account 444455556666. Collega questa policy alla tabella DynamoDB per abilitare l'accesso multi-account.

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/*"
            ]
        }
    ]
}
```

Considerazioni importanti

1. Assicurati che la chiave KMS si trovi nella stessa Regione della tabella DynamoDB.
2. La chiave KMS deve essere una chiave gestita dal cliente (CMK), non una. Chiave gestita da AWS
3. In caso di utilizzo di tabelle globali DynamoDB, configura le autorizzazioni per tutte le Regioni pertinenti.
4. Prendi in considerazione l'aggiunta di istruzioni condizionali per limitare l'accesso in base agli endpoint VPC o agli intervalli IP.
5. Per una maggiore sicurezza, prendi in considerazione l'utilizzo della condizione `aws:PrincipalOrgID` per limitare l'accesso all'organizzazione.
6. Monitora l'utilizzo delle chiavi KMS tramite CloudTrail e le metriche. CloudWatch

Integrazione Zero-ETL di DynamoDB con Amazon Redshift

L'integrazione Zero-ETL di Amazon DynamoDB con Amazon Redshift consente un'analisi fluida dei dati DynamoDB senza codice. Questa funzionalità completamente gestita replica automaticamente le tabelle DynamoDB in un database Amazon Redshift in modo che gli utenti possano eseguire query e analisi SQL sui propri dati DynamoDB senza dover configurare complessi processi ETL. L'integrazione funziona replicando i dati dalla tabella DynamoDB al database Amazon Redshift.

Per configurare l'integrazione, è sufficiente specificare una tabella DynamoDB come origine e un database Amazon Redshift come destinazione. All'attivazione, l'integrazione esporta la tabella DynamoDB completa per popolare il database Amazon Redshift. Il tempo richiesto per il completamento di questo processo iniziale dipende dalle dimensioni della tabella DynamoDB. L'integrazione Zero-ETL replica quindi in modo incrementale gli aggiornamenti da DynamoDB ad Amazon Redshift ogni 15 – 30 minuti utilizzando le esportazioni incrementali di DynamoDB. Ciò significa che i dati DynamoDB replicati in Amazon Redshift vengono conservati automaticamente. up-to-date

Una volta terminata la configurazione, gli utenti possono analizzare i dati DynamoDB in Amazon Redshift tramite client e strumenti SQL standard, senza influire sulle prestazioni delle tabelle DynamoDB. Eliminando il complesso processo ETL, questa integrazione Zero-ETL offre un modo semplice e veloce per ottenere approfondimenti da DynamoDB attraverso le funzionalità di analisi e machine learning di Amazon Redshift.

Argomenti

- Prerequisiti per creare un'integrazione Zero-ETL di DynamoDB con Amazon Redshift
- Limitazioni dell'utilizzo delle integrazioni Zero-ETL di DynamoDB con Amazon Redshift
- Creazione di un'integrazione Zero-ETL di DynamoDB con Amazon Redshift
- Visualizzazione delle integrazioni Zero-ETL di DynamoDB con Amazon Redshift
- Eliminazione di integrazioni Zero-ETL di DynamoDB con Amazon Redshift

Prerequisiti per creare un'integrazione Zero-ETL di DynamoDB con Amazon Redshift

1. Prima di procedere all'integrazione è necessario creare la tabella DynamoDB di origine e il cluster Amazon Redshift di destinazione. Queste informazioni sono trattate in Fase 1: configurare una tabella DynamoDB di origine e Fase 2: creare un data warehouse Amazon Redshift.
2. Un'integrazione zero-ETL tra Amazon DynamoDB e Amazon Redshift richiede che la tabella DynamoDB di origine abbia il ripristino (PITR) abilitato. Point-in-time
3. Per le policy basate su risorse, se l'integrazione viene creata in un contesto in cui la tabella DynamoDB e il data warehouse Amazon Redshift si trovano nello stesso account, durante la fase di creazione dell'integrazione è possibile utilizzare l'opzione Correggi per me per applicare automaticamente sia a DynamoDB che ad Amazon Redshift le policy basate su risorse richieste.

Se crei un'integrazione in cui la tabella DynamoDB e il data warehouse Amazon Redshift si trovano in account AWS diversi, dovrai applicare la seguente politica delle risorse sulla tabella 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:*"
                }
            }
        }
    ]
}
```

Potrebbe anche essere necessario configurare la policy basata su risorse sul proprio data warehouse Amazon Redshift. Per ulteriori informazioni, consulta Configurazione dell'autorizzazione usando l'API Amazon Redshift.

4. Per le policy IAM basate su identità:

   a. L'utente che crea l'integrazione necessita di una policy basata su identità che autorizzi le seguenti azioni: `GetResourcePolicy`, `PutResourcePolicy` e `UpdateContinuousBackups`.

   > **Note**
   >
   > I seguenti esempi di policy mostreranno la risorsa come `arn:aws:redshift{-serverless}`. Questo esempio dimostra che l'ARN può essere `arn:aws:redshift` o `arn:aws:redshift-serverless` a seconda del fatto che il namespace sia un cluster Amazon Redshift o un namespace Amazon Redshift serverless.

   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. L'utente responsabile della configurazione del namespace Amazon Redshift di destinazione necessita di una policy basata su identità che autorizzi le seguenti azioni: `PutResourcePolicy`, `DeleteResourcePolicy` e `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. Autorizzazioni della chiave di crittografia

Se la tabella DynamoDB di origine è crittografata tramite una chiave AWS KMS gestita dal cliente, sarà necessario aggiungere la seguente policy alla propria chiave KMS. Questa policy consente ad Amazon Redshift di poter esportare dati dalla tabella crittografata attraverso la chiave 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:*"
        }
    }
}
```

È anche possibile seguire la procedura riportata nella Getting started with zero-ETL integrations nella Guida alla gestione di Amazon Redshift per configurare le autorizzazioni del namespace Amazon Redshift.

Limitazioni dell'utilizzo delle integrazioni Zero-ETL di DynamoDB con Amazon Redshift

Le seguenti limitazioni generali si applicano alla versione corrente di questa integrazione e potrebbero cambiare nelle versioni successive.

> **Note**
>
> Oltre alle limitazioni riportate di seguito, consulta anche le considerazioni generali sull'utilizzo delle integrazioni Zero-ETL (vedi Considerations when using zero-ETL integrations with Amazon Redshift nella Guida alla gestione di Amazon Redshift).

- La tabella DynamoDB e il cluster Amazon Redshift devono trovarsi nella stessa Regione.

- La tabella DynamoDB di origine deve essere crittografata con una chiave di proprietà di Amazon o gestita dal cliente. AWS KMS La crittografia gestita da Amazon non è supportata per la tabella DynamoDB di origine.

Creazione di un'integrazione Zero-ETL di DynamoDB con Amazon Redshift

Prima di creare un'integrazione Zero-ETL è necessario configurare la tabella DynamoDB di origine e quindi il data warehouse Amazon Redshift di destinazione.

Fase 1: configurare una tabella DynamoDB di origine

Per creare un'integrazione zero-ETL con Amazon Redshift, devi abilitare il point-in-time ripristino (PITR) sulla tua tabella. Se il PITR non è attivato, la console può occuparsi di risolvere il problema durante il processo di configurazione dell'integrazione. Per i dettagli su come abilitare PITR, consulta recovery. Point-in-time

Fase 2: creare un data warehouse Amazon Redshift

Se non si dispone già di un data warehouse Amazon Redshift, è possibile crearne uno. Per creare un gruppo di lavoro Amazon Redshift serverless, consulta Creazione di un gruppo di lavoro con uno spazio dei nomi. Per creare un cluster Amazon Redshift, consulta Creating a cluster.

Affinché l'integrazione venga eseguita correttamente, il gruppo di lavoro o cluster Amazon Redshift di destinazione deve avere il parametro enable_case_sensitive_identifier attivato. Per ulteriori informazioni sull'attivazione della distinzione tra maiuscole e minuscole, consulta Turn on case sensitivity for your data warehouse nella Guida alla gestione di Amazon Redshift.

Una volta completata la configurazione del gruppo di lavoro o del cluster Amazon Redshift è necessario configurare il data warehouse. Per ulteriori informazioni, consulta Integrazioni Zero-ETL nella Guida alla gestione di Amazon Redshift.

Fase 3: creare un'integrazione Zero-ETL di DynamoDB

Prima di creare un'integrazione Zero-ETL, assicurati di completare le attività indicate nella sezione intitolata Prerequisiti per creare un'integrazione Zero-ETL di DynamoDB con Amazon Redshift.
La creazione di un'integrazione tra DynamoDB e Amazon Redshift è una procedura in due fasi. Innanzitutto crea un'integrazione da DynamoDB, quindi collega un database Amazon Redshift all'integrazione appena creata.

Creazione di un'integrazione Zero-ETL

1. Accedi alla console di AWS gestione e apri la console Amazon DynamoDB all'indirizzo. https://console.aws.amazon.com/dynamodbv2
2. Nel riquadro di navigazione, scegli Integrazioni.
3. Seleziona Crea un'integrazione Zero-ETL e seleziona Amazon Redshift.
4. Questo ti reindirizzerà alla Console Amazon Redshift. Per continuare con la procedura, consulta la sezione relativa a DynamoDB in Create a zero-ETL integration for DynamoDB.

Visualizzazione delle integrazioni Zero-ETL di DynamoDB con Amazon Redshift

È possibile visualizzare i dettagli di un'integrazione Zero-ETL per vederne le informazioni di configurazione e lo stato attuale.

Per visualizzare i dettagli di un'integrazione Zero-ETL nella console Amazon DynamoDB:

1. Accedi alla console di AWS gestione e apri la console Amazon DynamoDB all'indirizzo. https://console.aws.amazon.com/dynamodbv2
2. Nella console DynamoDB, seleziona Integrazioni.
3. Nel riquadro Integrazione Zero-ETL, seleziona l'integrazione Zero-ETL che desideri visualizzare.

Per visualizzare i dettagli di un'integrazione Zero-ETL nella console Amazon Redshift:

1. Accedi alla console di AWS gestione e apri la console Amazon Redshift all'indirizzo. https://console.aws.amazon.com/redshiftv2
2. Segui la procedura indicata in Viewing zero-ETL integrations.

> **Note**
>
> I possibili stati di un'integrazione Zero-ETL con Amazon Redshift sono elencati nella sezione Viewing zero-ETL integrations nella Guida alla gestione di Amazon Redshift.

Eliminazione di integrazioni Zero-ETL di DynamoDB con Amazon Redshift

Se si elimina un'integrazione Zero-ETL, i dati non vengono eliminati da DynamoDB o Amazon Redshift, ma DynamoDB interrompe l'invio di dati dalla tabella di origine alla destinazione Amazon Redshift.

Eliminazione di un'integrazione Zero-ETL

1. Accedi alla console di AWS gestione e apri la console Amazon DynamoDB all'indirizzo. https://console.aws.amazon.com/dynamodbv2
2. Nella console DynamoDB, seleziona Integrazioni.
3. Nel riquadro Integrazione Zero-ETL, seleziona l'integrazione Zero-ETL che desideri eliminare.
4. Scegli Gestisci. Questo ti reindirizzerà alla pagina dei dettagli dell'integrazione.
5. Per confermare l'eliminazione, scegliere Delete (Elimina).

Caricamento di dati da DynamoDB in Amazon Redshift con il comando COPY

Amazon Redshift funziona insieme ad Amazon DynamoDB con funzionalità avanzate di business intelligence e una potente interfaccia basata su SQL. Quando si copiano i dati da una tabella DynamoDB in Amazon Redshift, è possibile eseguire query di analisi di dati complesse su tali dati, inclusi join con altre tabelle nel cluster Amazon Redshift.

In termini di velocità effettiva assegnata, un'operazione di copia da una tabella DynamoDB viene conteggiata rispetto alla capacità di lettura di quella tabella. Una volta copiati i dati, le query SQL in Amazon Redshift non influiscono su DynamoDB in alcun modo. Ciò accade perché le query agiscono su una copia dei dati di DynamoDB e non sullo stesso DynamoDB.

Prima di poter caricare dati da una tabella DynamoDB, è necessario creare prima una tabella Amazon Redshift che funzioni da destinazione per i dati. Tieni presente che copi dati da un ambiente NoSQL a un ambiente SQL e che determinate regole valide in un ambiente non si applicano nell'altro. Ecco alcune differenze da considerare:

- I nomi delle tabelle DynamoDB possono contenere un massimo di 255 caratteri, tra cui i caratteri '.' (punto) e '-' (trattino), e fanno distinzione tra maiuscole e minuscole. I nomi delle tabelle Amazon Redshift sono limitati a 127 caratteri, non possono contenere punti o trattini e non prevedono una

distinzione tra lettere maiuscole e minuscole. Inoltre, i nomi delle tabelle non possono essere in conflitto con le parole riservate di Amazon Redshift.

- DynamoDB non supporta il concetto SQL di NULL. È necessario specificare in che modo Amazon Redshift interpreta i valori degli attributi vuoti o vuoti in DynamoDB, trattandoli come o come campi vuoti. NULLs
- I tipi di dati di DynamoDB non corrispondono direttamente a quelli di Amazon Redshift. È necessario assicurarsi che ogni colonna nella tabella Amazon Redshift abbia il tipo di dati e le dimensioni corrette per accogliere i dati da DynamoDB.

Di seguito è riportato un esempio di comando dall'SQL di 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;
```

In questo esempio la tabella di origine in DynamoDB è `my-favorite-movies-table`. La tabella di destinazione in Amazon Redshift è `favoritemovies`. La clausola `readratio 50` regola la percentuale di throughput assegnato utilizzata; in questo caso, il comando COPY utilizzerà non più del 50% delle unità di capacità di lettura assegnate a `my-favorite-movies-table`. È consigliabile impostare questa percentuale su un valore minore del throughput assegnato mediamente non utilizzato.

Per istruzioni dettagliate sul caricamento di dati da DynamoDB ad Amazon Redshift, consulta le sezioni seguenti nella Guida per gli sviluppatori di database di Amazon Redshift:

- Caricamento di dati da una tabella DynamoDB
- Il comando COPY
- Esempi di COPY

Elaborazione dei dati di DynamoDB con Apache Hive su Amazon EMR

Amazon DynamoDB è integrato con Apache Hive, un'applicazione di data warehouse che viene eseguita su Amazon EMR. Hive è in grado di leggere e scrivere dati nelle tabelle DynamoDB, consentendo di:

- Eseguire la query dei dati DynamoDB in tempo reale utilizzando un linguaggio simile a SQL (HiveQL).
- Copiare i dati da una tabella DynamoDB ad un bucket Amazon S3 e viceversa.
- Copiare i dati da una tabella DynamoDB in un file di sistema distribuito Hadoop (HDFS) e viceversa.
- Eseguire operazioni join sulle tabelle DynamoDB.

Argomenti

- Panoramica
- Tutorial: Utilizzo di Amazon DynamoDB e Apache Hive
- Creazione di una tabella esterna in Hive
- Elaborazione delle istruzioni HiveQL
- Esecuzioni di query sui dati in DynamoDB
- Copia di dati da e verso Amazon DynamoDB
- Ottimizzazione prestazioni

Panoramica

Amazon EMR è un servizio che semplifica l'elaborazione di grandi quantità di dati in modo rapido ed economico. Per utilizzare Amazon EMR, si avvia un cluster gestito di istanze Amazon EC2 che eseguono il framework open source Hadoop. Hadoop è un'applicazione distribuita che implementa l'algoritmo MapReduce, in cui un'attività è mappata a più nodi nel cluster. Ogni nodo elabora il suo lavoro designato, in parallelo con gli altri nodi. Infine, le uscite sono ridotte a un singolo nodo, restituendo il risultato finale.

Puoi scegliere di avviare il cluster Amazon EMR in modo che sia persistente o transitorio:

- Un cluster persistente viene eseguito fino a quando non lo si spegne. I cluster persistenti sono ideali per l'analisi dei dati, il data warehouse o qualsiasi altro uso interattivo.
- Un cluster transitorio viene eseguito abbastanza a lungo per elaborare un flusso di lavoro e quindi si arresta automaticamente. I cluster transitori sono ideali per attività di elaborazione periodica, ad esempio l'esecuzione di script.

Per informazioni sull'architettura e l'amministrazione di Amazon EMR, consulta la Guida alla gestione di Amazon EMR.

Quando avvii un cluster Amazon EMR, specifichi il numero iniziale e il tipo di istanze di Amazon EC2. Vengono specificate anche altre applicazioni distribuite (oltre a Hadoop stesso) che si desidera eseguire nel cluster. Queste applicazioni includono Hue, Mahout, Pig, Spark e altro ancora.

Per informazioni sulle applicazioni per Amazon EMR, consulta la Guida al rilascio di Amazon EMR.

A seconda della configurazione del cluster, potresti avere uno o più dei seguenti tipi di nodi:

- Nodo principale: gestisce il cluster, coordinando la distribuzione dell'eseguibile MapReduce e dei sottoinsiemi dei dati non elaborati, ai gruppi di istanze core e task. Inoltre, tiene traccia dello stato di ogni attività eseguita e monitora l'integrità dei gruppi di istanze. In un cluster esiste un solo nodo principale.
- Nodi principali: esegue le attività MapReduce e archivia i dati utilizzando il file di sistema distribuito Hadoop (HDFS).
- Nodi attività (facoltativo): esegue le attività MapReduce.

Tutorial: Utilizzo di Amazon DynamoDB e Apache Hive

In questo tutorial, viene avviato un cluster Amazon EMR e quindi si utilizza Apache Hive per elaborare i dati archiviati in una tabella DynamoDB.

Hive è un'applicazione di data warehouse per Hadoop che consente di elaborare e analizzare i dati da più origini. Hive fornisce un linguaggio simile a SQL, HiveQL, che consente di lavorare con i dati archiviati in locale nel cluster Amazon EMR o in un'origine dati esterna (ad esempio Amazon DynamoDB).

Per maggiori informazioni, vedi il tutorial Hive.

Argomenti

- Prima di iniziare
- Fase 1: Creazione di una coppia di chiavi di Amazon EC2
- Fase 2: avvio di un cluster Amazon EMR
- Fase 3: connessione al nodo principale
- Fase 4: caricamento di dati in HDFS

- Fase 5: copia dei dati in DynamoDB
- Fase 6: esecuzione di query sui dati nella tabella DynamoDB
- Fase 7: pulizia (opzionale)

Prima di iniziare

Per questo tutorial hai bisogno dei seguenti elementi:

- Un account AWS. Se non lo hai, consulta Iscrizione a AWS.
- Un client SSH (Secure Shell). È possibile utilizzare il client SSH per connettersi al nodo principale del cluster Amazon EMR ed eseguire comandi interattivi. I client SSH sono disponibili per impostazione predefinita nella maggior parte delle installazioni Linux, Unix e Mac OS X. Gli utenti Windows possono scaricare e installare il client PuTTY, che ha il supporto SSH.

Approfondimenti

Fase 1: Creazione di una coppia di chiavi di Amazon EC2

Fase 1: Creazione di una coppia di chiavi di Amazon EC2

In questo passaggio, verrà creata la coppia di chiavi Amazon EC2 necessaria per connetterti a un nodo principale Amazon EMR ed eseguire i comandi Hive.

1. Accedi a Console di gestione AWS e apri la console Amazon EC2 all'indirizzo https://console.aws.amazon.com/ec2/.
2. Seleziona una regione (ad esempio, `US West (Oregon)`). Questa deve essere la stessa regione in cui si trova la tabella DynamoDB.
3. Nel pannello di navigazione scegli Coppie di chiavi.
4. Scegli Crea coppia di chiavi.
5. In Nome della coppia di chiavi digita un nome per la coppia di chiavi (ad esempio, `mykeypair`), quindi seleziona Crea.
6. Scarica il file della chiave privata. Il nome del file terminerà con `.pem` (come `mykeypair.pem`). Salvare il file della chiave privata in un luogo sicuro. Ne avrai bisogno per accedere a qualsiasi cluster Amazon EMR che avvii con questa coppia di chiavi.

> **Important**
>
> Se perdi la coppia di chiavi, non potrai connetterti al nodo principale del cluster Amazon EMR.

Per maggiori informazioni sulle coppie di chiavi, consulta Coppie di chiavi Amazon EC2 nella Guida per l'utente di Amazon EC2.

Approfondimenti

Fase 2: avvio di un cluster Amazon EMR

Fase 2: avvio di un cluster Amazon EMR

In questa fase, verrà configurato e avviato un cluster Amazon EMR. Hive e un gestore di archiviazione per DynamoDB dovranno già essere installati nel cluster.

1. Apri la console Amazon EMR all'indirizzo https://console.aws.amazon.com/emr.
2. Scegli Crea cluster.
3. Nella pagina Crea cluster - Opzioni rapide, completa le seguenti operazioni:
   a. In Nome cluster, immettere un nome per il cluster (ad esempio, `My EMR cluster`).
   b. In Coppia di chiavi EC2), seleziona la coppia di chiavi creata in precedenza.

   Lascia le altre impostazioni ai valori predefiniti.
4. Scegli Crea cluster.

Per avviare il cluster, sono necessari diversi minuti. Puoi utilizzare la pagina Dettagli del cluster nella console Amazon EMR per monitorarne lo stato di avanzamento.

Quando lo stato del cluster diventa `Waiting`, allora il cluster è pronto.

File di registro del cluster e Amazon S3

Un cluster Amazon EMR genera file di log che contengono informazioni sullo stato del cluster e sul debug. Le impostazioni predefinite per Crea cluster - Opzioni rapide includono la configurazione della registrazione di Amazon EMR.

Se uno non esiste già, nella Console di gestione AWS viene creato un bucket Amazon S3. Il nome del bucket è `aws-logs-`*`account-id`*`-`*`region`*, dove *`account-id`* è il tuo numero di account AWS e *`region`* è la regione in cui è stato avviato il cluster (ad esempio, `aws-logs-123456789012-us-west-2`).

> **Note**
>
> Puoi utilizzare la console Amazon S3 per visualizzare i file di log. Per ulteriori informazioni, consulta Visualizzazione dei file di log nella Guida alla gestione di Amazon EMR.

È possibile utilizzare questo bucket per scopi diversi dalla registrazione. Ad esempio, è possibile utilizzarlo come posizione per archiviare uno script Hive o come destinazione quando si esportano dati da Amazon DynamoDB ad Amazon S3.

Approfondimenti

Fase 3: connessione al nodo principale

Fase 3: connessione al nodo principale

Quando lo stato del cluster Amazon EMR cambia in `Waiting`, sarai in grado di connetterti al nodo principale utilizzando SSH ed eseguire operazioni dalla riga di comando.

1. Nella console di Amazon EMR, scegli il nome del cluster per visualizzarne lo stato.
2. Nella pagina Dettagli del cluster, individua il campo DNS pubblico principale. Questo è il nome DNS pubblico del nodo principale del cluster Amazon EMR.
3. A destra del nome DNS, seleziona il link SSH.
4. Seguire le istruzioni riportate in Connessione al nodo principale tramite SSH.

   A seconda del sistema operativo in uso, seleziona la scheda Windows oppure la scheda Mac/Linux e segui le istruzioni riportate per la connessione al nodo principale.

Dopo aver effettuato la connessione al nodo principale utilizzando SSH o PuTTY, viene visualizzato un prompt dei comandi simile al seguente:

```
[hadoop@ip-192-0-2-0 ~]$
```

Approfondimenti

Fase 4: caricamento di dati in HDFS

Fase 4: caricamento di dati in HDFS

In questo passaggio, verrà copiato un file di dati nel file di sistema distribuito Hadoop (HDFS) e quindi verrà creata una tabella Hive esterna mappata al file di dati.

Download dei dati di esempio

1. Scaricare l'archivio dei dati di esempio (`features.zip`):

   ```
   wget https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/samples/
   features.zip
   ```

2. Estrarre il file `features.txt` dall'archivio:

   ```
   unzip features.zip
   ```

3. Visualizzare le prime righe del file `features.txt`:

   ```
   head features.txt
   ```

   I risultati dovrebbero essere simili a quanto segue:

   ```
   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
   ```

Il file `features.txt` contiene un sottoinsieme di dati della United States Board on Geographic Names (http://geonames.usgs.gov/domestic/download_data.htm). I campi di ogni riga rappresentano i seguenti elementi:

- ID funzione (identificatore univoco)
- Nome
- Classe (lake, forest, stream e così via)
- Stato
- Latitudine (gradi)
- Longitudine (gradi)
- Altezza (in piedi)

4. Al prompt dei comandi, inserisci il comando seguente:

```
hive
```

Il prompt dei comandi viene modificato in `hive>`.

5. Immettere la seguente istruzione HiveQL per creare una tabella 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. Immettere la seguente istruzione HiveQL per caricare la tabella con i dati:

```
LOAD DATA
LOCAL
INPATH './features.txt'
OVERWRITE
INTO TABLE hive_features;
```

7. Ora si dispone di una tabella Hive nativa popolata con i dati dal file `features.txt`. Per verificare, immettere la seguente istruzione HiveQL:

```
SELECT state_alpha, COUNT(*)
FROM hive_features
GROUP BY state_alpha;
```

L'output dovrebbe mostrare un elenco di stati e il numero di caratteristiche geografiche in ciascuno.

Approfondimenti

Fase 5: copia dei dati in DynamoDB

Fase 5: copia dei dati in DynamoDB

In questa fase, i dati verranno copiati dalla tabella Hive (`hive_features`) in una nuova tabella in DynamoDB.

1. Apri la console DynamoDB all'indirizzo https://console.aws.amazon.com/dynamodb/.
2. Scegliere Create Table (Crea tabella).
3. Nello schermata Crea tabella DynamoDB, procedi nel seguente modo:

   a. In Tabella, digitare **`Features`**.

   b. Per Chiave primaria, nel campo Chiave di partizione, digitare **`Id`**. Impostare il tipo di dati su Number (Numero).

      Deseleziona l'opzione Utilizza impostazioni predefinite. In Capacità con provisioning, specificare:

      - Unità di capacità in lettura-`10`
      - Unità di capacità in scrittura-`10`

   Scegli Crea.
4. Al prompt di Hive immettere la seguente istruzione HiveQL:

```
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
);
```

A questo punto, è stata stabilita una mappatura tra Hive e la tabella Caratteristiche in DynamoDB.

5. Immettere la seguente istruzione HiveQL per importare i dati in 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 invierà un processo MapReduce, che verrà elaborato dal cluster Amazon EMR. Il completamento del processo può richiedere diversi minuti.

6. Verificare che i dati siano stati caricati in DynamoDB:

   a. Nel pannello di navigazione della console DynamoDB, seleziona Tabelle.

   b. Scegli la tabella Caratteristiche, quindi seleziona la scheda Elementi per visualizzare i dati.

Approfondimenti

Fase 6: esecuzione di query sui dati nella tabella DynamoDB

Fase 6: esecuzione di query sui dati nella tabella DynamoDB

In questa fase HiveQL verrà utilizzato per eseguire query nella tabella Caratteristiche in DynamoDB. Prova le seguenti query Hive:

1. Tutti i tipi di caratteristiche (`feature_class`) in ordine alfabetico:

```
SELECT DISTINCT feature_class
FROM ddb_features
ORDER BY feature_class;
```

2. Tutti i lake che iniziano con la lettera "M":

```
SELECT feature_name, state_alpha
FROM ddb_features
WHERE feature_class = 'Lake'
AND feature_name LIKE 'M%'
ORDER BY feature_name;
```

3. Stati con almeno tre caratteristiche superiori a un miglio (5.280 piedi):

```
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;
```

Approfondimenti

Fase 7: pulizia (opzionale)

Fase 7: pulizia (opzionale)

Dopo aver completato il tutorial, potrai continuare a leggere questa sezione per avere ulteriori informazioni su come lavorare con i dati di DynamoDB in Amazon EMR. Potresti decidere di mantenere il tuo cluster Amazon EMR attivo e funzionante mentre esegui questa operazione.

Se il cluster non è più necessario, dovrai terminarlo e rimuovere le risorse ad esso associate. Questo ti aiuterà a evitare che ti vengano addebitate risorse che non ti servono.

1. Terminare il cluster Amazon EMR:
   a. Apri la console Amazon EMR all'indirizzo https://console.aws.amazon.com/emr.
   b. Scegli il cluster Amazon EMR, seleziona Termina, quindi conferma.
2. Eliminare la tabella Caratteristiche in DynamoDB:
   a. Apri la console DynamoDB all'indirizzo https://console.aws.amazon.com/dynamodb/.
   b. Nel pannello di navigazione, seleziona Tabelle.
   c. Scegli la tabella Caratteristiche. Nel menu Operazioni seleziona Elimina tabella.
3. Eliminare il bucket Amazon S3 contenente i file di log Amazon EMR:
   a. Apri la console di Amazon S3 su https://console.aws.amazon.com/s3/.
   b. Dall'elenco dei bucket, scegliere `aws-logs-` *`accountID`*`-`*`region`*, dove *`accountID`* è il tuo numero di account AWS e *`region`* è la regione in cui è stato avviato il cluster.
   c. Nel menu Operazioni seleziona Elimina.

Creazione di una tabella esterna in Hive

Nel Tutorial: Utilizzo di Amazon DynamoDB e Apache Hive, è stata creata una tabella Hive esterna mappata a una tabella DynamoDB. Quando sono state emesse istruzioni HiveQL sulla tabella esterna, le operazioni di lettura e scrittura sono state passate alla tabella DynamoDB.

È possibile considerare una tabella esterna come un puntatore a un'origine dati gestita e memorizzata altrove. In questo caso, l'origine dati sottostante è una tabella DynamoDB. La tabella deve esistere già. Non è possibile creare, aggiornare o eliminare una tabella DynamoDB da Hive. Per creare la tabella esterna, è possibile utilizzare l'istruzione `CREATE EXTERNAL TABLE`. Successivamente, sarà possibile utilizzare HiveQL per lavorare con i dati in DynamoDB come se tali dati fossero archiviati in locale all'interno di Hive.

> **Note**
>
> È possibile utilizzare le istruzioni `INSERT` per inserire dati in una tabella esterna e le istruzioni `SELECT` per selezionare i dati da essa. Tuttavia, non è possibile utilizzare le istruzioni `UPDATE` o `DELETE`per manipolare i dati nella tabella.

Se la tabella esterna non è più necessaria, puoi rimuoverla utilizzando l'istruzione `DROP TABLE`. In questo caso, `DROP TABLE` rimuove solo la tabella esterna in Hive. Non influisce sulla tabella DynamoDB sottostante o sui relativi dati.

Argomenti

- Sintassi di CREATE EXTERNAL TABLE
- Mappature dei tipi di dati

Sintassi di CREATE EXTERNAL TABLE

Di seguito viene illustrata la sintassi HiveQL per la creazione di una tabella Hive esterna mappata a una tabella 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 riga 1 è l'inizio dell'istruzione `CREATE EXTERNAL TABLE`, in cui si fornisce il nome della tabella Hive (hive_table) che si desidera creare.

La riga 2 specifica le colonne e i tipi di dati per hive_table. È necessario definire colonne e tipi di dati che corrispondono agli attributi nella tabella DynamoDB.

La riga 3 è la clausola `STORED BY`, in cui si specifica una classe incaricata della gestione dei dati tra Hive e la tabella DynamoDB. Per DynamoDB ,`STORED BY` deve essere impostato su `'org.apache.hadoop.hive.dynamodb.DynamoDBStorageHandler'`.

La riga 4 è l'inizio della clausola `TBLPROPERTIES`, in cui si definiscono i seguenti parametri per `DynamoDBStorageHandler`:

- `dynamodb.table.name`: il nome della tabella DynamoDB.

- `dynamodb.column.mapping`: le coppie di nomi di colonna nella tabella Hive e i relativi attributi nella tabella DynamoDB. Ogni coppia ha il formato hive_column_name:dynamodb_attribute_name e le coppie sono separate da virgole.

Tieni presente quanto segue:

- Il nome della tabella Hive non deve essere uguale al nome della tabella DynamoDB.
- Il nome delle colonne della tabella Hive non devono essere uguali a quelli nella tabella DynamoDB.
- La tabella specificata da `dynamodb.table.name` deve già essere presente in DynamoDB.
- Per `dynamodb.column.mapping`:
  - È necessario mappare gli attributi dello schema chiave per la tabella DynamoDB. Questa operazione include la chiave di partizione e la chiave di ordinamento (se presente).
  - Non è necessario mappare gli attributi non chiave della tabella DynamoDB. Tuttavia, quando si esegue una query sulla tabella Hive, non verrà visualizzato alcun dato da tali attributi.
  - Se i tipi di dati di una colonna di tabella Hive e di un attributo DynamoDB non sono compatibili, in queste colonne verrà visualizzato `NULL` quando si esegue una query sulla tabella Hive.

> **Note**
>
> L'istruzione `CREATE EXTERNAL TABLE` non esegue alcuna convalida sulla clausola `TBLPROPERTIES`. I valori forniti per `dynamodb.table.name` e `dynamodb.column.mapping` vengono valutati solo dalla classe `DynamoDBStorageHandler` quando si prova ad accedere alla tabella.

Mappature dei tipi di dati

Nella tabella seguente vengono illustrati i tipi di dati DynamoDB e i tipi di dati Hive compatibili:

Tipo di dati DynamoDB	Tipo di dati Hive
Stringa	`STRING`
Numero	`BIGINT` o `DOUBLE`
Binario	`BINARY`

Tipo di dati DynamoDB	Tipo di dati Hive
Set di stringhe	`ARRAY<STRING>`
Set numerico	`ARRAY<BIGINT>` o `ARRAY<DOUBLE>`
Set binario	`ARRAY<BINARY>`

> **Note**
>
> I seguenti tipi di dati DynamoDB non sono supportati dalla classe `DynamoDBStorageHandler`, quindi non possono essere utilizzati con `dynamodb.column.mapping`:
>
> - Eseguire la mappatura
> - Elenco
> - Booleano
> - Null
>
> Tuttavia, se devi lavorare con questi tipi di dati, puoi creare una singola entità chiamata `item` che rappresenti l'intero elemento DynamoDB come mappa di stringhe sia per le chiavi che per i valori della mappa. Per ulteriori informazioni, consulta Copia di dati senza mappature di colonne

Se desideri mappare un attributo DynamoDB di tipo Number, è necessario scegliere un tipo Hive appropriato:

- Il tipo Hive `BIGINT` è per interi firmati da 8 byte. È uguale al tipo di dati `long` in Java.
- Il tipo Hive `DOUBLE` è per numeri a virgola mobile doppi a 8 bit. È uguale al tipo `double` in Java.

Se si dispone di dati numerici memorizzati in DynamoDB con una precisione superiore rispetto al tipo di dati Hive scelto, l'accesso ai dati DynamoDB potrebbe causare una perdita di precisione.

Se si esportano dati di tipo Binary da DynamoDB in (Amazon S3) o HDFS, i dati vengono archiviati come stringa con codifica Base64. Se si importano dati da Amazon S3 o HDFS nel tipo Binary di DynamoDB, assicurarsi che siano codificati come stringa Base64.

Elaborazione delle istruzioni HiveQL

Hive è un'applicazione che viene eseguita su Hadoop, che è un framework orientato al batch per l'esecuzione di processi MapReduce. Quando si emette un'istruzione HiveQL, Hive determina se può restituire immediatamente i risultati o se deve inviare un processo MapReduce.

Ad esempio, si consideri la tabella ddb_features(da Tutorial: Utilizzo di Amazon DynamoDB e Apache Hive). La seguente query Hive stampa le abbreviazioni di stato e il numero di summit in ciascuno di essi:

```
SELECT state_alpha, count(*)
FROM ddb_features
WHERE feature_class = 'Summit'
GROUP BY state_alpha;
```

Hive non restituisce immediatamente i risultati. Invece, invia un processo MapReduce, che viene elaborato dal framework Hadoop. Hive attenderà fino al completamento del processo prima di visualizzare i risultati della query:

```
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)
```

Monitoraggio e annullamento dei processi

Quando Hive avvia un processo Hadoop, stampa l'output da quel processo. Lo stato di completamento del processo viene aggiornato man mano che il processo va avanti. In alcuni casi, lo stato potrebbe non essere aggiornato per un lungo periodo di tempo. Ciò può verificarsi, ad esempio, quando si esegue una query su una tabella DynamoDB di grandi dimensioni con un'impostazione di capacità di lettura con provisioning basso.

Se è necessario annullare il processo prima che sia completato, è possibile digitare **`Ctrl+C`** in qualsiasi momento.

Esecuzioni di query sui dati in DynamoDB

Negli esempi seguenti vengono illustrati i diversi modi in cui puoi utilizzare HiveQL per eseguire query sui dati archiviati in DynamoDB.

Questi esempi si riferiscono alla tabella ddb_features del tutorial (Fase 5: copia dei dati in DynamoDB).

Argomenti

- Uso delle funzioni di aggregazione
- Uso delle clausole GROUP BY e HAVING
- Unione di due tabelle DynamoDB
- Unione di tabelle da origini diverse

Uso delle funzioni di aggregazione

HiveQL fornisce funzioni integrate per il riepilogo dei valori dei dati. Ad esempio, è possibile utilizzare la funzione `MAX` per trovare il valore più grande per una colonna selezionata. Nell'esempio seguente viene restituita l'elevazione della caratteristica più alta nello stato del Colorado.

```
SELECT MAX(elev_in_ft)
FROM ddb_features
WHERE state_alpha = 'CO';
```

Uso delle clausole GROUP BY e HAVING

Puoi utilizzare la clausola `GROUP BY` per raccogliere dati all'interno di più record. Questa clausola viene spesso utilizzata con una funzione di aggregazione come `SUM`, `COUNT`, `MIN` o `MAX`. È possibile utilizzare la clausola `HAVING` anche per eliminare tutti i risultati che non soddisfano determinati criteri.

L'esempio seguente restituisce un elenco delle elevazioni più alte dagli stati che hanno più di cinque caratteristiche nella tabella ddb_features.

```
SELECT state_alpha, max(elev_in_ft)
FROM ddb_features
GROUP BY state_alpha
HAVING count(*) >= 5;
```

Unione di due tabelle DynamoDB

L'esempio seguente mappa un'altra tabella Hive (east_coast_states) a una tabella in DynamoDB. L'istruzione `SELECT` è un join tra queste due tabelle. Il join viene calcolato nel cluster e in seguito viene restituito. Il join non si verifica in DynamoDB.

Si consideri una tabella DynamoDB denominata EastCoastStates che contiene i seguenti dati:

```
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
```

Supponiamo che la tabella sia disponibile come tabella esterna Hive denominata 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");
```

Il join seguente restituisce gli stati sulla costa orientale degli Stati Uniti che hanno almeno tre caratteristiche:

```
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;
```

Unione di tabelle da origini diverse

In questo esempio, s3_east_coast_states è una tabella Hive associata a un file CSV archiviato in Amazon S3. La tabella ddb_features è associata ai dati in DynamoDB. Nell'esempio seguente vengono unite queste due tabelle, restituendo le caratteristiche geografiche dagli stati i cui nomi iniziano con "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 di dati da e verso Amazon DynamoDB

Nel Tutorial: Utilizzo di Amazon DynamoDB e Apache Hive, sono stati copiati i dati da una tabella Hive nativa in una tabella DynamoDB esterna e quindi è stata eseguita una query sulla tabella

DynamoDB esterna. La tabella è esterna perché esiste al di fuori di Hive. Anche se si rilascia la tabella Hive ad essa mappata, la tabella in DynamoDB non è interessata.

Hive è una eccellente soluzione per copiare i dati tra tabelle DynamoDB, bucket Amazon S3, tabelle Hive native e file di sistema distribuito Hadoop (HDFS). In questa sezione vengono forniti alcuni esempi di queste operazioni.

Argomenti

- Copia di dati tra DynamoDB e una tabella Hive nativa
- Copia di dati tra DynamoDB e Amazon S3
- Copia di dati tra DynamoDB e HDFS
- Utilizzando la compressione dati
- Lettura di dati di caratteri UTF-8 non stampabili

Copia di dati tra DynamoDB e una tabella Hive nativa

Se si dispone di dati in una tabella DynamoDB, è possibile copiarli in una tabella Hive nativa. Questo creerà uno snapshot di dati, a partire dal momento in cui li hai copiati.

Ciò potrebbe essere la scelta giusta nel caso in cui sia necessario eseguire molte query HiveQL, ma non si desidera consumare capacità di throughput assegnata da DynamoDB. Poiché i dati nella tabella Hive nativa sono una copia dei dati da DynamoDB e non sono dati "live", le query non dovrebbero aspettarsi che i dati siano aggiornati.

> **Note**
>
> Gli esempi in questa sezione presuppongono che sia stata seguita la procedura in Tutorial: Utilizzo di Amazon DynamoDB e Apache Hive che sia presente una tabella esterna in DynamoDB denominata ddb_features.

Example Da DynamoDB a una tabella Hive nativa

È possibile creare una tabella Hive nativa e popolarla con i dati da ddb_features in questo modo:

```
CREATE TABLE features_snapshot AS
SELECT * FROM ddb_features;
```

È quindi possibile aggiornare i set di dati in qualsiasi momento:

```
INSERT OVERWRITE TABLE features_snapshot
SELECT * FROM ddb_features;
```

In questi esempi, la query secondaria `SELECT * FROM ddb_features` recupererà tutti i dati da ddb_features. Se desideri copiare solo una parte dei dati, è possibile utilizzare una clausola `WHERE` nella query secondaria.

L'esempio seguente crea una tabella Hive nativa, contenente solo alcuni attributi per lake e summit:

```
CREATE TABLE lakes_and_summits AS
SELECT feature_name, feature_class, state_alpha
FROM ddb_features
WHERE feature_class IN ('Lake','Summit');
```

Example Da una tabella Hive nativa a DynamoDB

Utilizza l'istruzione HiveQL seguente per copiare i dati dalla tabella Hive nativa in ddb_features:

```
INSERT OVERWRITE TABLE ddb_features
SELECT * FROM features_snapshot;
```

Copia di dati tra DynamoDB e Amazon S3

Se si dispone di dati in una tabella DynamoDB, è possibile utilizzare Hive per copiarli in un bucket Amazon S3.

È possibile eseguire questa operazione se si desidera creare un archivio di dati nella tabella DynamoDB. Ad esempio, si supponga di disporre di un ambiente di test in cui è necessario lavorare con un set di base di dati di test in DynamoDB. È possibile copiare i dati di base in un bucket Amazon S3 e quindi eseguire i test. Successivamente, sarà possibile reimpostare l'ambiente di test ripristinando i dati di base dal bucket di Amazon S3 su DynamoDB.

Se è stato seguito il Tutorial: Utilizzo di Amazon DynamoDB e Apache Hive, si dispone già di un bucket Amazon S3 contenente i log Amazon EMR. È possibile utilizzare questo bucket per gli esempi in questa sezione, se si conosce il percorso principale per il bucket:

1. Apri la console Amazon EMR all'indirizzo https://console.aws.amazon.com/emr.

2. Per Nome, scegli il nome del cluster.
3. L'URI è riportato in URI log in Dettagli di configurazione.
4. Prendere nota del percorso completo dei bucket. La convenzione di denominazione è:

   `s3://aws-logs-`*`accountID`*`-`*`region`*

   dove *`accountID`* è il tuo ID account AWS e region è la regione AWS per il bucket.

> **Note**
>
> Per questi esempi, useremo un percorso secondario all'interno del bucket.
> `s3://aws-logs-123456789012-us-west-2/hive-test`

Le procedure seguenti presuppongono che sia stata seguita la procedura nel tutorial e che sia presente una tabella esterna in DynamoDB denominata ddb_features.

Argomenti

- Copia di dati utilizzando il formato predefinito Hive
- Copia di dati con un formato specificato dall'utente
- Copia di dati senza mappature di colonne
- Visualizzazione dei dati in Amazon S3

Copia di dati utilizzando il formato predefinito Hive

Example Da DynamoDB a Amazon S3

Utilizza un'istruzione `INSERT OVERWRITE` per scrivere direttamente in Amazon S3.

```
INSERT OVERWRITE DIRECTORY 's3://aws-logs-123456789012-us-west-2/hive-test'
SELECT * FROM ddb_features;
```

Il file di dati in Amazon S3 ha il seguente aspetto:

```
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
```

Ogni campo è separato da un carattere SOH (inizio dell'intestazione, 0x01). Nel file, SOH appare come **^A**.

Example Da Amazon S3 a DynamoDB

1. Crea una tabella esterna che punta ai dati non formattati in 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. Copiare i dati in DynamoDB.

```
INSERT OVERWRITE TABLE ddb_features
SELECT * FROM s3_features_unformatted;
```

Copia di dati con un formato specificato dall'utente

Se desideri specificare il proprio carattere separatore dei campi, è possibile creare una tabella esterna mappata al bucket Amazon S3. È possibile utilizzare questa tecnica per creare file di dati con valori separati da virgole (CSV).

Example Da DynamoDB a Amazon S3

1. Crea una tabella esterna Hive mappata ad Amazon S3. Quando si esegue questa operazione, assicurarsi che i tipi di dati siano coerenti con quelli della tabella esterna DynamoDB.

```
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. Copiare i dati da DynamoDB.

```
INSERT OVERWRITE TABLE s3_features_csv
SELECT * FROM ddb_features;
```

Il file di dati in Amazon S3 ha il seguente aspetto:

```
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 Da Amazon S3 a DynamoDB

Con una singola istruzione HiveQL, è possibile popolare la tabella DynamoDB utilizzando i dati di Amazon S3:

```
INSERT OVERWRITE TABLE ddb_features
SELECT * FROM s3_features_csv;
```

Copia di dati senza mappature di colonne

È possibile copiare i dati da DynamoDB in un formato non elaborato e scriverli in Amazon S3 senza specificare alcun tipo di dati o mappatura di colonne. Questo metodo può essere utilizzato per creare un archivio dei dati DynamoDB e memorizzarlo in Amazon S3.

Example Da DynamoDB a Amazon S3

1. Crea una tabella esterna associata alla tabella DynamoDB. In questa istruzione HiveQL non c'è `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. Crea un'altra tabella esterna associata al bucket 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. Copiare i dati da DynamoDB ad Amazon S3.

```
INSERT OVERWRITE TABLE s3_features_no_mapping
SELECT * FROM ddb_features_no_mapping;
```

Il file di dati in Amazon S3 ha il seguente aspetto:

```
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
```

Ogni campo inizia con un carattere STX (inizio del testo, 0x02) e termina con un carattere ETX (fine del testo, 0x03). Nel file, STX viene visualizzato come **^B** e ETX viene visualizzato come **^C**.

Example Da Amazon S3 a DynamoDB

Con una singola istruzione HiveQL, è possibile popolare la tabella DynamoDB utilizzando i dati di Amazon S3:

```
INSERT OVERWRITE TABLE ddb_features_no_mapping
```

```
SELECT * FROM s3_features_no_mapping;
```

Visualizzazione dei dati in Amazon S3

Se per connettersi al nodo principale si utilizza SSH, è possibile utilizzare il comando AWS Command Line Interface (AWS CLI) per accedere ai dati che Hive ha scritto su Amazon S3.

I passaggi seguenti presuppongono che siano stati copiati i dati da DynamoDB ad Amazon S3 utilizzando una delle procedure descritte in questa sezione.

1. Se ti trovi attualmente al prompt dei comandi di Hive, passa al prompt dei comandi di Linux.

   ```
   hive> exit;
   ```

2. Elencare il contenuto della directory hive-test nel bucket Amazon S3. Questo è il percorso in cui Hive ha copiato i dati da DynamoDB.

   ```
   aws s3 ls s3://aws-logs-123456789012-us-west-2/hive-test/
   ```

   La risposta dovrebbe essere simile alla seguente:

   ```
   2016-11-01 23:19:54 81983 000000_0
   ```

   Il nome del file (000000_0) è generato dal sistema.

3. (Facoltativo) È possibile copiare il file di dati da Amazon S3 nel file system locale sul nodo principale. Successivamente, sarà possibile utilizzare le utilità della riga di comando Linux standard per lavorare con i dati nel file.

   ```
   aws s3 cp s3://aws-logs-123456789012-us-west-2/hive-test/000000_0 .
   ```

   La risposta dovrebbe essere simile alla seguente:

   ```
   download: s3://aws-logs-123456789012-us-west-2/hive-test/000000_0
   to ./000000_0
   ```

   > **Note**
   >
   > Il file system locale sul nodo principale ha una capacità limitata. Non utilizzare questo comando con file più grandi dello spazio disponibile nel file system locale.

Copia di dati tra DynamoDB e HDFS

Se si dispone di dati in una tabella DynamoDB, è possibile utilizzare Hive per copiare i dati nel file di sistema distribuito Hadoop (HDFS).

È possibile eseguire questa operazione se si esegue un processo MapReduce che richiede dati da DynamoDB. Se si copiano i dati da DynamoDB in HDFS, Hadoop può elaborarli utilizzando tutti i nodi disponibili nel cluster Amazon EMR in parallelo. Al termine del processo MapReduce, è possibile scrivere i risultati da HDFS a DDB.

Negli esempi seguenti, Hive leggerà e scriverà nella seguente directory HDFS: `/user/hadoop/hive-test`

> **Note**
>
> Gli esempi in questa sezione presuppongono che sia stata seguita la procedura in Tutorial: Utilizzo di Amazon DynamoDB e Apache Hive e che sia presente una tabella esterna in DynamoDB denominata ddb_features.

Argomenti

- Copia di dati utilizzando il formato predefinito Hive
- Copia di dati con un formato specificato dall'utente
- Copia di dati senza mappature di colonne
- Accesso ai dati in HDFS

Copia di dati utilizzando il formato predefinito Hive

Example Da DynamoDB a HDFS

Utilizza un'istruzione `INSERT OVERWRITE` per scrivere direttamente su HDFS.

```
INSERT OVERWRITE DIRECTORY 'hdfs:///user/hadoop/hive-test'
SELECT * FROM ddb_features;
```

Il file di dati in HDFS ha il seguente aspetto:

```
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
```

Ogni campo è separato da un carattere SOH (inizio dell'intestazione, 0x01). Nel file, SOH appare come **^A**.

Example Da HDFS a DynamoDB

1. Crea una tabella esterna che si mappa ai dati non formattati in 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. Copiare i dati in DynamoDB.

```
INSERT OVERWRITE TABLE ddb_features
SELECT * FROM hdfs_features_unformatted;
```

Copia di dati con un formato specificato dall'utente

Se desideri utilizzare un carattere di separatore di campo diverso, è possibile creare una tabella esterna mappata alla directory HDFS. È possibile utilizzare questa tecnica per creare file di dati con valori separati da virgole (CSV).

Example Da DynamoDB a HDFS

1. Crea una tabella esterna Hive mappata ad HDFS. Quando si esegue questa operazione, assicurarsi che i tipi di dati siano coerenti con quelli della tabella esterna DynamoDB.

```
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. Copiare i dati da DynamoDB.

```
INSERT OVERWRITE TABLE hdfs_features_csv
SELECT * FROM ddb_features;
```

Il file di dati in HDFS ha il seguente aspetto:

```
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 Da HDFS a DynamoDB

Con una singola istruzione HiveQL, è possibile popolare la tabella DynamoDB utilizzando i dati da HDFS:

```
INSERT OVERWRITE TABLE ddb_features
SELECT * FROM hdfs_features_csv;
```

Copia di dati senza mappature di colonne

È possibile copiare i dati da DynamoDB in un formato non elaborato e scriverli in HDFS senza specificare alcun tipo di dati o mappatura di colonne. Questo metodo può essere utilizzato per creare un archivio dei dati DynamoDB e memorizzarlo in HDFS.

> **Note**
>
> Se la tabella DynamoDB contiene attributi di tipo Map, List, Boolean o Null, allora questo è l'unico modo per utilizzare Hive per copiare i dati da DynamoDB a HDFS.

Example Da DynamoDB a HDFS

1. Crea una tabella esterna associata alla tabella DynamoDB. In questa istruzione HiveQL non c'è `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. Crea un'altra tabella esterna associata alla directory 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. Copiare i dati da DynamoDB ad HDFS.

```
INSERT OVERWRITE TABLE hdfs_features_no_mapping
SELECT * FROM ddb_features_no_mapping;
```

Il file di dati in HDFS ha il seguente aspetto:

```
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
```

Ogni campo inizia con un carattere STX (inizio del testo, 0x02) e termina con un carattere ETX (fine del testo, 0x03). Nel file, STX viene visualizzato come **^B** e ETX viene visualizzato come **^C**.

Example Da HDFS a DynamoDB

Con una singola istruzione HiveQL, è possibile popolare la tabella DynamoDB utilizzando i dati da HDFS:

```
INSERT OVERWRITE TABLE ddb_features_no_mapping
SELECT * FROM hdfs_features_no_mapping;
```

Accesso ai dati in HDFS

HDFS è un file system distribuito, accessibile a tutti i nodi del cluster Amazon EMR. Se per connettersi al nodo principale si utilizza SSH, è possibile utilizzare gli strumenti a riga di comando per accedere ai dati che Hive ha scritto su HDFS.

HDFS non è la stessa cosa del file system locale sul nodo principale. Non è possibile lavorare con file e directory in HDFS utilizzando comandi Linux standard (come `cat`, `cp`, `mv` o `rm`). Invece, queste attività vengono eseguite utilizzando il comando `hadoop fs`.

I passaggi seguenti presuppongono che siano stati copiati i dati da DynamoDB ad HDFS utilizzando una delle procedure descritte in questa sezione.

1. Se ti trovi attualmente al prompt dei comandi di Hive, passa al prompt dei comandi di Linux.

   ```
   hive> exit;
   ```

2. Elencare il contenuto della directory /user/hadoop/hive-test in HDFS. Questo è il percorso in cui Hive ha copiato i dati da DynamoDB.

   ```
   hadoop fs -ls /user/hadoop/hive-test
   ```

   La risposta dovrebbe essere simile alla seguente:

   ```
   Found 1 items
   -rw-r--r-- 1 hadoop hadoop 29504 2016-06-08 23:40 /user/hadoop/hive-test/000000_0
   ```

   Il nome del file (000000_0) è generato dal sistema.

3. Visualizzare il contenuto del file :

   ```
   hadoop fs -cat /user/hadoop/hive-test/000000_0
   ```

> **Note**
>
> In questo esempio, il file è relativamente piccolo (circa 29 KB). Fare attenzione quando si utilizza questo comando con file molto grandi o contenenti caratteri non stampabili.

4. (Facoltativo) È possibile copiare il file di dati da HDFS nel file system locale sul nodo principale. Successivamente, sarà possibile utilizzare le utilità della riga di comando Linux standard per lavorare con i dati nel file.

```
hadoop fs -get /user/hadoop/hive-test/000000_0
```

Questo comando non sovrascriverà il file.

> **Note**
>
> Il file system locale sul nodo principale ha una capacità limitata. Non utilizzare questo comando con file più grandi dello spazio disponibile nel file system locale.

Utilizzando la compressione dati

Quando si utilizza Hive per copiare dati tra origini dati diverse, è possibile richiedere la compressione dei dati al volo. Hive fornisce diversi codec di compressione. È possibile sceglierne uno durante la sessione di Hive. In questo modo, i dati vengono compressi nel formato specificato.

L'esempio seguente comprime i file utilizzando l'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;
```

Il file risultante in Amazon S3 avrà un nome generato dal sistema con `.lzo` alla fine (ad esempio, `8d436957-57ba-4af7-840c-96c2fc7bb6f5-000000.lzo`).

I codec di compressione disponibili sono:

- `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`

Lettura di dati di caratteri UTF-8 non stampabili

Per leggere e scrivere dati di caratteri UTF-8 non stampabili, è possibile utilizzare la clausola `STORED AS SEQUENCEFILE` durante la creazione di una tabella Hive. Un SequenceFile è un formato di file binario Hadoop. Per leggere questo file è necessario usare Hadoop. Nell'esempio seguente viene illustrato come esportare dati da DynamoDB verso Amazon S3. Puoi utilizzare questa funzionalità per gestire i caratteri in codifica UTF-8 non stampabili.

```
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;
```

Ottimizzazione prestazioni

Quando si crea una tabella esterna Hive mappata a una tabella DynamoDB, non si utilizza alcuna capacità di lettura o scrittura da DynamoDB. Tuttavia, un'attività di lettura e scrittura nella tabella Hive (ad esempio `INSERT` o `SELECT`) si traduce direttamente in operazioni di lettura e scrittura nella tabella DynamoDB sottostante.

Apache Hive su Amazon EMR implementa la propria logica per bilanciare il carico I/O sulla tabella DynamoDB e cerca di ridurre al minimo la possibilità di superare il throughput fornito dalla tabella. Al termine di ogni query Hive, Amazon EMR restituisce parametri di runtime, incluso il numero di volte in cui il throughput assegnato è stato superato. È possibile utilizzare queste informazioni,

insieme ai parametri CloudWatch nella tabella DynamoDB, per migliorare le prestazioni nelle richieste successive.

La console di Amazon EMR fornisce strumenti di monitoraggio di base per il tuo cluster. Per maggiori informazioni, consulta Visualizzazione e monitoraggio di un cluster nella Guida alla gestione di Amazon EMR.

È inoltre possibile monitorare i processi cluster e Hadoop utilizzando strumenti basati sul Web, come Hue, Ganglia e l'interfaccia Web Hadoop. Per ulteriori informazioni, consulta Visualizzazione delle interfacce Web ospitate nei cluster Amazon EMR nella Guida alla gestione di Amazon EMR.

In questa sezione vengono descritti i passaggi che è possibile eseguire per ottimizzare le operazioni Hive nelle tabelle esterne di DynamoDB.

Argomenti

- Velocità di trasmissione effettiva assegnata a DynamoDB
- Regolazione dei mappatori
- Argomenti aggiuntivi

Velocità di trasmissione effettiva assegnata a DynamoDB

Quando si emettono istruzioni HiveQL sulla tabella DynamoDB esterna, la proprietà `DynamoDBStorageHandler` crea le richieste API DynamoDB di basso livello appropriate, che utilizzano la velocità effettiva di provisioning. Se nella tabella DynamoDB non vi è sufficiente capacità di lettura o scrittura, la richiesta verrà limitata, con conseguente rallentamento delle prestazioni HiveQL. Per questo motivo, è consigliabile assicurarsi che la tabella disponga di capacità di throughput sufficiente.

Ad esempio, si supponga di avere 100 unità di capacità di lettura assegnate per la tabella DynamoDB. Questo ti permetterà di leggere 409.600 byte al secondo (dimensioni dell'unità di capacità di lettura 100 × 4 KB). Supponiamo ora che la tabella contenga 20 GB di dati (21.474.836.480 byte) e che si desideri utilizzare la funzione `SELECT` per selezionare tutti i dati utilizzando HiveQL. È possibile stimare quanto tempo impiegherà la query per l'esecuzione in questo modo:

21.474.836.480 / 409.600 = 52.429 secondi = 14,56 ore

In questo scenario, la tabella DynamoDB è un collo di bottiglia. Non aiuterà ad aggiungere altri nodi Amazon EMR perché il throughput Hive è limitato a soli 409.600 byte al secondo. L'unico modo per

ridurre il tempo necessario per l'istruzione SELECT è di aumentare la capacità di lettura assegnata della tabella DynamoDB.

È possibile eseguire un calcolo simile per stimare il tempo necessario per caricare dati in massa in una tabella esterna Hive mappata a una tabella DynamoDB. Determina il numero totale di unità di capacità di scrittura necessarie per elemento (meno di 1 KB = 1, 1-2 KB = 2, ecc.) e moltiplicalo per il numero di elementi da caricare. Il risultato sarà il numero di unità di capacità di scrittura. Dividi questo risultato per il numero di unità di capacità di scrittura allocate al secondo. Ciò produrrà il numero di secondi necessari per caricare la tabella.

È necessario monitorare regolarmente i parametri CloudWatch per la tua tabella. Per una rapida panoramica nella console DynamoDB, scegli la tabella e seleziona la scheda Parametri. Da qui, è possibile visualizzare le unità di capacità di lettura e scrittura consumate e le richieste di lettura e scrittura che sono state limitate.

Capacità di lettura

Per impostazione predefinita, Amazon EMR gestisce il carico di richieste sulla tabella DynamoDB, in base alle impostazioni di velocità effettiva assegnata della tabella. Tuttavia, se si nota un grande numero di messaggi `ProvisionedThroughputExceeded` nell'output del processo, è possibile regolare la velocità di lettura predefinita. A tale scopo, puoi modificare la variabile di configurazione `dynamodb.throughput.read.percent`. Puoi utilizzare il comando `SET` per impostare questa variabile al prompt dei comandi di Hive:

```
SET dynamodb.throughput.read.percent=1.0;
```

Questa variabile persiste nella sessione di Hive corrente. Se esci da Hive e torni in un secondo momento, `dynamodb.throughput.read.percent`tornerà al suo valore predefinito.

Il valore di `dynamodb.throughput.read.percent` può essere compreso tra `0.1` e `1.5`, estremi inclusi. `0.5` rappresenta la velocità di lettura predefinita, il che significa che Hive cercherà di consumare metà della capacità di lettura della tabella. Se si aumenta il valore sopra `0.5`, Hive aumenterà il tasso di richiesta; diminuendo il valore al di sotto di `0.5` riduce la frequenza di richiesta di lettura. La velocità di lettura effettiva varia in base a fattori come la presenza di una distribuzione uniforme delle chiavi nella tabella DynamoDB.

Se si nota che Hive spesso esaurisce la capacità di lettura sottoposta a provisioning della tabella o se le richieste di lettura sono troppo limitate, provare a ridurre `dynamodb.throughput.read.percent` al di sotto di `0.5`. Se si dispone di capacità di lettura

sufficiente nella tabella e si desidera che le operazioni HiveQL siano più reattive, è possibile impostare il valore sopra `0.5`.

Capacità di scrittura

Per impostazione predefinita, Amazon EMR gestisce il carico di richieste sulla tabella DynamoDB, in base alle impostazioni di velocità effettiva assegnata della tabella. Tuttavia, se si nota un grande numero di messaggi `ProvisionedThroughputExceeded`nell'output del processo, è possibile regolare la velocità di scrittura predefinita. A tale scopo, puoi modificare la variabile di configurazione `dynamodb.throughput.write.percent`. Puoi utilizzare il comando `SET` per impostare questa variabile al prompt dei comandi di Hive:

```
SET dynamodb.throughput.write.percent=1.0;
```

Questa variabile persiste nella sessione di Hive corrente. Se esci da Hive e torni in un secondo momento, `dynamodb.throughput.write.percent`tornerà al suo valore predefinito.

Il valore di `dynamodb.throughput.write.percent` può essere compreso tra `0.1` e `1.5`, estremi inclusi. `0.5` rappresenta la velocità di scrittura predefinita, il che significa che Hive cercherà di consumare metà della capacità di scrittura della tabella. Se si aumenta il valore sopra `0.5`, Hive aumenterà il tasso di richiesta; diminuendo il valore al di sotto di `0.5` riduce la frequenza di richiesta di scrittura. La velocità di scrittura effettiva varia in base a fattori come la presenza di una distribuzione uniforme delle chiavi nella tabella DynamoDB.

Se si nota che Hive spesso esaurisce la capacità di lettura sottoposta a provisioning della tabella o se le richieste di scrittura sono troppo limitate, provare a ridurre `dynamodb.throughput.write.percent` al di sotto di `0.5`. Se si dispone di capacità di scrittura sufficiente nella tabella e si desidera che le operazioni HiveQL siano più reattive, è possibile impostare il valore sopra `0.5`.

Quando scrivi dati in DynamoDB utilizzando Hive, assicurati che il numero delle unità di capacità di scrittura sia maggiore del numero di mappatori del cluster. Ad esempio, si consideri un cluster Amazon EMR composto da 10 nodi m1.xlarge. Il tipo di nodo m1.xlarge fornisce 8 attività di mappatura, quindi il cluster avrebbe un totale di 80 mappatori (10 × 8). Se la tabella DynamoDB ha meno di 80 unità di capacità di scrittura, un'operazione di scrittura Hive potrebbe consumare tutta la velocità effettiva di scrittura per tale tabella.

Per determinare il numero di mappatori per i tipi di nodi di Amazon EMR, consulta Configurazione attività nella Guida per gli sviluppatori di Amazon EMR.

Per ulteriori informazioni sui mappatori, consulta Regolazione dei mappatori.

Regolazione dei mappatori

Quando Hive avvia un processo Hadoop, il processo viene elaborato da una o più attività di mappatura. Supponendo che la tabella DynamoDB disponga di capacità di throughput sufficiente, è possibile modificare il numero di mappatori nel cluster, migliorando potenzialmente le prestazioni.

> **Note**
>
> Il numero di attività di mappatura utilizzate in un processo Hadoop è influenzato dalle suddivisioni degli input, dove Hadoop suddivide i dati in blocchi logici. Se Hadoop non esegue un numero sufficiente di suddivisioni di input, le operazioni di scrittura potrebbero non essere in grado di consumare tutta la velocità effettiva di scrittura disponibile nella tabella DynamoDB.

Aumento del numero di mappatori

Ogni mappatore in un Amazon EMR ha una velocità massima di lettura di 1 MiB al secondo. Il numero di mappatori in un cluster dipende dalla dimensione dei nodi nel cluster. Per informazioni sulle dimensioni dei nodi e sul numero di mappatori per nodo, consulta Configurazione attività nella Guida per gli sviluppatori di Amazon EMR.

Se la tabella DynamoDB dispone di un'ampia capacità di throughput per le letture, è possibile provare ad aumentare il numero di mappatori completando una delle seguenti operazioni:

- Aumentare la dimensione dei nodi nel cluster. Ad esempio, se il cluster utilizza i nodi m1.large(tre mappatori per nodo), puoi provare ad eseguire l'aggiornamento ai nodi m1.xlarge (otto mappatori per nodo).
- Aumentare il numero di nodi del cluster. Ad esempio, se si dispone di un cluster a tre nodi di nodi m1.xlarge, disponi di un totale di 24 mappatori disponibili. Se si dovesse raddoppiare la dimensione del cluster, con lo stesso tipo di nodo, si avrebbero 48 mappatori.

È possibile utilizzare la Console di gestione AWS per gestire la dimensione o il numero di nodi nel cluster. Per rendere effettive le modifiche, potrebbe essere necessario riavviare il cluster.

In alternativa, puoi aumentare il numero di mappatori modificando il parametro di configurazione Hadoop `mapred.tasktracker.map.tasks.maximum`. Questo è un parametro Hadoop, non un

parametro Hive. Non pertanto è possibile modificarlo in modo interattivo dal prompt dei comandi. Se si aumenta il valore di `mapred.tasktracker.map.tasks.maximum`, è possibile aumentare il numero di mappatori senza aumentare la dimensione o il numero di nodi. Tuttavia, se si imposta il valore troppo alto, è possibile che i nodi del cluster esauriscano la memoria.

È possibile impostare il valore per `mapred.tasktracker.map.tasks.maximum` come operazione di bootstrap quando si avvia per la prima volta il cluster Amazon EMR. Per ulteriori informazioni, consulta (Facoltativo) Creazione di operazioni di bootstrap per l'installazione di software aggiuntivo nella Guida alla gestione di Amazon EMR.

Diminuzione del numero di mappatori

Se si utilizza l'istruzione `SELECT` per selezionare i dati da una tabella Hive esterna mappata a DynamoDB, il processo Hadoop può utilizzare tutte le attività necessarie, fino al numero massimo di mappatori nel cluster. In questo scenario, è possibile che una query Hive a lunga esecuzione possa utilizzare tutta la capacità di lettura assegnata della tabella DynamoDB, con un impatto negativo su altri utenti.

È possibile utilizzare il parametro `dynamodb.max.map.tasks` per impostare un limite superiore per le attività della mappa:

```
SET dynamodb.max.map.tasks=1
```

Questo valore deve essere maggiore o uguale a 1. Quando Hive elabora la query, il processo Hadoop risultante utilizzerà non più di `dynamodb.max.map.tasks` durante la lettura dalla tabella DynamoDB.

Argomenti aggiuntivi

Di seguito sono riportati alcuni altri modi per ottimizzare le applicazioni che utilizzano Hive per accedere a DynamoDB.

Retry duration (Durata nuovi tentativi)

Per impostazione predefinita, Hive rieseguirà un processo Hadoop se non ha restituito alcun risultato da DynamoDB entro due minuti. È possibile regolare questo intervallo modificando il parametro `dynamodb.retry.duration`:

```
SET dynamodb.retry.duration=2;
```

Il valore deve essere un numero intero diverso da zero, che rappresenta il numero di minuti nell'intervallo di nuovi tentativi. Il valore predefinito per `dynamodb.retry.duration` è 2 (minuti).

Richieste di dati in parallelo

Molteplici richieste di dati, sia da parte di più utenti sia da più applicazioni verso un'unica tabella, possono far esaurire il throughput di lettura assegnato e rallentare le prestazioni.

Durata dei processi

La consistenza dei dati in DynamoDB dipende dall'ordine delle operazioni di lettura e scrittura di ciascun nodo. Quando una query Hive è in avanzamento, un'altra applicazione potrebbe caricare nuovi dati nella tabella DynamoDB oppure modificare o eliminare dati esistenti. In questo caso, i risultati della query Hive potrebbe non riflettere le modifiche effettuate ai dati durante l'esecuzione della query.

Ora delle richieste

Le prestazioni possono essere migliorate pianificando query Hive che accedono a una tabella DynamoDB quando la richiesta nella tabella DynamoDB è minore. Ad esempio, se la maggior parte degli utenti della tua applicazione vive a San Francisco, potresti decidere di esportare i dati giornalieri alle 4:00 PST, quando la maggior parte degli utenti dorme e non aggiorna i record del database DynamoDB.

Integrazione di DynamoDB con Amazon S3

Le funzionalità di importazione ed esportazione di Amazon DynamoDB forniscono un modo semplice ed efficiente per spostare i dati tra le tabelle Amazon S3 e DynamoDB senza scrivere codice.

Le funzionalità di importazione ed esportazione di DynamoDB consentono di spostare, trasformare e copiare gli account di tabelle DynamoDB. È possibile importare da origini S3 ed esportare i dati di tabelle DynamoDB in Amazon S3 e utilizzare i servizi AWS, ad esempio Athena, Amazon SageMaker AI e AWS Lake Formation, per analizzare i dati ed estrarre informazioni di approfondimento. È inoltre possibile importare i dati direttamente nelle nuove tabelle DynamoDB per creare nuove applicazioni con prestazioni in millisecondi a una cifra su larga scala, facilitare la condivisione dei dati tra tabelle e account e semplificare i piani di ripristino di emergenza e continuità aziendale.

Argomenti

- Importazione di dati DynamoDB in Amazon S3: come funziona

- Esportazione dei dati DynamoDB in Amazon S3: come funziona

Importazione di dati DynamoDB in Amazon S3: come funziona

Per importare dati in DynamoDB, i dati devono trovarsi in un bucket Amazon S3 in formato CSV, DynamoDB JSON o Amazon Ion. I dati possono essere compressi in formato ZSTD o GZIP o possono essere importati direttamente in formato non compresso. I dati di origine possono essere un singolo oggetto Amazon S3 o più oggetti Amazon S3 che utilizzano lo stesso prefisso.

I dati verranno importati in una nuova tabella DynamoDB, che verrà creata quando avvii la richiesta di importazione. È possibile creare questa tabella con indici secondari, quindi eseguire query e aggiornare i dati su tutti gli indici primari e secondari non appena l'importazione è stata completata. È inoltre possibile aggiungere una replica di tabella globale al termine dell'importazione.

> **Note**
>
> Durante il processo di importazione di Amazon S3, DynamoDB crea una nuova tabella di destinazione di importazione. L'importazione in tabelle esistenti non è attualmente supportata da questa caratteristica.

L'importazione da Amazon S3 non consuma capacità di scrittura nella nuova tabella, quindi non è necessario effettuare il provisioning di capacità aggiuntiva per l'importazione dei dati in DynamoDB. I prezzi per l'importazione dei dati si basano sulle dimensioni non compresse dei dati di origine in Amazon S3, che vengono elaborati come risultato dell'importazione. Anche gli elementi elaborati ma che non vengono caricati nella tabella a causa della formattazione o di altre incongruenze nei dati di origine vengono fatturati come parte del processo di importazione. Per i dettagli sui prezzi, consulta Prezzi di Amazon DynamoDB.

Puoi importare i dati da un bucket S3 appartenente a un account diverso se disponi delle autorizzazioni corrette per leggere il bucket specifico. La nuova tabella potrebbe anche trovarsi in una regione diversa dal bucket Amazon S3 di origine. Per ulteriori informazioni, consulta la pagina relativa alla configurazione e alle autorizzazioni di Amazon Simple Storage Service.

I tempi di importazione sono direttamente correlati alle caratteristiche dei dati in Amazon S3. Ciò include la dimensione e il formato dei dati, lo schema di compressione, l'uniformità della distribuzione dei dati, il numero di oggetti Amazon S3 e altre variabili correlate. In particolare, i set di dati con chiavi distribuite uniformemente saranno più veloci da importare rispetto ai set di dati disallineati.

Ad esempio, se la chiave e l'indice secondario usano il mese e l'anno per creare partizioni e tutti i dati sono contenuti nel mese di dicembre, l'importazione di questi dati potrebbe richiedere molto più tempo.

Gli attributi associati alle chiavi dovrebbero essere univoci nella tabella di base. Se alcune chiavi non sono univoche, l'importazione sovrascriverà gli elementi associati fino a quando non rimarrà solo l'ultima sovrascrittura. Ad esempio, se la chiave primaria è il mese e più elementi sono impostati sul mese di settembre, ogni nuovo elemento sovrascriverà gli elementi scritti in precedenza e rimarrà solo un elemento con chiave primaria "mese" impostata su settembre. In questi casi, il numero di elementi elaborati nella descrizione della tabella di importazione non corrisponderà al numero di elementi nella tabella di destinazione.

AWS CloudTrail registra tutte le azioni della console e dell'API per l'importazione delle tabelle. Per ulteriori informazioni, consulta Registrazione delle operazioni di DynamoDB con AWS CloudTrail.

Il video seguente è un'introduzione all'importazione diretta da Amazon S3 in DynamoDB.

Importazione da Amazon S3

Argomenti

- Richiesta di importazione di una tabella in DynamoDB
- Formati di importazione Amazon S3 per DynamoDB
- Quote e convalida dei formati di importazione
- Best practice per l'importazione da Amazon S3 in DynamoDB

Richiesta di importazione di una tabella in DynamoDB

L'importazione DynamoDB consente di importare i dati da un bucket Amazon S3 a una nuova tabella DynamoDB. Puoi richiedere l'importazione di una tabella utilizzando la console DynamoDB, la CLI CloudFormationo l'API DynamoDB.

Se si desidera utilizzare il AWS CLI, è necessario prima configurarlo. Per ulteriori informazioni, consulta Accesso a DynamoDB.

> **Note**
>
> - La funzionalità Import Table interagisce con più AWS servizi diversi come Amazon CloudWatch S3 e. Prima di iniziare un'importazione, assicurati che l'utente o il ruolo che

richiama l'importazione APIs disponga delle autorizzazioni per tutti i servizi e le risorse da cui dipende la funzionalità.

- Non modificate gli oggetti Amazon S3 mentre l'importazione è in corso, poiché ciò può causare la non riuscita o l'annullamento dell'operazione.

Per ulteriori informazioni sugli errori e sulla risoluzione dei problemi, consulta Quote e convalida dei formati di importazione.

Argomenti

- Impostazione delle autorizzazioni IAM
- Richiesta di un'importazione utilizzando la Console di gestione AWS
- Ottenere dettagli sulle importazioni passate in Console di gestione AWS
- Richiedere un'importazione utilizzando il AWS CLI
- Ottenere dettagli sulle importazioni passate in AWS CLI

Impostazione delle autorizzazioni IAM

È possibile importare dati da qualsiasi bucket Amazon S3 per il quale si dispone dell'autorizzazione di lettura. Non è necessario che il bucket di origine si trovi nella stessa regione o abbia lo stesso proprietario della tabella di origine. Il tuo AWS Identity and Access Management (IAM) deve includere le azioni pertinenti sul bucket Amazon S3 di origine e le CloudWatch autorizzazioni necessarie per fornire informazioni di debug. Di seguito è riportata una policy di esempio.

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": "*"
    }
  ]
}
```

Autorizzazioni di Amazon S3

Quando si avvia un'importazione in bucket Amazon S3 di origine di proprietà di un altro account, assicurati che il ruolo o l'utente abbia accesso agli oggetti Amazon S3. A tale scopo, esegui il comando Amazon S3 `GetObject` e usa le credenziali. Quando si utilizza l'API, il parametro proprietario del bucket Amazon S3 viene impostato automaticamente sull'ID dell'account dell'utente corrente. Per le importazioni tra account, assicurati che questo parametro sia correttamente popolato

con l'ID dell'account del proprietario del bucket. Il codice seguente è un esempio di policy del bucket S3 nell'account di destinazione.

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

Quando si crea la nuova tabella per l'importazione, se si seleziona una chiave di crittografia a riposo che non è di proprietà di DynamoDB, è necessario fornire AWS KMS le autorizzazioni necessarie per utilizzare una tabella DynamoDB crittografata con chiavi gestite dal cliente. Per ulteriori informazioni, consulta Autorizzazione dell'uso della chiave. AWS KMS Se gli oggetti Amazon S3 sono crittografati con KMS di crittografia lato server (SSE-KMS), assicurati che il ruolo o l'utente che avvia l'importazione abbia accesso alla decrittografia utilizzando la chiave. AWS KMS Questa funzionalità non supporta l'uso di oggetti Amazon S3 crittografati mediante chiavi di crittografia fornite dal cliente (SSE-C).

CloudWatch autorizzazioni

Il ruolo o l'utente che sta avviando l'importazione necessita delle autorizzazioni di creazione e gestione per il gruppo di log e i flussi di log associati all'importazione.

Richiesta di un'importazione utilizzando la Console di gestione AWS

L'esempio seguente mostra come utilizzare la console DynamoDB per importare i dati esistenti in una nuova tabella denominata `MusicCollection`.

Come richiedere l'importazione di una tabella

1. Accedi Console di gestione AWS e apri la console DynamoDB all'indirizzo. https://console.aws.amazon.com/dynamodb/
2. Nel pannello di navigazione sul lato sinistro della console, scegliere Exports to S3 (Esportazioni su S3).
3. Nella pagina visualizzata, selezionare Import from S3 (Importazione da S3).
4. Scegliere Import from S3 (Importazione da S3).
5. In URL S3 di origine, immetti l'URL di origine di Amazon S3.

   Se possiedi il bucket di origine, seleziona Sfoglia S3 per cercarlo. In alternativa, inserisci l'URL del bucket nel seguente formato: `s3://bucket/prefix`. `prefix` è un prefisso della chiave Amazon S3. È un prefisso del nome dell'oggetto Amazon S3 che si desidera importare o il prefisso della chiave condiviso da tutti gli oggetti Amazon S3 che si desidera importare.

   > **Note**
   >
   > Non è possibile utilizzare lo stesso prefisso della richiesta di esportazione DynamoDB. La funzionalità di esportazione crea una struttura di cartelle e file manifesto per tutte le esportazioni. Se si utilizza lo stesso percorso Amazon S3, si verificherà un errore. Invece, indirizza l'importazione verso la cartella che contiene i dati di quella specifica esportazione. Il formato del percorso corretto in questo caso sarà `s3://bucket/prefix/AWSDynamoDB/<XXXXXXXX-XXXXXX>/Data/`, dove `XXXXXXXX-XXXXXX` è l'ID di esportazione. È possibile trovare l'ID di esportazione nell'ARN di esportazione, che ha il seguente formato: `arn:aws:dynamodb:<Region>:<AccountID>:table/<TableName>/export/<XXXXXXXX-XXXXXX>`. Ad esempio, `arn:aws:dynamodb:`*`us-east-1`*`:`*`123456789012`*`:table/`*`ProductCatalog`*`/export/`*`01234567890123-a1b2c3d4`*.

6. Specificare se si è il proprietario nella casella S3 bucket owner (Proprietario bucket S3). Se il bucket di origine è di proprietà di un account diverso, seleziona Un account diverso. AWS Immettere quindi l'ID dell'account del proprietario del bucket.

7. In Import file compression (Compressione file di importazione), selezionare No compression (Nessuna compressione), GZIP o ZSTD a seconda dei casi.
8. Selezionare il formato di file di importazione appropriato. Le opzioni sono DynamoDB JSON, Amazon Ion o CSV. Se si seleziona CSV, saranno disponibili due opzioni aggiuntive:CSV header (Intestazione CSV) e CSV delimiter character (Carattere delimitatore CSV).

   In CSV header (Intestazioni CSV), scegliere se l'intestazione verrà recuperata dalla prima riga del file o sarà personalizzata. Se si seleziona Customize your headers (Personalizza intestazioni), è possibile specificare i valori di intestazione in base ai quali eseguire l'importazione. Le intestazioni CSV specificate mediante questo metodo fanno distinzione tra maiuscole e minuscole e dovrebbero contenere le chiavi della tabella di destinazione.

   In CSV delimiter character (Carattere delimitatore CSV), impostare il carattere usato per separare gli elementi. Per impostazione predefinita è selezionata la virgola. Se si seleziona Custom delimiter character (Carattere delimitatore personalizzato), il delimitatore deve corrispondere al modello regex: `[,;:|\t ]`.
9. Selezionare il pulsante Next (Successivo) e quindi le opzioni per la nuova tabella creata per archiviare i dati.

   > **Note**
   >
   > La chiave primaria e la chiave di ordinamento devono corrispondere agli attributi nel file. In caso contrario, l'importazione avrà esito negativo. Gli attributi rispettano la distinzione tra maiuscole e minuscole.
10. Selezionare di nuovo Next (Successivo) per rivedere le opzioni di importazione, quindi fare clic su Import (Importa) per avviare l'attività di importazione. La nuova tabella verrà elencata nella sezione "Tables" (Tabelle) con lo stato "Creating" (Creazione in corso). A questo punto, la tabella non è accessibile.
11. Una volta completata l'importazione, verrà visualizzato lo stato "Active" (Attivo). A questo punto è possibile iniziare a utilizzare la tabella.

Ottenere dettagli sulle importazioni passate in Console di gestione AWS

Per informazioni sulle attività di importazione eseguite in passato, fai clic su Import from S3 (Importazione da S3) nella barra laterale di navigazione, quindi seleziona la scheda Imports (Importazioni). Il pannello delle importazioni contiene l'elenco di tutte le importazioni create negli

ultimi 90 giorni. Selezionando l'ARN di un'attività elencata nella scheda Imports (Importazioni) verranno recuperate le informazioni relative all'importazione specifica, incluse le impostazioni di configurazione avanzate scelte.

Richiedere un'importazione utilizzando il AWS CLI

L'esempio seguente importa dati in formato CSV da un bucket S3 chiamato bucket con un prefisso di prefisso in una nuova tabella denominata 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**
>
> Se scegli di crittografare l'importazione utilizzando una chiave protetta da AWS Key Management Service (AWS KMS), la chiave deve trovarsi nella stessa regione del bucket Amazon S3 di destinazione.

Ottenere dettagli sulle importazioni passate in AWS CLI

Le informazioni sulle attività di importazione eseguite in passato possono essere recuperate utilizzando il comando `list-imports`. Questo comando restituisce un elenco di tutte le importazioni create negli ultimi 90 giorni. Si noti che, sebbene le attività di importazione scadano dopo 90 giorni e i processi più vecchi non siano più presenti in questo elenco, DynamoDB non elimina nessuno degli oggetti nel bucket Amazon S3 o nella tabella creata durante l'importazione.

```
aws dynamodb list-imports
```

Per recuperare informazioni dettagliate su un'attività di importazione specifica, incluse le impostazioni di configurazione avanzate, utilizza il comando `describe-import`.

```
aws dynamodb describe-import \
    --import-arn arn:aws:dynamodb:us-east-1:123456789012:table/ProductCatalog/exp
```

Formati di importazione Amazon S3 per DynamoDB

DynamoDB può importare dati in tre formati: CSV, DynamoDB JSON e Amazon Ion.

Argomenti

- CSV
- DynamoDB JSON
- Amazon Ion

CSV

Un file in formato CSV è costituito da più elementi delimitati da nuove righe. Per impostazione predefinita, DynamoDB interpreta la prima riga di un file di importazione come intestazione e prevede che le colonne siano delimitate da virgole. È inoltre possibile definire le intestazioni che verranno applicate, purché corrispondano al numero di colonne nel file. Se si definiscono le intestazioni in modo esplicito, la prima riga del file verrà importata come valori.

> **Note**
>
> Durante l'importazione da file CSV, tutte le colonne diverse dall'intervallo hash e dalle chiavi della tabella di base e degli indici secondari vengono importate come stringhe DynamoDB.

Aggiunta di caratteri di escape alle virgolette doppie

Tutte le virgolette doppie presenti nel file CSV devono essere accompagnate da caratteri di escape. In caso contrario, come nell'esempio seguente, l'importazione avrà esito negativo:

```
id,value
"123",Women's Full "Length" Dress
```

Questa stessa importazione avrà esito positivo se invece dei caratteri di escape vengono usati due set di virgolette doppie:

```
id,value
"""123""","Women's Full ""Length"" Dress"
```

Una volta che il testo è stato correttamente preceduto da un carattere di escape e importato, apparirà come appare nel file CSV originale:

```
id,value
"123",Women's Full "Length" Dress
```

DynamoDB JSON

Un file in formato DynamoDB JSON può essere costituito da più oggetti elemento. Ogni singolo oggetto è nel formato JSON di marshalling standard di DynamoDB e le nuove righe vengono utilizzate come delimitatori di elementi. Come caratteristica aggiuntiva, le esportazioni da point-in-time sono supportate come origine di importazione per impostazione predefinita.

> **Note**
>
> Le nuove righe vengono utilizzate come delimitatori di elemento per un file in formato DynamoDB JSON e non devono essere utilizzate all'interno di un oggetto 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 è un formato gerarchico di serializzazione dei dati particolarmente tipizzato, autodescrittivo, creato per le esigenze di sviluppo rapido, disaccoppiamento ed efficienza quotidianamente associate alla progettazione di architetture orientate ai servizi su larga scala.

Quando si importano dati in formato Ion, i tipi di dati Ion vengono mappati ai tipi di dati DynamoDB nella nuova tabella DynamoDB.

N. S.	Conversione dei tipi di dati da Ion a DynamoDB	B
1	`Ion Data Type`	`DynamoDB Representation`
2	`string`	`String (s)`
3	`bool`	`Boolean (BOOL)`
4	`decimal`	`Number (N)`
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`

Gli elementi in un file Ion sono delimitati da nuove righe. Ogni riga inizia con un evidenziatore della versione Ion, seguito da un elemento in formato Ion.

> **Note**
>
> Nell'esempio seguente, sono stati formattati elementi da un file in formato Ion su più righe per migliorare la leggibilità.

```
$ion_1_0
[
  {
    Item:{
      Authors:$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"
    }
  }
]
```

Quote e convalida dei formati di importazione

Quote di importazione

La funzionalità DynamoDB di importazione da S3 può supportare fino a 50 processi di importazione simultanea, con una dimensione totale dell'oggetto di origine di importazione pari a 15 TB alla volta nelle regioni us-east-1, us-west-2 e eu-west-1. In tutte le altre regioni, sono supportate fino a 50

attività di importazione simultanee con una dimensione totale di 1 TB. Ogni processo di importazione può elaborare fino a 50.000 oggetti Amazon S3 in tutte le Regioni. Queste quote predefinite vengono applicate a tutti gli account. Se ritieni di dover modificare le quote, contatta il team del tuo account e la questione viene esaminata caso per caso. Per ulteriori informazioni sui limiti di DynamoDB, consulta Service Quotas.

Errori di convalida

Durante il processo di importazione, DynamoDB potrebbe riscontrare errori durante l'analisi dei dati. Per ogni errore, DynamoDB genera un registro CloudWatch e conserva un conteggio del numero totale di errori riscontrati. Se il formato dell'oggetto Amazon S3 è errato o se il suo contenuto non è in grado di formare un elemento DynamoDB, è possibile saltare l'elaborazione della parte rimanente dell'oggetto.

> **Note**
>
> Se l'origine dei dati di Amazon S3 ha più elementi che condividono la stessa chiave, gli elementi verranno sovrascritti finché non ne rimarrà uno. Potrebbe sembrare che un elemento sia stato importato e che gli altri siano stati ignorati. Gli elementi duplicati verranno sovrascritti in ordine casuale, non vengono conteggiati come errori e non vengono inviati ai file di log di CloudWatch.
>
> Una volta completata l'importazione, puoi vedere il numero totale di elementi importati, il numero totale di errori e il numero totale di elementi elaborati. Per un'ulteriore verifica, puoi anche controllare la dimensione totale degli elementi importati e la dimensione totale dei dati elaborati.

Esistono tre categorie di errori di importazione: errori di convalida API, errori di convalida dei dati ed errori di configurazione.

Errori di convalida API

Gli errori di convalida API sono errori a livello di elemento restituiti dall'API di sincronizzazione. Le cause più comuni sono dovuti a problemi di autorizzazione, parametri obbligatori mancanti ed errori di convalida dei parametri. I dettagli relativi al motivo per cui la chiamata API non è riuscita sono contenuti nelle eccezioni generate dalla richiesta `ImportTable`.

Errori di convalida dei dati

Gli errori di convalida dei dati possono verificarsi a livello di elemento o di file. Durante l'importazione, gli elementi vengono convalidati in base alle regole DynamoDB prima di venire importati nella tabella di destinazione. Quando un elemento non supera la convalida e non viene importato, il processo di importazione salta l'elemento e passa all'elemento successivo. Alla fine del processo, lo stato di importazione è impostato su FAILED (Non riuscito) con un codice FailureCode (codice di errore), ItemValidationError (errore di convalida elemento) e FailureMessage (messaggio di errore) indicante "Some of the items failed validation checks and were not imported. Please check CloudWatch error logs for more details." (Alcuni elementi non hanno superato i controlli di convalida e non sono stati importati. Controlla i registri degli errori CloudWatch per ulteriori dettagli.)

Le cause più comuni degli errori di convalida dei dati includono oggetti non analizzabili, oggetti in un formato errato (l'input specifica DYNAMODB_JSON ma l'oggetto non è in formato DYNAMODB_JSON) e la mancata corrispondenza tra lo schema e le chiavi della tabella di origine specificate.

Errori di configurazione

Gli errori di configurazione sono in genere errori del flusso di lavoro dovuti alla convalida delle autorizzazioni. Il flusso di lavoro di importazione verifica alcune autorizzazioni dopo aver accettato la richiesta. Se si verificano problemi durante la chiamata a una delle dipendenze richieste come Amazon S3 o CloudWatch, il processo contrassegna lo stato di importazione con FAILED (Non riuscito). I codici `failureCode` e `failureMessage` indicano la causa dell'errore. Se applicabile, il messaggio di errore contiene anche l'ID della richiesta che è possibile utilizzare per analizzare il motivo dell'errore in CloudTrail.

Gli errori di configurazione comuni includono URL errati per i bucket Amazon S3 e l'impossibilità di accedere ai bucket Amazon S3, ai registri CloudWatch Logs e alle chiavi AWS KMS utilizzate per decrittografare l'oggetto Amazon S3. Per ulteriori informazioni, consulta l'argomento relativo all'utilizzo e chiavi dati.

Convalida di oggetti Amazon S3 di origine

Per convalidare gli oggetti S3 di origine, esegui questi passaggi.

1. Convalidare il formato dei dati e il tipo di compressione

    - Assicurarsi che tutti gli oggetti Amazon S3 corrispondenti associati al prefisso specificato abbiano lo stesso formato (DYNAMODB_JSON, DYNAMODB_ION, CSV)

- Assicurarsi che tutti gli oggetti Amazon S3 corrispondenti associati al prefisso specificato siano compressi allo stesso modo (GZIP, ZSTD, NONE)

> **Note**
>
> Gli oggetti Amazon S3 non devono avere l'estensione corrispondente (.csv/.json/.ion/.gz/.zstd ecc.) poiché il formato di input specificato nella chiamata ImportTable ha la precedenza.

2. Verificare che i dati di importazione siano conformi allo schema di tabella desiderato

- Assicurarsi che ogni elemento nei dati di origine includa la chiave primaria. Per le importazioni una chiave di ordinamento è facoltativa.
- Accertarsi che il tipo di attributo associato alla chiave primaria e a qualsiasi chiave di ordinamento corrisponda al tipo di attributo nello schema di tabella e nello schema GSI, come specificato nei parametri di creazione della tabella

Risoluzione dei problemi

Log CloudWatch

Per i processi di importazione non riusciti, i messaggi di errore dettagliati vengono pubblicati nei registri CloudWatch. Per accedere a questi registri, recupera prima i valori importArn dall'output e describe-import usando questo comando:

```
aws dynamodb describe-import --import-arn arn:aws:dynamodb:us-east-1:ACCOUNT:table/
target-table/import/01658528578619-c4d4e311
}
```

Output di esempio:

```
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."
    }
}
```

Recupera il gruppo di log e l'ID di importazione dalla risposta precedente e utilizza questi valori per recuperare i registri degli errori. L'ID di importazione è l'ultimo elemento del percorso del campo `ImportArn`. Il nome del gruppo di log è `/aws-dynamodb/imports`. Il nome del flusso di log degli errori è `import-id/error`. Per questo esempio, sarebbe `01658528578619-c4d4e311/error`.

Chiave pk mancante nell'elemento

Se l'oggetto S3 di origine non contiene la chiave primaria specificata come parametro, l'importazione avrà esito negativo, ad esempio quando si definisce la chiave primaria per l'importazione come nome di colonna "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"}'
```

La colonna "pk" non è presente nell'oggetto di origine `import-test.csv` che include i seguenti contenuti:

```
title,artist,year_of_release
The Dark Side of the Moon,Pink Floyd,1973
```

Questa importazione non avrà esito negativo a causa di un errore di convalida dell'elemento perché manca la chiave primaria nell'origine dei dati.

Esempio di registro degli errori CloudWatch:

```
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"
}
```

Questo registro degli errori riporta che uno o più valori di parametro non sono validi e che manca la chiave "pk" nell'elemento. Poiché questo processo di importazione non è riuscito, la tabella "target-table" ora esiste ed è vuota perché non sono stati importati elementi. Il primo elemento è stato elaborato e l'oggetto non ha superato la convalida dell'elemento.

Per risolvere il problema, elimina innanzitutto la tabella "target-table" se non è più necessaria. Usa quindi un nome di colonna chiave primaria esistente nell'oggetto di origine oppure aggiorna i dati di origine nel modo seguente:

```
pk,title,artist,year_of_release
Albums::Rock::Classic::1973::AlbumId::ALB25,The Dark Side of the Moon,Pink Floyd,1973
```

Tabella di destinazione esistente

Se si avvia un processo di importazione e si riceve una risposta come segue:

```
An error occurred (ResourceInUseException) when calling the ImportTable operation:
 Table already exists: target-table
```

Per correggere questo errore, è necessario scegliere un nome di tabella che non esiste già e provare a rieseguire l'importazione.

Il bucket specificato non esiste.

Se il bucket di origine non esiste, l'importazione avrà esito negativo e registrerà i dettagli del messaggio di errore in CloudWatch.

Esempio di descrizione dell'importazione:

```
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` è `S3NoSuchBucket`, con `FailureMessag` contenente dettagli come l'ID della richiesta e il servizio che ha generato l'errore. Poiché l'errore è stato rilevato prima dell'importazione dei dati nella tabella, non viene creata una nuova tabella DynamoDB. In alcuni casi, quando si verificano questi errori dopo l'avvio dell'importazione dei dati, la tabella con i dati parzialmente importati viene conservata.

Per correggere questo errore, assicurati che il bucket Amazon S3 di origine esista e quindi riavvia il processo di importazione.

Best practice per l'importazione da Amazon S3 in DynamoDB

Di seguito sono elencate le best practice per eseguire l'importazione dei dati da Amazon S3 in DynamoDB.

Rimanere al di sotto del limite di 50.000 oggetti S3

Ogni processo di importazione supporta un massimo di 50.000 oggetti S3. Se il set di dati contiene più di 50.000 oggetti, valuta la possibilità di consolidarli in oggetti più grandi.

Evitare oggetti S3 eccessivamente grandi

Gli oggetti S3 vengono importati in parallelo. La presenza di numerosi oggetti S3 di medie dimensioni consente l'esecuzione parallela senza sovraccarichi eccessivi. Per gli articoli inferiori a 1 KB, considera che puoi inserire 4.000.000 elementi in ogni oggetto S3. Se la dimensione media degli elementi è maggiore, inserisci proporzionalmente meno elementi in ogni oggetto S3.

Randomizzare i dati ordinati

Se un oggetto S3 contiene i dati disposti secondo un ordinamento, può creare una partizione hot di implementazione, ossia la situazione in cui tutta l'attività viene ricevuta da una partizione, quindi dalla partizione successiva e così via. I dati ordinati sono definiti come elementi in sequenza nell'oggetto S3 che vengono scritti nella stessa partizione di destinazione durante l'importazione. Una situazione comune di dati ordinati è un file CSV in cui gli elementi sono ordinati per chiave di partizione in modo che gli elementi ripetuti condividano la stessa chiave di partizione.

Per evitare una partizione hot di implementazione, in questi casi si consiglia di randomizzare l'ordine. In tal modo si possono migliorare le prestazioni tramite la distribuzione delle operazioni di scrittura. Per ulteriori informazioni, consulta Distribuzione efficiente dell'attività di scrittura durante il caricamento dei dati in DynamoDB.

Comprimere i dati per mantenere la dimensione totale dell'oggetto S3 al di sotto del limite regionale

Nel processo di importazione da S3 esiste un limite per la dimensione totale dei dati dell'oggetto S3 da importare. Il limite è di 15 TB nelle regioni us-east-1, us-west-2 ed eu-west-1 e di 1 TB in tutte le altre regioni. Il limite si basa sulle dimensioni degli oggetti S3 non elaborati.

La compressione consente di far rientrare nel limite un numero maggiore di dati non elaborati. Se la compressione da sola non è sufficiente a far rientrare l'importazione entro il limite, puoi anche contattare il Supporto AWS Premium per aumentare la quota.

Essere consapevoli di come le dimensioni degli articoli influiscono sulle prestazioni

Se la dimensione media degli elementi è molto piccola (inferiore a 200 byte), il processo di importazione potrebbe richiedere più di tempo rispetto agli elementi di dimensioni maggiori.

Prendere in considerazione l'importazione senza indici secondari globali

La durata di un'operazione di importazione può dipendere dalla presenza di uno o più indici secondari globali (GSI). Se intendi creare indici con chiavi di partizione a bassa cardinalità, è possibile accelerare l'importazione se rimandi la creazione dell'indice al termine dell'operazione di importazione (anziché includerla nel processo di importazione).

> **Note**
>
> La creazione di un GSI durante l'importazione non comporta costi di scrittura, al contrario della creazione di un GSI dopo l'importazione.

Esportazione dei dati DynamoDB in Amazon S3: come funziona

L'esportazione da DynamoDB in S3 è una soluzione completamente gestita per esportare i dati DynamoDB in un bucket Amazon S3 su larga scala. Utilizzando l'esportazione da DynamoDB in S3, puoi esportare dati da una tabella Amazon DynamoDB da qualsiasi momento all'interno della finestra di ripristino (PITR) verso un point-in-time bucket Amazon S3. Devi abilitare PITR sulla tabella per utilizzare la funzionalità di esportazione. Questa funzionalità ti consente di eseguire analisi e query complesse sui tuoi dati utilizzando altri AWS servizi come Athena AWS Glue, Amazon AI, SageMaker Amazon EMR e. AWS Lake Formation

L'esportazione da DynamoDB a S3 consente di esportare dati completi e incrementali dalla tabella DynamoDB. Le esportazioni sono asincrone, non consumano unità di capacità di lettura (RCUs) e non hanno alcun impatto sulle prestazioni e sulla disponibilità delle tabelle. I formati di file di esportazione supportati sono i formati DynamoDB JSON e Amazon Ion. Puoi anche esportare i dati in un bucket S3 di proprietà di un altro AWS account e in un'altra regione. AWS I tuoi dati sono sempre crittografati. end-to-end

Le esportazioni complete di DynamoDB vengono addebitate in base alle dimensioni della tabella DynamoDB (dati della tabella e indici secondari locali) nel point-in-time in cui viene effettuata l'esportazione. Le esportazioni incrementali di DynamoDB vengono addebitate in base alla dimensione dei dati elaborati dai backup continui per il periodo di tempo esportato. L'esportazione incrementale ha un costo minimo di 10 MB. Sono previsti costi aggiuntivi per l'archiviazione dei dati esportati in Amazon S3 e per le richieste PUT effettuate sul bucket Amazon S3. Per ulteriori informazioni su questi costi, consulta Prezzi di Amazon DynamoDB e Prezzi di Amazon S3.

Per informazioni specifiche sulle quote di servizio, consulta Esportazione delle tabelle in Amazon S3.

Argomenti

- Richiesta di esportazione di una tabella in DynamoDB
- Formato di output di esportazione della tabella DynamoDB

Richiesta di esportazione di una tabella in DynamoDB

Le esportazioni di tabelle DynamoDB ti consentono di esportare i dati delle tabelle in un bucket Amazon S3, consentendoti di eseguire analisi e query complesse sui tuoi dati utilizzando altri servizi AWS come Athena, Amazon AI, Amazon AWS Glue EMR e. SageMaker AWS Lake Formation Puoi richiedere l'esportazione di una tabella utilizzando Console di gestione AWS l' AWS CLI API DynamoDB.

> **Note**
>
> Il richiedente paga i bucket Amazon S3 non sono supportati.

DynamoDB supporta sia l'esportazione completa che l'esportazione incrementale:

- Con le esportazioni complete, puoi esportare un'istantanea completa della tabella da qualsiasi momento all'interno della finestra di point-in-time ripristino (PITR) nel tuo bucket Amazon S3.
- Con le esportazioni incrementali, puoi esportare i dati dalla tabella DynamoDB che sono stati modificati, aggiornati o eliminati in un periodo di tempo specificato, all'interno della finestra PITR, nel bucket Amazon S3.

Argomenti

- Prerequisiti

- Richiesta di un'esportazione utilizzando la Console di gestione AWS
- Ottenere dettagli sulle esportazioni passate in Console di gestione AWS
- Richiesta di un'esportazione utilizzando la AWS CLI
- Ottenere dettagli sulle esportazioni passate in AWS CLI
- Richiesta di un'esportazione utilizzando l' AWS SDK
- Ottieni dettagli sulle esportazioni precedenti utilizzando l'SDK AWS

Prerequisiti

Abilitazione del PITR

Per utilizzare la funzionalità di esportazione in S3, è necessario abilitare il PITR sulla tabella. Per dettagli su come abilitare PITR, consulta P recovery. oint-in-time Se richiedi un'esportazione per una tabella che non ha PITR abilitato, la richiesta avrà esito negativo e verrà visualizzato un messaggio di eccezione: «Si è verificato un errore (PointInTimeRecoveryUnavailableException) durante la chiamata dell'`ExportTableToPointInTime`operazione: Il ripristino in tempo reale non è abilitato per la tabella '». my-dynamodb-table È possibile effettuare richieste ed esportare solo da un punto temporale all'interno del PITR `RecoveryPeriodInDays` configurato.

Impostazione delle autorizzazioni S3

È possibile esportare i dati della tabella in qualsiasi bucket Amazon S3 per cui si ha l'autorizzazione di scrittura. Non è necessario che il bucket di destinazione si trovi nella stessa AWS regione o abbia lo stesso proprietario del proprietario della tabella di origine. La tua policy AWS Identity and Access Management (IAM) deve consentirti di eseguire le azioni S3 (`s3:AbortMultipartUploads3:PutObject`, `es3:PutObjectAcl`) e l'azione di esportazione di DynamoDB (). `dynamodb:ExportTableToPointInTime` Ecco una di policy di esempio che concederà all'utente le autorizzazioni per eseguire esportazioni in un bucket 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/*"
        }
    ]
}
```

Se si necessita di scrivere su un bucket Amazon S3 che si trova in un altro account o non si dispone delle autorizzazioni di scrittura, il proprietario del bucket Amazon S3 dovrà aggiungere una policy di bucket che consenta di esportare da DynamoDB verso quel bucket. Ecco un esempio di policy sul bucket Amazon S3 di destinazione.

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/*"
        }
    ]
}
```

La revoca di queste autorizzazioni mentre è in corso un'esportazione comporterà la creazione di file parziali.

> **Note**
>
> Se la tabella o il bucket che stai esportando sono crittografati con chiavi gestite dal cliente, è necessario che le policy della chiave KMS forniscano a DynamoDB l'autorizzazione a usarla. Questa autorizzazione viene concessa tramite l'IAM User/Role che attiva il processo di esportazione. Per ulteriori informazioni sulla crittografia, comprese le best practice, consulta gli articoli relativi a come DynamoDB utilizza AWS KMS e all'utilizzo di una chiave KMS personalizzata.

Richiesta di un'esportazione utilizzando la Console di gestione AWS

Nell'esempio seguente viene mostrato come utilizzare la console DynamoDB per esportare una tabella esistente denominata `MusicCollection`.

> **Note**
>
> Questa procedura presuppone che tu abbia abilitato point-in-time il ripristino. Per abilitarla per la `MusicCollection` tabella, nella scheda Panoramica della tabella, nella sezione Dettagli della tabella, scegli Abilita il oint-in-timeripristino P.

Come richiedere l'esportazione di una tabella

1. Accedi Console di gestione AWS e apri la console DynamoDB all'indirizzo. https://console.aws.amazon.com/dynamodb/
2. Nel riquadro di navigazione sul lato sinistro della console, scegli Exports to S3 (Esportazioni su S3).
3. Seleziona il pulsante Esporta in S3.
4. Scegli una tabella di origine e un bucket S3 di destinazione. Se il bucket di destinazione è di proprietà dell'account, è possibile utilizzare il pulsante Browse S3 (Sfoglia S3) per trovarlo. In caso contrario, immetti l'URL del bucket utilizzando il `s3://bucketname/prefix format.` . **`prefix`** è una cartella facoltativa che consente di mantenere organizzato il bucket di destinazione.

5. Scegli Esportazione completa o Esportazione incrementale. Un'esportazione completa restituisce lo snapshot dell'intera tabella così com'era nel point-in-time specificato. Un'esportazione incrementale restituisce le modifiche apportate alla tabella durante il periodo di esportazione specificato. L'output viene compattato in modo tale da contenere solo lo stato finale dell'elemento relativo al periodo di esportazione. L'elemento verrà visualizzato solo una volta nell'esportazione anche se presenta più aggiornamenti nello stesso periodo di esportazione.

   Full export

   1. Seleziona il point-in-time di cui desideri esportare lo snapshot completo della tabella. Questo può avvenire in qualsiasi point-in-time all'interno della finestra PITR. In alternativa, puoi selezionare Ora corrente per esportare lo snapshot più recente.
   2. Per Formato di file esportato, scegli tra DynamoDB JSON e Amazon Ion. Per impostazione predefinita, la tabella verrà esportata in formato DynamoDB JSON dall'ultima ora ripristinabile nella finestra di ripristino point-in-time (PITR) e sarà crittografata utilizzando una chiave Amazon S3 (SSE-S3). Se necessario, puoi modificare queste impostazioni di esportazione.

      > **Note**
      >
      > Se scegli di crittografare l'esportazione utilizzando una chiave protetta da AWS Key Management Service (AWS KMS), la chiave deve trovarsi nella stessa regione del bucket S3 di destinazione.

   Incremental export

   1. Seleziona il Periodo di esportazione di cui desideri esportare i dati incrementali. Scegli un'ora di inizio nella finestra PITR. La durata del periodo di esportazione deve essere di almeno 15 minuti e non superiore a 24 ore. L'ora di inizio del periodo di esportazione è inclusa e l'ora di fine è esclusa.
   2. Scegli tra Modalità assoluta o Modalità relativa.
      a. Modalità assoluta esporterà i dati incrementali per il periodo di tempo specificato.
      b. Modalità relativa esporterà i dati incrementali per un periodo di esportazione relativo all'ora di invio del processo di esportazione.

3. Per Formato di file esportato, scegli tra DynamoDB JSON e Amazon Ion. Per impostazione predefinita, la tabella verrà esportata in formato DynamoDB JSON dall'ultima ora ripristinabile nella finestra di ripristino point-in-time (PITR) e sarà crittografata utilizzando una chiave Amazon S3 (SSE-S3). Se necessario, puoi modificare queste impostazioni di esportazione.

   > **Note**
   >
   > Se scegli di crittografare l'esportazione utilizzando una chiave protetta da AWS Key Management Service (AWS KMS), la chiave deve trovarsi nella stessa regione del bucket S3 di destinazione.

4. Per Esporta il tipo di visualizzazione, seleziona Nuove e vecchie immagini o Solo nuove immagini. La nuova immagine fornisce lo stato più recente dell'elemento. La vecchia immagine mostra lo stato dell'elemento subito prima della "data e ora di inizio" specificate. L'impostazione predefinita è Nuove e vecchie immagini. Per ulteriori informazioni su nuove e vecchie immagini, consulta Output di esportazione incrementale.

6. Seleziona Esporta per iniziare.

I dati esportati non sono coerenti a livello di transazione. Le operazioni di transazione possono essere suddivise tra due output di esportazione. Un sottoinsieme di elementi modificati da un'operazione di transazione può riflettersi nell'esportazione, mentre un altro sottoinsieme di modifiche nella stessa transazione non si riflette nella stessa richiesta di esportazione. Tuttavia, le esportazioni presentano tutte una consistenza finale. Se una transazione viene interrotta durante un'esportazione, la restante transazione sarà presente nella prossima esportazione contigua, senza duplicati. I periodi di tempo utilizzati per le esportazioni si basano su un orologio di sistema interno e possono variare di un minuto rispetto all'orologio locale dell'applicazione.

Ottenere dettagli sulle esportazioni passate in Console di gestione AWS

Per informazioni sui processi di esportazione eseguiti in passato, seleziona Esportazioni in S3 nella barra laterale di navigazione. Questa scheda contiene un elenco di tutte le esportazioni create negli ultimi 90 giorni. Seleziona l'ARN di un processo elencato nella scheda Esportazioni per recuperare le informazioni relative all'esportazione, incluse le impostazioni di configurazione avanzate scelte. Notare che, sebbene i metadati delle attività di esportazione scadano dopo 90 giorni e i processi più vecchi non siano più presenti in questo elenco, gli oggetti nel bucket S3 rimangono finché le

relative policy di bucket lo consentono. DynamoDB non elimina mai alcun oggetto creato nel bucket S3 durante un'esportazione.

Richiesta di un'esportazione utilizzando la AWS CLI

L'esempio seguente mostra come utilizzare per AWS CLI esportare una tabella esistente denominata `MusicCollection` in un bucket S3 chiamato. `ddb-export-musiccollection`

> **Note**
>
> Questa procedura presuppone che tu abbia abilitato il ripristino. point-in-time Per abilitarlo per la tabella `MusicCollection`, emettere il comando seguente.
>
> ```
> aws dynamodb update-continuous-backups \
>     --table-name MusicCollection \
>     --point-in-time-recovery-specification PointInTimeRecoveryEnabled=True
> ```

Full export

Il comando esporta `MusicCollection` in un bucket S3 denominato `ddb-export-musiccollection-9012345678` con un prefisso `2020-Nov`. I dati della tabella verranno esportati in formato DynamoDB JSON da un momento specifico nella finestra di ripristino point-in-time (PITR) e vengono crittografati utilizzando una chiave Amazon S3 (SSE-S3).

> **Note**
>
> Se si richiede l'esportazione di una tabella tra più account, assicurarsi di includere l'opzione `--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

Il comando seguente esegue un'esportazione incrementale fornendo un nuovo `--export-type` e `--incremental-export-specification`. Sostituisci i tuoi valori con qualsiasi cosa in corsivo. I tempi sono specificati in secondi dall'epoca.

```
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**
>
> Se scegli di crittografare l'esportazione utilizzando una chiave protetta da AWS Key Management Service (AWS KMS), la chiave deve trovarsi nella stessa regione del bucket S3 di destinazione.

Ottenere dettagli sulle esportazioni passate in AWS CLI

Le informazioni sulle richieste di esportazione eseguite in passato possono essere recuperate utilizzando il comando `list-exports`. Questo comando restituisce un elenco di tutte le esportazioni create negli ultimi 90 giorni. Notare che, sebbene i metadati dei processi di esportazione scadano dopo 90 giorni e i processi più vecchi non siano restituiti dal comando `list-exports`, gli oggetti nel bucket S3 rimangono finché le relative policy di bucket lo consentono. DynamoDB non elimina mai alcun oggetto creato nel bucket S3 durante un'esportazione.

Lo stato delle esportazioni è `PENDING` fino a quando non hanno esito positivo o negativo. In caso di esito positivo, lo stato cambierà in `COMPLETED`. In caso di esito negativo, lo stato passerà a `FAILED` con `failure_message` e `failure_reason`.

Nell'esempio seguente viene utilizzato il parametro `table-arn` facoltativo per elencare solo le esportazioni di una tabella specifica.

```
aws dynamodb list-exports \
```

```
    --table-arn arn:aws:dynamodb:us-east-1:123456789012:table/ProductCatalog
```

Per recuperare informazioni dettagliate su un processo di esportazione specifico, incluse le impostazioni di configurazione avanzate, utilizza il comando `describe-export`.

```
aws dynamodb describe-export \
    --export-arn arn:aws:dynamodb:us-east-1:123456789012:table/ProductCatalog/
export/01234567890123-a1b2c3d4
```

Richiesta di un'esportazione utilizzando l' AWS SDK

Usa questi frammenti di codice per richiedere l'esportazione di una tabella utilizzando l' AWS SDK di tua scelta.

Python

Esportazione 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'
)
```

Esportazione incrementale

```
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'
)
```

Ottieni dettagli sulle esportazioni precedenti utilizzando l'SDK AWS

Usa questi frammenti di codice per ottenere dettagli sulle esportazioni di tabelle precedenti utilizzando l' AWS SDK di tua scelta.

Python

Elenco

```
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',
  )
)
```

Descriva

```
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 di output di esportazione della tabella DynamoDB

Un'esportazione di tabelle DynamoDB, oltre ai file contenenti i dati della tabella, include file manifesto. Tutti questi file vengono salvati nel bucket Amazon S3 specificato nella richiesta di esportazione. Nelle sezioni seguenti sono descritti il formato e il contenuto di ciascun oggetto di output.

Output di esportazione completo

File manifesto

Per ogni richiesta di esportazione, oltre ai file checksum, DynamoDB crea file manifesto nel bucket S3 specificato.

```
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
```

Quando si richiede l'esportazione di una tabella viene scelto un parametro **`export-prefix`**. Questo ti aiuta a mantenere organizzati i file nel bucket S3 di destinazione. L'**`ExportId`** è un token univoco generato dal servizio per garantire che più esportazioni nello stesso bucket S3 e `export-prefix` non si sovrascrivano l'un l'altro.

L'esportazione crea anche almeno 1 file per partizione. Per le partizioni vuote, la richiesta di esportazione creerà un file vuoto. Tutti gli elementi di ogni file provengono dal keyspace con hash di quella particolare partizione.

> **Note**
>
> DynamoDB crea anche un file vuoto denominato `_started` nella stessa directory dei file manifesto. Questo file verifica che il bucket di destinazione sia scrivibile e che l'esportazione sia iniziata. Può essere tranquillamente eliminato.

Il manifesto di riepilogo

Il file `manifest-summary.json` contiene le informazioni di riepilogo del processo di esportazione. Ciò consente di sapere quali file di dati nella cartella di dati condivisa sono associati a questa esportazione. Il formato è il seguente:

```
{
   "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"
}
```

I file manifesto

Il file `manifest-files.json` contiene informazioni sui file contenenti i dati della tabella esportati. Il file ha un formato JSON lines, pertanto le nuove righe vengono utilizzate come delimitatori di elementi. Nell'esempio seguente, i dettagli di un file di dati dai file manifesto vengono formattati su più righe per motivi di leggibilità.

```
{
```

```
"itemCount": 8,
   "md5Checksum": "sQMSpEILNgoQmarvDFonGQ==",
   "etag": "af83d6f217c19b8b0fff8023d8ca4716-1",
   "dataFileS3Key": "AWSDynamoDB/01693685827463-2d8752fd/data/asdl123dasas.json.gz"
}
```

File di dati

DynamoDB può esportare i dati della tabella in due formati: DynamoDB JSON e Amazon Ion. Indipendentemente dal formato scelto, i dati verranno scritti in più file compressi denominati dalle chiavi. Anche questi file sono elencati nel file `manifest-files.json`.

La struttura di directory del bucket Amazon S3 dopo un'esportazione completa conterrà tutti i file manifesto e i file di dati nella cartella Export ID.

```
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

Un'esportazione di tabella in formato DynamoDB JSON è costituita da più oggetti `Item`. Ogni singolo oggetto è nel formato JSON di marshalling standard di DynamoDB.

Quando si creano parser personalizzati per i dati di esportazione in DynamoDB JSON, il formato generato è JSON lines. Ciò significa che le nuove righe saranno utilizzate come delimitatori di elementi. Numerosi servizi AWS, come Athena e AWS Glue, analizzeranno automaticamente questo formato.

Nell'esempio seguente, un singolo elemento da un'esportazione DynamoDB JSON è stato formattato su più righe per motivi di leggibilità.

```
{
```

```
    "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 è un formato gerarchico di serializzazione dei dati particolarmente tipizzato, autodescrittivo, creato per le esigenze di sviluppo rapido, disaccoppiamento ed efficienza quotidianamente associate alla progettazione di architetture orientate ai servizi su larga scala. DynamoDB supporta l'esportazione dei dati delle tabelle in formato di testo di Ion, che è un superset di JSON.

Quando si esporta una tabella in formato Ion, i tipi di dati DynamoDB utilizzati nella tabella vengono mappati ai tipi di dati Ion. I set DynamoDB usano annotazioni del tipo Ionper disambiguare il tipo di dati utilizzato nella tabella di origine.

La tabella seguente elenca la mappatura dei tipi di dati DynamoDB con tipi di dati Ion:

Tipo di dati DynamoDB	Rappresentazione Ion
String (S)	string
Boolean (BOOL)	bool
Number (N)	decimal
Binary (B)	blob
Set (SS, NS, BS)	list (con annotazione del tipo $dynamodb_SS, $dynamodb_NS o $dynamodb_BS)
Elenco	list
Map	struct

Gli elementi in un'esportazione Ion sono delimitati da nuove righe. Ogni riga inizia con un evidenziatore della versione Ion, seguito da un elemento in formato Ion. Nell'esempio seguente, un singolo elemento da un'esportazione DynamoDB JSON è stato formattato su più righe per motivi di leggibilità.

```
$ion_1_0 {
    Item:{
        Authors:$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"
    }
```

```
}
```

Output di esportazione incrementale

File manifesto

Per ogni richiesta di esportazione, oltre ai file checksum, DynamoDB crea file manifesto nel bucket S3 specificato.

```
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
```

Quando si richiede l'esportazione di una tabella viene scelto un parametro **`export-prefix`**. Questo ti aiuta a mantenere organizzati i file nel bucket S3 di destinazione. L'**`ExportId`** è un token univoco generato dal servizio per garantire che più esportazioni nello stesso bucket S3 e `export-prefix` non si sovrascrivano l'un l'altro.

L'esportazione crea anche almeno 1 file per partizione. Per le partizioni vuote, la richiesta di esportazione creerà un file vuoto. Tutti gli elementi di ogni file provengono dal keyspace con hash di quella particolare partizione.

> **Note**
>
> DynamoDB crea anche un file vuoto denominato `_started` nella stessa directory dei file manifesto. Questo file verifica che il bucket di destinazione sia scrivibile e che l'esportazione sia iniziata. Può essere tranquillamente eliminato.

Il manifesto di riepilogo

Il file `manifest-summary.json` contiene le informazioni di riepilogo del processo di esportazione. Ciò consente di sapere quali file di dati nella cartella di dati condivisa sono associati a questa esportazione. Il formato è il seguente:

```
{
 "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"
}
```

I file manifesto

Il file `manifest-files.json` contiene informazioni sui file contenenti i dati della tabella esportati. Il file ha un formato JSON lines, pertanto le nuove righe vengono utilizzate come delimitatori di elementi. Nell'esempio seguente, i dettagli di un file di dati dai file manifesto vengono formattati su più righe per motivi di leggibilità.

```
{
"itemCount": 8,
   "md5Checksum": "sQMSpEILNgoQmarvDFonGQ==",
   "etag": "af83d6f217c19b8b0fff8023d8ca4716-1",
   "dataFileS3Key": "AWSDynamoDB/data/sgad6417s6vss4p7owp0471bcq.json.gz"
}
```

File di dati

DynamoDB può esportare i dati della tabella in due formati: DynamoDB JSON e Amazon Ion. Indipendentemente dal formato scelto, i dati verranno scritti in più file compressi denominati dalle chiavi. Anche questi file sono elencati nel file `manifest-files.json`.

I file di dati per le esportazioni incrementali sono tutti contenuti in una cartella di dati comune nel bucket S3. I file manifesto si trovano nella cartella Export ID.

```
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
    #   ...
```

Nei file di esportazione, l'output di ogni elemento include un timestamp che indica quando quell'elemento è stato aggiornato nella tabella e una struttura di dati che indica se si è trattato di un'operazione `insert`, `update` o `delete`. Il timestamp si basa su un orologio di sistema interno e può differire dall'orologio dell'applicazione. Per le esportazioni incrementali, puoi scegliere tra due tipi di visualizzazione di esportazione per la struttura di output: immagini nuove e vecchie o solo immagini nuove.

- La nuova immagine fornisce lo stato più recente dell'elemento
- La vecchia immagine mostra lo stato dell'elemento subito prima della data e dell'ora di inizio specificate

I tipi di visualizzazione possono essere utili se desideri vedere come è stato modificato l'elemento durante il periodo di esportazione. Possono anche essere utili per aggiornare in modo efficiente i sistemi downstream, specialmente se tali sistemi hanno una chiave di partizione diversa dalla chiave di partizione di DynamoDB.

È possibile dedurre se un elemento dell'output di esportazione incrementale era `insert`, `update` o `delete` osservando la struttura dell'output. La struttura di esportazione incrementale e le operazioni corrispondenti sono riepilogate nella tabella seguente per entrambi i tipi di visualizzazione di esportazione.

Operazione	Solo nuove immagini	Immagini nuove e vecchie
Insert	Chiavi + nuova immagine	Chiavi + nuova immagine
Aggiornamento	Chiavi + nuova immagine	Chiave + nuova immagine + vecchia immagine
Eliminazione	Chiavi	Chiavi + vecchia immagine
Insert + delete	Nessun output	Nessun output

DynamoDB JSON

Un'esportazione di tabelle in formato DynamoDB JSON è costituita da un timestamp dei metadati che indica il tempo di scrittura dell'elemento, seguito dalle chiavi dell'elemento e dai valori. Di seguito viene illustrato un esempio di output JSON di DynamoDB che utilizza il tipo di visualizzazione di esportazione come Immagini nuove e vecchie.

```
// 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 è un formato gerarchico di serializzazione dei dati particolarmente tipizzato, autodescrittivo, creato per le esigenze di sviluppo rapido, disaccoppiamento ed efficienza quotidianamente associate alla progettazione di architetture orientate ai servizi su larga scala. DynamoDB supporta l'esportazione dei dati delle tabelle in formato di testo di Ion, che è un superset di JSON.

Quando si esporta una tabella in formato Ion, i tipi di dati DynamoDB utilizzati nella tabella vengono mappati ai tipi di dati Ion. I set DynamoDB usano annotazioni del tipo Ionper disambiguare il tipo di dati utilizzato nella tabella di origine.

La tabella seguente elenca la mappatura dei tipi di dati DynamoDB con tipi di dati Ion:

Tipo di dati DynamoDB	Rappresentazione Ion
String (S)	string
Boolean (BOOL)	bool
Number (N)	decimal
Binary (B)	blob
Set (SS, NS, BS)	list (con annotazione del tipo $dynamodb_SS, $dynamodb_NS o $dynamodb_BS)
Elenco	list
Map	struct

Gli elementi in un'esportazione Ion sono delimitati da nuove righe. Ogni riga inizia con un evidenziatore della versione Ion, seguito da un elemento in formato Ion. Nell'esempio seguente, un singolo elemento da un'esportazione DynamoDB JSON è stato formattato su più righe per motivi di leggibilità.

```
$ion_1_0 {
    Record:{
        Keys:{
             ISBN:"333-3333333333"
         },
        Metadata:{
            WriteTimestampMicros:1684374845117899.
        },
        OldImage:{
            Authors:$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"
        }
    }
}
```

Integrazione Zero-ETL di DynamoDB con Amazon SageMaker Lakehouse

L'integrazione Zero-ETL di DynamoDB con Amazon SageMaker Lakehouse elimina la necessità di creare pipeline personalizzate per lo spostamento dei dati replicando automaticamente i dati DynamoDB su Amazon SageMaker Lakehouse. Questa integrazione no code aiuta i clienti a eseguire carichi di lavoro di analisi sui propri dati DynamoDB con Amazon SageMaker Lakehouse senza utilizzare la capacità delle tabelle DynamoDB. L'integrazione esporta automaticamente i dati dalla tabella e mantiene aggiornata la destinazione, in genere entro 15 – 30 minuti.

Argomenti

- Integrazione Zero-ETL di DynamoDB con Amazon SageMaker Lakehouse

Integrazione Zero-ETL di DynamoDB con Amazon SageMaker Lakehouse

La configurazione di un'integrazione tra la tabella DynamoDB e Amazon SageMaker Lakehouse presenta dei prerequisiti, come la configurazione dei ruoli IAM, che AWS Glue utilizza per accedere ai dati dall'origine e scrivere sulla destinazione, e l'uso di chiavi KMS per crittografare i dati nella posizione intermedia o di destinazione.

Argomenti

- Prerequisiti per creare un'integrazione Zero-ETL di DynamoDB con Amazon SageMaker Lakehouse
- Creazione di un'integrazione Zero-ETL di DynamoDB con Amazon SageMaker Lakehouse
- Visualizzazione di metriche CloudWatch per l'integrazione

Prerequisiti per creare un'integrazione Zero-ETL di DynamoDB con Amazon SageMaker Lakehouse

Per configurare un'integrazione Zero-ETL con un'origine DynamoDB, è necessario impostare una policy di accesso basato su risorse (RBAC) che consenta a AWS Glue di accedere ai dati della tabella DynamoDB ed esportarli. La policy dovrebbe includere autorizzazioni specifiche come `ExportTableToPointInTime`, `DescribeTable` e `DescribeExport` con condizioni che limitino l'accesso a un Account AWS e a una Regione specifici. Per ulteriori informazioni, consulta Configurazione di un'origine Amazon DynamoDB.

È necessario abilitare il recupero point-in-time (PITR) per la tabella ed è possibile applicare la policy utilizzando i comandi AWS CLI. La policy può essere ulteriormente perfezionata specificando l'ARN di integrazione completo per un controllo degli accessi più restrittivo. Per ulteriori informazioni, consulta Prerequisites for setting up a zero-ETL integration.

Creazione di un'integrazione Zero-ETL di DynamoDB con Amazon SageMaker Lakehouse

Una volta completata la configurazione dei prerequisiti per l'integrazione è possibile creare, modificare o eliminare l'integrazione Zero-ETL seguendo le istruzioni riportate di seguito:

Creazione di un'integrazione

Per creare un'integrazione

1. Esegui l'accesso alla Console di gestione AWS e apri la console Amazon DynamoDB all'indirizzo https://console.aws.amazon.com/dynamodbv2.
2. Nel riquadro di navigazione, scegli Integrazioni.
3. Seleziona Crea un'integrazione Zero-ETL con Amazon SageMaker Lakehouse, quindi seleziona Successivo.
4. Per creare un'integrazione, consulta Creating an integration.
5. Per modificare un'integrazione, consulta Modifying an integration.
6. Per eliminare un'integrazione, consulta Deleting an integration.
7. Per configurare un'integrazione tra account, consulta Setting up cross-account integration.

Abilitazione della compattazione su tabelle Amazon S3 di destinazione

È possibile abilitare la compattazione per migliorare le prestazioni delle query in Amazon Athena.

Innanzitutto, completa la configurazione dei prerequisiti per le risorse di compattazione, inclusa la configurazione del ruolo IAM necessario. Consulta la documentazione di Lake Formation per la procedura dettagliata di configurazione dei ruoli IAM. Consulta Optimizing tables for compaction.

Per abilitare la compattazione sulla tabella AWS Glue creata durante l'integrazione, segui la procedura di abilitazione della compattazione di Lake Formation. Ciò contribuirà a ottimizzare le prestazioni della tabella e l'efficienza delle query.

Visualizzazione di metriche CloudWatch per l'integrazione

Una volta completata l'integrazione, è possibile visualizzare notifiche su metriche CloudWatch ed EventBridge generate nell'account per ciascun processo AWS Glue. Per ulteriori informazioni, consulta Monitoring and integration.

Integrazione Zero-ETL di Amazon DynamoDB con il Servizio OpenSearch di Amazon

Amazon DynamoDB offre un'integrazione Zero-ETL con il Servizio OpenSearch di Amazon tramite il plug-in DynamoDB per OpenSearch Ingestion. Amazon OpenSearch Ingestion offre un'esperienza completamente gestita e no code per l'importazione di dati nel Servizio OpenSearch di Amazon.

Con il plugin DynamoDB per OpenSearch Ingestion, è possibile utilizzare una o più tabelle DynamoDB come origine per l'inserimento in uno o più indici del Servizio OpenSearch. È possibile sfogliare e configurare le proprie pipeline di OpenSearch Ingestion con DynamoDB come origine da OpenSearch Ingestion o DynamoDB Integrations nella Console di gestione AWS.

- Nozioni di base su OpenSearch Ingestion sono consultabili nella pagina OpenSearch Ingestion getting started guide.
- I prerequisiti e tutte le opzioni di configurazione per il plugin DynamoDB sono reperibili nella pagina relativa alla documentazione del plugin DynamoDB plugin per OpenSearch Ingestion.

Come funziona

Il plug-in utilizza l'esportazione da DynamoDB in Amazon S3 per creare uno snapshot iniziale da caricare in OpenSearch. Dopo il caricamento dello snapshot, il plugin utilizza i flussi DynamoDB per replicare eventuali ulteriori modifiche quasi in tempo reale. Ogni elemento viene elaborato come evento in OpenSearch Ingestion e può essere modificato con i plugin del processore. È possibile eliminare gli attributi o creare attributi compositi e inviarli a diversi indici tramite percorsi.

È necessario che il recupero point-in-time (PITR) sia abilitato per utilizzare l'esportazione su Amazon S3. È inoltre necessario che i flussi DynamoDB siano abilitati (con l'opzione Immagini nuove e vecchie selezionata) per poterli utilizzare. È possibile creare una pipeline senza bisogno di uno snapshot escludendo le impostazioni di esportazione.

È possibile anche creare una pipeline con solo uno snapshot e nessun aggiornamento escludendo le impostazioni dei flussi. Il plugin non utilizza il throughput di lettura o scrittura sulla tabella, quindi è sicuro da usare senza influire sul traffico di produzione. Esistono dei limiti al numero di consumer paralleli su un flusso che dovrebbero essere considerati prima di creare questa o altre integrazioni. Per altre considerazioni, consulta the section called "Best practice per l'integrazione".

Per pipeline semplici, una singola unità di calcolo OpenSearch (OCU, OpenSearch Compute Unit) è in grado di elaborare circa 1 MB di scrittura al secondo, ossia l'equivalente di circa 1000 unità di richiesta di scrittura (WCU). A seconda della complessità della pipeline e di altri fattori, è possibile anche un risultato superiore.

OpenSearch Ingestion supporta una coda DLQ per gli eventi che causano errori irreversibili. Inoltre, la pipeline può riprendere da dove era stata interrotta senza l'intervento dell'utente anche in caso di interruzione del servizio con DynamoDB, la pipeline o il Servizio OpenSearch di Amazon.

Se l'interruzione si protrae per più di 24 ore, ciò può causare la perdita degli aggiornamenti. Tuttavia, la pipeline continuerà a elaborare gli aggiornamenti che erano ancora disponibili una volta ripristinata la disponibilità. È necessaria una nuova creazione dell'indice per correggere eventuali irregolarità dovute agli eventi eliminati, a meno che non si trovino nella coda DLQ.

Per tutte le impostazioni e i dettagli del plugin, consulta la documentazione del plugin DynamoDB plugin per OpenSearch Ingestion.

Esperienza di creazione integrata tramite la console

DynamoDB e il Servizio OpenSearch offrono un'esperienza integrata nella Console di gestione AWS, che semplifica il processo di avvio. Dopo aver eseguito questa procedura, il servizio selezionerà automaticamente il blueprint DynamoDB e aggiungerà le informazioni DynamoDB appropriate al caso.

Per creare un'integrazione, segui OpenSearch Ingestion getting started guide. Alla Fase 3 relativa alla creazione di una pipeline, sostituisci le Fasi 1 e 2 con le seguenti:

1. Vai alla console DynamoDB.

2. Nel pannello di navigazione a sinistra, seleziona Integrazione.
3. Seleziona la tabella DynamoDB da replicare su OpenSearch.
4. Scegli Create (Crea).

Da qui è possibile continuare con il resto del tutorial.

Passaggi successivi

Per una migliore comprensione di come DynamoDB si integra con il Servizio OpenSearch, consulta quanto segue:

- Getting started with Amazon OpenSearch Ingestion
- DynamoDB plugin configuration and requirements

Gestione delle modifiche sostanziali all'indice

OpenSearch può aggiungere dinamicamente nuovi attributi all'indice. Tuttavia, dopo aver impostato il modello di mappatura per una determinata chiave, sarà necessario intraprendere ulteriori azioni per modificarlo. Inoltre, se la modifica richiede la rielaborazione di tutti i dati nella tabella DynamoDB, è necessario adottare misure per avviare una nuova esportazione.

> **Note**
>
> In tutte queste opzioni, potrebbe comunque essere possibile riscontrare problemi se la tabella DynamoDB presenta conflitti di tipo con il modello di mappatura specificato. Assicurati che sia abilitata una coda DLQ (anche in fase di sviluppo). Ciò consente di individuare più facilmente eventuali problemi nel record che causano un conflitto al momento dell'indicizzazione nell'indice su OpenSearch.

Argomenti

- Come funziona
- Eliminazione dell'indice e reimpostazione della pipeline (opzione incentrata sulla pipeline)
- Ricreazione dell'indice e reimpostazione della pipeline (opzione incentrata sull'indice)
- Creazione di un nuovo indice e di un nuovo sink (opzione online)
- Best practice per evitare i conflitti di tipo ed eseguirne il debug

Come funziona

Ecco una rapida panoramica delle azioni intraprese durante la gestione delle modifiche sostanziali all'indice. Consulta le procedure dettagliate nelle sezioni seguenti.

- Arresto e avvio della pipeline: questa opzione ripristina lo stato della pipeline e la pipeline verrà riavviata con una nuova esportazione completa. Non è distruttiva, quindi non elimina l'indice o i dati in DynamoDB. Se non si crea un nuovo indice prima di procedere con questa opzione, si potrebbe riscontrare un numero elevato di errori dovuti ai conflitti di versione perché l'esportazione tenta di inserire nell'indice documenti più vecchi rispetto alla `_version` corrente. È possibile ignorare questi errori senza correre rischi. Non verrà addebitato il costo della pipeline durante il suo arresto.
- Aggiornamento della pipeline: questa opzione aggiorna la configurazione nella pipeline con un approccio blu/verde, senza perdere alcuno stato. Se si apportano modifiche significative alla pipeline (ad esempio aggiungendo nuovi percorsi, indici o chiavi agli indici esistenti), potrebbe essere necessario reimpostare completamente la pipeline e ricreare l'indice. Questa opzione non esegue un'esportazione completa.
- Eliminazione e ricreazione dell'indice: questa opzione rimuove i dati e le impostazioni di mappatura dall'indice. È necessario eseguire questa operazione prima di apportare modifiche sostanziali alle mappature. Questa procedura interromperà tutte le applicazioni che si basano sull'indice finché l'indice non verrà ricreato e sincronizzato. L'eliminazione dell'indice non avvia una nuova esportazione. Sarebbe opportuno eliminare l'indice solo dopo aver aggiornato la pipeline. Altrimenti, l'indice potrebbe venire ricreato prima di aggiornare le impostazioni.

Eliminazione dell'indice e reimpostazione della pipeline (opzione incentrata sulla pipeline)

Questo metodo è spesso l'opzione più veloce se si è ancora in fase di sviluppo. Si procederà a eliminare l'indice nel Servizio OpenSearch, quindi a interrompere e avviare la pipeline per iniziare una nuova esportazione di tutti i dati. Ciò garantisce che i modelli di mappatura non entrino in conflitto con gli indici esistenti e che non si verifichino perdite di dati da una tabella elaborata incompleta.

1. Arresta la pipeline tramite la Console di gestione AWS o utilizzando l'operazione API StopPipeline con la AWS CLI o un SDK.
2. Aggiorna la configurazione della pipeline con le nuove modifiche.
3. Elimina l'indice nel Servizio OpenSearch, tramite una chiamata REST API o la dashboard di OpenSearch.

4. Avvia la pipeline tramite la console o utilizzando l'operazione API `StartPipeline` con la AWS CLI o un SDK.

   > **Note**
   >
   > Ciò avvia una nuova esportazione completa, che comporterà costi aggiuntivi.

5. Monitora eventuali problemi imprevisti perché viene generata una nuova esportazione per creare il nuovo indice.
6. Verifica che l'indice corrisponda alle aspettative nel Servizio OpenSearch.

Una volta completata l'esportazione e ripresa la lettura dal flusso, i dati della tabella DynamoDB saranno ora disponibili nell'indice.

Ricreazione dell'indice e reimpostazione della pipeline (opzione incentrata sull'indice)

Questo metodo funziona bene se è necessario eseguire molte iterazioni sulla progettazione dell'indice nel Servizio OpenSearch prima di riprendere la pipeline da DynamoDB. Può essere utile per lo sviluppo quando si desidera iterare molto rapidamente sui modelli di ricerca ed evitare di attendere il completamento di nuove esportazioni tra ogni iterazione.

1. Arresta la pipeline tramite Console di gestione AWS o chiamando l'operazione API StopPipeline con la AWS CLI o un SDK.
2. Elimina e ricrea l'indice in OpenSearch con il modello di mappatura che desideri utilizzare. È possibile inserire manualmente alcuni dati di esempio per verificare che le ricerche funzionino come previsto. Se i dati di esempio potrebbero entrare in conflitto con i dati di DynamoDB, assicurati di eliminarli prima di passare alla fase successiva.
3. Se è presente un modello di indicizzazione nella pipeline, rimuovilo o sostituiscilo con quello già creato nel Servizio OpenSearch. Assicurati che il nome dell'indice corrisponda al nome nella pipeline.
4. Avvia la pipeline tramite console o chiamando l'operazione API `StartPipeline` con la AWS CLI o un SDK.

   > **Note**
   >
   > Ciò avvierà una nuova esportazione completa, che comporterà costi aggiuntivi.

5. Monitora eventuali problemi imprevisti perché viene generata una nuova esportazione per creare il nuovo indice.

Una volta completata l'esportazione e ripresa la lettura dal flusso, i dati della tabella DynamoDB saranno ora disponibili nell'indice.

Creazione di un nuovo indice e di un nuovo sink (opzione online)

Questo metodo funziona bene se è necessario aggiornare il modello di mappatura ma attualmente si utilizza l'indice in produzione. Questo crea un indice completamente nuovo, sul quale sarà necessario spostare l'applicazione dopo la sincronizzazione e la convalida.

> **Note**
>
> Questa opzione creerà un altro consumer nel flusso. Ciò può rappresentare un problema se sono presenti anche altri consumer come AWS Lambda o le tabelle globali. Potrebbe essere necessario sospendere gli aggiornamenti della pipeline esistente per creare la capacità necessaria per caricare il nuovo indice.

1. Crea una nuova pipeline con nuove impostazioni e un nome di indice diverso.
2. Monitora il nuovo indice per eventuali problemi imprevisti.
3. Esegui la transizione dell'applicazione verso il nuovo indice.
4. Arresta ed elimina la vecchia pipeline dopo aver verificato che tutto funzioni correttamente.

Best practice per evitare i conflitti di tipo ed eseguirne il debug

- Utilizza sempre una coda DLQ per semplificare il debug in caso di conflitti di tipo.
- Usa sempre un modello di indice con mappature e imposta `include_keys`. Sebbene il Servizio OpenSearch mappi dinamicamente le nuove chiavi, ciò può causare problemi con comportamenti imprevisti (ad esempio aspettativa che qualcosa sia un `GeoPoint`, ma viene creato come una `string` o un `object`) o errori (come avere un valore `number` che è un misto di `long` e `float`).
- Se c'è bisogno di mantenere l'indice esistente funzionante in produzione, è possibile anche sostituire una delle precedenti fasi di eliminazione dell'indice semplicemente rinominando l'indice nel file di configurazione della pipeline. In questo modo viene creato un indice completamente

nuovo. L'applicazione dovrà quindi essere aggiornata in modo che punti al nuovo indice una volta completata.

- In caso di problemi di conversione dei tipi correggibili con un processore, è possibile testare questa opzione con `UpdatePipeline`. A tale scopo, sarà necessario interrompere e riavviare o elaborare le code DLQ per correggere eventuali documenti precedentemente ignorati che contenevano errori.

Best practice per utilizzare l'integrazione Zero-ETL di DynamoDB e del Servizio OpenSearch

DynamoDB dispone di un'integrazione Zero-ETL con il Servizio OpenSearch di Amazon. Per ulteriori informazioni, consulta il plug-in DynamoDB per OpenSearch Ingestion e la pagina specific best practices for Amazon OpenSearch Service.

Configurazione

- Indicizza solo i dati su cui devi eseguire ricerche. Usa sempre un modello di mappatura (`template_type: index_template` e `template_content`) e `include_keys` per implementare questa opzione.
- Monitora i log per verificare la presenza di errori correlati ai conflitti di tipo. Il Servizio OpenSearch si aspetta che tutti i valori di una determinata chiave abbiano lo stesso tipo. Genera eccezioni in caso di mancata corrispondenza. Se si verifica uno di questi errori, è possibile aggiungere un processore per verificare che una determinata chiave abbia sempre lo stesso valore.
- In genere, utilizza il valore di metadati `primary_key` per il valore `document_id`. Nel Servizio OpenSearch, l'ID del documento è l'equivalente della chiave primaria in DynamoDB. L'utilizzo della chiave primaria faciliterà la ricerca del documento e garantirà che gli aggiornamenti vengano replicati in modo coerente senza conflitti.

  È possibile utilizzare la funzione helper `getMetadata` per ottenere la chiave primaria (ad esempio, `document_id: "${getMetadata('primary_key')}"`). Se si utilizza una chiave primaria composita, la funzione helper si occuperà di concatenare le chiavi insieme.
- In generale, utilizza il valore di metadati `opensearch_action` per l'impostazione `action`. Ciò garantirà che gli aggiornamenti vengano replicati in modo tale che i dati nel Servizio OpenSearch corrispondano allo stato più recente di DynamoDB.

  È possibile utilizzare la funzione helper `getMetadata` per ottenere la chiave primaria (ad esempio, `action: "${getMetadata('opensearch_action')}"`). È possibile anche visualizzare il tipo

di evento di flusso attraverso `dynamodb_event_name` per casi d'uso come il filtraggio. Tuttavia, in genere non sarebbe da utilizzare per l'impostazione `action`.

Osservabilità

- Usa sempre una coda DLQ sui sink OpenSearch per gestire gli eventi eliminati. DynamoDB è generalmente meno strutturato del Servizio OpenSearch ed è sempre possibile che accada qualcosa di inaspettato. Con una coda DLQ è possibile ripristinare singoli eventi e persino automatizzare il processo di recupero. Questo aiuterà a evitare di dover ricostruire l'intero indice.
- Imposta sempre avvisi perché il ritardo di replica non superi la quantità prevista. Di norma, un minuto è un valore sicuro che evita notifiche eccessive. Questo può variare a seconda dell'intensità del traffico di scrittura e delle impostazioni dell'unità di calcolo OpenSearch (OCU, OpenSearch Compute Unit) sulla pipeline.

  Se il ritardo di replica supera le 24 ore, il flusso inizierà a ridurre gli eventi e si riscontreranno problemi di precisione, a meno di non eseguire una ricostruzione completa dell'indice da zero.

Dimensionamento

- Usa il dimensionamento automatico per le pipeline per aumentare o ridurre verticalmente le OCU per adattarle al meglio al carico di lavoro.
- Per le tabelle con throughput allocato senza dimensionamento automatico, si consiglia di impostare le OCU in base alle unità di capacità di scrittura (WCU, Write Capacity Units) divise per 1000. Imposta il minimo su 1 OCU al di sotto di tale quantità (ma almeno su 1) e imposta il massimo su almeno 1 OCU al di sopra di tale quantità.
  - Formula:

    ```
    OCU_minimum = GREATEST((table_WCU / 1000) - 1, 1)
    OCU_maximum = (table_WCU / 1000) + 1
    ```

  - Esempio: nella tabella sono state allocate 25000 WCU. Le OCU della pipeline devono essere impostate con un minimo di 24 (25000/1000 – 1) e un massimo di almeno 26 (25000/1000 + 1).
- Per le tabelle con throughput allocato con dimensionamento automatico, si consiglia di impostare le OCU in base alle WCU minime e massime, divise per 1000. Imposta il minimo su 1 OCU al di sotto del minimo di DynamoDB e imposta il massimo su almeno 1 OCU al di sopra del massimo di DynamoDB.

- Formula:

```
OCU_minimum = GREATEST((table_minimum_WCU / 1000) - 1, 1)
OCU_maximum = (table_maximum_WCU / 1000) + 1
```

- Esempio: la tabella ha una policy di dimensionamento automatico con un minimo di 8000 e un massimo di 14000. Le OCU della pipeline devono essere impostate con un minimo di 7 (8000/1000 – 1) e un massimo di 15 (14000/1000 + 1).

- Per le tabelle con throughput on demand, si consiglia di impostare le OCU in base al picco e alla valle tipici per le unità di richiesta di scrittura al secondo. Potrebbe essere necessario calcolare la media su un periodo di tempo più lungo, a seconda dell'aggregazione disponibile. Imposta il minimo su 1 OCU al di sotto del minimo di DynamoDB e imposta il massimo su almeno 1 OCU al di sopra del massimo di DynamoDB.
  - Formula:

```
# 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
```

  - Esempio: la tabella ha una valle media di 300 unità di richiesta di scrittura al secondo e un picco medio di 4300. Le OCU della pipeline devono essere impostate con un minimo di 1 (300/1000 – 1, ma almeno 1) e un massimo di almeno 5 (4300/1000 + 1).
- Segui le best practice per scalare gli indici del Servizio OpenSearch di destinazione. Se gli indici sono sottodimensionati, l'acquisizione da DynamoDB rallenterà e potrebbe causare ritardi.

> **Note**
>
> `GREATEST` è una funzione SQL che, in base a una serie di argomenti, restituisce l'argomento con il valore massimo.

Integrazione di DynamoDB con Amazon EventBridge

Amazon DynamoDB offre i flussi DynamoDB per il Change Data Capture, consentendo l'acquisizione di modifiche a livello di elemento nelle tabelle DynamoDB. I flussi DynamoDB possono invocare funzioni Lambda per elaborare tali modifiche, permettendo l'integrazione basata sugli eventi con altri

servizi e applicazioni. Questo servizio supporta anche il filtraggio, che consente un'elaborazione degli eventi efficiente e mirata.

I flussi DynamoDB supportano fino a due consumer simultanei per shard, nonché operazioni di filtro tramite il filtraggio di eventi Lambda, così che vengano elaborati solo gli elementi che soddisfano criteri specifici. Alcuni clienti potrebbero avere l'esigenza di supportare più di due consumer, altri potrebbero dover arricchire gli eventi di modifica prima dell'elaborazione o utilizzare un filtraggio e un instradamento più avanzati.

L'integrazione di DynamoDB con EventBridge può supportare tali requisiti.

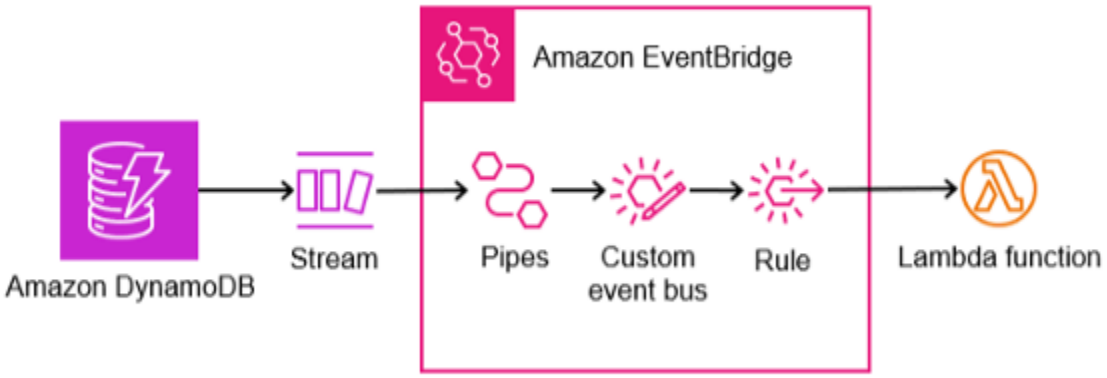
Amazon EventBridge è un servizio serverless che utilizza gli eventi per connettere tra loro i componenti delle applicazioni, semplificando la creazione di applicazioni scalabili basate sugli eventi. EventBridge offre un'integrazione nativa con Amazon DynamoDB tramite EventBridge Pipes, permettendo un flusso di dati ottimizzato da DynamoDB a un router EventBridge. Tale router può quindi essere distribuito in modalità fan-out a più applicazioni e servizi attraverso una serie di regole e destinazioni.

Argomenti

- Come funziona
- Creazione di un'integrazione tramite la console
- Passaggi successivi

Come funziona

L'integrazione tra DynamoDB ed EventBridge Pipes sfrutta i flussi DynamoDB per acquisire una sequenza cronologica di modifiche a livello di elemento in una tabella DynamoDB. Ogni record acquisito in questo modo contiene i dati modificati nella tabella.



Una pipe EventBridge utilizza gli eventi provenienti dai flussi DynamoDB e li instrada verso una destinazione come un router EventBridge (un router di eventi è un router che riceve eventi e li invia a destinazioni, dette anche target). La distribuzione avviene in base alle regole che rispecchiano il contenuto dell'evento. La pipe include anche la possibilità facoltativa di filtrare eventi specifici e di eseguire arricchimenti sui dati degli eventi prima di inviarli alla destinazione.

Sebbene EventBridge supporti più tipi di destinazioni, una scelta comune per l'implementazione di una progettazione di tipo fan-out consiste nell'utilizzare una funzione Lambda come destinazione. L'esempio seguente illustra un'integrazione con una funzione Lambda come destinazione.

Creazione di un'integrazione tramite la console

Segui la procedura riportata di seguito per creare un'integrazione tramite la Console di gestione AWS.

1. Abilita i flussi DynamoDB nella tabella di origine seguendo le istruzioni riportate nella sezione Abilitazione di un flusso della Guida per gli sviluppatori di DynamoDB. Se i flussi DynamoDB sono già abilitati nella tabella di origine, verifica che attualmente siano presenti meno di due consumer. I consumer potrebbero essere funzioni Lambda, tabelle globali DynamoDB, integrazioni Zero-ETL di Amazon DynamoDB con il Servizio OpenSearch di Amazon o applicazioni che eseguono operazioni di lettura direttamente dai flussi, ad esempio tramite l'adattatore Kinesis per i flussi DynamoDB.
2. Crea un router di eventi EventBridge seguendo la procedura riportata nella sezione Creazione di un router di eventi Amazon EventBridge della Guida per l'utente di EventBridge.
   a. Durante la creazione del router di eventi abilita Individuazione schema.

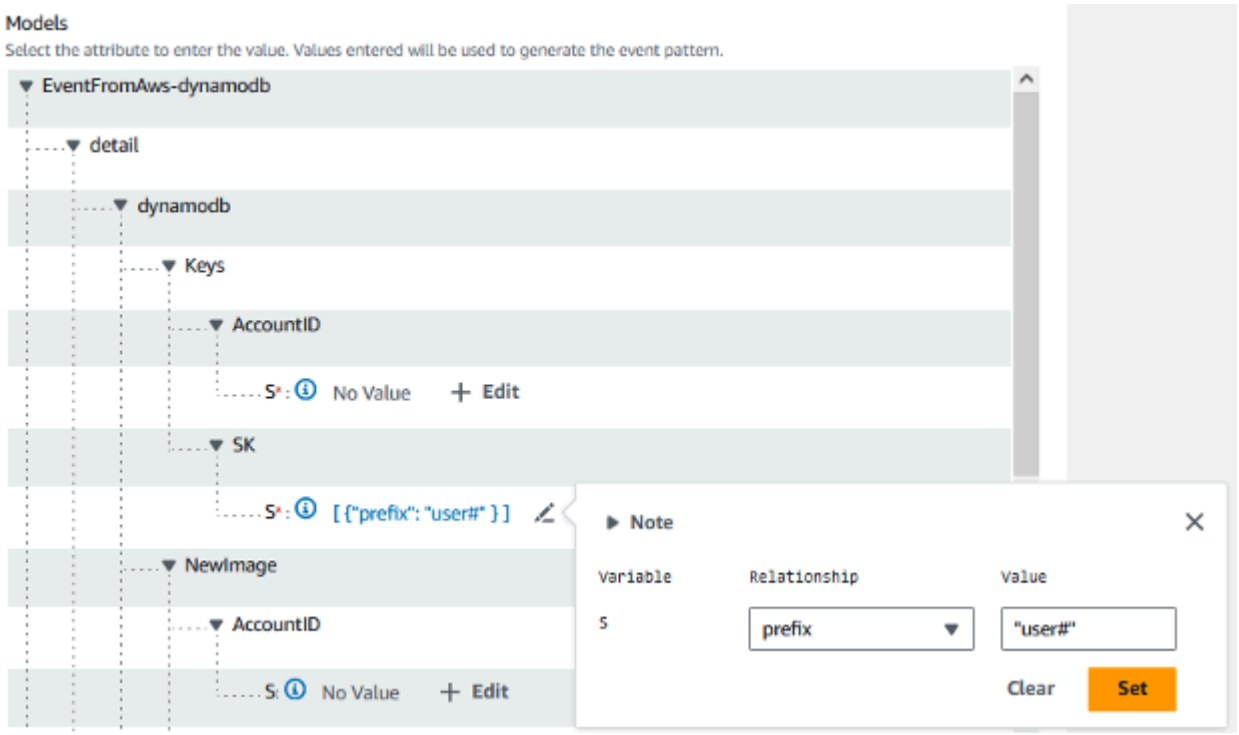
3. Crea una pipe EventBridge seguendo le istruzioni riportate nella sezione Creazione di una pipe Amazon EventBridge della Guida per l'utente di EventBridge.
   a. Durante la configurazione dell'origine, nel campo Origine seleziona DynamoDB e nel campo Flussi DynamoDB seleziona il nome del flusso della tabella di origine.
   b. Durante la configurazione della destinazione, nel campo Servizio di destinazione seleziona Router di eventi EventBridge e nel campo Bus di eventi come destinazione seleziona il router di eventi creato nella fase 2.
4. Scrivi un elemento di esempio nella tabella DynamoDB di origine per attivare un evento, così che EventBridge possa dedurre lo schema dall'elemento stesso. Questo schema può essere utilizzato per creare regole per l'instradamento degli eventi. Ad esempio, se è in corso l'implementazione di un modello di progettazione che prevede l'overload degli attributi, si potrebbero voler attivare regole diverse a seconda del valore della chiave di ordinamento. I dettagli sulla scrittura di un elemento in DynamoDB sono disponibili nella sezione Utilizzo di elementi e attributi della Guida per gli sviluppatori di DynamoDB.
5. Seguendo la procedura riportata nella sezione Compilazione di funzioni Lambda con Python della Guida per gli sviluppatori di Lambda, crea una funzione Lambda di esempio in Python da utilizzare come destinazione. Durante la creazione della funzione è possibile utilizzare il codice di esempio riportato di seguito per illustrare l'integrazione. Una volta invocata, la funzione stamperà le rappresentazioni `NewImage` e `OldImage` ricevute con l'evento e queste risulteranno visibili in CloudWatch Logs.

```
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. Crea una regola EventBridge che instraderà gli eventi sulla nuova funzione Lambda seguendo la procedura riportata nella sezione Creazione di una regola che reagisce agli eventi della Guida per l'utente di EventBridge.

a. Durante la definizione dei dettagli della regola seleziona il nome del router di eventi creato nella fase 2 come Router di eventi.

b. Durante la creazione del modello di eventi, segui la guida per l'opzione Schema esistente. Qui è possibile selezionare il registro discovered-schemas e lo schema individuato per l'evento. In questo modo sarà possibile configurare un modello di eventi specifico per il caso d'uso, che instrada solo i messaggi corrispondenti ad attributi specifici. Ad esempio, se si desidera creare una corrispondenza solo sugli elementi DynamoDB dove la chiave di ordinamento (SK) inizia con "`user#`", verrà utilizzata una configurazione come questa.



c. Una volta terminata la progettazione del modello in base allo schema, fai clic su Genera un modello di eventi in JSON. Se invece desideri creare una corrispondenza con tutti gli eventi che compaiono nei flussi DynamoDB, usa il seguente codice JSON per il modello di eventi.

```
{
  "source": ["aws.dynamodb"]
}
```

d. Per selezionare le destinazioni segui la guida per il servizio AWS. Nel campo Seleziona una destinazione scegli "Funzione Lambda". Nel campo Funzione seleziona la funzione Lambda creata nella fase 5.

7. È ora possibile interrompere l'individuazione degli schemi sul router di eventi seguendo la procedura riportata nella sezione Starting or stopping schema discovery on event buses della Guida per l'utente di EventBridge.
8. Scrivi un secondo elemento di esempio nella tabella DynamoDB di origine per attivare un evento. Verifica che l'evento sia stato elaborato correttamente in ogni fase.
   a. Visualizza la metrica CloudWatch PutEventsApproximateSuccessCount per il router di eventi seguendo la sezione Monitoraggio di Amazon EventBridge della Guida per l'utente di EventBridge.
   b. Visualizza i log delle funzioni per la funzione Lambda seguendo la sezione Monitoraggio e risoluzione dei problemi delle funzioni Lambda della Guida per gli sviluppatori di Lambda. Se la funzione Lambda utilizza il codice di esempio fornito, nel gruppo di log di CloudWatch Logs dovrebbero risultare stampate e visibili le rappresentazioni `NewImage` e `OldImage` provenienti dai flussi DynamoDB.
   c. Visualizza la metrica Numero di errori e percentuale di successo (%) per la funzione Lambda seguendo la sezione Monitoraggio e risoluzione dei problemi delle funzioni Lambda della Guida per gli sviluppatori di Lambda.

Passaggi successivi

In questo esempio viene illustrata un'integrazione di base con una singola funzione Lambda come destinazione. Per una migliore comprensione di configurazioni più complesse, come la creazione di più regole, la configurazione di più destinazioni, l'integrazione con altri servizi e l'arricchimento degli eventi, consulta la Guida per l'utente completa di EventBridge: Nozioni di base su EventBridge.

> **Note**
>
> Tieni presenti le quote EventBridge potenzialmente pertinenti all'applicazione in questione. Sebbene la capacità dei flussi DynamoDB scali in base alla tabella, le quote EventBridge sono separate. Quelle più comuni da tenere in considerazione in un'applicazione di grandi dimensioni sono la Soglia di limitazione (della larghezza di banda della rete) per le invocazioni, espressa in transazioni al secondo e la Soglia di limitazione (della larghezza di banda della rete) di PutEvents, espressa in transazioni al secondo. Queste quote specificano il numero di invocazioni che possono essere inviate alle destinazioni e il numero di eventi che possono essere inseriti nel router al secondo.

Integrazione di DynamoDB con Streaming gestito da Amazon per Apache Kafka

Streaming gestito da Amazon per Apache Kafka (Amazon MSK) semplifica l'acquisizione e l'elaborazione dei dati in streaming in tempo reale con un servizio completamente gestito per Apache Kafka e ad alta disponibilità.

Apache Kafka è un datastore distribuito ottimizzato per l'inserimento e l'elaborazione dei dati in streaming in tempo reale. Kafka è in grado di elaborare flussi di record, archiviarli in modo efficace nell'ordine in cui sono stati generati i record e pubblicare e sottoscrivere flussi di record.

Grazie a queste funzionalità, Apache Kafka viene spesso utilizzato per creare pipeline di dati in streaming in tempo reale. Una pipeline di dati elabora e sposta i dati in modo affidabile da un sistema all'altro e può essere una parte importante dell'adozione di una strategia di database dedicati, facilitando l'uso di più database, ciascuno dei quali supporta casi d'uso diversi.

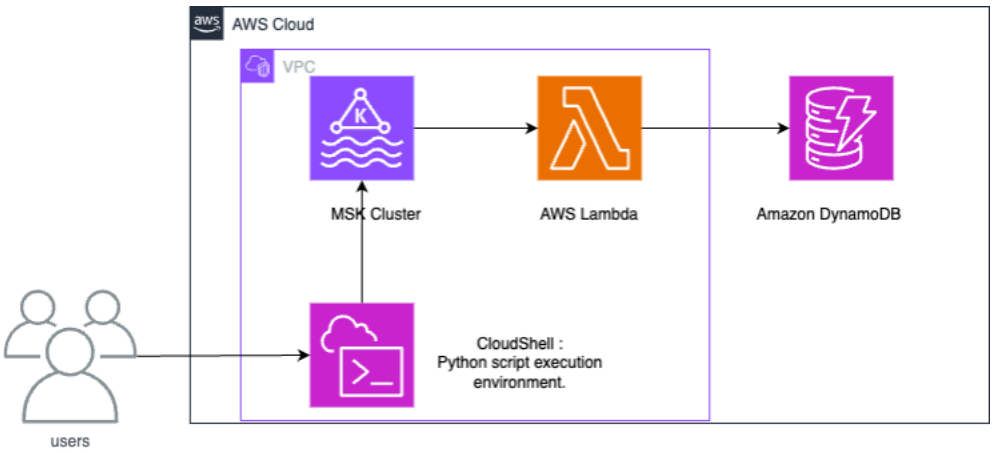
Amazon DynamoDB è l'obiettivo comune di queste pipeline di dati per supportare applicazioni che utilizzano modelli di dati chiave-valore o documentali e desiderano una scalabilità illimitata con prestazioni costanti con risposte in pochi millisecondi.

Argomenti

- Come funziona
- Configurazione di un'integrazione tra Amazon MSK e DynamoDB
- Passaggi successivi

Come funziona

Un'integrazione tra Amazon MSK e DynamoDB utilizza una funzione Lambda per utilizzare i record da Amazon MSK e scriverli su DynamoDB.



Lambda esegue internamente il polling di nuovi messaggi da Amazon MSK, quindi invoca in modo sincrono la funzione Lambda di destinazione. Il payload degli eventi della funzione Lambda contiene batch di messaggi da Amazon MSK. Per l'integrazione tra Amazon MSK e DynamoDB, la funzione Lambda scrive questi messaggi su DynamoDB.

Configurazione di un'integrazione tra Amazon MSK e DynamoDB

> **Note**
>
> È possibile scaricare le risorse utilizzate in questo esempio nel seguente repository GitHub.

La procedura seguente mostra come configurare un esempio di integrazione tra Amazon MSK e Amazon DynamoDB. L'esempio rappresenta i dati generati da dispositivi Internet delle cose (IoT) e inseriti in Amazon MSK. Man mano che i dati vengono inseriti in Amazon MSK, possono essere integrati con servizi di analisi o strumenti di terze parti compatibili con Apache Kafka, rendendo possibili vari casi d'uso di analisi. L'integrazione di DynamoDB fornisce anche la ricerca dei valori delle chiavi dei record dei singoli dispositivi.

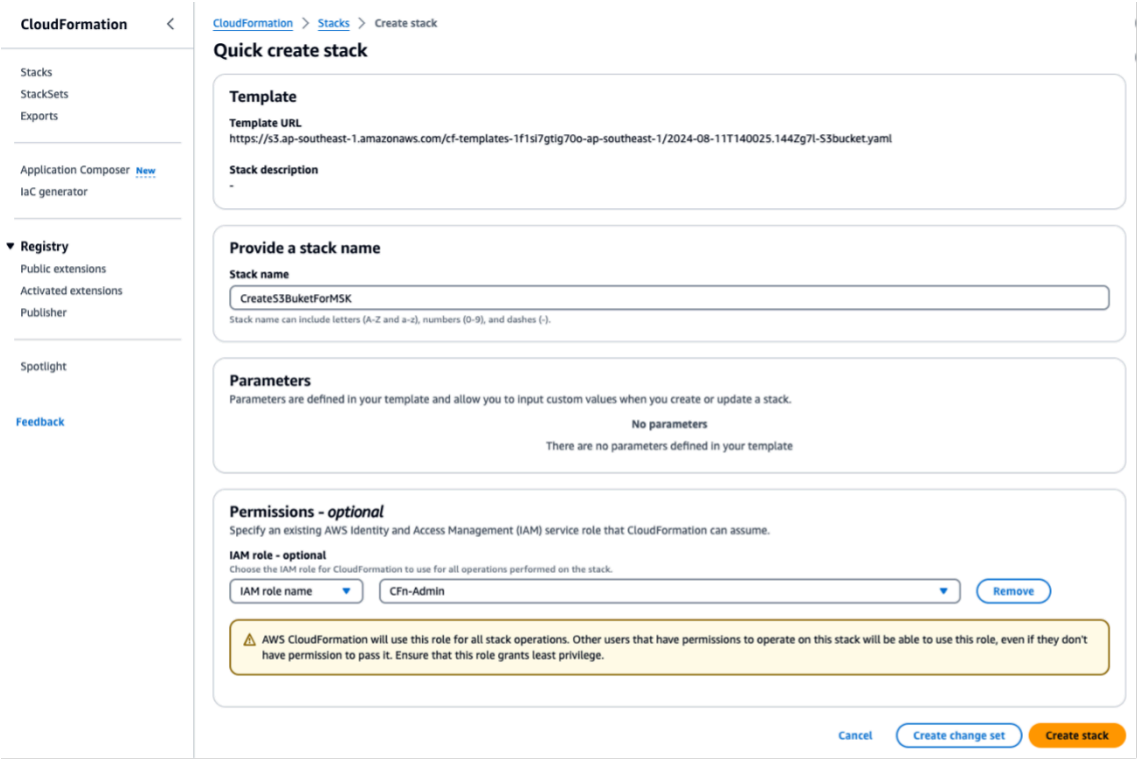
Questo esempio dimostrerà come uno script Python scrive i dati di sensori IoT su Amazon MSK. Quindi, una funzione Lambda scrive elementi con la chiave di partizione "`deviceid`" in DynamoDB.

Il modello CloudFormation fornito creerà le seguenti risorse: un bucket Amazon S3, un Amazon VPC, un cluster Amazon MSK e un AWS CloudShell per testare le operazioni sui dati.

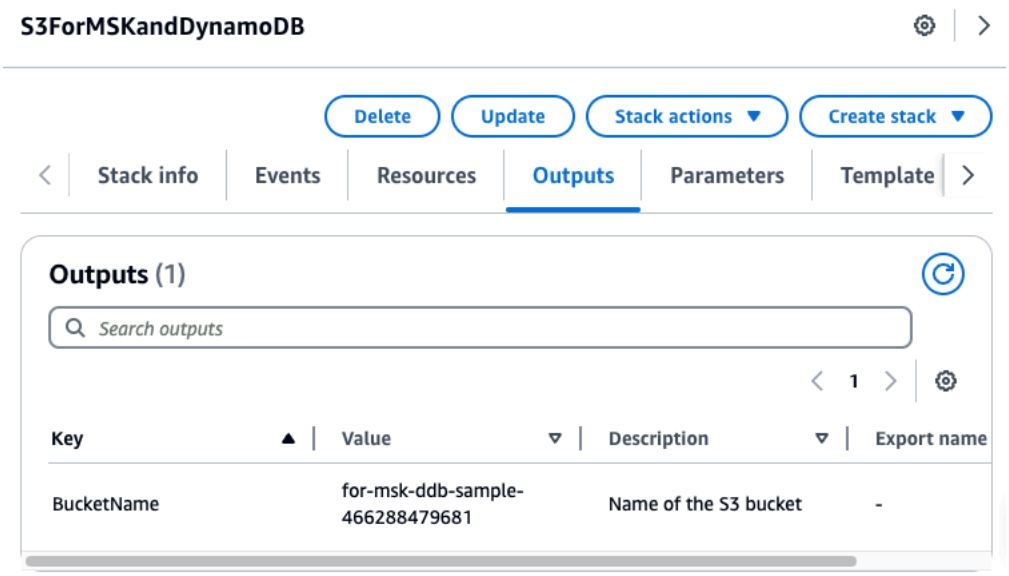
Per generare dati di test, crea un argomento Amazon MSK e quindi crea una tabella DynamoDB. È possibile utilizzare Session Manager dalla console di gestione per accedere al sistema operativo di CloudShell ed eseguire script Python.

Dopo aver eseguito il modello CloudFormation, è possibile completare la creazione di questa architettura eseguendo le seguenti operazioni.

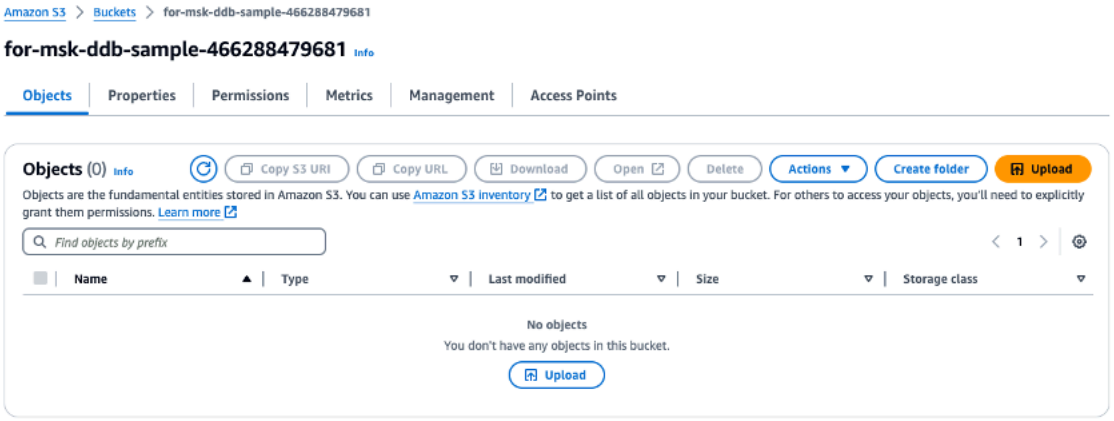
1. Esegui il modello CloudFormation `S3bucket.yaml` per creare un bucket S3. Esegui eventuali script o operazioni successive nella stessa Regione. Immetti `ForMSKTestS3` come nome dello stack CloudFormation.



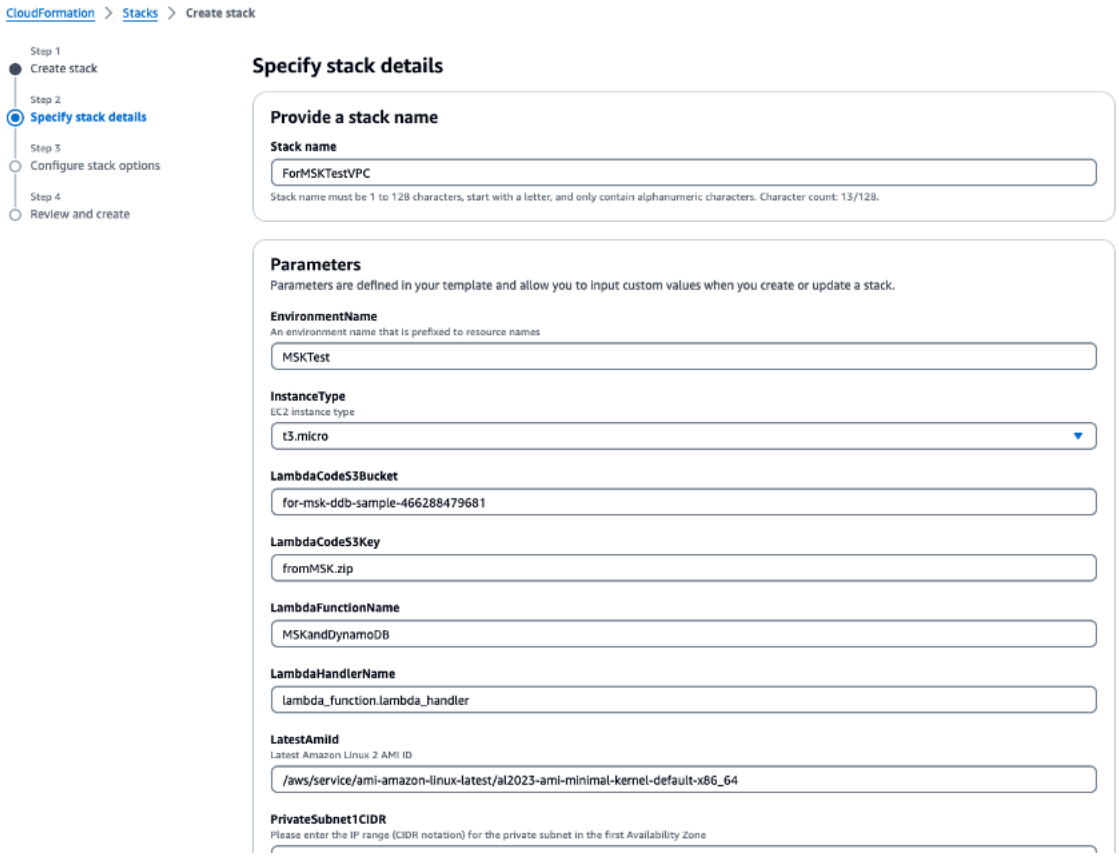
   Al termine, annota il nome del bucket S3 in uscita nella sezione Output. Ne avrai bisogno nella Fase 3.



2. Carica il file ZIP `fromMSK.zip` scaricato nel bucket S3 appena creato.

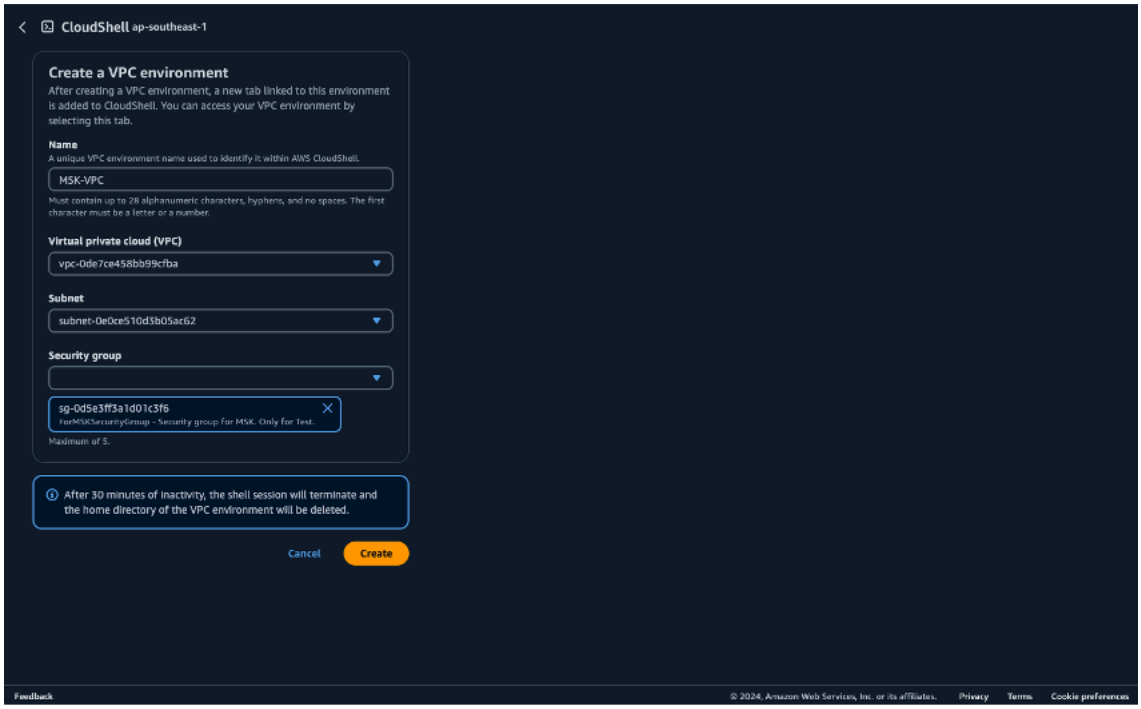


3. Esegui il modello CloudFormation `VPC.yaml` per creare un VPC, un cluster Amazon MSK e una funzione Lambda. Nella schermata di immissione dei parametri, inserisci il nome del bucket S3 creato nella Fase 1, in cui viene richiesto il bucket S3. Assegna allo stack CloudFormation il nome `ForMSKTestVPC`.



4. Prepara l'ambiente per l'esecuzione di script Python in CloudShell. È possibile utilizzare CloudShell sulla Console di gestione AWS. Per ulteriori informazioni sull'utilizzo di CloudShell, consulta Getting started with AWS CloudShell. Dopo aver avviato CloudShell, crea un CloudShell che appartiene al VPC appena creato per connetterti al cluster Amazon MSK. Crea il CloudShell in una sottorete privata. Riempi i seguenti campi:

    1. Nome: può essere impostato su qualsiasi nome. Un esempio è MSK-VPC
    2. VPC: seleziona MSKTest
    3. Sottorete: seleziona Sottorete privata MSKTest (AZ1)
    4. SecurityGroup: seleziona ForMSKSecurityGroup



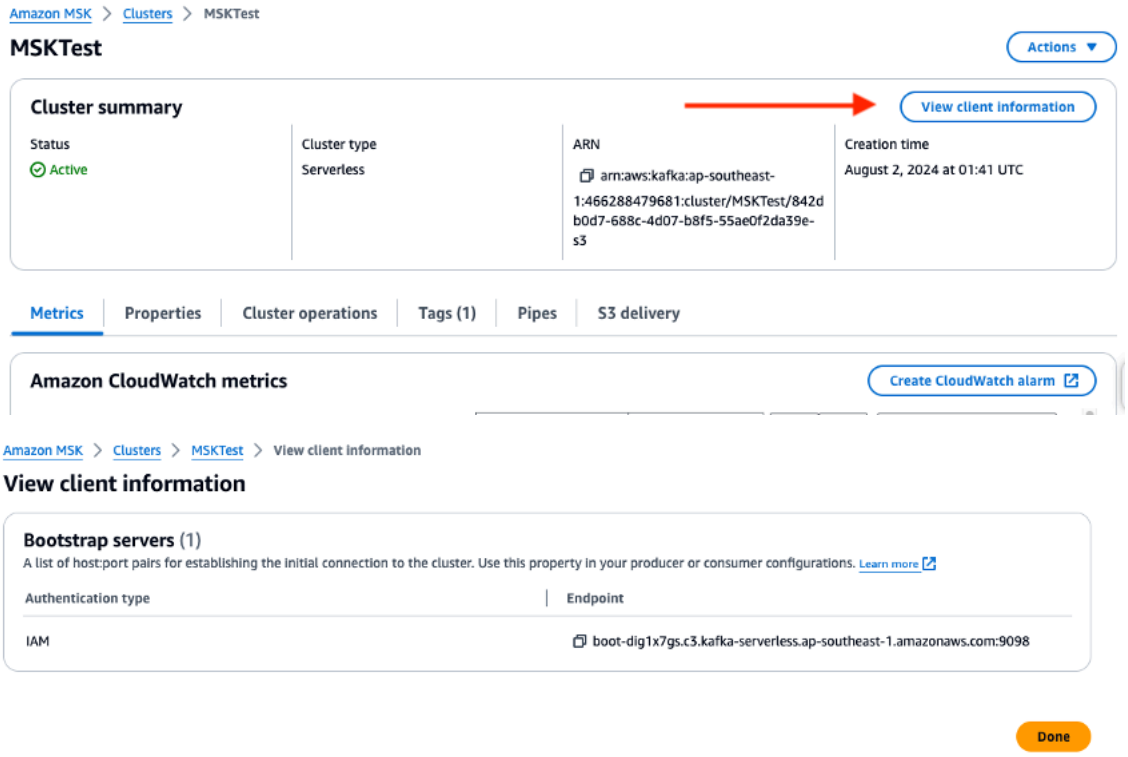
Una volta avviato il CloudShell appartenente alla sottorete privata, esegui il comando seguente:

```
pip install boto3 kafka-python aws-msk-iam-sasl-signer-python
```

5. Scarica gli script Python dal bucket S3.

```
aws s3 cp s3://[YOUR-BUCKET-NAME]/pythonScripts.zip ./
unzip pythonScripts.zip
```

6. Controlla la console di gestione e imposta le variabili di ambiente per l'URL del broker e il valore della Regione negli script Python. Controlla l'endpoint del broker del cluster Amazon MSK nella console di gestione.



7. Imposta le variabili di ambiente sul CloudShell. Se utilizzi la Regione Stati Uniti occidentali (Oregon):

```
export AWS_REGION="us-west-2"
export MSK_BROKER="boot-YOURMSKCLUSTER.c3.kafka-serverless.ap-
southeast-1.amazonaws.com:9098"
```

8. Esegui i seguenti script Python.

   Crea un argomento Amazon MSK:

```
python ./createTopic.py
```

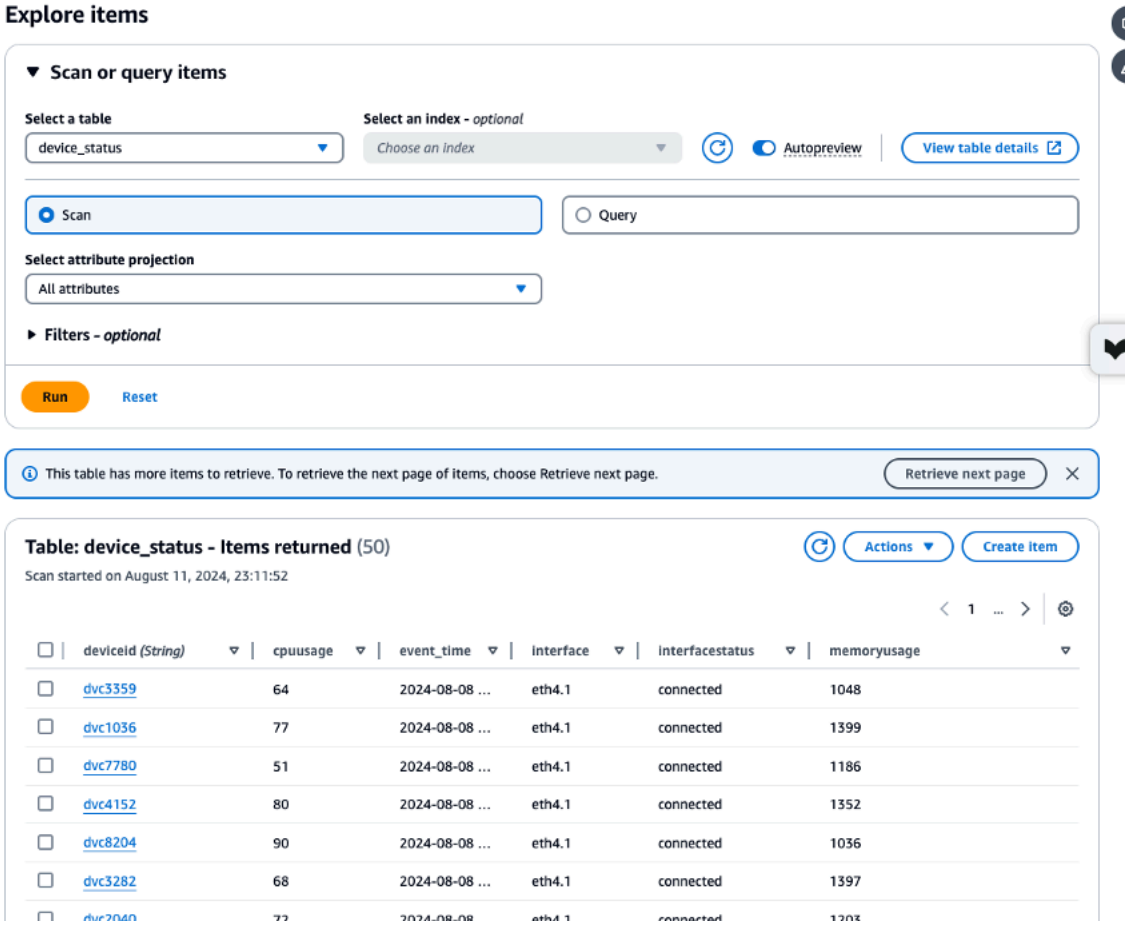
   Crea una tabella DynamoDB:

```
python ./createTable.py
```

Scrivi i dati di test nell'argomento Amazon MSK:

```
python ./kafkaDataGen.py
```

9. Controlla le metriche CloudWatch per le risorse Amazon MSK, Lambda e DynamoDB create e verifica i dati archiviati nella tabella `device_status` utilizzando DynamoDB Data Explorer per garantire che tutti i processi funzionino correttamente. Se ogni processo viene eseguito senza errori, è possibile verificare che i dati dei test scritti da CloudShell in Amazon MSK vengano scritti anche in DynamoDB.



10. Al termine, elimina le risorse create in questo tutorial. Elimina i due stack CloudFormation: `ForMSKTestS3` e `ForMSKTestVPC`. Se l'eliminazione dello stack viene completata correttamente, tutte le risorse verranno eliminate.

Passaggi successivi

> **Note**
>
> Se hai creato risorse seguendo questo esempio, ricordati di eliminarle per evitare addebiti imprevisti.

L'integrazione ha identificato un'architettura che collega Amazon MSK e DynamoDB per consentire il supporto dei carichi di lavoro OLTP da parte dei dati di flusso. Da qui, è possibile realizzare ricerche più complesse collegando DynamoDB con il Servizio OpenSearch. Prendi in considerazione l'integrazione con EventBridge per esigenze più complesse basate sugli eventi e con estensioni come Servizio gestito da Amazon per Apache Flink per un throughput più elevato e requisiti di latenza inferiori.

Best practice per l'integrazione con DynamoDB

Durante l'integrazione di DynamoDB con altri servizi, è sempre necessario seguire le best practice per l'utilizzo di ogni singolo servizio. Tuttavia, per l'integrazione occorre prendere in considerazione alcune best practice specifiche.

Argomenti

- Creazione di uno snapshot in DynamoDB
- Acquisizione di dati di modifica in DynamoDB

Creazione di uno snapshot in DynamoDB

- In genere, si consiglia di utilizzare l'esportazione in Amazon S3 per creare snapshot per la replica iniziale. È allo stesso tempo conveniente e non compete con il traffico dell'applicazione in termini di throughput. È inoltre possibile prendere in considerazione un backup e un ripristino su una nuova tabella seguiti da un'operazione di scansione. In questo modo si eviterà di competere con l'applicazione in termini di throughput, ma in genere questa opzione si rivelerà molto meno conveniente rispetto all'esportazione.
- Imposta sempre un `StartTime` durante un'esportazione. In questo modo è facile determinare da dove iniziare il Change Data Capture (CDC).

- Durante l'utilizzo di un'esportazione su S3, imposta un'azione del ciclo di vita sul bucket S3. In genere, un'azione con scadenza impostata su 7 giorni è sicura, ma è opportuno seguire tutte le possibili linee guida dell'azienda. Anche se si eliminano esplicitamente gli elementi dopo l'ingestione, questa azione può aiutare a intercettare i problemi, riducendo i costi inutili e prevenendo le violazioni delle norme.

Acquisizione di dati di modifica in DynamoDB

- In caso di bisogno di CDC quasi in tempo reale, usa i flussi DynamoDB o il flusso di dati Amazon Kinesis (KDS). Una volta deciso quale usare, in genere considera qual è l'opzione di più facile utilizzo con il servizio a valle. Se si deve fornire l'elaborazione degli eventi in ordine a livello di chiave di partizione o se si hanno elementi di dimensioni eccezionalmente grandi, è consigliabile usare i flussi DynamoDB.
- Se non si ha bisogno di un CDC quasi in tempo reale, è possibile utilizzare l'esportazione su Amazon S3 con esportazioni incrementali per esportare solo le modifiche avvenute tra due momenti nel tempo.

  Se si è utilizzata l'esportazione in S3 per generare uno snapshot, ciò può essere particolarmente utile perché è possibile utilizzare un codice simile per elaborare esportazioni incrementali. In genere, l'esportazione su S3 è leggermente più economica rispetto alle precedenti opzioni di flussi, ma in genere il costo non è il fattore principale per la scelta dell'opzione da utilizzare.
- In genere è possibile avere solo due consumer simultanei di un flusso DynamoDB. Considera questo aspetto al momento di pianificare la strategia di integrazione.
- Non utilizzare scansioni per rilevare le modifiche: questo approccio potrebbe funzionare su piccola scala, ma diventa poco pratico abbastanza rapidamente.

Utilizzo dell'IA generativa con DynamoDB

Amazon DynamoDB è un database NoSQL distribuito senza server, completamente gestito, con prestazioni a una cifra in millisecondi su qualsiasi scala. DynamoDB è ottimizzato per carichi di lavoro ad alto throughput ed è possibile estenderne le funzionalità attraverso l'integrazione con modelli di IA generativa. Utilizzando modelli di IA generativa è possibile lavorare con i dati archiviati nelle tabelle DynamoDB in tempo reale e creare applicazioni che possono adattarsi al contesto e altamente personalizzate. È possibile anche migliorare l'esperienza dell'utente finale sfruttando appieno i dati aziendali, degli utenti e delle applicazioni per personalizzare le proprie soluzioni di IA generativa.

Per ulteriori informazioni sull'IA di generazione e sulle soluzioni AWS fornite per creare applicazioni AI di generazione, consulta Trasforma il tuo business con l'intelligenza artificiale generativa.

Argomenti

- Casi d'uso di IA generativa per DynamoDB
- Blog di IA generativa per DynamoDB
- Sfruttare l'integrazione zero-ETL di DynamoDB con Service OpenSearch

Casi d'uso di IA generativa per DynamoDB

DynamoDB è ampiamente utilizzato nelle applicazioni conversazionali basate sull'intelligenza artificiale, come chatbot e call center creati con un modello di fondazione (FM). Puoi accedere FMs tramite Amazon Bedrock, Amazon SageMaker AI o altri fornitori di modelli. Tali applicazioni utilizzano comunemente DynamoDB per migliorare la personalizzazione e migliorare l'esperienza utente su tre modelli di dati: dati applicativi, dati aziendali e dati degli utenti. Alcuni esempi di questi modelli di dati sono i seguenti:

- Archiviazione dei dati dell'applicazione, come la cronologia dei messaggi di chat, tramite integrazioni con LangChaino un codice personalizzato. LlamaIndex Questo contesto migliora l'esperienza dell'utente consentendo al modello una conversazione bidirezionale con l'utente.
- Creazione di un'esperienza utente personalizzata sfruttando i dati aziendali, come inventario, prezzi e documentazione.
- Applicazione dei dati utente, come la cronologia web, gli ordini passati e le preferenze dell'utente, per fornire risposte personalizzate.

Ad esempio, una compagnia di assicurazioni può creare un chatbot utilizzando DynamoDB per fornire al proprio modello di IA generativa basato sulla generazione potenziata da recupero dati (RAG, Retrieval-Augmented Generation) l'accesso a dati quasi in tempo reale. Esempi di tali dati sono i tassi ipotecari in tempo reale, i prezzi dei prodotti, la copia del compliant/standard contratto, la cronologia web degli utenti e le preferenze degli utenti. La combinazione di DynamoDB con RAG aggiunge informazioni approfondite e aggiornate sui prodotti assicurativi e sui dati degli utenti. Ciò arricchisce i prompt e le risposte per fornire agli utenti finali un'esperienza accurata, personalizzata e quasi in tempo reale.

Allo stesso modo, i clienti del settore dei servizi finanziari utilizzano DynamoDB, le knowledge base Amazon Bedrock e gli agenti Amazon Bedrock per creare applicazioni di IA generativa basate su RAG. Queste applicazioni possono utilizzare report sugli utili e trascrizioni delle chiamate open source. Possono inoltre utilizzare la cronologia del portafoglio e delle transazioni specifica dell'utente per generare un riepilogo on demand del portafoglio che include una prospettiva per il futuro.

Blog di IA generativa per DynamoDB

I seguenti articoli offrono casi d'uso dettagliati, best practice e step-by-step guide per aiutarvi a sfruttare le funzionalità di DynamoDB nella creazione di applicazioni avanzate basate sull'intelligenza artificiale.

- Amazon DynamoDB data models for generative AI chatbots
- Crea un chatbot scalabile e sensibile al contesto con Amazon DynamoDB, Amazon Bedrock e LangChain

Sfruttare l'integrazione zero-ETL di DynamoDB con Service OpenSearch

Puoi utilizzare Amazon Bedrock con DynamoDB per fornire accesso serverless a modelli fondamentali (FMs), come Amazon Titan e altri modelli di terze parti. Puoi sfruttare l'integrazione zero-ETL con Amazon OpenSearch Service per abilitare le funzionalità di ricerca vettoriale durante la creazione di applicazioni AI generative. Il workshop Generative AI with DynamoDB zero-ETL OpenSearch to integration e Amazon Bedrock ti offrono un'esperienza pratica nella configurazione dell'integrazione DynamoDB zero-ETL con. OpenSearch Questo workshop svolge le seguenti attività:

- Crea una pipeline dalla tabella DynamoDB a. OpenSearch

- Crea un connettore Amazon Bedrock in OpenSearch.
- Interroga Amazon Bedrock OpenSearch come archivio vettoriale.
- Utilizza Claude FM in Amazon Bedrock per creare una risposta scritta in inglese semplice che spieghi i risultati della ricerca restituiti da. OpenSearch

Questo workshop consente di integrare DynamoDB per creare applicazioni OpenSearch AI generative. Dimostra inoltre la capacità flessibile di interrogazione tra i motori di database per aiutarti a integrare DynamoDB e per i casi d'uso tradizionali. OpenSearch Questo workshop è uno dei sette moduli dell'Amazon DynamoDB Immersion Day. Puoi tenere questo workshop in qualsiasi luogo. Account AWS

Puoi anche fare riferimento al seguente post di blog su come configurare un'integrazione zero-ETL tra DynamoDB e Service. OpenSearch Questo post del blog descrive anche come configurare i connettori modello in OpenSearch Service per generare automaticamente incorporamenti utilizzando Amazon Bedrock per i dati in entrata. Ricerca vettoriale per Amazon DynamoDB con zero ETL per Amazon Service. OpenSearch

Quote e vincoli per Amazon DynamoDB

Questo argomento descrive le quote correnti, precedentemente definite limiti, all'interno di Amazon DynamoDB. Questo argomento descrive anche come eseguire le attività di gestione delle quote, ad esempio visualizzare le quote correnti e richiederne un aumento.

Argomenti

- Esecuzione di attività di gestione delle quote in DynamoDB
- Richiesta di un aumento delle quote in DynamoDB
- Quote in Amazon DynamoDB
- Vincoli in Amazon DynamoDB

Esecuzione di attività di gestione delle quote in DynamoDB

Amazon DynamoDB dispone di diversi componenti di servizio, come tabelle, flussi, indici e altro ancora. Alla creazione dell'Account AWS, su questi componenti vengono impostate quote predefinite (precedentemente denominate limiti). Salvo diversa indicazione, ogni quota si applica a una regione specifica. Per alcune quote è possibile richiedere aumenti. Una volta raggiunta la quota di una risorsa, le ulteriori richieste di creazione di tale risorsa restituiranno un errore con un'eccezione.

Accesso alle quote di DynamoDB

È possibile lavorare con Service Quotas di DynamoDB attraverso le seguenti modalità:

- Console di gestione AWS

  La console Service Quotas è un'interfaccia basata su browser che è possibile utilizzare per visualizzare e gestire le quote di servizio. L'accesso a Service Quotas è possibile da qualsiasi pagina della Console di gestione AWS selezionando il servizio nella barra di navigazione in alto o cercando Service Quotas nella Console di gestione AWS.
- Strumenti AWS Command Line Interface

  Durante l'utilizzo degli strumenti della AWS Command Line Interface, è possibile inviare comandi alla riga di comando del sistema per eseguire attività correlate a Service Quotas. Gli strumenti a riga di comando sono utili per creare script che eseguono attività AWS.
- AWS SDK

È possibile utilizzare gli SDK AWS per diversi linguaggi di programmazione e piattaforme (ad esempio Java, Python, Ruby, .NET, iOS e Android, ecc.) per eseguire attività correlate a Service Quotas.

Se una quota regolabile non è disponibile nella console Service Quotas, utilizza AWS Support Center Console per creare un caso di aumento delle quote di servizio.

Visualizzazione delle quote correnti nella console

Come visualizzare le quote DynamoDB correnti attraverso la console Service Quotas

1. Apri la console Service Quotas all'indirizzo https://console.aws.amazon.com/servicequotas/home/services/dynamodb/quotas/
2. Nella barra di navigazione, nella parte superiore della schermata, seleziona una Regione.
3. La console mostra i dettagli su Nome quota di DynamoDB, Valore della quota a livello di account applicato, Valore della quota predefinito di AWS, Utilizzo e Regolabilità della quota a livello di account o di risorsa.

   Se il valore o l'utilizzo della quota applicata non sono disponibili, la console mostra Non disponibile. È possibile richiedere il valore della quota applicata tramite la console del Centro di supporto.
4. Seleziona un Nome quota specifico per visualizzare la pagina Dettagli, che mostra la Descrizione, il Codice di quota, l'ARN di quota, l'Utilizzo, il Valore della quota a livello di account applicato, la Regolabilità e il Valore della quota predefinito di AWS.

   Se applicabile, la pagina Dettagli mostra anche le opzioni di Monitoraggio, gli Allarmi, la Cronologia delle richieste e gli eventuali Tag della quota.

Visualizzazione delle quote correnti tramite la AWS CLI

Per visualizzare i valori predefiniti per le quote DynamoDB:

- Chiama l'operazione `ListDefaultServiceQuotas` con il codice di servizio DynamoDB (dynamodb) per recuperare i valori predefiniti per le quote di servizio di 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
        }......
    ]
}
```

Per visualizzare i valori delle quote applicate:

- Chiama l'operazione `ListServiceQuotas` con il codice di servizio DynamoDB (dynamodb) per recuperare i valori di tutte le quote applicate a livello di account, di risorsa o a tutti i livelli passando `ACCOUNT`, `RESOURCE` o `ALL` rispettivamente come valore per il parametro `QuotaAppliedAtLevel`. L'esempio di CLI seguente recupera i valori delle quote applicate a livello di account.

```
$ 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"
        }.....

        }
    ]
}
```

Richiesta di un aumento delle quote in DynamoDB

È possibile richiedere un aumento delle quote per ciascuna Regione utilizzando la console Service Quotas, la AWS CLI o un caso di supporto. Se una quota regolabile non è disponibile nella console Service Quotas, utilizza AWS Support Center Console per creare un caso di aumento delle quote di servizio.

Il Supporto potrebbe approvare, rifiutare o approvare parzialmente le richieste di aumento delle quote. Gli aumenti non vengono concessi immediatamente e possono essere necessari alcuni giorni per renderli effettivi.

Per richiedere un aumento utilizzando la console Service Quotas

1. Apri la console Service Quotas all'indirizzo https://console.aws.amazon.com/servicequotas/home/services/dynamodb/quotas/
2. Nella barra di navigazione, nella parte superiore della schermata, seleziona una Regione.
3. Filtra l'elenco in base al nome della risorsa. Ad esempio, inserisci On demand per individuare le quote per le istanze on demand.
4. Se la quota è modificabile, seleziona la quota e quindi scegli Richiedi aumento della quota.
5. In Modifica valore quota, inserisci il nuovo valore della quota.
6. Scegli Richiedi.
7. Per visualizzare eventuali richieste in sospeso o risolte di recente nella console, scegli Pannello di controllo dal riquadro di navigazione. Per le richieste in sospeso, scegliere lo stato della richiesta per aprire la ricevuta della richiesta. Lo stato iniziale di una richiesta è Pending (In attesa). Dopo che lo stato è cambiato in Quota richiesta, vedrai il numero del caso con il Supporto. Scegli il numero del caso per aprire il ticket della tua richiesta.

Per ulteriori informazioni, incluso come utilizzare la AWS CLI o gli SDK per richiedere un aumento della quota, consulta la pagina Richiesta di un aumento della quota nella Guida per l'utente delle Service Quotas.

Quote in Amazon DynamoDB

Questa sezione descrive le quote correnti, precedentemente definite limiti, all'interno di Amazon DynamoDB. Salvo dove diversamente specificato, ogni quota si applica a una Regione specifica.

Argomenti

- Throughput di lettura/scrittura
- Capacità prenotata
- Tabelle
- Tabelle globali
- Indici secondari
- Attributi degli indici secondari proiettati
- DynamoDB Streams

- Importazione da Amazon S3
- Esportazione delle tabelle in Amazon S3
- Backup e ripristino
- Contributor Insights

Throughput di lettura/scrittura

Quote predefinite della velocità di trasmissione effettiva

AWS imposta alcune quote predefinite sulla velocità di trasmissione effettiva che il tuo account può allocare e usare in una regione.

La velocità di trasmissione effettiva di lettura a livello di account e le quote di velocità di trasmissione effettiva di scrittura a livello di account si applicano a livello di account. Queste quote a livello di account si applicano alla somma della capacità di velocità di trasmissione effettiva fornita per tutte le tabelle dell'account e gli indici secondari globali in una determinata regione. Tutte le velocità di trasmissione effettiva disponibili per l'account possono essere fornite tramite provisioning per una tabella singola o per più tabelle. Queste quote si applicano solo alle tabelle che utilizzano la modalità di capacità con provisioning.

La velocità di trasmissione effettiva di lettura a livello di tabella e le quote di velocità di trasmissione effettiva di scrittura a livello di tabella si applicano in modo diverso alle tabelle che utilizzano la modalità di capacità allocata e alle tabelle che utilizzano la modalità capacità on demand.

Per le tabelle in modalità di capacità allocata e gli indici secondari globali, la quota è la quantità massima di unità di capacità di lettura e scrittura che possono essere fornite per qualsiasi tabella o uno dei relativi indici secondari globali nella regione. Il totale di ogni singola tabella e di tutti i relativi indici secondari globali deve inoltre rimanere al di sotto della quota di velocità di trasmissione effettiva di lettura e scrittura a livello di account. Ciò si aggiunge al requisito che il totale di tutte le tabelle allocate e dei relativi indici secondari globali deve rimanere al di sotto della quota di velocità di trasmissione effettiva di lettura e scrittura a livello di account.

Per le tabelle in modalità capacità on demand e gli indici secondari globali, la quota a livello di tabella è il massimo numero di unità di capacità di lettura e scrittura disponibili per qualsiasi tabella o per ogni singolo indice secondario globale all'interno di quella tabella. Nessuna quota di velocità di trasmissione effettiva di lettura e scrittura a livello di account viene applicata alle tabelle in modalità on demand.

Di seguito sono riportate le quote di throughput applicabili al proprio account per impostazione predefinita.

> **Note**
>
> È possibile richiedere un numero qualsiasi di unità di capacità di lettura (RCU, Read Capacity Units) o unità di capacità di scrittura (WCU, Write Capacity Units) per le tabelle DynamoDB tramite un aumento delle quote di servizio. I valori elencati nella tabella sottostante rappresentano le quote predefinite iniziali e non i limiti massimi per le tabelle.

Nome quota di throughput	On demand	Assegnata	Regolabile
Per tabella	40.000 unità di richiesta di lettura e 40.000 unità di richiesta di scrittura	40.000 unità di capacità in lettura e 40.000 unità di capacità in scrittura	Sì
Per account	Non applicabile	80.000 unità di capacità in lettura e 80.000 unità di capacità in scrittura	Sì
Throughput minimo per qualsiasi tabella o indice secondario globale	Non applicabile	1 unità di capacità di lettura e 1 unità di capacità di scrittura	Sì

Aumento o diminuzione della velocità di trasmissione effettiva (per tabelle assegnate)

Aumento della velocità di trasmissione effettiva assegnata

Puoi aumentare `ReadCapacityUnits` o `WriteCapacityUnits` ogni qualvolta sia necessario, utilizzando l'operazione Console di gestione AWS o `UpdateTable`. In un'unica chiamata, puoi aumentare la velocità effettiva assegnata di una tabella, di tutti gli indici secondari globali di tale

tabella o di una combinazione di questi. Le nuove impostazioni non avranno effetto fino a quando l'operazione `UpdateTable` non sarà completata.

Non è possibile superare le quote per account quando si aggiunge una capacità con provisioning e DynamoDB non consente di aumentare la capacità con provisioning molto rapidamente. Oltre a queste limitazioni, puoi aumentare la capacità assegnata delle tabelle in base alle tue necessità. Per ulteriori informazioni sulle quote di ogni account, consulta la sezione precedente, Quote predefinite della velocità di trasmissione effettiva.

Riduzione della velocità di trasmissione effettiva assegnata

Per ogni tabella e indice secondario globale in un'operazione `UpdateTable`, puoi ridurre `ReadCapacityUnits` o `WriteCapacityUnits` (o entrambi). Le nuove impostazioni non hanno effetto fino a quando l'operazione `UpdateTable` non è terminata.

Esiste una quota predefinita sul numero di riduzioni di capacità con provisioning che è possibile eseguire su DynamoDB al giorno. Un giorno è definito in base a Universal Coordinated Time (UTC). In un dato giorno, è possibile iniziare eseguendo fino a quattro diminuzioni in un'ora, purché non si siano ancora state eseguite altre diminuzioni nel corso dello stesso giorno. Successivamente, è possibile eseguire una diminuzione aggiuntiva all'ora (una volta ogni 60 minuti). Ciò porta efficacemente il numero massimo di riduzioni in un giorno a 27 volte.

> **Important**
>
> I limiti di riduzione della tabella e degli indici secondari globali non sono associati; ciò significa che tutti gli indici secondari associati di una determinata tabella avranno i propri limiti di riduzione. Tuttavia, se una singola richiesta diminuisce la velocità effettiva di una tabella e di un indice secondario globale, questa verrà rifiutata se supera i limiti correnti. Le richieste non vengono parzialmente elaborate.

Example

Nelle prime quattro ore di una giornata, una tabella con un indice secondario globale può essere modificata come segue:

- Riduci i valori `WriteCapacityUnits` o `ReadCapacityUnits` (o entrambi) di una tabella 4 volte.
- Riduci i valori `WriteCapacityUnits` o `ReadCapacityUnits` (o entrambi) dell'indice secondario globale di 4 volte.

Alla fine dello stesso giorno, la velocità effettiva della tabella e dell'indice secondario globale può essere potenzialmente diminuita per un totale di 27 volte ciascuna.

Capacità prenotata

AWS impone una quota predefinita sulla quantità di capacità riservata attiva che il tuo account può acquistare. Il limite di quota è una combinazione di capacità riservata per le unità di capacità di scrittura (WCU) e le unità di capacità di lettura (RCU).

Quota di capacità riservata	Capacità riservata attiva	Regolabile
Per account	1.000.000 di unità di capacità assegnata (WCU_ RCU)	Sì

Se tenti di acquistare più di 1.000.000 di unità di capacità assegnata in un unico acquisto, riceverai un errore per questo limite di quota di servizio. Se disponi di capacità riservata attiva e tenti di acquistare capacità riservata aggiuntiva che comporterebbe più di 1.000.000 di unità di capacità assegnate attive, riceverai un errore relativo a questo limite di quota di servizio.

Tabelle

Dimensione della tabella

Non vi è un limite pratico sulla dimensione della tabella. Le tabelle non hanno restrizioni in termini di numero di item o di byte.

Numero massimo di tabelle per regione per account

Per qualsiasi account AWS vi è una quota iniziale di 2500 tabelle per Regione AWS.

Se hai bisogno di più di 2.500 tabelle per un singolo account, contatta il team del tuo account AWS per valutare un aumento fino a un massimo di 10.000 tabelle. Per più di 10.000, la best practice consigliata è quella di configurare più account, ognuno dei quali può servire fino a 10.000 tabelle.

Tabelle globali

Le seguenti quote predefinite si applicano quando si utilizzano tabelle globali.

Quote di tabelle globali predefinite	On demand	Assegnata
Numero di tabelle globali MRSC (consulta the section called “Modalità di coerenza”)	400 tabelle globali MRSC in totale in qualsiasi modalità di capacità	400 tabelle globali MRSC in totale in qualsiasi modalità di capacità
Throughput per tabella configurato per la coerenza finale multi-Regione (MREC)	40.000 unità di richiesta di lettura e 40.000 unità di richiesta di scrittura	40.000 unità di capacità in lettura e 40.000 unità di capacità in scrittura
Throughput per tabella configurato per elevata consistenza multi-Regione (MRSC)	40.000 unità di richiesta di lettura e 40.000 unità di richiesta di scrittura	40.000 unità di capacità in lettura e 40.000 unità di capacità in scrittura
Dati compilati per nuove repliche per account, per Regione, al giorno	10 TB	10 TB

Note

In alcuni casi sarà necessario richiedere un aumento del limite di quota mediante Supporto AWS. Se una delle seguenti condizioni si riferisce allo scenario corrente, consulta https://aws.amazon.com/support:

- Affinché la creazione delle repliche abbia esito positivo, le quote di throughput delle tabelle globali devono essere uguali o superiori alle quote di throughput per tabella. Sono previste quote di throughput delle tabelle globali separate per le tabelle globali MREC e quelle MRSC.
- Se aggiungi una replica o repliche a una regione di destinazione entro un periodo di 24 ore con un totale combinato superiore a 10 TB, è necessario richiedere un aumento della quota di servizio per la quota di backfill dei dati di replica aggiunta.
- Se si verifica un errore simile al seguente:

- Impossibile creare una replica della tabella 'tabella_esempio' nella regione 'Regione_esempio_A' perché supera il limite dell'account corrente nella regione 'Regione_esempio_B'.

Indici secondari

È possibile definire fino a 5 indici secondari locali per tabella.

Esiste una quota di default di 20 indici secondari globali per tabella.

Attributi degli indici secondari proiettati

È possibile proiettare un massimo di 100 attributi combinati per tutti gli indici secondari locali e globali di una tabella. Tale quota si applica solo agli attributi proiettati specificati dall'utente.

Per l'operazione `CreateTable`, se si specifica un `ProjectionType` di `INCLUDE`, il numero totale degli attributi specificati nei `NonKeyAttributes` sommato tra tutti gli indici secondari non dovrà essere superiore a 100. La proiezione di uno stesso nome attributo su due indici diversi è considerata come due attributi separati nel calcolo della quota.

Tale quota non si applica agli indici secondari con un `ProjectionType` di `KEYS_ONLY` o `ALL`.

DynamoDB Streams

Lettori simultanei di una partizione in DynamoDB Streams

Per le tabelle a Regione singola che non sono tabelle globali, è possibile progettare in modo che fino a due processi simultanei possano leggere dallo stesso shard dei flussi DynamoDB. Il superamento di questo limite comporta una limitazione delle richieste. Per le tabelle globali consigliamo di limitare il numero di lettori simultanei a uno per evitare richieste di limitazione della larghezza di banda della rete.

Capacità di scrittura massima per una tabella con DynamoDB Streams abilitato

AWS stabilisce alcune quote predefinite sulla capacità di scrittura delle tabelle di DynamoDB con DynamoDB Streams abilitato. Queste quote predefinite sono applicabili solo per le tabelle in modalità di capacità di lettura/scrittura con provisioning.

- Per tabella: 40.000 unità di capacità in scrittura

Importazione da Amazon S3

La funzionalità DynamoDB di importazione da S3 può supportare fino a 50 processi di importazione simultanea, con una dimensione totale dell'oggetto di origine di importazione pari a 15 TB alla volta nelle regioni us-east-1, us-west-2 e eu-west-1. In tutte le altre regioni, sono supportate fino a 50 attività di importazione simultanee con una dimensione totale di 1 TB. Ogni processo di importazione può elaborare fino a 50.000 oggetti Amazon S3 in tutte le Regioni. Per ulteriori informazioni sull'importazione e la convalida, consultare le quote del formato di importazione e la convalida.

Esportazione delle tabelle in Amazon S3

Esportazione completa: è possibile esportare fino a 300 attività simultanee di esportazione o fino a un totale di 100 TB di tutte le esportazioni di tabelle in corso. Entrambi questi limiti vengono verificati prima che un'esportazione venga messa in coda.

Esportazione incrementale: l'esportazione incrementale di DynamoDB in Amazon S3 può supportare contemporaneamente fino a 300 attività di esportazione o fino a un totale di 100 TB di tutte le esportazioni di tabelle in corso. I limiti del periodo di esportazione sono un minimo di 15 minuti e un massimo di 24 ore.

Backup e ripristino

DynamoDB supporta contemporaneamente fino a 50 ripristini per un totale di 50 TB tramite backup DynamoDB on demand o continui. AWS Backup supporta contemporaneamente fino a 50 ripristini per un totale di 25 TB.

Contributor Insights

L'abilitazione di Customer Insights su una tabella DynamoDB non esonera dai limiti imposti dalle regole di Contributor Insights. Per ulteriori informazioni, consulta Quote del servizio di CloudWatch.

Vincoli in Amazon DynamoDB

Questa sezione descrive i vincoli correnti, precedentemente definiti limiti, all'interno di Amazon DynamoDB.

Argomenti

- Modalità di capacità in lettura/scrittura

- Indici secondari
- Chiavi di partizione e chiavi di ordinamento
- Regole di denominazione
- Tipi di dati
- Item
- Attributes
- Parametri di espressione
- Transazioni di DynamoDB
- DynamoDB Streams
- DynamoDB Accelerator (DAX)
- Vincoli specifici delle API
- Crittografia a riposo per DynamoDB

Modalità di capacità in lettura/scrittura

È possibile cambiare le tabelle dalla modalità con capacità allocata alla modalità on demand fino a quattro volte in una finestra variabile di 24 ore. È possibile passare le tabelle dalla modalità on demand alla modalità con capacità allocata in qualsiasi momento.

Per ulteriori informazioni sul passaggio dalla modalità di capacità di lettura a quella di scrittura, consulta Considerazioni sul passaggio tra modalità di capacità in DynamoDB.

Dimensioni dell'unità di capacità (per tabelle assegnate)

Una unità di capacità di lettura equivale a una lettura fortemente consistente al secondo o a due letture a consistenza finale al secondo, per elementi di dimensioni fino a 4 KB.

Una unità di capacità di scrittura equivale a una scrittura al secondo per elementi di dimensioni fino a 1 KB.

Le richieste di lettura transazionale richiedono due unità di capacità in lettura per eseguire una lettura al secondo per elementi di dimensioni fino a 4 KB.

Le richieste di scrittura transazionale richiedono due unità di capacità in scrittura per eseguire una scrittura al secondo per elementi di dimensioni fino a 1 KB.

Dimensioni dell'unità di richiesta (per tabelle on demand)

Una unità di richiesta di lettura equivale a una elevata consistenza di lettura al secondo o a due letture a coerenza finale al secondo, per elementi di dimensioni fino a 4 KB.

Un'unità di richiesta di scrittura equivale a una scrittura al secondo per elementi di dimensioni fino a 1 KB.

Le richieste di lettura transazionali richiedono due unità di richiesta di lettura al secondo per eseguire una lettura per elementi fino a 4 KB.

Le richieste di scrittura transazionale richiedono due unità di richiesta di scrittura al secondo per eseguire una scrittura per elementi fino a 1 KB.

Indici secondari

Attributi di indice secondario proiettati per tabella

Puoi proiettare un massimo di 100 attributi in tutti gli indici secondari locali e globali di una tabella. Tale approccio si applica solo agli attributi proiettati specificati dall'utente.

In un'operazione `CreateTable`, se si specifica un `ProjectionType` di `INCLUDE`, il numero totale degli attributi specificati in `NonKeyAttributes`, sommato tra tutti gli indici secondari non dovrà essere superiore a 100. Se proietti lo stesso nome di attributo in due indici diversi, questo sarà considerato come parte di due attributi distinti quando si dovrà determinare il totale.

Questo limite non si applica agli indici secondari il cui valore `ProjectionType` è `KEYS_ONLY` o `ALL`.

Chiavi di partizione e chiavi di ordinamento

Lunghezza della chiave di partizione

La lunghezza minima del valore di una chiave di partizione è di 1 byte. La lunghezza massima è 2048 byte.

Valori della chiave di partizione

Non vi è un limite pratico relativo al numero dei valori di chiavi di partizione distinte, sia per le tabelle che per gli indici secondari.

Lunghezza della chiave di ordinamento

La lunghezza minima del valore di una chiave di ordinamento è di 1 byte. La lunghezza massima è 1024 byte.

Valori della chiave di ordinamento

In generale, non vi è un limite pratico relativo al numero dei valori delle chiavi di ordinamento distinte per ogni valore della chiave di partizione.

Fanno eccezione le tabelle con indici secondari. Una raccolta di elementi è l'insieme di elementi che hanno lo stesso valore dell'attributo della chiave di partizione. In un indice secondario globale la raccolta di elementi è indipendente dalla tabella di base (e può avere un attributo chiave di partizione diverso), ma in un indice secondario locale la vista indicizzata è co-locata nella stessa partizione dell'elemento nella tabella e condivide lo stesso attributo della chiave di partizione. Di conseguenza, quando una tabella ha uno o più LSI, la raccolta di elementi non può essere distribuita su più partizioni.

Per una tabella con uno o più indici secondari locali (LSI), le raccolte di elementi non possono avere dimensioni superiori a 10 GB. Ciò include tutti gli elementi della tabella di base e tutte le visualizzazioni LSI previste che hanno lo stesso valore dell'attributo della chiave di partizione. La dimensione massima di una partizione è di 10 GB. Per informazioni più dettagliate, consulta Limite delle dimensioni delle raccolte di elementi.

Regole di denominazione

Nomi di tabelle e nomi di indici secondari

I nomi delle tabelle e degli indici secondari devono contenere un minimo di 3 caratteri e un massimo di 255. Di seguito sono riportati i caratteri consentiti:

- `A-Z`
- `a-z`
- `0-9`
- `_` (carattere di sottolineatura)
- `-` (trattino)
- `.` (punto)

Nomi di attributi

In generale, il nome di un attributo deve contenere almeno un carattere, ma non deve superare i 64 KB.

Di seguito sono elencate le eccezioni. Questi nomi di attributo non devono superare i 255 caratteri:

- Nomi delle chiavi di partizione degli indici secondari.
- Nomi delle chiavi di ordinamento degli indici secondari.
- I nomi degli attributi proiettati specificati dall'utente (applicabile solo agli indici secondari locali). In un'operazione `CreateTable`, se specifichi il valore `INCLUDE` per `ProjectionType`, i nomi degli attributi del parametro `NonKeyAttributes` saranno soggetti a restrizioni di lunghezza. I tipi di proiezione `KEYS_ONLY` e `ALL` non sono interessati.

Questi nomi di attributo devono essere codificati tramite UTF-8, mentre la dimensione totale di ogni nome (dopo la codifica) non può superare i 255 byte.

Tipi di dati

Stringa

La lunghezza di una stringa è vincolata dal limite massimo della dimensione dell'elemento, che è di 400 KB.

Le stringhe sono Unicode con codifica binaria UTF-8. Poiché UTF-8 è una codifica a larghezza variabile, DynamoDB determina la lunghezza di una stringa utilizzando i byte UTF-8.

Numero

Un numero può avere fino a 38 cifre di precisione e può essere positivo, negativo o zero.

- Intervallo positivo: da `1E-130` a `9.99999999999999999999999999999999999999E+125`
- Intervallo negativo: da `-9.99999999999999999999999999999999999999E+125` a `-1E-130`

DynamoDB utilizza stringhe JSON per rappresentare dati di tipo Number nelle richieste e nelle risposte. Per ulteriori informazioni, consulta API DynamoDB di basso livello.

Se la precisione del numero è importante, è necessario passare i numeri a DynamoDB utilizzando le stringhe convertite da un tipo di numero.

Binario

La lunghezza di un dato Binary è vincolata dal limite massimo della dimensione dell'elemento, che è di 400 KB.

Le applicazioni che funzionano con gli attributi di tipo Binary devono codificare i dati nel formato Base64 prima di inviarli a DynamoDB. Quando i dati vengono ricevuti, DynamoDB li decodifica in un array di byte senza firma e utilizza questo array come lunghezza dell'attributo.

Item

Dimensione degli elementi

La dimensione massima di un elemento in DynamoDB è 400 KB, che include sia la lunghezza binaria del nome dell'attributo (lunghezza UTF-8) che le lunghezze dei valori dell'attributo (ancora la lunghezza binaria). Il nome attributo viene conteggiato per il limite di dimensione.

Ad esempio, considera un elemento con due attributi: un attributo denominato "shirt-color" con valore "R" e un attributo denominato "shirt-size" con valore "M". La dimensione totale di quell'elemento è di 23 byte.

Dimensione degli elementi per le tabelle con indici secondari locali

Per ogni indice secondario locale di una tabella, vi è un limite di 400 KB sul totale dei seguenti elementi:

- La dimensione dei dati di un elemento nella tabella;
- La dimensione delle voci corrispondenti, inclusi i valori delle chiavi e gli attributi proiettati, in tutti gli indici secondari locali.

Attributes

Coppie nome/valore degli attributi per elemento

La dimensione cumulativa degli attributi per elemento deve rientrare nella dimensione massima dell'elemento di DynamoDB (400 KB).

Numero di valori in un elenco, una mappa o un insieme

Non vi è alcun limite al numero di valori in un attributo List, Map o Set purché l'elemento contenente i valori rientri nel limite di dimensione dell'elemento di 400 KB.

Valori di attributi

Sono consentiti valori degli attributi String e Binary vuoti, se l'attributo non viene utilizzato come attributo chiave per una tabella o un indice. I valori String e Binary vuoti sono consentiti all'interno dei tipi elenco, lista e mappa. Il valore di attributo non può essere un set vuoto (String Set, Number Set o Binary Set). Tuttavia, sono consentiti elenchi e mappe vuoti.

Profondità degli attributi nidificati

DynamoDB supporta gli attributi nidificati fino a un massimo di 32 livelli di profondità.

Parametri di espressione

I parametri di espressione includono `ProjectionExpression`, `ConditionExpression`, `UpdateExpression` e `FilterExpression`.

Lunghezze

La lunghezza massima di qualsiasi stringa di espressione è di 4 KB. Ad esempio, la dimensione di `ConditionExpression a=b` è di 3 byte.

La lunghezza massima di qualsiasi nome di attributo di espressione individuale o di un valore attributo di espressione è di 255 byte. Ad esempio, `#name` è 5 byte; `:val` è 4 byte.

La lunghezza massima di tutte le variabili di sostituzione in un'espressione è di 2 MB. Questo valore rappresenta la somma tra la lunghezza di tutti i `ExpressionAttributeNames` e `ExpressionAttributeValues`.

Operatori e operandi

Il numero massimo degli operatori e delle funzioni consentite in un `UpdateExpression` è di 300. Ad esempio, UpdateExpression `SET a = :val1 + :val2 + :val3` contiene due operatori "+".

Il numero massimo di operandi per il comparatore `IN` è 100

Parole riservate

DynamoDB non impedisce di utilizzare nomi in conflitto con le parole riservate. (Per un elenco completo, consulta Parole riservate in DynamoDB).

Tuttavia, se utilizzi una parola prenotata in un parametro di espressione, devi specificare anche `ExpressionAttributeNames`. Per ulteriori informazioni, consulta Nomi di attributi di espressione (alias) in DynamoDB.

Transazioni di DynamoDB

Le operazioni API transazionali di DynamoDB hanno i seguenti vincoli:

- Una transazione non può contenere più di 100 elementi univoci.
- Una transazione non può contenere più di 4 MB di dati.
- Una transazione non può avere due operazioni che agiscono sullo stesso elemento nella stessa tabella. Ad esempio, non puoi utilizzare sia `ConditionCheck` che `Update` per lo stesso elemento in una transazione.
- Una transazione non può operare su tabelle in più di un account o regione AWS.
- Le operazioni transazionali forniscono garanzie di atomicità, consistenza, isolamento e durabilità (Atomicity, Consistency, Isolation and Durability, ACID) solo all'interno della regione AWS in cui è stata effettuata originariamente l'operazione di scrittura. Le transazioni non sono supportate tra le regioni nelle tabelle globali. Ad esempio, supponi di avere una tabella globale con repliche nelle Regioni Stati Uniti orientali (Ohio) e Stati Uniti occidentali (Oregon) e di eseguire un'operazione `TransactWriteItems` nella Regione Stati Uniti orientali (Virginia settentrionale). In questo caso, puoi osservare transazioni parzialmente completate nella Regione degli Stati Uniti occidentali (Oregon) mentre le modifiche vengono replicate. Le modifiche vengono replicate in altre Regioni solo dopo essere state confermate nella Regione di origine.

DynamoDB Streams

Lettori simultanei di una partizione in DynamoDB Streams

Per le tabelle di una regione singola che non sono tabelle globali, è possibile progettare contemporaneamente fino a due processi da leggere dallo stesso shard DynamoDB Streams. Il superamento di questo limite comporta una limitazione delle richieste. Per le tabelle globali

consigliamo di limitare il numero di lettori simultanei a uno per evitare richieste di limitazione della larghezza di banda della rete.

DynamoDB Accelerator (DAX)

Disponibilità nelle regioni AWS

Per un elenco di regioni AWS in cui è disponibile DAX, consulta Acceleratore DynamoDB (DAX) in Riferimenti generali di AWS.

Nodi

Un cluster DAX è costituito esattamente da un nodo primario e un numero di nodi di replica di lettura compreso tra 0 e 10.

Il numero totale di nodi (per account AWS) non può essere superiore a 50 in una singola regione AWS.

Gruppi di parametri

Puoi creare fino a 20 gruppi di parametri DAX per regione.

Gruppi di sottoreti

Puoi creare fino a 50 gruppi di sottoreti DAX per regione.

All'interno di un gruppo di sottoreti, puoi definire fino a un massimo di 20 sottoreti.

> **Important**
>
> Un cluster DAX supporta un massimo di 500 tabelle DynamoDB. Una volta superate le 500 tabelle DynamoDB, il cluster potrebbe subire un peggioramento della disponibilità e delle prestazioni.

Vincoli specifici delle API

`CreateTable/UpdateTable/DeleteTable/PutResourcePolicy/DeleteResourcePolicy`

In generale, sono possibili fino a 500 richieste CreateTable, UpdateTable, DeleteTable, PutResourcePolicy e DeleteResourcePolicy in esecuzione contemporaneamente in qualsiasi

combinazione. Di conseguenza, il numero totale delle tabelle nello stato `CREATING`, `UPDATING` o `DELETING` non può essere superiore a 500.

È possibile inviare fino a 2.500 richieste al secondo di richieste API del piano di controllo (control-plane) mutabili (`CreateTable`, `DeleteTable`, `UpdateTable`, `PutResourcePolicy` e `DeleteResourcePolicy`) in qualsiasi combinazione tra un gruppo di tabelle. Tuttavia, le richieste `PutResourcePolicy` e `DeleteResourcePolicy` hanno limiti individuali inferiori. Per ulteriori informazioni, consulta i seguenti dettagli sulle quote per `PutResourcePolicy` e`DeleteResourcePolicy`.

Le richieste `CreateTable` e `PutResourcePolicy` che includono una policy basata su risorse verranno conteggiate come due richieste aggiuntive per ogni KB della policy. Ad esempio, una richiesta `CreateTable` o `PutResourcePolicy` con un criterio di dimensione 5 KB verrà conteggiata come 11 richieste: 1 per la richiesta `CreateTable` e 10 per la policy basata su risorse (2 x 5 KB). Analogamente, una policy di dimensioni 20 KB conterà come 41 richieste: 1 per la richiesta `CreateTable` e 40 per la policy basata su risorse (2 x 20 KB).

`PutResourcePolicy`

È possibile inviare fino a 25 richieste API `PutResourcePolicy` al secondo su un gruppo di tabelle. Dopo una richiesta riuscita per una singola tabella, non vengono supportate nuove `PutResourcePolicy` richieste per i successivi 15 secondi.

La dimensione massima supportata per un documento di policy basata su risorse è di 20 KB. DynamoDB non calcola gli spazi vuoti nella determinazione delle dimensioni di una policy rispetto a tale limite.

`DeleteResourcePolicy`

È possibile inviare fino a 50 richieste API `DeleteResourcePolicy` al secondo su un gruppo di tabelle. Dopo una richiesta `PutResourcePolicy` riuscita per una singola tabella, non vengono supportate nuove richieste `DeleteResourcePolicy` per i successivi 15 secondi.

`BatchGetItem`

Una singola operazione `BatchGetItem`può recuperare un massimo di 100 elementi. La dimensione totale di tutti gli elementi recuperati non può essere superiore a 16 MB.

`BatchWriteItem`

Una singola operazione `BatchWriteItem` può contenere fino a 25 richieste `PutItem` o `DeleteItem`. La dimensione totale di tutti gli elementi scritti non può essere superiore a 16 MB.

`DescribeStream`

È possibile chiamare `DescribeStream` a una frequenza massima di 10 volte al secondo.

`DescribeTableReplicaAutoScaling`

`DescribeTableReplicaAutoScaling` metodo supporta solo 10 richieste al secondo.

`DescribeLimits`

`DescribeLimits` dovrebbe essere chiamato solo periodicamente. È possibile che si verifichino errori di limitazione se lo chiami più di una volta al minuto.

`DescribeContributorInsights`/`ListContributorInsights`/`UpdateContributorInsights`

`DescribeContributorInsights`, `ListContributorInsights` e `UpdateContributorInsights` dovrebbero essere richiamate solo periodicamente. DynamoDB supporta un massimo di cinque richieste al secondo per ciascuna di queste API.

`DescribeTable`/`ListTables`/`GetResourcePolicy`

È possibile inviare fino a 2.500 richieste al secondo di richieste di una combinazione di richieste API del piano di controllo (control-plane) di sola lettura (`DescribeTable`, `ListTables` e `GetResourcePolicy`). L'API `GetResourcePolicy` ha un limite individuale inferiore di 100 richieste al secondo.

`DescribeTimeToLive`

L'operazione `DescribeTimeToLive` è sottoposta a limitazione (della larghezza di banda della rete) a 10 unità di richieste di lettura al secondo. Se si supera questo limite, DynamoDB restituisce un errore `ThrottlingException`.

`Query`

Il set di risultati di un'operazione `Query` è limitato a 1 MB per chiamata. Puoi utilizzare `LastEvaluatedKey` dalla risposta alle query per recuperare più risultati.

`Scan`

Il set di risultati di un'operazione `Scan` è limitato a 1 MB per chiamata. Puoi utilizzare `LastEvaluatedKey` dalla risposta alle verifiche per recuperare più risultati.

`UpdateKinesisStreamingDestination`

Quando si eseguono operazioni `UpdateKinesisStreamingDestination`, è possibile impostare `ApproximateCreationDateTimePrecision` su un nuovo valore per un massimo di 3 volte in un periodo di 24 ore.

`UpdateTableReplicaAutoScaling`

Il metodo `UpdateTableReplicaAutoScaling` supporta solo 10 richieste al secondo.

`UpdateTableTimeToLive`

Il metodo `UpdateTableTimeToLive` supporta solo una richiesta per tabella specificata all'ora per abilitare o disabilitare `Time to Live (TTL)`. Questa modifica potrebbe richiedere fino a un'ora per l'elaborazione completa. Qualsiasi ulteriore chiamata `UpdateTimeToLive` alla stessa tabella durante questo lasso di tempo di un'ora ha come risultato una ValidationException.

Crittografia a riposo per DynamoDB

Puoi passare da una Chiave di proprietà di AWS, a una Chiave gestita da AWS, a una chiave gestita dal cliente per un massimo di quattro volte, in qualunque momento durante un periodo di 24 ore, dal momento della creazione della tabella. E se non vi sono state modifiche nelle ultime 6 ore, sarà consentita un'ulteriore modifica. Questo porta il numero massimo di modifiche in un giorno a 8 (quattro modifiche nelle prime sei ore e una modifica per ciascuna delle successive finestre di sei ore in un giorno).

Puoi cambiare le chiavi di crittografia per utilizzare una Chiave di proprietà di AWS ogni volta che lo ritieni necessario, anche se la quota di cui sopra è stata esaurita.

A meno che tu non ne richieda una quantità maggiore, le quote sono le seguenti. Per richiedere un aumento delle quote di servizio, consulta https://aws.amazon.com/support.

Documentazione di riferimento dell'API DynamoDB

Il riferimento alle API di Amazon DynamoDB contiene un elenco completo delle operazioni supportate da:

- DynamoDB
- DynamoDB Streams
- DynamoDB Accelerator (DAX)

Risoluzione dei problemi di Amazon DynamoDB

Negli argomenti seguenti vengono forniti suggerimenti per la risoluzione dei problemi relativi a errori e problemi che potrebbero verificarsi durante l'utilizzo di Amazon DynamoDB. In caso di problemi non elencati qui di seguito, è possibile utilizzare il pulsante Feedback in questa pagina per segnalarne l'assenza.

Per ulteriori suggerimenti sulla risoluzione dei problemi e per risposte a domande comuni relative al supporto, visitare il Knowledge Center di AWS.

Argomenti

- Risoluzione degli errori interni del server in Amazon DynamoDB
- Risoluzione dei problemi di latenza in Amazon DynamoDB
- Risoluzione dei problemi di limitazione (della larghezza di banda della rete) in Amazon DynamoDB

Risoluzione degli errori interni del server in Amazon DynamoDB

In DynamoDB, gli errori interni del server (500 errori) indicano che il servizio non è in grado di soddisfare la richiesta. Questi errori possono verificarsi per vari motivi, ad esempio problemi transitori di rete nel parco istanze, problemi di infrastruttura, problemi relativi ai nodi di archiviazione e altro ancora.

È possibile che si verifichino alcuni errori interni del server durante il ciclo di vita della tabella DynamoDB. Questa evenienza è prevista in ragione della natura distribuita del servizio e non dovrebbe di norma costituire motivo di preoccupazione. DynamoDB ripara e risolve automaticamente gli eventuali problemi temporanei con il servizio in tempo reale, senza richiedere alcun intervento da parte dell'utente. Tuttavia, in caso di numeri costantemente elevati di errori interni del server nelle richieste alla tabella (come mostrato nella metrica the section called "SystemErrors"), è opportuno indagare ulteriormente.

Argomenti

- Indagine degli errori interni del server
- Ridurre al minimo l'impatto degli errori interni del server
- Miglioramento della consapevolezza operativa

Indagine degli errori interni del server

In caso di riscontro di errori interni del server nella tabella DynamoDB, considera queste opzioni:

1. Controlla la Dashboard AWS Health.

   Per identificare il problema, la prima fase è controllare la Dashboard AWS Service Health e la Dashboard Stato dell'account AWS. Queste dashboard forniscono informazioni preziose su eventuali problemi a livello di servizio, tabelle interessate, problemi in corso e sulla causa principale una volta risolto il problema.

   L'analisi dei dettagli presenti in queste dashboard consentirà di comprendere meglio lo stato corrente dei Servizi AWS in uso e gli eventuali problemi potenziali che interessano l'account. Queste informazioni possono aiutare a determinare le fasi successive per risolvere il problema e ridurre al minimo le interruzioni delle operazioni.
2. Contatta il Supporto.

   Errori prolungati nelle richieste potrebbero indicare un problema con il servizio. In linea generale, un tasso di errore complessivo pari o superiore all'1% negli ultimi 15 minuti indica che è opportuno procedere con l'inoltro del problema al team del Supporto AWS. Per maggiori informazioni, consulta l'Accordo sul livello di servizio di DynamoDB.

   Durante l'apertura di un caso con il team del Supporto AWS, fornisci i seguenti dettagli per velocizzare il processo di risoluzione dei problemi:

   - Database DDB, tabelle o indici secondari interessati
   - Finestra temporale in cui sono stati osservati gli errori
   - ID di richiesta DynamoDB, come `4KBNVRGD25RG1KE09UT4V3FQDJVV4KQNS05AEMVJF66Q9ASUAAJG`, reperibile nei log delle applicazioni.

   L'inclusione di questi dettagli nella richiesta di supporto aiuterà il team AWS a comprendere il problema e offrire una risoluzione più rapida. Qualora non si disponga degli ID delle richieste è comunque necessario registrare log del caso con gli altri dettagli disponibili.

Ridurre al minimo l'impatto degli errori interni del server

In caso di errori interni del server durante l'utilizzo di DynamoDB, prendi in considerazione le seguenti best practice per ridurne al minimo l'impatto sull'applicazione:

- Usa backoff e ripetizioni dei tentativi: i comportamenti degli SDK predefiniti di DynamoDB sono progettati per trovare il giusto equilibrio per la maggior parte delle applicazioni in termini di strategia di backoff e ripetizioni dei tentativi. Tuttavia, è possibile regolare queste impostazioni in base alla tolleranza dell'applicazione ai tempi di inattività e ai requisiti delle prestazioni. Approfondisci i backoff e le ripetizioni dei tentativi per capire come ottimizzare queste impostazioni di ripetizione dei tentativi.
- Usa letture a coerenza finale: se l'applicazione non richiede un'elevata consistenza di lettura, prendi in considerazione l'utilizzo di letture a coerenza finale. Queste letture hanno un costo inferiore e rendono meno probabile il verificarsi di problemi temporanei dovuti a errori interni del server, in quanto vengono servite da uno qualsiasi dei nodi di archiviazione disponibili. Per ulteriori informazioni, consulta Coerenza di lettura di DynamoDB.

Miglioramento della consapevolezza operativa

Mantenere un'elevata disponibilità e affidabilità delle applicazioni è fondamentale nel panorama digitale odierno. Un aspetto chiave è il monitoraggio proattivo degli errori interni del server nelle tabelle DynamoDB e negli indici secondari globali (GSI). Creando allarmi CloudWatch per monitorare questi errori è possibile acquisire una migliore consapevolezza operativa e ricevere avvisi sui potenziali problemi prima che abbiano un impatto sugli utenti finali. Questo approccio è in linea con il pilastro Eccellenza operativa del framework AWS Well-Architected, poiché garantisce che il carico di lavoro DynamoDB sia ottimizzato per prestazioni, sicurezza e affidabilità.

Creazione di allarmi CloudWatch

È necessario impostare gli allarmi CloudWatch sulle tabelle DynamoDB per ricevere notifiche in caso di numeri costantemente elevati di errori interni del server anziché, così da non dover analizzare le metriche manualmente. Questo approccio si lega al pilastro Eccellenza operativa del framework Well-Architected per qualsiasi carico di lavoro su AWS. Consulta Utilizzo di DynamoDB Well-Architected Lens per ottimizzare il carico di lavoro di DynamoDB per saperne di più su come progettare le tabelle DynamoDB secondo i principi del Well-Architecting.

Questi allarmi utilizzano metriche matematiche personalizzate per calcolare la percentuale di richieste non riuscite per un periodo di 5 minuti. La best practice consigliata consiglia di configurare l'allarme in modo che entri nello stato `ALARM` quando 3 punti dati consecutivi superano la soglia dell'1%, il che significa che complessivamente l'1% delle richieste non va a buon fine entro un periodo di 15 minuti.

L'esempio seguente è un modello CloudFormation che può aiutare a creare allarmi CloudWatch nella tabella e nel GSI.

```
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'
```

Risoluzione dei problemi di latenza in Amazon DynamoDB

Se la latenza del carico di lavoro è elevata, è possibile analizzare la metrica CloudWatch `SuccessfulRequestLatency` e controllare la latenza media e la latenza mediana attraverso le metriche percentili (p50) per controllare se è correlata a DynamoDB. Una certa variabilità nella metrica `SuccessfulRequestLatency` restituita è normale e i picchi occasionali (in particolare nelle statistiche `Maximum` e nei percentili elevati) non devono destare preoccupazione. Tuttavia, se le statistiche `Average` o il p50 (mediana) mostrano un forte aumento persistente, è opportuno controllare la Dashboard AWS Service Health e la Dashboard Personal Health per ulteriori informazioni. Alcune possibili cause includono la dimensione dell'elemento nella tabella (un elemento da 1 KB e un elemento da 400 KB avranno una latenza diversa) o la dimensione della query (10 elementi rispetto a 100 elementi).

Le metriche percentili (p99, p90, ecc.) possono aiutare a comprendere meglio la distribuzione della latenza. Ad esempio:

- p50 (mediana) mostra la latenza tipica del carico di lavoro.
- p90 mostra che il 90% delle richieste è più veloce di questo valore.
- p99 aiuta a identificare la latenza peggiore che riguarda l'1% delle richieste.

Valori di p99 elevati con valori p50 normali potrebbero indicare problemi sporadici che interessano una piccola parte delle richieste, mentre valori p50 costantemente elevati potrebbero suggerire un peggioramento delle prestazioni.

> **Note**
>
> Per analizzare valori percentili personalizzati (come p99,9), è possibile inserire manualmente il percentile desiderato (ad esempio, p99,9) nel campo delle statistiche delle metriche CloudWatch. Ciò consente di valutare le distribuzioni di latenza oltre i percentili predefiniti elencati nel menu a discesa.

Sono previste alcune variazioni nelle metriche di latenza, in particolare nei percentili più elevati, che possono essere considerate il risultato di operazioni in background basate su DynamoDB che aiutano a mantenere un'elevata disponibilità e durabilità dei dati archiviati nelle tabelle DynamoDB oppure il risultato di problemi transitori dell'infrastruttura.

Se necessario, considera la possibilità di aprire una richiesta di supporto con Supporto AWS e continua a valutare le opzioni di fallback disponibili per la tua applicazione (ad esempio l'evacuazione di una regione se disponi di un'architettura a più regioni) in base ai tuoi runbook. All'apertura di un caso di supporto è necessario registrare log degli ID delle richieste per le richieste lente e fornire questi ID al Supporto AWS.

La metrica `SuccessfulRequestLatency` misura solo la latenza interna al servizio DynamoDB; l'attività lato client e i tempi di accesso alla rete non sono inclusi. Per avere altri dettagli sulla latenza complessiva delle chiamate dal client al servizio DynamoDB, puoi abilitare la registrazione dei parametri di latenza nel tuo SDK AWS.

> **Note**
>
> Per la maggior parte delle operazioni singleton (operazioni che si applicano a un singolo elemento specificando completamente il valore della chiave primaria), DynamoDB fornisce un valore di `Average SuccessfulRequestLatency` pari a pochissimi millisecondi. Questo valore non include il sovraccarico di trasporto per il codice chiamante che accede all'endpoint DynamoDB. Per le operazioni sui dati con più elementi, la latenza varia in base a fattori quali la dimensione del set di risultati, la complessità delle strutture di dati restituite e le eventuali espressioni di condizione e di filtro applicate. Per operazioni con più elementi ripetute sullo stesso set di dati con gli stessi parametri, DynamoDB fornisce un'elevata coerenza per `Average SuccessfulRequestLatency`.

Prendi in considerazione una o più delle seguenti strategie per ridurre la latenza:

- Riutilizzo delle connessioni: le richieste DynamoDB vengono effettuate tramite una sessione autenticata che per impostazione predefinita è HTTPS. L'avvio della connessione richiede più round trip e richiede tempo, pertanto la latenza della prima richiesta è superiore al normale rispetto alle richieste successive che riutilizzano la connessione. Le richieste su una connessione già inizializzata forniscono una bassa latenza costante di DynamoDB. Per evitare il sovraccarico della latenza necessaria nello stabilire nuove connessioni, è possibile implementare un meccanismo

"keep-alive" inviando una richiesta `GetItem` ogni 30 secondi quando non vengono effettuate altre richieste.

- Usa letture a coerenza finale: se la tua applicazione non richiede un'elevata consistenza di lettura, puoi utilizzare le letture a coerenza finale. Le letture a coerenza finale hanno un costo inferiore e possono provenire da più zone di disponibilità, permettendo la selezione di una zona di disponibilità ubicata nella stessa posizione del richiedente, il che riduce la latenza. Per ulteriori informazioni, consulta Coerenza di lettura di DynamoDB.
- Implementazione dell'hedging delle richieste: per requisiti di latenza p99 molto bassi, prendi in considerazione l'implementazione dell'hedging delle richieste. Con l'hedging delle richieste, se la richiesta iniziale non riceve una risposta abbastanza rapidamente, invia una seconda richiesta equivalente e considera valida la prima risposta ricevuta. Ciò migliora la latenza di coda al costo di alcune richieste aggiuntive. È possibile decidere quanto aspettare prima di inviare la seconda richiesta. L'hedging è più facile per le letture. Per quanto riguarda le scritture, utilizza l'ordinamento basato sul timestamp per garantire che le richieste con hedging vengano trattate come se si verificassero al momento del primo tentativo, evitando aggiornamenti fuori ordine. Questo approccio è stato discusso in Timestamp writes for write hedging in Amazon DynamoDB.
- Regolazione del timeout delle richieste e del comportamento delle ripetizioni dei tentativi: il percorso dal client a DynamoDB attraversa molti componenti, ognuno dei quali è progettato sulla base della ridondanza. Prendi in considerazione i seguenti aspetti:
  - Resilienza della rete
  - Timeout dei pacchetti TCP
  - L'architettura distribuita di DynamoDB

  I comportamenti degli SDK predefiniti sono ottimizzati per la maggior parte delle applicazioni. Tuttavia, è possibile implementare una strategia per anticipare l'errore (fail-fast) e regolare le impostazioni di timeout. Le richieste che richiedono molto più tempo del normale hanno meno probabilità di esito finale positivo. Attraverso l'anticipo dell'errore (fail fast) e la ripetizione dei tentativi, è possibile ottenere rapidamente un esito positivo attraverso un percorso diverso. La procedura è simile all'hedging delle richieste, ma termina la prima richiesta invece di consentirle di procedere.

  Evita di impostare valori di timeout troppo bassi. Timeout troppo bassi possono causare problemi di disponibilità indotti dal client. Ad esempio, un timeout del socket di 50 millisecondi potrebbe causare errori di connessione durante i picchi di latenza di rete, ad esempio quando ci si avvicina ai limiti di larghezza di banda delle istanze Amazon EC2 per il traffico a flusso singolo. Valuta

attentamente i vantaggi di timeout più bassi rispetto ai potenziali rischi per la disponibilità delle applicazioni e preferisci l'hedging rispetto ai timeout brevi.

Per un'utile discussione su questo argomento, consulta Tuning AWS Java SDK HTTP request settings for latency-aware Amazon DynamoDB applications.

- Riduci la distanza tra il client e l'endpoint DynamoDB: se hai utenti distribuiti a livello globale, prendi in considerazione l'utilizzo di Tabelle globali: replica multi-attiva e multi-Regione. Con le tabelle globali è possibile replicare la tabella su Regioni AWS specificate in cui si desidera che la tabella sia disponibile. È possibile posizionare una copia dei dati più vicino all'utente finale per ridurre la latenza di rete durante le operazioni di lettura e scrittura. Per ulteriori informazioni sull'uso efficace delle tabelle globali di DynamoDB, consulta Utilizzo delle tabelle globali di Amazon DynamoDB nel Prontuario AWS.
- Utilizzo della caching: in presenza di traffico a lettura intensiva, prendi in considerazione l'adozione di uno dei seguenti servizi di caching:
  - DynamoDB Accelerator (DAX): un sistema di cache in memoria completamente gestito e a elevata disponibilità per DynamoDB che migliora fino a 10 volte le prestazioni anche in presenza di milioni di richieste al secondo, consentendo di ottenere risposte in microsecondi invece che in millisecondi. Per ulteriori informazioni su DAX, consulta Accelerazione in memoria con DynamoDB Accelerator (DAX):
  - Amazon ElastiCache: un servizio di caching in memoria completamente gestito che può essere integrato con DynamoDB per migliorare le prestazioni di lettura attraverso il modello cache-aside. Per ulteriori informazioni, consulta Integrating Amazon DynamoDB and Amazon ElastiCache by using read-through caching nel Prontuario AWS.

Risoluzione dei problemi di limitazione (della larghezza di banda della rete) in Amazon DynamoDB

DynamoDB implementa la limitazione (della larghezza di banda della rete) per due scopi principali: mantenere le prestazioni complessive del servizio e il controllo dei costi. La limitazione (della larghezza di banda della rete) serve sia come protezione intenzionale che impedisce il peggioramento delle prestazioni quando i tassi di consumo superano la capacità, sia come meccanismo di controllo dei costi quando si raggiungono i limiti massimi di throughput o delle quote di servizio. Quando si verifica la limitazione (della larghezza di banda della rete), DynamoDB restituisce eccezioni specifiche con informazioni dettagliate sul motivo per cui la richiesta è stata limitata e su quale risorsa è stata interessata. Ogni motivo di limitazione (della larghezza di banda della rete)

corrisponde a metriche CloudWatch specifiche che forniscono informazioni aggiuntive sulla frequenza e sui modelli degli eventi di limitazione.

Il diagramma seguente illustra i quattro scenari principali in cui DynamoDB implementa la limitazione (della larghezza di banda della rete) protettiva:

1. Superamento del throughput dell'intervallo di chiavi (in entrambe le modalità):

   L'utilizzo diretto in partizioni specifiche supera i limiti di throughput a livello di partizione interna.

2. Superamento del throughput allocato (in modalità con provisioning):

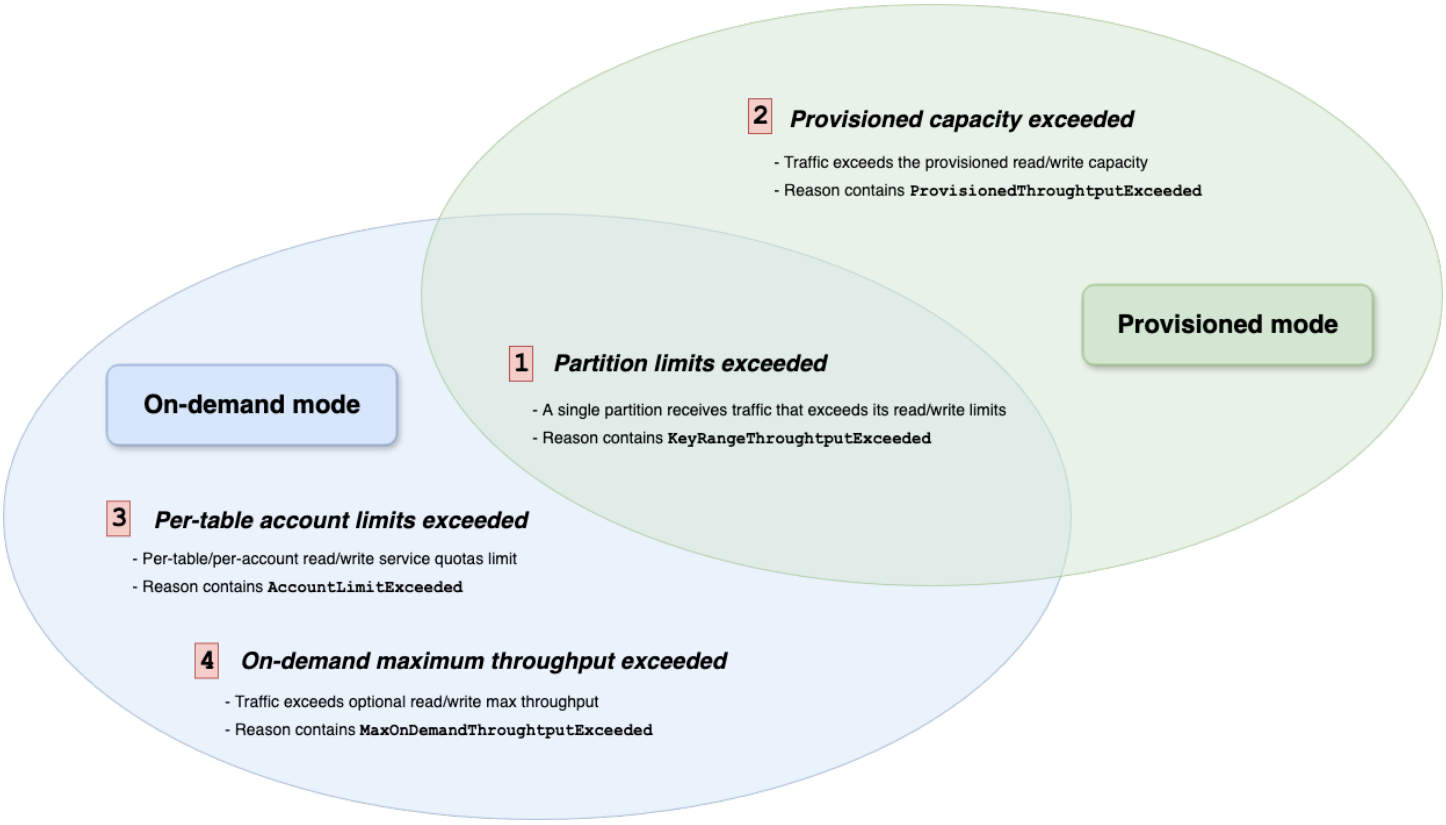
   L'utilizzo supera le unità di capacità allocate (lettura o scrittura) configurate per una tabella o un indice secondario globale (GSI).

3. Superamento delle quote di servizio a livello di account (in modalità on demand):

   L'utilizzo fa sì che una tabella o un GSI superino le quote di servizio a livello di account per tabella per il throughput di lettura/scrittura nella Regione AWS corrente. Queste quote fungono da protezione backstop e possono essere aumentate.

4. Superamento del throughput massimo on demand (in modalità on demand):

   L'utilizzo supera i limiti di throughput massimo configurati per una tabella o un GSI. Si tratta di limiti configurati specificamente per il controllo dei costi.



Questa guida è organizzata per conoscere e lavorare con la limitazione (della larghezza di banda della rete) in DynamoDB. Nella prima parte viene illustrato come a identificare il tipo specifico di limitazione (della larghezza di banda della rete) che interessa il carico di lavoro attraverso un quadro diagnostico.

Successivamente, la sezione relativa alla guida alla risoluzione offre indicazioni specifiche per ogni scenario di limitazione (della larghezza di banda della rete), incluse le metriche CloudWatch da monitorare per il rilevamento e l'analisi e le procedure consigliate per l'ottimizzazione. Seguendo questo approccio strutturato, è possibile diagnosticare meglio la causa principale della limitazione (della larghezza di banda della rete) e implementare la soluzione appropriata per garantire che le tabelle DynamoDB funzionino in modo efficiente.

Per iniziare, segui the section called "Diagnosi della limitazione (della larghezza di banda della rete)" per scoprire come identificare quale tipo di limitazione (della larghezza di banda della rete) influisce sul tuo carico di lavoro e come implementare la strategia di risoluzione consigliata.

Argomenti

- Diagnosi della limitazione (della larghezza di banda della rete)
- Guida alla risoluzione della limitazione (della larghezza di banda della rete) in DynamoDB

- Funzionamento della limitazione (della larghezza di banda della rete) e della contropressione di scrittura sugli indici secondari globali in DynamoDB
- Metriche di limitazione (della larghezza di banda della rete) di CloudWatch

Diagnosi della limitazione (della larghezza di banda della rete)

Quando l'applicazione subisce una limitazione (della larghezza di banda della rete), DynamoDB fornisce informazioni dettagliate sulle eccezioni e metriche CloudWatch mirate per aiutare a diagnosticare questi eventi.

In questa sezione viene presentato un approccio sistematico alla comprensione degli eventi di limitazione (della larghezza di banda della rete) nelle applicazioni DynamoDB. Viene mostrato come interpretare le eccezioni di limitazione (della larghezza di banda della rete), correlarle alle metriche CloudWatch per informazioni più approfondite e capire quali modifiche potrebbero ridurre questo tipo di limitazione nelle applicazioni DynamoDB.

Funzionamento delle eccezioni di limitazione (della larghezza di banda della rete)

Quando DynamoDB limita sottopone a limitazione (della larghezza di banda della rete) una richiesta, restituisce eccezioni specifiche con informazioni diagnostiche dettagliate. Ad esempio, in Java, queste includono `ProvisionedThroughputExceededException`, `RequestLimitExceeded` o `ThrottlingException`.

Ogni eccezione include `ThrottlingReasons` una raccolta di dati `ThrottlingReason` singoli contenenti due campi chiave per aiutare a identificare e comprendere la limitazione (della larghezza di banda della rete):

- Un motivo: un campo concatenato che segue il formato `<ResourceType><OperationType><LimitType>`
- Un ARN della risorsa: il nome della risorsa Amazon (ARN) della tabella o dell'indice interessati

Il campo del motivo segue un modello coerente che aiuta a capire esattamente cosa sta succedendo:

- ResourceType (A cosa viene applicata la limitazione (della larghezza di banda della rete)): `Table` o `Index`
- OperationType (che tipo di operazione): `Read` o `Write`
- LimitType (Perché si è verificata la limitazione (della larghezza di banda della rete)):

- `KeyRangeThroughputExceeded`: si verifica quando una partizione specifica che supporta la tabella o l'indice ha utilizzato una capacità di lettura o scrittura oltre i limiti di throughput interni per partizione.
- `ProvisionedThroughputExceeded`: si verifica su una tabella con provisioning o su un indice secondario globale quando il tasso di consumo di lettura o scrittura ha superato la quantità allocata.
- `AccountLimitExceeded`: si verifica in una tabella o in un indice on demand quando il tasso di consumo di lettura o scrittura ha superato il tasso di consumo massimo per una tabella e i relativi indici impostato a livello di account. È possibile richiedere un aumento di questa quota.
- `MaxOnDemandThroughputExceeded`: si verifica su una tabella o un indice on demand quando il tasso di consumo di lettura o scrittura ha superato il tasso di consumo massimo fornito dall'utente configurato per la tabella o l'indice. È possibile aumentare in prima persona questo valore a qualsiasi valore fino al limite dell'account o impostarlo su –1 per indicare che non ci sono limiti forniti dall'utente.

L'ARN della risorsa identifica esattamente quale tabella o quale indice sono sottoposti a limitazione (della larghezza di banda della rete):

- Per le tabelle: `arn:aws:dynamodb:[region]:[account-id]:table/[table-name]`
- Per gli indici: `arn:aws:dynamodb:[region]:[account-id]:table/[table-name]/index/[index-name]`

Esempi di motivi di limitazione (della larghezza di banda della rete) completa:

- `TableReadProvisionedThroughputExceeded`
- `IndexWriteAccountLimitExceeded`

In questo modo è possibile identificare esattamente quale risorsa viene sottoposta a limitazione (della larghezza di banda della rete), che tipo di operazione l'ha causata e perché si è verificata la limitazione.

Eccezioni di esempio

Esempio 1: superamento della capacità allocata su 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
        }
    }
}
```

In questo esempio, l'applicazione riceve una `ProvisionedThroughputExceededException` con il motivo `IndexWriteProvisionedThroughputExceeded`. Alle scritture su `OrderDateIndex` viene applicata una limitazione (della larghezza di banda della rete) perché l'utilizzo di scrittura ha superato la capacità di scrittura allocata configurata del GSI.

Esempio 2: superamento del throughput massimo on demand

```
{
    "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
        }
    }
```

```
}
```

In questo esempio, le letture dalla tabella `UserSessions` subiscono una limitazione (della larghezza di banda della rete) perché superano il limite massimo di throughput on demand configurato nella tabella.

Framework di diagnosi della limitazione (della larghezza di banda della rete) di DynamoDB

Quando l'applicazione riscontra una limitazione (della larghezza di banda della rete), segui questa procedura per diagnosticare e risolvere il problema.

Fase 1: analizzare i dettagli di **`ThrottlingReason`**

1. Controlla il campo motivo per identificare il motivo specifico della limitazione (della larghezza di banda della rete). Il motivo descrive in dettaglio il tipo di risorsa sottoposta a limitazione (della larghezza di banda della rete) (tabella o indice), il tipo di operazione che ha causato l'evento di limitazione (lettura o scrittura) e il tipo di limite superato (partizione, throughput allocato, limite dell'account).
2. Controlla il campo resourceArn per identificare quale risorsa (tabella o GSI) subisce la limitazione (della larghezza di banda della rete).
3. Utilizza queste informazioni combinate per comprendere il contesto completo del problema della limitazione (della larghezza di banda della rete).

   Ad esempio, si consideri questo scenario in cui compare la seguente eccezione `ProvisionedThroughputExceededException` con motivo di limitazione (della larghezza di banda della rete) `TableWriteKeyRangeThroughputExceeded`. Il resourceARN interessato è `arn:aws:dynamodb:us-west-2:123456789012:table/CustomerOrders`.

   Questa combinazione informa che le operazioni di scrittura sulla tabella `CustomerOrders` sono sottoposte a limitazione (della larghezza di banda della rete) e la limitazione si verifica a livello di partizione (non a livello di tabella, che verrebbe visualizzato come `TableWriteProvisionedThroughputExceeded`). La causa principale è che è stata superata la capacità di throughput massima per un valore o un intervallo specifico della chiave di partizione, il che indica un problema di partizione calda.

   La comprensione di questa relazione tra gli elementi di eccezione consente di implementare la strategia di mitigazione appropriata, in questo caso, risolvere il problema della partizione calda anziché aumentare la capacità complessiva allocata alla tabella.

Fase 2: identificare e analizzare le metriche CloudWatch correlate

1. Identifica le metriche: ogni motivo di limitazione (della larghezza di banda della rete) in DynamoDB corrisponde direttamente a metriche CloudWatch specifiche che è possibile monitorare per tracciare e analizzare gli eventi di limitazione. È possibile ricavare sistematicamente i nomi delle metriche CloudWatch appropriate a partire dal motivo della limitazione (della larghezza di banda della rete).
2. Abbina il motivo della limitazione (della larghezza di banda della rete) alle metriche CloudWatch corrispondenti utilizzando questa tabella di riferimento:

   Motivi completi della limitazione (della larghezza di banda della rete) e riferimento alle metriche CloudWatch

Categoria	Motivo della limitazione (della larghezza di banda della rete)	Metriche CloudWatch principali
Superamento della capacità allocata	TableReadProvisionedThroughputExceeded	ReadProvisionedThroughputThrottleEvents
	TableWriteProvisionedThroughputExceeded	WriteProvisionedThroughputThrottleEvents
	IndexReadProvisionedThroughputExceeded	ReadProvisionedThroughputThrottleEvents (GSI)
	IndexWriteProvisionedThroughputExceeded	WriteProvisionedThroughputThrottleEvents (GSI)
Superamento dei limiti di partizione	TableReadKeyRangeThroughputExceeded	ReadKeyRangeThroughputThrottleEvents
	TableWriteKeyRangeThroughputExceeded	WriteKeyRangeThroughputThrottleEvents

Categoria	Motivo della limitazione (della larghezza di banda della rete)	Metriche CloudWatch principali
	IndexReadKeyRangeThroughputExceeded	ReadKeyRangeThroughputThrottleEvents (GSI)
	IndexWriteKeyRangeThroughputExceeded	WriteKeyRangeThroughputThrottleEvents (GSI)
Superamento del limite massimo on demand	TableReadMaxOnDemandThroughputExceeded	ReadMaxOnDemandThroughputThrottleEvents
	TableWriteMaxOnDemandThroughputExceeded	WriteMaxOnDemandThroughputThrottleEvents
	IndexReadMaxOnDemandThroughputExceeded	ReadMaxOnDemandThroughputThrottleEvents (GSI)
	IndexWriteMaxOnDemandThroughputExceeded	WriteMaxOnDemandThroughputThrottleEvents (GSI)
Superamento dei limiti dell’account	TableReadAccountLimitExceeded	ReadAccountLimitThrottleEvents
	TableWriteAccountLimitExceeded	WriteAccountLimitThrottleEvents
	IndexReadAccountLimitExceeded	ReadAccountLimitThrottleEvents (GSI)

Categoria	Motivo della limitazione (della larghezza di banda della rete)	Metriche CloudWatch principali
	IndexWriteAccountLimitExceeded	WriteAccountLimitThrottleEvents (GSI)

Ad esempio, se si riceve `IndexWriteProvisionedThroughputExceeded`, come minimo si dovrebbe monitorare la metrica CloudWatch `WriteProvisionedThroughputThrottleEvents` per l'indice specifico identificato nel `ResourceArn`.

3. Monitora queste metriche in CloudWatch per comprendere la frequenza e la tempistica degli eventi di limitazione (della larghezza di banda della rete), distinguere tra limitazione di lettura e scrittura, identificare gli schemi temporali in cui la limitazione aumenta e tenere traccia degli andamenti di utilizzo della capacità.

   DynamoDB pubblica metriche dettagliate per ogni tabella e indice secondario globale. Le metriche (`ReadThrottleEvents`,`WriteThrottleEvents` e `ThrottledRequests`) aggregano tutti gli eventi di limitazione (della larghezza di banda della rete) della tabella e dei relativi indici.

Fase 3: identificare le chiavi con limitazione (della larghezza di banda della rete) e i tassi di accesso elevati con Contributor Insights di CloudWatch (per la limitazione relativa alle partizioni)

Se sono stati identificati problemi relativi alle partizioni nella Fase 1 (come errori `KeyRangeThroughputExceeded`), Contributor Insights di CloudWatch per DynamoDB può aiutare a diagnosticare quali chiavi specifiche generano traffico e registrano eventi di limitazione (della larghezza di banda della rete) nella tabella o nell'indice.

1. Abilita Contributor Insights di CloudWatch per la tabella o l'indice sottoposto a limitazione (della larghezza di banda della rete) in base al `ResourceARN`.

   È possibile scegliere la modalità Chiavi con limitazione (della larghezza di banda della rete) per concentrarsi sulle chiavi con maggiore limitazione. Questa modalità è ideale per il monitoraggio continuo in quanto elabora gli eventi solo quando si verifica una limitazione (della larghezza di banda della rete). In alternativa, la modalità Chiavi con accessi e limitazione (della larghezza di banda della rete) consente di osservare i modelli nelle chiavi a cui si accede più spesso.

2. Analizza i report per identificare i modelli problematici. Cerca chiavi con tassi di accesso o di limitazione (della larghezza di banda della rete) sproporzionatamente elevati e metti in correlazione i modelli di limitazione e traffico. È possibile creare dashboard integrate combinando grafici di Contributor Insights e metriche CloudWatch DynamoDB.

Per informazioni dettagliate sull'attivazione e l'utilizzo di Contributor Insights di CloudWatch, consulta Using CloudWatch Contributor Insights for DynamoDB.

Fase 4: determinare la soluzione appropriata

Dopo aver diagnosticato la causa specifica della limitazione (della larghezza di banda della rete), implementa la soluzione consigliata in base al contesto specifico. La soluzione appropriata dipende da diversi fattori, tra cui lo scenario di limitazione (della larghezza di banda della rete), la modalità di capacità della tabella, le decisioni di progettazione delle chiavi e delle tabelle, i modelli di accesso e l'efficienza delle query, la configurazione degli indici globali e secondari, l'architettura generale del sistema e i punti di integrazione.

Per soluzioni dettagliate per affrontare gli scenari di limitazione (della larghezza di banda della rete) specifici, consulta la sezione the section called "Guida alla risoluzione". Questa risorsa fornisce strategie di correzione mirate personalizzate in base al particolare motivo di limitazione (della larghezza di banda della rete) e alla configurazione della modalità di capacità.

Fase 5: monitorare i progressi

1. Tieni traccia delle metriche CloudWatch che corrispondono allo scenario di limitazione (della larghezza di banda della rete).
2. L'efficacia delle strategie di mitigazione può essere verificata tramite la diminuzione degli eventi di questo tipo.

Guida alla risoluzione della limitazione (della larghezza di banda della rete) in DynamoDB

In questa sezione vengono fornite indicazioni mirate per la risoluzione per ogni motivo di limitazione (della larghezza di banda della rete) specifico che DynamoDB può restituire. Ogni voce include approcci di risoluzione suggeriti basati sulle best practice e le metriche CloudWatch corrispondenti da monitorare.

DynamoDB implementa 16 diversi motivi di limitazione (della larghezza di banda della rete) in quattro categorie principali. Utilizza i motivi di limitazione (della larghezza di banda della rete) indicati nell'eccezione dell'applicazione per accedere direttamente alle linee guida pertinenti.

Superamento del throughput dell'intervallo di chiavi (partizioni calde)

Questi motivi di limitazione (della larghezza di banda della rete) si verificano quando singole partizioni superano i limiti di throughput, con ripercussioni sia sulla modalità con provisioning che on demand:

- TableReadKeyRangeThroughputExceeded
- TableWriteKeyRangeThroughputExceeded
- IndexReadKeyRangeThroughputExceeded
- IndexWriteKeyRangeThroughputExceeded

Superamento del throughput allocato

Questi motivi di limitazione (della larghezza di banda della rete) si verificano quando i tassi di consumo superano i limiti di capacità allocata in modalità con provisioning:

- TableReadProvisionedThroughputExceeded
- TableWriteProvisionedThroughputExceeded
- IndexReadProvisionedThroughputExceeded
- IndexWriteProvisionedThroughputExceeded

Superamento dei limiti dell'account

Questi motivi di limitazione (della larghezza di banda della rete) si verificano quando i tassi di consumo superano le quote di throughput a livello di account nella Regione AWS:

- TableReadAccountLimitExceeded
- TableWriteAccountLimitExceeded
- IndexReadAccountLimitExceeded
- IndexWriteAccountLimitExceeded

Superamento del throughput massimo on demand

Questi motivi di limitazione (della larghezza di banda della rete) si verificano quando i tassi di consumo superano i limiti di throughput massimo configurati in modalità on demand:

- TableReadMaxOnDemandThroughputExceeded
- TableWriteMaxOnDemandThroughputExceeded
- IndexReadMaxOnDemandThroughputExceeded
- IndexWriteMaxOnDemandThroughputExceeded

1. Superamento del throughput dell'intervallo di chiavi (partizioni calde)

Amazon DynamoDB applica limiti di throughput specifici a livello di partizione sia per la tabella che per l'indice secondario globale (GSI). Ogni partizione ha un numero massimo di unità di capacità di lettura (RCUs) e unità di capacità di scrittura (WCUs) al secondo. Quando le partizioni ricevono un traffico concentrato che supera questi limiti, subiscono una limitazione (della larghezza di banda della rete), mentre altre operazioni possono rimanere sottoutilizzate, creando "partizioni calde". La limitazione (della larghezza di banda della rete) a livello di partizione di DynamoDB opera in modo indipendente per le letture e le scritture: una partizione può limitare le letture mentre le scritture continuano normalmente o viceversa. Tale limitazione (della larghezza di banda della rete) può verificarsi anche quando la tabella o il GSI hanno una capacità complessiva sufficiente. Per ulteriori informazioni su:

- Limiti delle partizioni di DynamoDB e progettazione efficace delle chiavi di partizione per la prevenzione delle partizioni calde, consulta Best practice per la progettazione delle chiavi di partizione e relativo uso in DynamoDB.
- Concetti generali sulle partizioni e la distribuzione dei dati, consulta Partizioni in DynamoDB.
- Ulteriori indicazioni e scenari reali per la gestione delle chiavi di partizione e del throughput, consulta the section called "Risorse aggiuntive".

Quando singole partizioni superano i limiti di throughput, DynamoDB restituisce un tipo di motivo di limitazione (della larghezza di banda della rete) `KeyRangeThroughputExceeded` nell'eccezione di limitazione. Le informazioni identificano che una partizione sta registrando un traffico elevato e il tipo di operazione (lettura o scrittura) causa il problema.

Misure di mitigazione per il superamento del throughput dell'intervallo delle chiavi

Questa sezione fornisce indicazioni sulla risoluzione degli scenari di limitazione (della larghezza di banda della rete) a livello di partizione. Prima di utilizzare questa guida è necessario assicurarsi di aver identificato i motivi specifici della limitazione (della larghezza di banda della rete) nella gestione delle eccezioni dell'applicazione e di aver determinato il nome della risorsa Amazon (ARN) della risorsa interessata. Per informazioni su come recuperare i motivi della limitazione (della larghezza di banda della rete) e identificare le risorse limitate, consulta the section called "Framework di diagnosi".

Prima di approfondire gli specifici scenari di limitazione (della larghezza di banda della rete), verifica innanzitutto se il problema si risolve automaticamente:

- DynamoDB si adatta spesso alle partizioni calde tramite il suo meccanismo automatico. split-for-heat In caso di eventi di limitazione (della larghezza di banda della rete) che si interrompono dopo un breve periodo, è possibile che la tabella si sia già adattata suddividendo la partizione calda. Quando le partizioni si dividono, ogni nuova partizione gestisce una sezione più piccola dello spazio del keyspace, il che può aiutare a distribuire il carico in modo più uniforme. In molti casi, non sono richieste altre operazioni in quanto DynamoDB ha risolto automaticamente il problema.

  Per ulteriori informazioni sul meccanismo, vedere. split-for-heat the section called "Risorse aggiuntive"

Se la limitazione (della larghezza di banda della rete) persiste, consulta gli scenari specifici riportati di seguito per opzioni di correzione mirate:

- the section called "TableReadKeyRangeThroughputExceeded"
- the section called "TableWriteKeyRangeThroughputExceeded"
- the section called "IndexReadKeyRangeThroughputExceeded"
- the section called "IndexWriteKeyRangeThroughputExceeded"

TableReadKeyRangeThroughputExceeded

Quando avviene

Quando il tasso di consumo di una o più partizioni nella tabella DynamoDB supera il limite di throughput di lettura della partizione. Questa limitazione (della larghezza di banda della rete) si verifica indipendentemente dalla capacità totale allocata alla tabella e influisce sia sulle tabelle con

provisioning che on demand. Puoi monitorare le CloudWatch metriche the section called “Diagnosi e monitoraggio comuni” per analizzare il tuo evento di throttling.

Opzioni di correzione

Prendi in considerazione questa procedura per risolvere i problemi di limitazione (della larghezza di banda della rete):

Sia per le modalità con provisioning che on demand:

- Pre-riscalda la capacità: se la limitazione (della larghezza di banda della rete) persiste, controlla se la capacità della tabella è limitata dalla sua capacità the section called “Throughput a caldo”. Utilizza il throughput a caldo o aumenta in anticipo la capacità di lettura allocata per far fronte agli aumenti di traffico previsti. L’aumento del throughput a caldo migliora la capacità della tabella di gestire picchi di traffico improvvisi prima che si verifichi la limitazione (della larghezza di banda della rete). Nel tempo, se il throughput effettivo si avvicina costantemente ai livelli di throughput a caldo, DynamoDB può suddividere le partizioni occupate in base ai modelli di utilizzo osservati.
- Identifica le chiavi calde: se la tabella non ha risolto il problema automaticamente e il throughput a caldo è elevato o un aumento non è risultato utile, sarà necessario identificare le chiavi calde specifiche. Utilizza the section called “Identificazione dei tasti di scelta rapida mediante Contributor Insights CloudWatch” per determinare se alcuni valori di una particolare chiave di partizione non sono validi. È un primo passo per indirizzare gli sforzi di mitigazione in modo efficace. L’identificazione potrebbe non essere sempre semplice, in particolare nel caso di partizioni calde continue (in cui partizioni diverse si surriscaldano nel tempo) o quando la limitazione (della larghezza di banda della rete) è innescata da operazioni come le scansioni. Per questi scenari complessi, potrebbe essere necessario analizzare i modelli di accesso dell’applicazione e correlarli con la tempistica degli eventi di limitazione (della larghezza di banda della rete).
- A seconda del caso d'uso, prendi in considerazione l'utilizzo di letture che alla fine sono consistenti: passa da letture fortemente coerenti a letture sostanzialmente coerenti, che consumano la metà RCUs e possono immediatamente raddoppiare la capacità di lettura effettiva. Per le best practice sull’implementazione di letture a coerenza finale per ridurre l’utilizzo di capacità di lettura, consulta the section called “Coerenza di lettura di DynamoDB”.
- Migliora la progettazione delle chiavi di partizione: come soluzione a lungo termine, prendi in considerazione the section called “Miglioramento della progettazione delle chiavi di partizione” per distribuire l’accesso in modo più uniforme tra le partizioni. Questo approccio spesso fornisce la soluzione più completa ai problemi relativi alle partizioni calde risolvendo la causa principale. Tuttavia, richiede un’attenta pianificazione in quanto comporta sfide di migrazione significative.

TableWriteKeyRangeThroughputExceeded

Quando avviene

Quando il tasso di consumo di una o più partizioni nella tabella DynamoDB supera il limite di throughput di scrittura della partizione. Questa limitazione (della larghezza di banda della rete) si verifica indipendentemente dalla capacità totale allocata alla tabella e influisce sia sulle tabelle con provisioning che on demand. Puoi monitorare le CloudWatch metriche the section called “Diagnosi e monitoraggio comuni” per analizzare il tuo evento di limitazione.

Opzioni di correzione

Prendi in considerazione questa procedura per risolvere i problemi di limitazione (della larghezza di banda della rete):

Sia per le modalità con provisioning che on demand:

- Pre-riscalda la capacità: se la limitazione (della larghezza di banda della rete) persiste, controlla se la capacità della tabella è limitata dalla sua capacità Funzionamento del throughput di DynamoDB. Utilizza il throughput a caldo o aumenta in anticipo la capacità di scrittura allocata per far fronte agli aumenti di traffico previsti. L’aumento del throughput a caldo migliora la capacità della tabella di gestire picchi di traffico improvvisi prima che si verifichi la limitazione (della larghezza di banda della rete). Nel tempo, se il throughput effettivo si avvicina costantemente ai livelli di throughput a caldo, DynamoDB può suddividere le partizioni occupate in base ai modelli di utilizzo osservati.
- Identifica le chiavi calde: se la tabella non ha risolto il problema automaticamente e il throughput a caldo è elevato o un aumento non è risultato utile, sarà necessario identificare le chiavi calde specifiche. Utilizza the section called “Identificazione dei tasti di scelta rapida mediante Contributor Insights CloudWatch” per determinare se alcuni valori di una particolare chiave di partizione non sono validi. È un primo passo per indirizzare gli sforzi di mitigazione in modo efficace. Prendi in considerazione questi modelli comuni:
  - Se la stessa chiave di partizione appare spesso nei dati di limitazione (della larghezza di banda della rete), ciò significa che si tratta di una chiave calda concentrata.
  - Se non ci sono chiavi ripetute ma si stanno scrivendo i dati in modo ordinato (ad esempio timestamp sequenziali o operazioni basate sulla scansione che seguono l’ordine del keyspace), probabilmente sono presenti partizioni calde continue in cui chiavi diverse diventano calde nel tempo man mano che le scritture si spostano nel keyspace.

  La limitazione (della larghezza di banda della rete) di scrittura può verificarsi anche con operazioni come `BatchWriteItem` o transazioni che riguardano più elementi contemporaneamente.

Quando i singoli elementi di una richiesta `BatchWriteItem` vengono limitati, DynamoDB non propaga questi errori di limitazione (della larghezza di banda della rete) al codice dell'applicazione. Piuttosto, DynamoDB restituisce informazioni sugli elementi non elaborati nella risposta, che l'applicazione deve gestire effettuando ripetizioni di tentativi per quegli elementi specifici. Per le transazioni, l'intera operazione ha esito negativo e viene segnalata una `TransactionCanceledException` in caso di eventuale limitazione (della larghezza di banda della rete) di un elemento. Per questi scenari complessi, potrebbe essere necessario analizzare i modelli di scrittura e i flussi di lavoro di importazione dei dati dell'applicazione, correlarli alla tempistica degli eventi di limitazione (della larghezza di banda della rete) e implementare strategie appropriate di gestione delle ripetizioni dei tentativi.

- Migliora la progettazione delle chiavi di partizione: come soluzione a lungo termine, prendi in considerazione the section called "Miglioramento della progettazione delle chiavi di partizione" per distribuire l'accesso in modo più uniforme tra le partizioni. Questo approccio spesso fornisce la soluzione più completa ai problemi relativi alle partizioni calde risolvendo la causa principale. Tuttavia, richiede un'attenta pianificazione in quanto comporta sfide di migrazione significative.

IndexReadKeyRangeThroughputExceeded

Quando avviene

Quando il tasso di consumo di una o più partizioni nel GSI DynamoDB supera il limite di throughput di lettura della partizione. Questa limitazione (della larghezza di banda della rete) si verifica indipendentemente dalla capacità totale allocata al GSI e influisce sia sulle tabelle con provisioning che on demand. Puoi monitorare le CloudWatch metriche per analizzare il tuo evento the section called "Diagnosi e monitoraggio comuni" di limitazione.

Opzioni di correzione

Prendi in considerazione questa procedura per risolvere i problemi di limitazione (della larghezza di banda della rete):

- Pre-riscalda la capacità: se la limitazione (della larghezza di banda della rete) persiste, controlla se il GSI è limitato dalla sua capacità Funzionamento del throughput di DynamoDB. Utilizza il throughput a caldo o aumenta in anticipo la capacità di lettura allocata per far fronte agli aumenti di traffico previsti. L'aumento del throughput a caldo migliora la capacità del GSI di gestire picchi di traffico improvvisi prima che si verifichi la limitazione (della larghezza di banda della rete). Nel tempo, se il throughput effettivo si avvicina costantemente ai livelli di throughput a caldo, DynamoDB può suddividere le partizioni occupate in base ai modelli di utilizzo osservati.

- Identifica le chiavi calde: se il GSI non ha risolto il problema automaticamente e il throughput a caldo è elevato o un aumento non è risultato utile, sarà necessario identificare le chiavi calde specifiche. Utilizza the section called “Identificazione dei tasti di scelta rapida mediante Contributor Insights CloudWatch” per determinare se alcuni valori di una particolare chiave di partizione non sono validi. È un primo passo per indirizzare gli sforzi di mitigazione in modo efficace. Tieni presente che GSIs, per, la distribuzione delle chiavi di partizione può differire notevolmente dalla tabella di base, creando diversi modelli di tasti di scelta rapida.
- Riprogetta le chiavi di partizione del GSI: valuta se la progettazione delle chiavi del GSI potrebbe creare hot spot artificiali (come indicatori di stato, chiavi con sola data o attributi booleani) che concentrano le letture su un numero limitato di partizioni. Prendi in considerazione l’utilizzo di chiavi composite che combinano l’attributo a bassa cardinalità con un attributo ad alta cardinalità (ad esempio, “ACTIVE#customer123” anziché solo “ACTIVE”) o applicano tecniche the section called “Partizionamento di scrittura” agli elementi della tabella di base che influiscono sulla distribuzione del GSI per distribuire le scritture su più partizioni. Sebbene l’esecuzione di query su dati sottoposti a sharding richieda una logica applicativa aggiuntiva per aggregare i risultati, questo approccio impedisce la limitazione (della larghezza di banda della rete) distribuendo i modelli di accesso in modo più uniforme.

IndexWriteKeyRangeThroughputExceeded

Quando avviene

Quando il tasso di consumo di una o più partizioni nel GSI DynamoDB supera il limite di throughput di scrittura della partizione. Questa limitazione (della larghezza di banda della rete) si verifica indipendentemente dalla capacità totale allocata al GSI e influisce sia sulle tabelle con provisioning che on demand. Puoi monitorare le CloudWatch metriche the section called “Diagnosi e monitoraggio comuni” per analizzare il tuo evento di throttling.

Opzioni di correzione

Prendi in considerazione questa procedura per risolvere i problemi di limitazione (della larghezza di banda della rete):

- Riprogetta la chiave di partizione del GSI: controlla la progettazione delle chiavi di partizione del GSI per verificare che abbia una cardinalità (unicità) sufficiente per distribuire le scritture in modo uniforme. Una causa comune della limitazione (della larghezza di banda della rete) di scrittura nei GSI è l’uso di attributi a bassa cardinalità come chiavi di partizione degli stessi (come i flag di stato con solo pochi valori possibili). Anche quando la tabella di base ha chiavi di partizione ben

distribuite, nel GSI possono comunque essere riscontrate partizioni calde se la chiave di partizione concentra le scritture su un numero limitato di valori. Ad esempio, se l'80% degli elementi presenta status="ACTIVE", ciò crea una partizione estremamente calda in un GSI basato sullo stato. Prendi in considerazione l'utilizzo di chiavi composite che combinano l'attributo a bassa cardinalità con un attributo ad alta cardinalità (ad esempio, "ACTIVE#customer123" anziché solo "ACTIVE") o applicano tecniche the section called "Partizionamento di scrittura" agli elementi della tabella di base che influiscono sulla distribuzione del GSI per distribuire le scritture su più partizioni. Sebbene l'esecuzione di query su dati sottoposti a sharding richieda una logica applicativa aggiuntiva per aggregare i risultati, questo approccio impedisce la limitazione (della larghezza di banda della rete) distribuendo i modelli di accesso in modo più uniforme.

- Pre-riscalda la capacità: controlla se il GSI è limitato dalla sua capacità Funzionamento del throughput di DynamoDB. Utilizza il throughput a caldo o aumenta in anticipo la capacità di lettura allocata per far fronte agli aumenti di traffico previsti. L'aumento del throughput a caldo migliora la capacità del GSI di gestire picchi di traffico improvvisi prima che si verifichi la limitazione (della larghezza di banda della rete). Nel tempo, se il throughput effettivo si avvicina costantemente ai livelli di throughput a caldo, DynamoDB può suddividere le partizioni occupate in base ai modelli di utilizzo osservati.
- Ottimizza le proiezioni GSI: applica the section called "Ottimizzazione delle proiezioni dei GSI" tecniche per ridurre il volume di scrittura. GSIs La proiezione di un numero inferiore di attributi può ridurre significativamente la capacità di scrittura utilizzata da ogni aggiornamento dei GSI.

Diagnosi e monitoraggio comuni

Durante la risoluzione dei problemi di limitazione a livello di partizione, diverse CloudWatch metriche possono aiutare a identificare le partizioni calde e a confermare la causa principale.

Metriche essenziali CloudWatch

Monitora queste metriche chiave per diagnosticare la limitazione (della larghezza di banda della rete) a livello di partizione:

- Eventi di limitazione (della larghezza di banda della rete) a livello di partizione: `ReadKeyRangeThroughputThrottleEvents` e `WriteKeyRangeThroughputThrottleEvents` monitorano quando singole partizioni superano i limiti di throughput. `ReadThrottleEvents` e `WriteThrottleEvents` monitorano quando le richieste di lettura o scrittura superano la capacità allocata.

- Utilizzo di capacità: `ConsumedReadCapacityUnits` e `ConsumedWriteCapacityUnits` mostrano i modelli di utilizzo complessivi.

Procedure di risoluzione

Identificazione dei tasti di scelta rapida mediante Contributor Insights CloudWatch

Utilizza questa procedura per identificare quali chiavi di partizione causano la limitazione (della larghezza di banda della rete).

1. Abilita CloudWatch Contributor Insights sulla tabella o sul GSI per tenere traccia delle chiavi con più limitazioni. Prendi in considerazione la possibilità di mantenere costantemente attivo CloudWatch Contributor Insights per ricevere avvisi di limitazione in tempo reale utilizzando la modalità Throttled keys. Questa modalità si concentra esclusivamente sulle richieste sottoposte a limitazione (della larghezza di banda della rete) ed elabora gli eventi solo quando si verifica la limitazione. Questo monitoraggio mirato è un modo conveniente per mantenere una visibilità continua sui problemi di limitazione (della larghezza di banda della rete).
2. Identifica le chiavi che causano i problemi di partizioni calde.
3. (Se è abilitata la Modalità delle chiavi con accessi e limitazione (della larghezza di banda della rete) completa) Analizza i modelli di accesso nel tempo per determinare se le chiavi calde sono coerenti o se si verificano in periodi specifici.

Miglioramento della progettazione delle chiavi di partizione

Utilizza questo approccio quando è possibile modificare lo schema della tabella per distribuire meglio il traffico tra le partizioni. Quando possibile, questa è la soluzione a lungo termine più efficace per i problemi relativi alle partizioni calde. Idealmente, la progettazione delle chiavi di partizione dovrebbe essere attentamente considerata durante la fase iniziale di progettazione della tabella.

La riprogettazione della chiave di partizione rappresenta una modifica fondamentale del modello di dati che ha un impatto sull'intero ecosistema applicativo. Prima di procedere con questo approccio, prendi attentamente in considerazione queste limitazioni significative:

- Complessità della migrazione dei dati: la riprogettazione delle chiavi di partizione richiede la migrazione di tutti i dati esistenti, operazione che può richiedere molte risorse e tempo per tabelle di grandi dimensioni.
- Modifiche al codice dell'applicazione: tutto il codice dell'applicazione che legge o scrive nella tabella deve essere aggiornato per utilizzare la nuova struttura delle chiavi.

- Impatto sulla produzione: la migrazione a una nuova progettazione delle chiavi spesso richiede tempi di inattività o complesse strategie di doppia scrittura durante la transizione.

Per le linee guida e i principi completi sulla progettazione delle chiavi di partizione, consulta the section called “Progettazione delle chiavi di partizione” e the section called “Distribuzione dei carichi di lavoro”.

Ottimizzazione delle proiezioni dei GSI

Analisi dei modelli di query dell’applicazione per determinare esattamente quali attributi devono essere disponibili quando si eseguono query sul GSI e limitazione delle proiezioni solo a tali attributi. Quando si aggiornano attributi che non sono proiettati in un GSI, non viene eseguita alcuna operazione di scrittura su tale GSI, riducendo l’utilizzo di throughput di scrittura durante gli aggiornamenti. Questa strategia di proiezione mirata ottimizza sia le prestazioni che i costi, pur continuando a supportare i requisiti di query dell’applicazione. Si noti che la proiezione di un numero inferiore di attributi riduce l’utilizzo di capacità di scrittura, ma potrebbe richiedere letture aggiuntive della tabella di base.

Per ulteriori informazioni sulle strategie di proiezione efficienti, consulta Best practice per l’utilizzo degli indici secondari in DynamoDB.

Risorse aggiuntive

I seguenti post del blog forniscono esempi e dettagli pratici sui concetti trattati in questa guida:

- Per ulteriori informazioni sull'utilizzo GSIs per distribuire il traffico di lettura, consulta Using Global Secondary Indexes per creare indici secondari coerenti in Amazon DynamoDB.
- Per una guida pratica sul dimensionamento di DynamoDB e la gestione delle partizioni calde, consulta Part 1: Scaling DynamoDB - How partitions, hot keys, and split for heat impact performance.
- Per informazioni dettagliate su come funziona il split-for-heat meccanismo di DynamoDB, i suoi vantaggi e i dettagli di implementazione, consulta la Parte 3: Riepilogo e best practice.
- Per le strategie dettagliate di sharding di scrittura, consulta the section called “Partizionamento di scrittura”.

2. Superamento del throughput allocato

La limitazione della capacità assegnata si verifica quando il tasso di consumo dell'applicazione supera le unità di capacità di lettura o scrittura (RCUs/WCUs) configurate per le tabelle o gli indici secondari globali. Sebbene DynamoDB fornisca una capacità di espansione per gestire picchi di traffico occasionali, le richieste sostenute oltre i limiti prestabiliti comportano una limitazione (della larghezza di banda della rete). Quando ciò accade, DynamoDB restituisce un tipo di motivo di limitazione (della larghezza di banda della rete) `ProvisionedThroughputExceeded` nell'eccezione di limitazione. Il motivo indica se il problema riguarda le operazioni di lettura o scrittura e se influisce sulla tabella di base o su un indice secondario globale.

La limitazione (della larghezza di banda della rete) può verificarsi indipendentemente dal fatto che il dimensionamento automatico sia abilitato o meno. Il dimensionamento automatico si adatta agli aumenti dei consumi, ma non risponde istantaneamente ed è vincolato dai limiti di capacità massima configurati. Ciò significa che può ancora verificarsi la limitazione (della larghezza di banda della rete) durante improvvisi picchi di traffico o quando l'utilizzo supera i limiti massimi di dimensionamento automatico.

Misure di mitigazione per il superamento del throughput allocato

Questa sezione fornisce indicazioni per la risoluzione degli scenari di limitazione (della larghezza di banda della rete) della capacità allocata. Prima di utilizzare questa guida è necessario assicurarsi di aver identificato il motivo specifico della limitazione (della larghezza di banda della rete) nella gestione delle eccezioni dell'applicazione e di aver determinato il nome della risorsa Amazon (ARN) della risorsa interessata. Per informazioni su come recuperare i motivi della limitazione (della larghezza di banda della rete) e identificare le risorse limitate, consulta the section called "Framework di diagnosi".

Prima di addentrarsi in scenari specifici di limitazione, occorre valutare innanzitutto se la limitazione (della larghezza di banda della rete) è effettivamente un problema che deve essere risolto:

- Una limitazione (della larghezza di banda della rete) occasionale è normale e prevista in applicazioni DynamoDB ben ottimizzate; in questo caso la limitazione (della larghezza di banda della rete) indica semplicemente che si sta consumando il 100% di quanto allocato. Se l'applicazione gestisce correttamente la limitazione (della larghezza di banda della rete) con la ripetizione dei tentativi e le prestazioni complessive soddisfano i requisiti, potrebbe non essere necessaria un'azione immediata.

- Tuttavia, se la latenza causata è inaccettabile sul lato client, peggiorando l'esperienza utente o impedendo il completamento tempestivo delle operazioni cruciali, procedi con le opzioni di mitigazione riportate di seguito.

Per risolvere i problemi di limitazione (della larghezza di banda della rete), stabilisci innanzitutto se la limitazione è causata da:

- Picchi di traffico temporanei: aumenti di traffico di breve durata che superano la capacità allocata ma non sono prolungati. Questi richiedono strategie diverse rispetto a un traffico elevato e continuo.
- Traffico elevato e continuo: carichi di lavoro prolungati che superano costantemente la capacità allocata.

Per quanto riguarda i picchi di traffico, prendi in considerazione le strategie tratte dal blog Handle traffic spikes with Amazon DynamoDB provisioned capacity in the section called "Risorse aggiuntive".

Per un traffico elevato e continuo, prendi in considerazione le seguenti opzioni di regolazione della capacità:

- the section called "TableReadProvisionedThroughputExceeded"
- the section called "TableWriteProvisionedThroughputExceeded"
- the section called "IndexReadProvisionedThroughputExceeded"
- the section called "IndexWriteProvisionedThroughputExceeded"

TableReadProvisionedThroughputExceeded

Quando avviene

Il tasso di consumo in lettura dell'applicazione supera le unità di capacità di lettura assegnate () configurate per la tabella. RCUs Puoi monitorare le CloudWatch metriche the section called "Diagnosi e monitoraggio comuni" per analizzare il tuo evento di limitazione.

Approccio di risoluzione

Prendi in considerazione queste strategie per risolvere il problema della limitazione (della larghezza di banda della rete) della capacità di lettura:

- Passaggio alla modalità con capacità on demand: valuta la possibilità di passare la tabella alla modalità on demand in caso di riscontro di frequente limitazione (della larghezza di banda della rete) dovuta ai picchi di traffico. La modalità on demand elimina i problemi di provisioning e si adatta automaticamente al carico di lavoro.
- In caso di mantenimento della modalità con provisioning e se il dimensionamento automatico non è abilitato:
  - Valuta la possibilità di aumentare la capacità di lettura delle tabelle.
  - Abilita il dimensionamento automatico per la capacità di lettura sulla tabella.
- Se il dimensionamento automatico è abilitato (impostazione predefinita per le tabelle create nella console):
  - Ottimizza i parametri di dimensionamento automatico di lettura della tabella.

TableWriteProvisionedThroughputExceeded

Quando avviene

Il tasso di consumo di scrittura dell'applicazione supera le unità di capacità di scrittura assegnate (WCUs) configurate per la tabella. Puoi monitorare le CloudWatch metriche per analizzare il tuo evento the section called “Diagnosi e monitoraggio comuni” di throttling.

Approccio di risoluzione

Prendi in considerazione queste strategie per risolvere il problema della limitazione (della larghezza di banda della rete) della capacità di scrittura:

- Passaggio alla modalità con capacità on demand: valuta la possibilità di passare la tabella alla modalità on demand in caso di riscontro di frequente limitazione (della larghezza di banda della rete) dovuta ai picchi di traffico. La modalità on demand elimina i problemi di provisioning e si adatta automaticamente al carico di lavoro.
- In caso di mantenimento della modalità con provisioning e se il dimensionamento automatico non è abilitato:
  - Valuta la possibilità di aumentare la capacità di scrittura delle tabelle.
  - Abilita il dimensionamento automatico per la capacità di scrittura sulla tabella.
- Se il dimensionamento automatico è abilitato (impostazione predefinita per le tabelle create nella console):
  - Ottimizza i parametri di dimensionamento automatico di scrittura della tabella.

IndexReadProvisionedThroughputExceeded

Quando avviene

Il consumo di lettura su un Global Secondary Index (GSI) supera le unità di capacità di lettura fornite dal GSI (). RCUs Puoi monitorare le CloudWatch metriche per analizzare il tuo evento di the section called “Diagnosi e monitoraggio comuni” throttling.

Approccio di risoluzione

Prendi in considerazione queste strategie per risolvere il problema della limitazione (della larghezza di banda della rete) della capacità di lettura nei GSI:

- Passaggio alla modalità con capacità on demand: valuta la possibilità di passare la tabella di base alla modalità on demand in caso di riscontro di frequente limitazione (della larghezza di banda della rete) dovuta ai picchi di traffico. La modalità on demand elimina i problemi di provisioning e si adatta automaticamente al carico di lavoro.
- In caso di mantenimento della modalità con provisioning e se il dimensionamento automatico non è abilitato:
  - Prendi in considerazione l’aumento della capacità di lettura del GSI.
  - Abilita il dimensionamento automatico per la capacità di lettura sul GSI.
- Se il dimensionamento automatico è abilitato (impostazione predefinita per le tabelle create nella console):
  - Ottimizza i parametri di dimensionamento automatico di lettura del GSI.

IndexWriteProvisionedThroughputExceeded

Quando avviene

Quando gli aggiornamenti agli elementi nella tabella di base attivano la scrittura su un GSI che supera la capacità di scrittura allocata del GSI. Ciò causa una limitazione (della larghezza di banda della rete) da contropressione nelle scritture della tabella di base. Puoi monitorare le CloudWatch metriche per analizzare il tuo evento the section called “Diagnosi e monitoraggio comuni” di limitazione.

Approccio di risoluzione

Prendi in considerazione queste strategie per risolvere il problema della limitazione (della larghezza di banda della rete) della capacità di scrittura nei GSI:

- Passaggio alla modalità con capacità on demand: valuta la possibilità di passare la tabella di base alla modalità on demand in caso di riscontro di frequente limitazione (della larghezza di banda della rete) dovuta ai picchi di traffico. La modalità on demand elimina i problemi di provisioning e si adatta automaticamente al carico di lavoro.
- In caso di mantenimento della modalità con provisioning e se il dimensionamento automatico non è abilitato:
  - Prendi in considerazione l'aumento della capacità di scrittura del GSI.
  - Abilita il dimensionamento automatico per la capacità di scrittura sul GSI.
- Se il dimensionamento automatico è abilitato (impostazione predefinita per le tabelle create nella console):
  - Ottimizza i parametri di dimensionamento automatico di scrittura del GSI.

Diagnosi e monitoraggio comuni

Durante la risoluzione degli errori di throughput, diverse CloudWatch metriche possono aiutare a identificare la causa principale.

Metriche essenziali CloudWatch

Monitora queste metriche chiave per diagnosticare la limitazione (della larghezza di banda della rete) della capacità allocata:

- Eventi di limitazione (della larghezza di banda della rete): `ReadProvisionedThroughputThrottleEvents` e `WriteProvisionedThroughputThrottleEvents` monitorano quando le richieste vengono limitate per questo motivo. `ReadThrottleEvents` e `WriteThrottleEvents` monitorano quando le richieste di lettura o scrittura superano la capacità allocata.
- Utilizzo di capacità: `ConsumedReadCapacityUnits` e `ConsumedWriteCapacityUnits` mostrano l'utilizzo effettivo.
- Capacità allocata: `ProvisionedReadCapacityUnits` e `ProvisionedWriteCapacityUnits` mostrano i limiti configurati.

Procedure di risoluzione

Aumento della capacità di throughput delle tabelle

Utilizza questa procedura quando il dimensionamento automatico non è abilitato ed è necessario un aumento immediato della capacità.

1. Aggiorna la capacità fornita della tabella utilizzando la console AWS CLI DynamoDB o l'SDK:
   - Per la capacità di lettura: aumenta il parametro `ReadCapacityUnits`, che specifica il numero massimo di operazioni a elevata consistenza di lettura utilizzate al secondo prima che DynamoDB sottoponga a limitazione (della larghezza di banda della rete) le richieste.
   - Per la capacità di scrittura: aumenta il parametro `WriteCapacityUnits`, che specifica il numero massimo di scritture utilizzate al secondo prima che DynamoDB sottoponga a limitazione (della larghezza di banda della rete) le richieste.
2. Verifica che le nuove impostazioni di capacità non superino le quote di throughput per tabella e che l'utilizzo totale dell'account rimanga al di sotto delle quote di throughput per account della Regione. Se questi limiti si stanno avvicinando, valuta invece la possibilità di passare alla modalità con capacità on demand.

Configurazione del dimensionamento automatico delle tabelle per regolare la capacità di lettura o scrittura della tabella o del GSI

Configura il dimensionamento automatico di DynamoDB per regolare automaticamente la capacità di lettura o scrittura in base ai modelli di traffico. È possibile configurare Auto Scaling in modo indipendente per entrambe le tabelle e GSIs con controlli separati per le unità di capacità di lettura e scrittura.

1. Abilita il dimensionamento automatico per la capacità di lettura, la capacità di scrittura o entrambe sulla tabella o sul GSI.
2. Stabilisci una percentuale di utilizzo di destinazione con margine di manovra per i picchi di traffico.

> **Note**
>
> Un minore utilizzo di destinazione aumenta i costi e la frequenza di dimensionamento. Destinazioni inferiori al 40% possono causare un eccesso di provisioning. Monitora i modelli di utilizzo e i costi per bilanciare prestazioni ed efficienza.

3. Imposta limiti di capacità:

   - Minimo RCUs/WCUs: mantiene una capacità sufficiente durante i periodi di traffico ridotto.
   - Massimo RCUs/WCUs: soddisfa le richieste di traffico di picco e protegge da eventi di scalabilità incontrollata.

Per indicazioni sulla configurazione e la gestione del dimensionamento automatico di DynamoDB, consulta Gestione automatica della capacità effettiva di trasmissione con il dimensionamento automatico di DynamoDB.

> **Note**
>
> Il dimensionamento automatico richiede in genere diversi minuti per rispondere alle variazioni del traffico. In caso di picchi di traffico improvvisi, la capacità di espansione della tabella offre una protezione immediata mentre il dimensionamento automatico si adatta. Configura l'utilizzo di destinazione con un margine di manovra adeguato per dare tempo alle operazioni di dimensionamento e preservare la capacità di espansione in caso di richieste impreviste.

Ottimizzazione delle impostazioni di dimensionamento automatico di lettura o scrittura della tabella o dell'indice

Utilizza questa procedura quando il dimensionamento automatico è abilitato ma si verifica comunque una limitazione (della larghezza di banda della rete). È possibile ottimizzare Auto Scaling in modo indipendente sia per le tabelle che per gli indici secondari globali (GSIs), con controlli separati per le unità di capacità di lettura e scrittura.

- Modifica l'utilizzo degli obiettivi: valuta la possibilità di ridurre l'utilizzo target per la tabella o di attivare il ridimensionamento prima che si verifichi GSIs la limitazione. Assicurati di monitorare il traffico dopo aver apportato queste modifiche. Per ulteriori informazioni sul consumo di capacità e sulle implicazioni in termini di costi, consulta the section called "Configurazione del dimensionamento automatico delle tabelle per regolare la capacità di lettura o scrittura della tabella o del GSI".
- Controlla i limiti di capacità: assicurati che le impostazioni di capacità minima e massima siano in linea con i modelli di carico di lavoro effettivi.

Passaggio alla modalità con capacità on demand

Per informazioni generali sul passaggio tra modalità di capacità, consulta the section called "Passaggio tra modalità di capacità". Fai riferimento a Service Quotas per conoscere i vincoli specifici relativi al passaggio da una modalità all'altra.

Aumento della capacità di throughput dei GSI

Utilizza questa procedura quando il dimensionamento automatico non è abilitato sul GSI ed è necessario un aumento immediato della capacità.

1. Aggiorna la capacità fornita del GSI utilizzando la console DynamoDB o l'SDK: AWS CLI

   - Per la capacità di lettura: aumenta il parametro `ReadCapacityUnits` per il GSI specifico, che specifica il numero massimo di letture utilizzabili al secondo dal GSI prima che DynamoDB sottoponga a limitazione (della larghezza di banda della rete) le richieste. Tieni presente che supportano GSIs solo letture alla fine coerenti.
   - Per la capacità di scrittura: aumenta il parametro `WriteCapacityUnits` per il GSI specifico, che specifica il numero massimo di scritture utilizzabili al secondo dal GSI prima che DynamoDB sottoponga a limitazione (della larghezza di banda della rete) le richieste.
2. Assicurati che la capacità di throughput allocata del GSI rimanga entro le quote di throughput per account e per tabella.

Risorse aggiuntive

- Per informazioni dettagliate sulla gestione dei picchi di traffico nelle tabelle DynamoDB con capacità allocata, comprensive di varie strategie, dall'utilizzo del dimensionamento automatico e della capacità di espansione alla gestione strategica della limitazione (della larghezza di banda della rete), consulta Handle traffic spikes with Amazon DynamoDB provisioned capacity.
- Per informazioni su come utilizzare un'espressione Cron per pianificare una policy di dimensionamento, consulta Optimize costs by scheduling provisioned capacity for DynamoDB.
- Per informazioni pratiche sul monitoraggio e l'analisi dei modelli di utilizzo del throughput per le tabelle DynamoDB in modalità con capacità allocata, consulta How to evaluate throughput utilization for Amazon DynamoDB tables in provisioned mode.

3. Superamento dei limiti dell'account

Le tabelle on demand non dispongono di livelli di capacità allocata da gestire, ma DynamoDB impone limiti di throughput a livello di account per prevenire l'esecuzione incontrollata e garantire un uso equo delle risorse da parte di tutti i clienti. Questi limiti di account per tabella fungono da protezioni regolabili, impostati per ogni combinazione di account e Regione. Quando il tasso di consumo in lettura o scrittura supera questi limiti, DynamoDB restituisce un tipo di motivo di limitazione (della larghezza di banda della rete) `AccountLimitExceeded` nell'eccezione di limitazione. I limiti di account predefiniti per tabella si applicano automaticamente quando le tabelle non hanno impostazioni di throughput massimo personalizzate configurate. Facoltativamente, è possibile configurare le impostazioni di throughput massimo per un controllo più preciso dei costi e la prevedibilità, oppure richiedere aumenti delle quote tramite la console the section called "Quote" se i requisiti dell'applicazione superano i limiti predefiniti.

Misure di mitigazione per il superamento dei limiti dell'account

Questa sezione fornisce linee guida per la risoluzione degli scenari di limitazione (della larghezza di banda della rete) dei limiti dell'account. Prima di utilizzare questa guida è necessario assicurarsi di aver identificato i motivi specifici della limitazione (della larghezza di banda della rete) nella gestione delle eccezioni dell'applicazione e di aver determinato il nome della risorsa Amazon (ARN) della risorsa interessata. Per informazioni su come recuperare i motivi della limitazione (della larghezza di banda della rete) e identificare le risorse limitate, consulta the section called "Framework di diagnosi".

Prima di approfondire specifici scenari di limitazione (della larghezza di banda della rete), stabilisci innanzitutto se è effettivamente necessaria un'azione:

- Valuta l'impatto sulle prestazioni: verifica se l'applicazione soddisfa ancora i requisiti prestazionali nonostante la limitazione (della larghezza di banda della rete). Molte applicazioni funzionano correttamente con o in prossimità dei limiti degli account, in particolare durante operazioni in blocco o migrazioni di dati.
- Esamina i modelli di limitazione (della larghezza di banda della rete): se la limitazione è intermittente e l'applicazione gestisce le ripetizioni di tentativi in modo efficace, i limiti attuali potrebbero essere sufficienti per il carico di lavoro.

Se l'applicazione funziona in modo accettabile anche quando occasionalmente raggiunge i limiti dell'account, si potrebbe scegliere di monitorare semplicemente la situazione piuttosto che implementare modifiche immediate.

Se ritieni che la limitazione (della larghezza di banda della rete) stia causando problemi di prestazioni o affidabilità inaccettabili, seleziona uno specifico motivo della limitazione di seguito per trovare le opzioni di mitigazione consigliate:

- the section called “TableReadAccountLimitExceeded”
- the section called “TableWriteAccountLimitExceeded”
- the section called “IndexReadAccountLimitExceeded”
- the section called “IndexWriteAccountLimitExceeded”

TableReadAccountLimitExceeded

Quando avviene

Quando l’utilizzo di lettura della tabella ha superato la quota di throughput di lettura per tabella a livello di account per la Regione. Puoi monitorare le CloudWatch metriche per analizzare il tuo evento di limitazione. the section called “Diagnosi e monitoraggio comuni”

Approccio di risoluzione

Segui la procedura riportata di seguito per questa limitazione (della larghezza di banda della rete).

- Richiedi aumenti della quota:

  Richiedi un aumento del limite di velocità di lettura per tabella (codice Quota L-CF0). CBE56 Per la procedura dettagliata di invio della richiesta, consulta Richiesta di aumenti della quota per tabella.

TableWriteAccountLimitExceeded

Quando avviene

Quando l’utilizzo di scrittura della tabella ha superato la quota di throughput di scrittura per tabella a livello di account per la Regione. Puoi monitorare le CloudWatch metriche per analizzare il tuo evento di throttlingthe section called “Diagnosi e monitoraggio comuni”.

Approccio di risoluzione

Segui la procedura riportata di seguito per questa limitazione (della larghezza di banda della rete).

- Richiedi aumenti della quota: richiedi un aumento del limite di velocità di scrittura per tabella (codice quota L-). AB614373 Per la procedura dettagliata di invio della richiesta, consulta Richiesta di aumenti della quota per tabella.

IndexReadAccountLimitExceeded

Quando avviene

Quando le operazioni di lettura dirette a un indice secondario globale (GSI) consumano un throughput superiore a quello consentito dalla quota di lettura per tabella dell'account nella Regione AWS corrente. La quota di velocità effettiva di lettura per tabella a livello di account si applica collettivamente a una tabella e a tutte le relative combinazioni. GSIs Puoi monitorare le CloudWatch metriche per analizzare il tuo evento di limitazione. the section called "Diagnosi e monitoraggio comuni"

Approccio di risoluzione

Scegli l'opzione di risoluzione appropriata in base alla distribuzione della capacità dell'account:

- Richiedi aumenti della quota: richiedi un aumento del limite di velocità di lettura per tabella (codice quota L-CF0). CBE56 Per la procedura dettagliata di invio della richiesta, consulta Richiesta di aumenti della quota per tabella.
- Ottimizza l'utilizzo del GSI: controlla i modelli di progettazione del GSI e di esecuzione di query per ridurre l'utilizzo non necessario di capacità di lettura.

IndexWriteAccountLimitExceeded

Quando avviene

Le operazioni di scrittura nella tabella di base generano gli aggiornamenti corrispondenti a un GSI che complessivamente superano la quota di velocità di scrittura per tabella a livello di account per regione. AWS Ogni scrittura su un elemento della tabella di base che contiene attributi indicizzati da un GSI attiva un'operazione di scrittura corrispondente a quel GSI. Queste operazioni di scrittura combinate vengono conteggiate ai fini della quota di throughput di scrittura per tabella. È possibile monitorare le CloudWatch metriche the section called "Diagnosi e monitoraggio comuni" per analizzare i modelli e la tempistica di questi eventi di limitazione e identificare quali operazioni stanno causando l'eccessiva attività di scrittura del GSI.

Approccio di risoluzione

Scegli l'opzione di risoluzione appropriata in base alla distribuzione della capacità dell'account:

- Richiedi aumenti della quota: richiedi un aumento del limite di velocità effettiva di scrittura per tabella (codice quota L-AB614373) per far fronte a un traffico di scrittura GSI più elevato derivante dalle operazioni della tabella di base. La quota di velocità effettiva di scrittura per tabella si applica all'intera tabella, inclusa tutta la tabella. GSIs Per la procedura dettagliata di invio della richiesta, consulta Richiesta di aumenti della quota per tabella.
- Ottimizza le proiezioni GSI: rivedi le proiezioni GSI e progetta per ridurre il volume di scrittura. GSIs

Diagnosi e monitoraggio comuni

Per la risoluzione dei problemi relativi al superamento degli eventi di limitazione del limite di account, diverse CloudWatch metriche possono aiutare a identificare se si stanno raggiungendo i limiti per tabella o a livello di account e a comprendere i modelli di distribuzione della capacità.

Metriche essenziali CloudWatch

Monitora queste metriche chiave per diagnosticare la limitazione (della larghezza di banda della rete) dei limiti degli account:

- Eventi di limitazione (della larghezza di banda della rete) dovuti a limiti dell'account: `ReadAccountLimitThrottleEvents` e `WriteAccountLimitThrottleEvents` monitorano quando le richieste vengono sottoposte a limitazione (della larghezza di banda della rete) a causa dei limiti a livello di account. `ReadThrottleEvents` e `WriteThrottleEvents` monitorano quando le richieste di lettura o scrittura superano la capacità allocata.
- Utilizzo di capacità per risorsa: `ConsumedReadCapacityUnits` e `ConsumedWriteCapacityUnits` per ogni tabella e GSI mostrano l'utilizzo delle singole risorse.
- Consumo a livello di account: utilizza le espressioni CloudWatch matematiche metriche per sommare la capacità consumata in tutte le tabelle e GSIs per monitorare l'utilizzo totale dell'account.

Procedure di risoluzione

Richiesta di aumenti di quota per tabella

Se le applicazioni devono funzionare oltre gli attuali limiti di throughput per tabella, è necessario inviare una richiesta di aumento della quota utilizzando la procedura seguente. Ogni tabella

DynamoDB del AWS tuo account (insieme a tutte le tabelle GSIs associate) è soggetta a queste quote di throughput all'interno di una regione specifica. Queste quote rappresentano la capacità massima di lettura o scrittura che ogni singola tabella e la relativa tabella GSIs possono consumare collettivamente e si applicano indipendentemente a ciascuna tabella anziché come aggregato su tutte le tabelle dell'account.

Facoltativamente, è possibile anche impostare limiti inferiori per tabella o per GSI configurando le relative impostazioni di throughput massimo on demand.

1. Identifica la quota specifica che deve essere aumentata:
   - Limite di velocità effettiva di lettura per tabella (codice quota CBE56 L-CF0): impostazione predefinita: 40.000 per tabella RCUs
   - Limite di velocità effettiva di scrittura per tabella (codice quota L-): impostazione predefinita: 40.000 per tabella AB614373 WCUs
2. Utilizza la console AWS Service Quotas per richiedere un aumento:
   - Accedi al servizio DynamoDB in Service Quotas
   - Trova la quota appropriata utilizzando il codice quota
   - Richiedi un aumento in base al picco di utilizzo previsto
3. Fornisci una giustificazione per l'aumento, tra cui:
   - Modelli di utilizzo attuali e requisiti di traffico di picco
   - Giustificazione aziendale dell'aumento della capacità
   - Cronologia dei casi in cui è necessaria una maggiore capacità

> **Note**
>
> L'elaborazione degli aumenti delle quote richiede in genere 24 – 48 ore. Pianifica le richieste di conseguenza e prendi in considerazione strategie di mitigazione temporanee in attesa dell'approvazione.

Ottimizzazione delle proiezioni e della progettazione dei GSI

Ottimizzazione delle proiezioni e della progettazione degli indici secondari globali (GSI) per ridurre l'utilizzo di capacità e migliorare le prestazioni.

Strategie di proiezione selettive

Se le query devono accedere solo a pochi attributi, la proiezione solo di tali attributi riduce la quantità di dati scritti nel GSI quando gli elementi della tabella di base cambiano. Per i dettagli sui tipi di proiezione, consulta Projections for Global Secondary Indexes.

1. Analizza i modelli di query: esamina i modelli di query dell'applicazione per identificare a quali attributi si accede effettivamente tramite il GSI.
2. Utilizza proiezioni selettive: effettua la proiezione solo degli attributi che sono effettivamente necessari nelle query per ridurre il volume di scrittura.
3. Prendi in considerazione `KEYS_ONLY`: se le query richiedono solo gli attributi chiave, usa la proiezione `KEYS_ONLY` per ridurre al minimo il volume di scrittura.
4. Bilancia i compromessi tra lettura e scrittura: la proiezione di un minor numero di attributi riduce l'utilizzo di capacità di scrittura, ma può richiedere letture aggiuntive della tabella di base.

Implementazione di GSI di tipo sparse

Sparse GSIs contiene solo gli elementi che hanno l'attributo indicizzato, anziché tutti gli elementi della tabella di base. Ciò riduce la densità delle partizioni e migliora le prestazioni quando si interrogano frequentemente sottoinsiemi di dati specifici.

1. Design GSIs che include solo elementi con valori di attributo specifici.
2. Implementa l'indicizzazione condizionale impostando l'attributo della chiave di partizione del GSI solo sugli elementi che devono essere indicizzati.
3. Utilizza chiavi composite in formato sparso GSIs (ad esempio, status #timestamp) per distribuire ulteriormente il traffico all'interno del sottoinsieme di elementi indicizzati.

Per ulteriori informazioni sull'implementazione di queste strategie, consulta Best practice per l'utilizzo degli indici secondari in DynamoDB.

4. Superamento del throughput massimo on demand

Quando si configura una tabella on demand o un GSI, è possibile facoltativamente impostare un limite massimo di throughput (`MaxReadRequestUnits` e `MaxWriteRequestUnits`) a livello di tabella o indice per evitare costi eccessivi o proteggere i sistemi a valle dal sovraccarico. Per ulteriori informazioni sul throughput massimo, consulta the section called "Throughput massimo di DynamoDB per le tabelle on demand".

Quando l'utilizzo di lettura o scrittura supera questi limiti autoimposti, le richieste aggiuntive che supererebbero tale limite ricevono risposte rapide. DynamoDB restituisce eccezioni con `MaxOnDemandThroughputExceeded` un tipo di motivo di limitazione (della larghezza di banda della rete), che indica quale risorsa ha raggiunto il limite di throughput.

Limitazione (della larghezza di banda della rete) a causa del superamento del throughput massimo on demand

Questa sezione fornisce indicazioni sulla risoluzione per scenari di limitazione (della larghezza di banda della rete) a causa del superamento del throughput massimo on demand. Prima di utilizzare questa guida è necessario assicurarsi di aver identificato i motivi specifici della limitazione (della larghezza di banda della rete) nella gestione delle eccezioni dell'applicazione e di aver determinato il nome della risorsa Amazon (ARN) della risorsa interessata. Per informazioni su come recuperare i motivi della limitazione (della larghezza di banda della rete) e identificare le risorse limitate, consulta the section called "Framework di diagnosi".

Prima di approfondire specifici scenari di limitazione (della larghezza di banda della rete), considera innanzitutto se è effettivamente necessaria un'azione:

- Valuta le impostazioni di throughput massimo: questi limiti sono stati configurati intenzionalmente per controllare i costi o proteggere i sistemi a valle. In caso di ricezione di eventi di limitazione (della larghezza di banda della rete) `MaxOnDemandThroughputExceeded`, i limiti funzionano come previsto. Valuta se l'aumento di questi limiti è in linea con gli obiettivi originali di controllo dei costi o protezione del sistema.
- Valuta l'impatto sulle applicazioni: stabilisci se la limitazione (della larghezza di banda della rete) sta effettivamente causando problemi alle applicazioni o agli utenti. Se le applicazioni gestiscono le ripetizioni di tentativi in modo efficace e soddisfano i requisiti prestazionali nonostante occasionali limitazioni (della larghezza di banda della rete), mantenere i limiti attuali potrebbe essere la scelta appropriata.
- Analizza i modelli di traffico: analizza se la limitazione (della larghezza di banda della rete) rappresenta uno schema di traffico previsto o un picco insolito. Per modelli di traffico prevedibili e ricorrenti che superano costantemente i limiti, potrebbe essere necessario modificare le impostazioni di throughput massimo. Per i picchi temporanei, l'implementazione di migliori strategie di distribuzione delle richieste potrebbe essere più appropriata rispetto all'innalzamento dei limiti.

Se, dopo un'attenta valutazione, ritieni che le impostazioni di throughput massimo debbano essere modificate, fai riferimento agli scenari di limitazione (della larghezza di banda della rete) specifici riportati di seguito per opzioni di correzione mirate:

- the section called "TableReadMaxOnDemandThroughputExceeded"
- the section called "TableWriteMaxOnDemandThroughputExceeded"
- the section called "IndexReadMaxOnDemandThroughputExceeded"
- the section called "IndexWriteMaxOnDemandThroughputExceeded"

TableReadMaxOnDemandThroughputExceeded

Quando avviene

Quando la tabella on demand ha superato la capacità di throughput massimo di lettura configurata. Puoi monitorare le CloudWatch metriche the section called "Diagnosi e monitoraggio comuni" per analizzare il tuo evento di limitazione.

Opzioni di correzione

Prendi in considerazione questa procedura per risolvere i problemi di limitazione (della larghezza di banda della rete):

- Aumenta il limite massimo di throughput: utilizza la console DynamoDB, la AWS CLI, o l'API `UpdateTable` DynamoDB per aumentare il valore `MaxReadRequestUnits` per la tabella interessata, quindi monitora e regola. Ciò consente alla tabella di gestire un throughput di lettura più elevato prima che si verifichi la limitazione (della larghezza di banda della rete).
- Rimuovi il limite massimo: imposta `MaxReadRequestUnits` su `-1` per rimuovere il limite massimo, permettendo il dimensionamento in base alla domanda fino alle quote di throughput a livello di account. In questo modo viene rimosso il limite personalizzato, ma vengono comunque mantenute le misure di protezione a livello di account di AWS. Tuttavia, è importante monitorare attentamente la spesa dopo aver rimosso questo limite, poiché ora la tabella può utilizzare molta più capacità prima di raggiungere le quote a livello di account.

TableWriteMaxOnDemandThroughputExceeded

Quando avviene

Quando la tabella on demand ha superato la capacità di throughput massimo di scrittura configurata. Puoi monitorare le CloudWatch metriche per analizzare il tuo evento the section called “Diagnosi e monitoraggio comuni” di limitazione.

Opzioni di correzione

Prendi in considerazione questa procedura per risolvere i problemi di limitazione (della larghezza di banda della rete):

- Aumenta il limite massimo di throughput: utilizza la console DynamoDB, la AWS CLI, o l'API `UpdateTable` DynamoDB per aumentare il valore `MaxWriteRequestUnits` per la tabella interessata, quindi monitora e regola.
- Rimuovi il limite massimo: imposta `MaxWriteRequestUnits` su `-1` per rimuovere il limite massimo, permettendo il dimensionamento in base alla domanda fino alle quote di throughput a livello di account. In questo modo viene rimosso il limite personalizzato, ma vengono comunque mantenute le misure di protezione a livello di account di AWS. Tuttavia, è importante monitorare attentamente la spesa dopo aver rimosso questo limite, poiché ora la tabella può utilizzare molta più capacità prima di raggiungere le quote a livello di account.

IndexReadMaxOnDemandThroughputExceeded

Quando avviene

Quando le richieste di lettura a un GSI in modalità on demand hanno superato la capacità di throughput massimo di lettura configurata del GSI. Puoi monitorare le CloudWatch metriche per analizzare il tuo evento the section called “Diagnosi e monitoraggio comuni” di limitazione.

Opzioni di correzione

Prendi in considerazione questa procedura per risolvere i problemi di limitazione (della larghezza di banda della rete):

- Aumenta il limite massimo di throughput del GSI: utilizza la console DynamoDB, la AWS CLI, o l'API `UpdateTable` DynamoDB per aumentare il valore `MaxReadRequestUnits` per il GSI interessato, quindi monitora e regola.
- Rimuovi il limite massimo del GSI: imposta `MaxReadRequestUnits` su `-1` perché il GSI rimuova il limite, permettendo il dimensionamento in base alla domanda fino alle quote di throughput a livello di account. Ciò rimuove il limite personalizzato ma mantiene comunque le misure di protezione a

livello AWS di account. È tuttavia importante monitorare attentamente la spesa dopo aver rimosso questo limite.

IndexWriteMaxOnDemandThroughputExceeded

Quando avviene

Quando gli aggiornamenti agli elementi nella tabella di base attivano la scrittura su un GSI in modalità on demand e superano la capacità di throughput di scrittura massima configurata del GSI, con conseguente limitazione (della larghezza di banda della rete) da contropressione. Puoi monitorare le CloudWatch metriche per analizzare il tuo evento the section called “Diagnosi e monitoraggio comuni” di limitazione.

Opzioni di correzione

Prendi in considerazione questa procedura per risolvere i problemi di limitazione (della larghezza di banda della rete):

- Aumenta il limite massimo di throughput del GSI: utilizza la console DynamoDB, la AWS CLI, o l'API `UpdateTable` DynamoDB per aumentare il valore `MaxWriteRequestUnits` per il GSI interessato, quindi monitora e regola.
- Rimuovi il limite massimo del GSI: imposta `MaxWriteRequestUnits` su `-1` perché il GSI rimuova il limite, permettendo il dimensionamento in base alla domanda fino alle quote di throughput a livello di account. Ciò rimuove il limite personalizzato ma mantiene comunque le misure di protezione a livello AWS di account. È tuttavia importante monitorare attentamente la spesa dopo aver rimosso questo limite.

Diagnosi e monitoraggio comuni

Durante la risoluzione dei problemi relativi alla risoluzione dei problemi relativi alla velocità effettiva massima su richiesta superiore agli eventi di limitazione, diverse CloudWatch metriche possono aiutare a identificare la causa principale e i modelli di scalabilità.

Metriche essenziali CloudWatch

Monitora queste metriche chiave per diagnosticare che il throughput massimo on demand ha superato la limitazione (della larghezza di banda della rete):

- Eventi di limitazione (della larghezza di banda della rete) da throughput massimo: `ReadMaxOnDemandThroughputThrottleEvents` e

`WriteMaxOnDemandThroughputThrottleEvents` monitorano quando le richieste vengono sottoposte a limitazione (della larghezza di banda della rete) a causa del superamento dei limiti massimi. `ReadThrottleEvents` e `WriteThrottleEvents` monitorano quando le richieste di lettura o scrittura superano la capacità allocata.

- Throughput massimo attuale configurato per una tabella o un indice secondario globale: `OnDemandMaxReadRequestUnits` e `OnDemandMaxWriteRequestUnits` mostrano gli attuali limiti di capacità massima.
- Utilizzo effettivo della capacità: `ConsumedReadCapacityUnits` e `ConsumedWriteCapacityUnits` mostrano i modelli di utilizzo effettivi.

Approccio di analisi

Segui questa procedura per verificare la diagnosi relativa al superamento del throughput massimo on demand:

1. Confronta la capacità utilizzata con i limiti di capacità massima: verifica se l'utilizzo si avvicina ai limiti massimi o li supera costantemente.
2. Esamina la frequenza e la tempistica degli eventi di limitazione (della larghezza di banda della rete) per identificare i modelli. Cerca aumenti improvvisi della capacità utilizzata che coincidono con l'evento di limitazione (della larghezza di banda della rete).
3. Utilizzate CloudWatch Contributor Insights per identificare quali elementi o chiavi di partizione consumano più capacità.

Funzionamento della limitazione (della larghezza di banda della rete) e della contropressione di scrittura sugli indici secondari globali in DynamoDB

La limitazione (della larghezza di banda della rete) da contropressione sugli indici secondari globali (GSI) rappresenta uno degli scenari di limitazione (della larghezza di banda della rete) più complessi in DynamoDB perché crea una relazione indiretta tra le operazioni di scrittura e la limitazione: l'applicazione scrive su una tabella di base ma subisce una limitazione (della larghezza di banda della rete) a causa dei vincoli di capacità imposti su uno o più indici.

Funzionamento della limitazione (della larghezza di banda della rete) da contropressione sui GSI

Durante la scrittura su una tabella DynamoDB, tutti i GSI nella tabella vengono aggiornati in modo asincrono secondo un modello di consistenza finale. Se un GSI non dispone di una capacità sufficiente per gestire questi aggiornamenti, DynamoDB sottopone a limitazione (della larghezza di banda della rete) le scritture nella tabella di base per mantenere la coerenza dei dati. Questa operazione è denominata contropressione sui GSI. Per ulteriori informazioni sull'utilizzo dei GSI in DynamoDB, consulta la pagina relativa agli indici secondari globali in DynamoDB.

A differenza della limitazione (della larghezza di banda della rete) diretta della tabella, in cui la risorsa a cui si accede è anche la risorsa che causa la limitazione, la contropressione sui GSI crea una dipendenza tra la tabella di base e i suoi indici. Anche se la tabella di base ha un'ampia capacità, le scritture verranno limitate se un GSI associato non è in grado di gestire il volume di aggiornamento. La comprensione di questa relazione è particolarmente importante perché i vincoli a livello di partizione si applicano indipendentemente sia alla tabella di base che a ciascun GSI: ognuno ha la propria struttura di partizione e i corrispondenti limiti di throughput.

Il partizionamento di un GSI si basa sulla relativa chiave di partizione, che spesso è diversa dalla chiave di partizione della tabella di base. Anche se l'accesso alla tabella di base è perfettamente distribuito tra le partizioni, gli aggiornamenti a un GSI potrebbero comunque concentrarsi su partizioni specifiche, creando punti caldi nel GSI. Per le best practice generali sulla progettazione delle chiavi di partizione sia per le tabelle che per i GSI, consulta la risorsa relativa alla progettazione delle chiavi di partizione di DynamoDB.

Ad esempio, se la tabella di base utilizza `customerId` come chiave di partizione (distribuita equamente) ma il GSI utilizza `status` come chiave di partizione (con valori possibili limitati come "attivo", "in sospeso", "chiuso"), gli aggiornamenti agli elementi con i valori di stato più diffusi possono creare partizioni calde del GSI anche quando l'accesso alla tabella di base è bilanciato. Ciò crea uno scenario particolarmente impegnativo in cui l'applicazione potrebbe subire una limitazione (della larghezza di banda della rete) a causa delle partizioni calde del GSI anche se sia la tabella di base che il GSI hanno una capacità complessiva sufficiente e il modello di accesso della tabella di base appare ben distribuito.

Anche se l'eccezione di limitazione (della larghezza di banda della rete) si riferisce al GSI (tramite `ResourceArn`), l'operazione effettiva che viene limitata è la scrittura nella tabella di base. Ciò può creare confusione perché l'applicazione sta scrivendo sulla tabella di base ma riceve un'eccezione relativa al GSI.

Tipi di limitazione (della larghezza di banda della rete) sui GSI

La limitazione (della larghezza di banda della rete) da contropressione sui GSI si manifesta attraverso diversi tipi di eccezioni a seconda del vincolo di capacità specifico:

- Superamento della capacità allocata per il GSI: si verifica quando il GSI non dispone di unità di capacità di scrittura sufficienti per gestire gli aggiornamenti dalle operazioni della tabella di base. Ciò produce un'eccezione `ProvisionedThroughputExceededException` con il motivo `IndexWriteProvisionedThroughputExceeded` e `ResourceArn` indica direttamente che il GSI specifico presenta vincoli di capacità.
- Superamento del throughput massimo on demand per il GSI: si verifica quando le operazioni di scrittura sul GSI superano i limiti massimi configurati nelle tabelle on demand. Ciò produce un'eccezione `ThrottlingException` con il motivo `IndexWriteMaxOnDemandThroughputExceeded`, che identifica il GSI specifico con restrizioni di throughput configurate.
- Superamento dei limiti di partizione del GSI: si verifica quando le singole partizioni del GSI superano i limiti di throughput (partizioni calde), anche se la capacità complessiva del GSI sembra sufficiente. Ciò genera un'eccezione `ThrottlingException` con il motivo `IndexWriteKeyRangeThroughputExceeded`, che indica problemi di partizioni calde sul GSI specifico identificato in `ResourceArn`. Ciò è particolarmente importante perché la distribuzione delle partizioni del GSI può differire in modo significativo dalla distribuzione delle partizioni della tabella di base, creando hot spot nel GSI anche quando l'accesso alla tabella di base è distribuito equamente.
- Superamento dei limiti dell'account per il GSI: si attiva quando le operazioni di scrittura su un GSI specifico superano i limiti di throughput Regionali per tabella (o per ogni singolo GSI all'interno di quella tabella) stabiliti a livello di account. DynamoDB restituisce un'eccezione `ThrottlingException` con il motivo `IndexWriteAccountLimitExceeded`, che indica il GSI che ne ha spinto l'utilizzo oltre i limiti dell'account. Questa limitazione (della larghezza di banda della rete) avviene indipendentemente per ogni GSI che supera il limite. Per informazioni sulle quote per tabella, per account, Regionali e di servizio, consulta the section called "Quote".

Metriche di limitazione (della larghezza di banda della rete) di CloudWatch

In questa pagina viene fornita una guida completa alle metriche CloudWatch progettate specificamente per aiutare a identificare, diagnosticare e risolvere i problemi di limitazione (della larghezza di banda della rete) nelle tabelle e negli indici DynamoDB.

Metriche generali di limitazione (della larghezza di banda della rete)

- `ThrottledRequests`
  - Incrementa di uno quando un evento all'interno di una richiesta viene sottoposto a limitazione (della larghezza di banda della rete), indipendentemente dal numero di singoli eventi all'interno di quella richiesta. Ad esempio, se si aggiorna un elemento in una tabella con indici secondari globali (GSI), si verificano più eventi, ovvero un'operazione di scrittura nella tabella di base e un'operazione di scrittura in ciascun indice. Se uno di questi singoli eventi viene sottoposto a limitazione (della larghezza di banda della rete), la metrica `ThrottledRequests` viene incrementata una sola volta.

    È importante comprendere questo comportamento per il monitoraggio e la risoluzione dei problemi delle prestazioni di DynamoDB, poiché potrebbe mascherare la reale portata della limitazione (della larghezza di banda della rete). Per informazioni più complete, confronta la metrica `ThrottledRequests` con le metriche specifiche a livello di evento come `ReadThrottleEvents`, `WriteThrottleEvents` e con metriche mirate, come ad esempio `ReadKeyRangeThroughputThrottleEvents`. L'elenco completo di queste metriche specifiche per le cause è disponibile in questa pagina. Ogni metrica corrisponde a particolari motivi di limitazione (della larghezza di banda della rete) che vengono rilevati nell'eccezione di limitazione. Per indicazioni su come recuperare e interpretare questi motivi durante gli eventi di limitazione (della larghezza di banda della rete), consulta la sezione the section called "Diagnosi della limitazione (della larghezza di banda della rete)" che fornisce istruzioni per identificare e risolvere le cause principali di questo tipo di problemi.
- `ReadThrottleEvents`
  - Controlla le richieste che superano l'RCU assegnato per una tabella o un GSI.
- `WriteThrottleEvents`
  - Controlla le richieste che superano l'RCU assegnato per una tabella o un GSI.

Metriche di limitazione (della larghezza di banda della rete) dettagliate per causa

Limitazione (della larghezza di banda della rete) del throughput on demand

- `ReadMaxOnDemandThroughputThrottleEvents`
  - Numero di richieste di lettura sottoposte a limitazione (della larghezza di banda della rete) a causa del throughput massimo on demand.
- `WriteMaxOnDemandThroughputThrottleEvents`

- Numero di richieste di scrittura sottoposte a limitazione (della larghezza di banda della rete) a causa del throughput massimo on demand.

Limitazione (della larghezza di banda della rete) a livello di account

- `ReadAccountLimitThrottleEvents`
  - Numero di richieste di lettura sottoposte a limitazione (della larghezza di banda della rete) a causa dei limiti dell'account.
- `WriteAccountLimitThrottleEvents`
  - Numero di richieste di scrittura sottoposte a limitazione (della larghezza di banda della rete) a causa dei limiti dell'account.

Limitazione (della larghezza di banda della rete) a livello di partizione

- `ReadKeyRangeThroughputThrottleEvents`
  - Numero di richieste di lettura sottoposte a limitazione (della larghezza di banda della rete) a causa dei limiti della partizione.
- `WriteKeyRangeThroughputThrottleEvents`
  - Numero di richieste di scrittura sottoposte a limitazione (della larghezza di banda della rete) a causa dei limiti della partizione.

Metriche di analisi della capacità

- `OnlineIndexConsumedWriteCapacity`
  - Quando si aggiunge un nuovo GSI a una tabella esistente, DynamoDB esegue un'operazione di riempimento che copia i dati dalla tabella di base al nuovo indice. Questo processo consuma unità di capacità di scrittura. La metrica `OnlineIndexConsumedWriteCapacity` tiene traccia di questo consumo specifico.

    Questo consumo è separato e aggiuntivo rispetto alle normali operazioni di scrittura tracciate da `ConsumedWriteCapacityUnits`. La metrica standard `ConsumedWriteCapacityUnits` per il GSI non include la capacità di scrittura utilizzata durante il processo iniziale di creazione dell'indice.
- `ProvisionedReadCapacityUnits` e `ProvisionedWriteCapacityUnits`
  - Visualizza quante unità di capacità di lettura o di scrittura assegnate sono state consumate nel periodo di tempo specificato, per una tabella o un indice secondario globale specificato.
  - Notare che la dimensione `TableName` restituisce `ProvisionedReadCapacityUnits` per la tabella solo per impostazione predefinita. Per visualizzare il numero di unità di capacità di

lettura o di scrittura allocate per un indice secondario globale, è necessario specificare anche `TableName` e `GlobalSecondaryIndexName`.

- `ConsumedReadCapacityUnits` e `ConsumedWriteCapacityUnits`
  - Visualizza quante unità di capacità di lettura o scrittura sono state utilizzate nel periodo di tempo specificato. `ConsumedWriteCapacityUnits` non include la capacità di scrittura utilizzata durante il processo iniziale di creazione dell'indice.

Per ulteriori informazioni sulle metriche CloudWatch per DynamoDB, consulta Parametri e dimensioni di DynamoDB.

Appendice DynamoDB

Argomenti

- Risoluzione dei problemi di creazione della SSL/TLS connessione con DynamoDB
- Tabelle e dati di esempio da utilizzare in DynamoDB
- Creazione di tabelle di esempio e caricamento di dati in DynamoDB
- Applicazione di esempio DynamoDB che utilizza: AWS SDK for Python (Boto) Tic-tac-toe
- Parole riservate in DynamoDB
- AWS Esempi di SDK for Java 1.x
- AWS Esempi di SDK for Go 1.x
- AWS SDK per esempi di Node.js 2.x

Risoluzione dei problemi di creazione della SSL/TLS connessione con DynamoDB

Amazon DynamoDB sposterà a breve gli endpoint per proteggere i certificati firmati dall'autorità di certificazione Amazon Trust Services (ATS) anziché dall'autorità di certificazione di terze parti. A dicembre 2017 abbiamo avviato la regione EU-WEST-3 (Parigi) con i certificati protetti rilasciati da Amazon Trust Services. Tutte le nuove regioni avviate dopo dicembre 2017 hanno gli endpoint con i certificati rilasciati da Amazon Trust Services. In questa guida viene indicato come convalidare e risolvere i problemi della connessione SSL/TLS.

Test dell'applicazione o del servizio

La maggior parte delle AWS SDKs interfacce a riga di comando (CLIs) supportano l'autorità di certificazione Amazon Trust Services. Se utilizzi una versione dell' AWS SDK per Python o della CLI rilasciata prima del 29 ottobre 2013, devi eseguire l'aggiornamento. .NET, Java, PHP JavaScript, Go e C++ SDKs e CLIs non raggruppano alcun certificato, i relativi certificati provengono dal sistema operativo sottostante. L'SDK di Ruby include almeno uno dei requisiti richiesti CAs dal 10 giugno 2015. Prima di quella data, l'SDK Ruby V2 non creava bundle di certificati. Se utilizzi una versione non supportata, personalizzata o modificata dell' AWS SDK o se utilizzi un trust store personalizzato, potresti non disporre del supporto necessario per Amazon Trust Services Certificate Authority.

Per convalidare l'accesso agli endpoint DynamoDB, devi sviluppare un test che acceda all'API DynamoDB o all'API DynamoDB Streams nella regione EU-WEST-3 e verificare che l'handshake TLS venga superato. Gli endpoint specifici a cui devi accedere in questo test sono:

- DynamoDB: https://dynamodb.eu-west-3.amazonaws.com
- Streams DynamoDB: https://streams.dynamodb.eu-west-3.amazonaws.com

Se l'applicazione non supporta l'autorità di certificazione Amazon Trust Services, verrà restituita una delle seguenti tipologie di errore:

- SSL/TLS Errori di negoziazione
- Un lungo ritardo prima che il software riceva un errore che indica un fallimento SSL/TLS della negoziazione. La durata del ritardo dipende dalla strategia relativa ai nuovi tentativi e dalla configurazione del timeout del client.

Test del browser del client

Per verificare che il tuo browser sia in grado di connettersi ad Amazon DynamoDB, apri il seguente URL:. https://dynamodb.eu-west-3.amazonaws.com Se il test viene superato, vedrai un messaggio come questo:

```
healthy: dynamodb.eu-west-3.amazonaws.com
```

Se il test non ha esito positivo, verrà visualizzato un errore simile a questo:/. https://untrusted-root.badssl.com

Aggiornamento del client dell'applicazione software

Le applicazioni che accedono agli endpoint API DynamoDB o DynamoDB Streams (tramite browser o a livello di codice) dovranno aggiornare l'elenco delle CA attendibili sui computer client se non supportano già nessuno dei seguenti elementi: CAs

- Autorità di certificazione root Amazon 1
- Autorità di certificazione root dei servizi Starfield - G2
- Autorità di certificazione Starfield classe 2

Se i client si fidano già di UNO qualsiasi dei tre precedenti, questi CAs si fideranno dei certificati utilizzati da DynamoDB e non è richiesta alcuna azione. Tuttavia, se i tuoi clienti non si fidano già di nessuno dei precedenti CAs, le connessioni HTTPS a DynamoDB o DynamoDB Streams falliranno. APIs Per ulteriori informazioni, consulta questo post del blog: https://aws.amazon.com/blogs/ security/ - -/. how-to-prepare-for aws-move-to-its own-certificate-authority

Aggiornamento del browser del client

Puoi aggiornare il bundle di certificati del browser semplicemente applicando l'aggiornamento del browser. Le istruzioni per i browser più comuni sono disponibili nei relativi siti Web:

- Chrome: risposta/95414https://support.google.com/chrome/? hl=it
- Firefox: https://support.mozilla. org/en-US/kb/update- firefox-latest-version
- Safari: https://support.apple.com/en-us/HT204:16
- Internet Explorer: https://support.microsoft.com/en-us/windows-internet-explorer-whichhelp/17295/ - version #ie =altro

Aggiornamento manuale del bundle di certificati

Se non è possibile accedere all'API DynamoDB o all'API dei flussi DynamoDB, è necessario aggiornare il bundle di certificati. Per fare ciò, devi importare almeno uno dei file richiesti. CAs Puoi trovarli all'indirizzo https://www.amazontrust.com/repository/.

I sistemi operativi e i linguaggi di programmazione seguenti supportano i certificati Amazon Trust Services:

- Versioni di Microsoft Windows con gli aggiornamenti di gennaio 2005 o successivi, Windows Vista, Windows 7, Windows Server 2008 e versioni più recenti.
- MacOS X 10.4 con Java per MacOS X 10.4 Release 5, MacOS X 10.5 e versioni più recenti.
- Red Hat Enterprise Linux 5 (marzo 2007), Linux 6 e Linux 7 e CentOS 5, CentOS 6 e CentOS 7
- Ubuntu 8.10
- Debian 5.0
- Amazon Linux (tutte le versioni)
- Java 1.4.2_12, Java 5 update 2 e tutte le versioni più recenti, incluse Java 6, Java 7 e Java 8

Se non riesci ancora a connetterti, consulta la documentazione del software, il fornitore del sistema operativo o contatta AWS https://aws.amazon.comSupport/support per ulteriore assistenza.

Tabelle e dati di esempio da utilizzare in DynamoDB

Nella Guida per gli sviluppatori di Amazon DynamoDB vengono utilizzate tabelle di esempio per illustrare vari aspetti di DynamoDB.

Nome tabella	Chiave primaria
ProductCatalog	Chiave primaria semplice: • `Id` (numero).
Forum	Chiave primaria semplice: • `Name` (stringa)
Thread	Chiave primaria composita: • `ForumName` (stringa) • `Subject` (stringa)
Reply	Chiave primaria composita: • `Id` (stringa) • `ReplyDateTime` (stringa)

La tabella Reply ha un indice secondario globale denominato PostedBy-Message-Index. Questo indice facilita le query su due attributi non chiave della tabella Reply.

Nome indice	Chiave primaria
PostedBy-Message-Index	Chiave primaria composita: • `PostedBy` (stringa) • `Message` (stringa)

Per ulteriori informazioni su queste tabelle, consulta Fase 1: creare una tabella DynamoDB e Fase 2: scrivere dati in una tabella DynamoDB.

File di dati di esempio

Argomenti

- Dati di esempio ProductCatalog
- Dati di esempio Forum
- Dati di esempio Thread
- Dati di esempio Reply

Nelle sezioni seguenti vengono illustrati i file di dati di esempio utilizzati per caricare le tabelle ProductCatalog, Forum, Thread e Reply.

Ogni file di dati contiene più elementi `PutRequest`, ognuno dei quali contiene un singolo elemento. Questi elementi `PutRequest` vengono utilizzati come input per l'operazione `BatchWriteItem` tramite AWS Command Line Interface (AWS CLI).

Dati di esempio 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"
                    }
                }
            }
        }
    ]
}
```

Dati di esempio 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"}
```

```
                }
            }
        }
    ]
}
```

Dati di esempio 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"
                            }
```

```
                        ]
                    }
                }
            }
        }
    ]
}
```

Dati di esempio 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"
                    }
                }
            }
        }
    ]
}
```

Creazione di tabelle di esempio e caricamento di dati in DynamoDB

Questa appendice fornisce il codice per creare le tabelle e aggiungere i dati a livello di programmazione.

Argomenti

- Creazione di tabelle di esempio e caricamento di dati utilizzando AWS SDK per Java
- Creazione di tabelle di esempio e caricamento di dati utilizzando AWS SDK per .NET

Creazione di tabelle di esempio e caricamento di dati utilizzando AWS SDK per Java

Il seguente esempio di codice Java consente di creare le tabelle e caricare i dati nelle tabelle. Per step-by-step istruzioni su come eseguire questo codice utilizzando Eclipse, consulta. Esempi di codice 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());

        }
    }

}
```

Creazione di tabelle di esempio e caricamento di dati utilizzando AWS SDK per .NET

Il seguente esempio di codice C# consente di creare le tabelle e caricare i dati nelle tabelle. Per step-by-step istruzioni su come eseguire questo codice in Visual Studio, consulta. Esempi di codice .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);
        }
    }
}
```

Applicazione di esempio DynamoDB che utilizza: AWS SDK for Python (Boto) Tic-tac-toe

Il Tic-Tac-Toe gioco è un esempio di applicazione web basata su Amazon DynamoDB. L'applicazione utilizza il AWS SDK for Python (Boto) per effettuare le chiamate DynamoDB necessarie per archiviare i dati di gioco in una tabella DynamoDB e il framework web Python Flask per end-to-end illustrare lo sviluppo di applicazioni in DynamoDB, incluso come modellare i dati. Vengono illustrate anche le best practice per la modellazione dei dati in DynamoDB, tra cui la tabella creata per l'applicazione di gioco, la chiave primaria definita, gli indici aggiuntivi necessari in base ai requisiti di query e l'uso di attributi con valori concatenati.

L'applicazione sul Web viene utilizzata nel modo seguente: Tic-Tac-Toe

1. Accedi alla home page dell'applicazione.
2. Invita quindi un altro utente a partecipare al gioco come avversario.

   Fino a quando un altro utente non accetta il tuo invito, lo stato della partita rimane `PENDING`. Quando un avversario accetta l'invito, lo stato della partita diventa `IN_PROGRESS`.
3. La partita inizia dopo che l'avversario accede e accetta l'invito.
4. L'applicazione archivia tutte le mosse della partita e le informazioni sullo stato in una tabella DynamoDB.
5. La partita finisce con una vittoria o un pareggio e lo stato passa a `FINISHED`.

L'esercizio di creazione dell' end-to-endapplicazione è descritto nei passaggi seguenti:

- Fase 1: distribuzione e test in locale: in questa sezione, viene scaricata, implementata e testata l'applicazione di gioco tris nel computer locale. Creerai le tabelle necessarie nella versione scaricabile di DynamoDB.
- Fase 2: esame del modello di dati e dei dettagli di implementazione : in questa sezione viene descritto in primo luogo in dettaglio il modello di dati, inclusi gli indici e l'uso dell'attributo valore concatenato. Viene quindi illustrato il funzionamento dell'applicazione.
- Fase 3: distribuzione in produzione tramite il servizio DynamoDB: questa sezione è incentrata sulle considerazioni per l'implementazione in un ambiente di produzione. In questa fase viene creata una tabella usando il servizio Amazon DynamoDB e l'applicazione viene distribuita usando AWS Elastic Beanstalk. Quando l'applicazione è in produzione, vengono concesse inoltre le autorizzazioni appropriate in modo che l'applicazione possa accedere alla tabella DynamoDB. Le istruzioni contenute in questa sezione illustrano la distribuzione in end-to-end produzione.
- Fase 4: Eliminazione delle risorse: in questa sezione vengono evidenziate le aree non coperte da questo esempio. La sezione fornisce inoltre le istruzioni per rimuovere le AWS risorse create nei passaggi precedenti in modo da evitare di incorrere in addebiti.

Fase 1: distribuzione e test in locale

Argomenti

- 1.1: Download e installazione dei pacchetti richiesti
- 1.2: Test dell'applicazione di gioco

In questo passaggio scarichi, distribuisci e testerai l'applicazione di Tic-Tac-Toe gioco sul tuo computer locale. Invece di usare il servizio Web Amazon DynamoDB, scaricherai DynamoDB nel computer e creerai la tabella necessaria in questa posizione.

1.1: Download e installazione dei pacchetti richiesti

Per testare l'applicazione in locale, è necessario quanto segue:

- Python
- Flask (un microframework per Python)
- AWS SDK for Python (Boto)
- DynamoDB in esecuzione nel computer
- Git

Per ottenere questi strumenti, procedi come indicato di seguito:

1. Installare Python. Per step-by-step istruzioni, consulta Download Python.

   L' Tic-Tac-Toeapplicazione è stata testata utilizzando Python versione 2.7.
2. Installa Flask e AWS SDK for Python (Boto) usa Python Package Installer (PIP):
   - Installa PIP.

     Per istruzioni, consulta Installazione di PIP. Nella pagina di installazione scegli il collegamento get-pip.py e quindi salva il file. Apri un terminale dei comandi come amministratore e inserisci quanto segue al prompt dei comandi:

     ```
     python.exe get-pip.py
     ```

     In Linux non devi specificare l'estensione `.exe`. Specifichi solo `python get-pip.py`.
   - Usando PIP, installa i pacchetti Flask e Boto tramite il codice seguente.

     ```
     pip install Flask
     pip install boto
     pip install configparser
     ```
3. Scaricare DynamoDB sul computer. Per istruzioni sull'esecuzione, consulta Configurazione di DynamoDB locale (versione scaricabile) .

4. Scarica l'applicazione: Tic-Tac-Toe
   a. Installa Git. Per istruzioni, consulta la pagina relativa ai download di git.
   b. Per scaricare l'applicazione, eseguire il codice descritto di seguito.

```
git clone https://github.com/awslabs/dynamodb-tictactoe-example-app.git
```

1.2: Test dell'applicazione di gioco

Per testare l' Tic-Tac-Toeapplicazione, è necessario eseguire DynamoDB localmente sul computer.

Per eseguire l'applicazione tic-tac-toe

1. Avviare DynamoDB.
2. Avviare il server Web per l' Tic-Tac-Toeapplicazione.

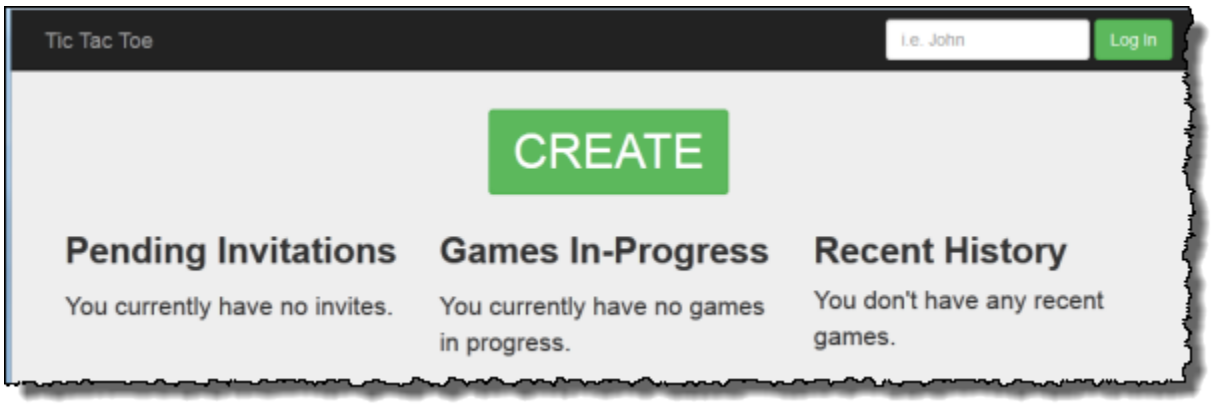
   A tale scopo, aprite un terminale di comando, accedete alla cartella in cui avete scaricato l' Tic-Tac-Toeapplicazione ed eseguite l'applicazione localmente utilizzando il codice seguente.

```
python.exe application.py --mode local --serverPort 5000 --port 8000
```

   In Linux non devi specificare l'estensione `.exe`.
3. Apri il browser Web e inserisci quanto segue.

```
http://localhost:5000/
```

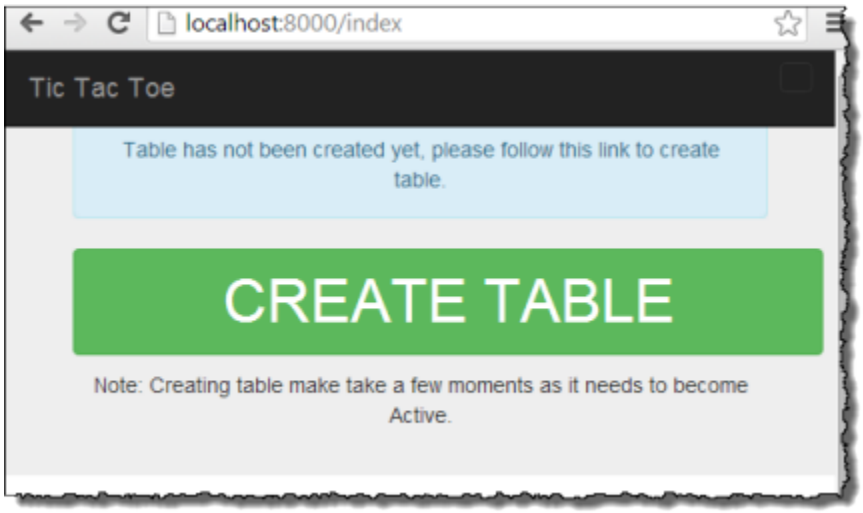
   Nel browser viene visualizzata la home page.



4. Inserisci **`user1`** nella casella Log in (Accesso) per accedere come user1.

> **Note**
>
> Questa applicazione di esempio non esegue l'autenticazione utente. L'ID utente viene usato solo per identificare i giocatori. Se due giocatori accedono con lo stesso alias, l'applicazione funziona come se stessi giocando in due browser diversi.

5. Se è la prima volta che si gioca, viene visualizzata una pagina in cui viene richiesto di creare la tabella necessaria (`Games`) in DynamoDB. Scegli CREATE TABLE (CREA TABELLA).



6. Scegli CREATE per creare il primo tic-tac-toe gioco.
7. Inserisci **`user2`** nella casella Scegli un avversario e scegli Crea partita.

   In questo modo, viene creata la partita aggiungendo un item nella tabella `Games`. Lo stato della partita viene impostato su `PENDING`.

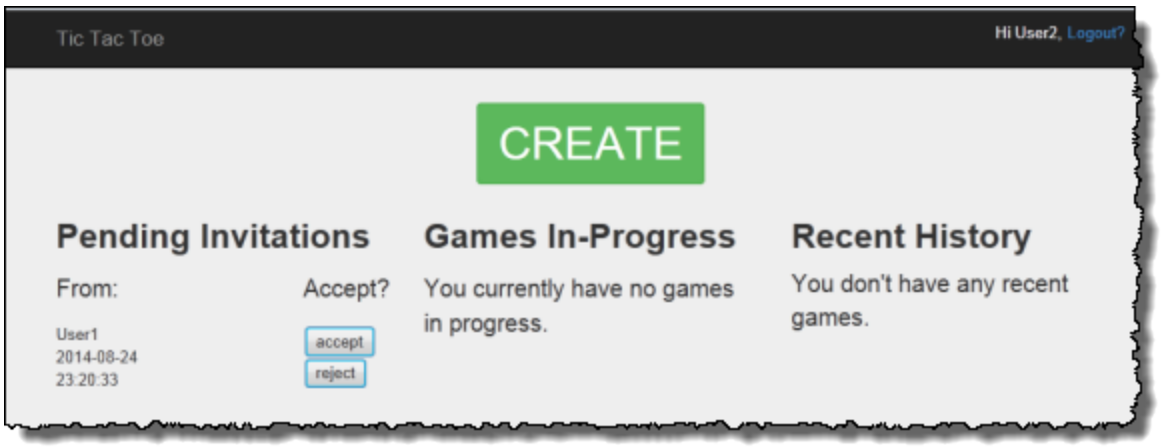
8. Apri un'altra finestra del browser e inserisci quanto segue.

```
http://localhost:5000/
```

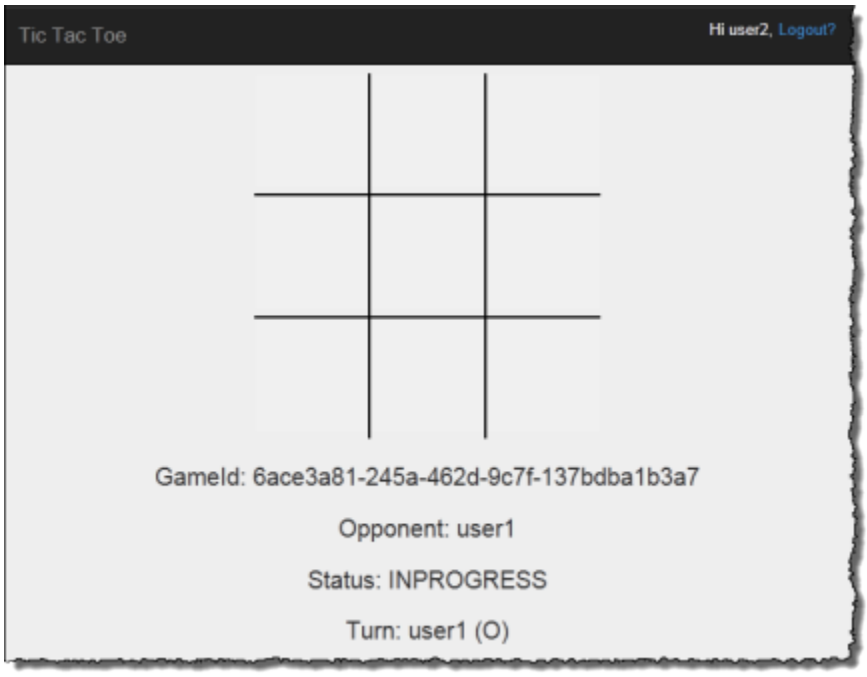
Il browser passa le informazioni tramite cookie, quindi devi usare la modalità di navigazione in incognito o di esplorazione privata, in modo che i cookie non vengano trasferiti.

9. Accedi come user2.

   Viene visualizzata una pagina che mostra un invito in attesa da user1.



10. Scegli accept (accetta) per accettare l'invito.



   La pagina del gioco appare con una tic-tac-toe griglia vuota. Nella pagina sono visualizzate anche informazioni rilevanti sulla partita, come l'ID della partita, il giocatore di turno e lo stato della partita.

11. Gioca la partita.

Per ogni mossa dell'utente, il servizio Web invia una richiesta a DynamoDB per l'aggiornamento condizionale dell'elemento di gioco nella tabella `Games`. Le condizioni garantiscono, ad esempio, che la mossa sia valida, che il quadrato scelto dall'utente sulla griglia sia disponibile e che fosse il turno dell'utente che ha fatto la mossa. Per una mossa valida, l'operazione di aggiornamento aggiunge un nuovo attributo corrispondente alla selezione sulla griglia. L'operazione di aggiornamento imposta anche il valore dell'attributo esistente sull'utente che può fare la mossa successiva.

Nella pagina del gioco, l'applicazione effettua JavaScript chiamate asincrone ogni secondo, per un massimo di 5 minuti, per verificare se lo stato del gioco in DynamoDB è cambiato. In caso affermativo, l'applicazione aggiorna la pagina con le nuove informazioni. Dopo 5 minuti, l'applicazione smette di eseguire richieste ed è necessario aggiornare la pagina manualmente per ottenere le informazioni aggiornate.

Fase 2: esame del modello di dati e dei dettagli di implementazione

Argomenti

- 2.1: Modello di dati di base
- 2.2: Funzionamento dell'applicazione (procedura guidata per il codice)

2.1: Modello di dati di base

Questa applicazione di esempio mette in evidenza i concetti seguenti relativi al modello di dati di DynamoDB:

- Tabella: in DynamoDB, una tabella è una raccolta di elementi (ovvero record) e ogni elemento è una raccolta di coppie nome-valore denominate attributi.

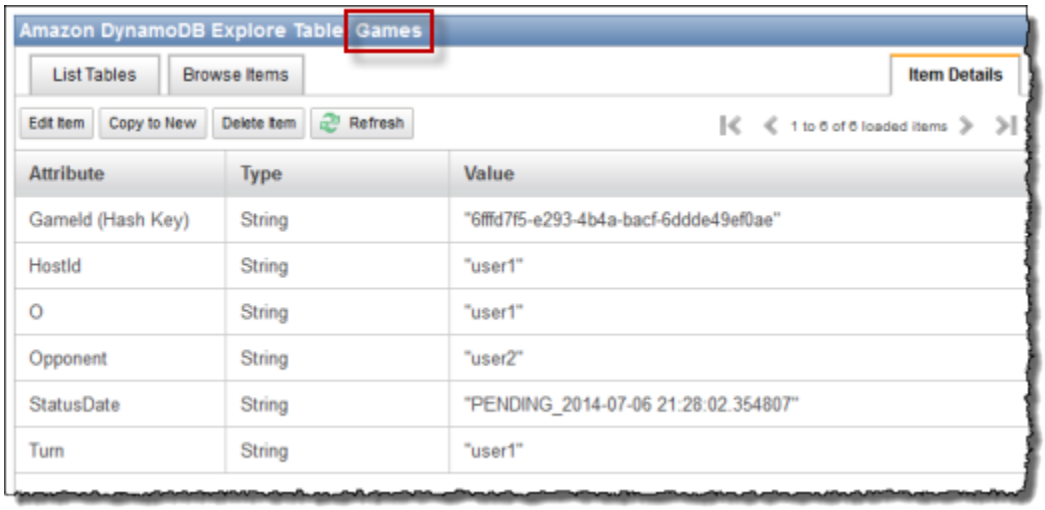
  In questo Tic-Tac-Toe esempio, l'applicazione memorizza tutti i dati di gioco in una tabella,. `Games` L'applicazione crea una item nella tabella per ogni partita e archivia tutti i dati di gioco come attributi. Una tic-tac-toe partita può avere fino a nove mosse. Poiché le tabelle DynamoDB non hanno uno schema nei casi in cui l'attributo richiesto è solo la chiave primaria, l'applicazione può archiviare un numero variabile di attributi per elemento di gioco.

  La tabella `Games` ha una chiave primaria semplice costituita da un attributo, `GameId`, di tipo stringa. L'applicazione assegna un ID univoco a ogni partita. Per ulteriori informazioni sulle chiavi primarie di DynamoDB, vedi Chiave primaria.

Quando un utente inizia una tic-tac-toe partita invitando un altro utente a giocare, l'applicazione crea un nuovo elemento nella `Games` tabella con attributi che memorizzano i metadati del gioco, come i seguenti:

- `HostId`, l'utente che ha avviato la partita.
- `Opponent`, l'utente che è stato invitato a giocare.
- Utente che deve giocare il turno successivo. L'utente che avvia la partita gioca per primo.
- Utente che usa il simbolo O sulla griglia di gioco. L'utente che avvia la partita usa il simbolo O.

L'applicazione crea anche un attributo concatenato `StatusDate`, contrassegnando lo stato iniziale della partita come `PENDING`. Lo screenshot seguente mostra un elemento di esempio come appare nella console DynamoDB:

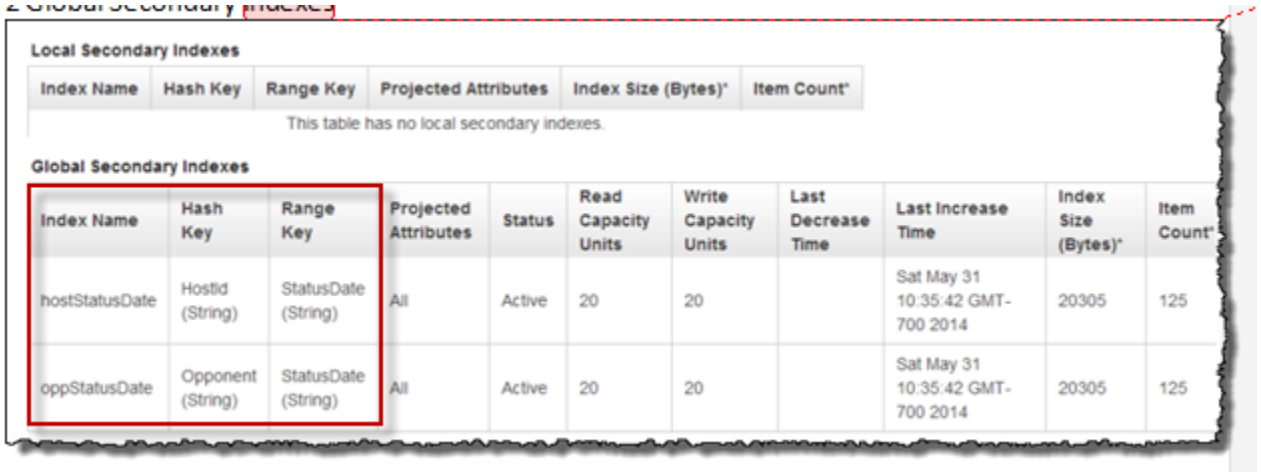


Man mano che la partita procede, l'applicazione aggiunge un attributo alla tabella per ogni mossa nel gioco. Il nome dell'attributo è costituito dalla posizione sulla griglia, ad esempio `TopLeft` o `BottomRight`. Una mossa, ad esempio, può avere un attributo `TopLeft` con il valore `0`, un attributo `TopRight` con il valore 0 e un attributo `BottomRight` con il valore `X`. Il valore dell'attributo è 0 o `X`, a seconda dell'utente che ha fatto la mossa. Considera, ad esempio, la griglia seguente.

- Attributi dei valori concatenati: l'attributo `StatusDate`illustra un attributo valore concatenato. In base a questo approccio, anziché creare attributi separati per archiviare lo stato della partita (`PENDING`, `IN_PROGRESS` e `FINISHED`) e la data (dell'ultima mossa), i valori vengono combinati in un unico attributo, ad esempio `IN_PROGRESS_2014-04-30 10:20:32`.

  L'applicazione usa quindi l'attributo `StatusDate` per la creazione di indici secondari specificando `StatusDate` come chiave di ordinamento per l'indice. Il vantaggio dell'uso dell'attributo con valori concatenati `StatusDate` è spiegato ulteriormente negli indici illustrati più avanti.

- Indici secondari globali: è possibile utilizzare la chiave primaria della tabella, `GameId`, per interrogare in modo efficiente la tabella per trovare un oggetto di gioco. Per eseguire query sulla tabella in base ad attributi diversi da quelli della chiave primaria, DynamoDB supporta la creazione di indici secondari. In questa applicazione di esempio vengono creati i due indici secondari seguenti:



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- -indice. StatusDate Questo indice ha `HostId` come chiave di partizione e `StatusDate` come chiave di ordinamento. Puoi usare l'indice per eseguire query su `HostId`, ad esempio per trovare le partite ospitate da un utente specifico.
- OpponentId- StatusDate -indice. Questo indice ha `OpponentId` come chiave di partizione e `StatusDate` come chiave di ordinamento. Puoi usare l'indice per eseguire query su `Opponent`, ad esempio per trovare le partite in cui un utente specifico è l'avversario.

Questi indici vengono detti indici secondari globali perché la loro chiave di partizione non corrisponde alla chiave di partizione (`GameId`) usata nella chiave primaria della tabella.

Entrambi gli indici hanno `StatusDate` come chiave di ordinamento. Ciò permette quanto segue:

- Puoi eseguire query usando l'operatore di confronto `BEGINS_WITH`. Puoi ad esempio trovare tutte le partite con l'attributo `IN_PROGRESS` ospitate da un utente specifico. In questo caso, l'operatore `BEGINS_WITH` cerca il valore `StatusDate` che inizia con `IN_PROGRESS`.
- DynamoDB archivia gli elementi nell'indice ordinandoli in base al valore della chiave di ordinamento. Se pertanto tutti i prefissi relativi allo stato sono uguali (ad esempio, `IN_PROGRESS`), il formato ISO usato per la parte relativa alla data farà in modo che gli elementi vengano ordinati dal meno recente al più recente. Questo approccio permette di eseguire in modo efficace determinate query, come le seguenti:
  - Recupera fino a 10 partite `IN_PROGRESS` più recenti ospitate dall'utente che ha eseguito l'accesso. Per questa query, specifica l'indice `HostId-StatusDate-index`.
  - Recupera fino a 10 partite `IN_PROGRESS` più recenti in cui l'utente che ha eseguito l'accesso è l'avversario. Per questa query, specifica l'indice `OpponentId-StatusDate-index`.

Per ulteriori informazioni sugli indici secondari, consulta Miglioramento dell'accesso ai dati con gli indici secondari in DynamoDB.

2.2: Funzionamento dell'applicazione (procedura guidata per il codice)

Questa applicazione ha due pagine principali:

- Pagina iniziale: questa pagina fornisce all'utente un semplice accesso, un pulsante CREA per creare una nuova tic-tac-toe partita, un elenco delle partite in corso, la cronologia delle partite e tutti gli inviti a giocare attivi in sospeso.

  La home page non si aggiorna automaticamente, ma deve essere aggiornata manualmente per aggiornare gli elenchi.

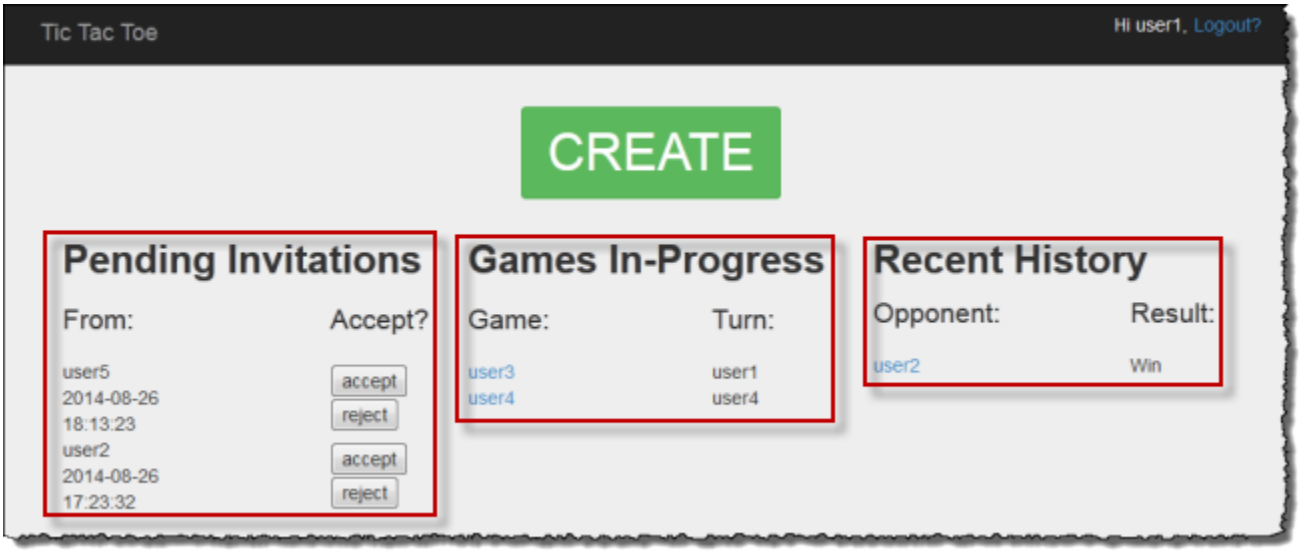
- Pagina del gioco: questa pagina mostra la tic-tac-toe griglia in cui giocano gli utenti.

  L'applicazione aggiorna la pagina del gioco automaticamente ogni secondo. JavaScript Nel tuo browser richiama il server web Python ogni secondo per chiedere alla tabella Giochi se gli elementi del gioco nella tabella sono cambiati. In caso affermativo, JavaScript attiva un aggiornamento della pagina in modo che l'utente veda la bacheca aggiornata.

Analizziamo nel dettaglio il funzionamento dell'applicazione.

Home page

Dopo l'accesso dell'utente, l'applicazione visualizza i tre elenchi di informazioni seguenti.



- Inviti: in questo elenco vengono visualizzati fino a 10 inviti più recenti da altri che sono in attesa di accettazione da parte dell'utente che ha eseguito l'accesso. Nello screenshot precedente l'utente user1 ha inviti in attesa da user5 e user2.
- Giochi in corso: questo elenco mostra fino ai 10 giochi più recenti in corso. Si tratta delle partite a cui l'utente sta giocando, che hanno stato `IN_PROGRESS`. Nello screenshot, l'utente1 sta giocando attivamente con user3 e tic-tac-toe user4.
- Cronologia recente: questo elenco mostra fino ai 10 giochi più recenti che l'utente ha terminato, che hanno lo stato `FINISHED`. Nello screenshot user1 ha giocato in precedenza con user2. Per ogni partita completata viene indicato il risultato.

Nel codice la funzione `index` (in `application.py`) esegue le tre chiamate seguenti per recuperare informazioni sullo stato della partita:

```
inviteGames     = controller.getGameInvites(session["username"])
inProgressGames = controller.getGamesWithStatus(session["username"], "IN_PROGRESS")
finishedGames   = controller.getGamesWithStatus(session["username"], "FINISHED")
```

Ognuna di queste chiamate restituisce un elenco di elementi di DynamoDB inclusi in oggetti `Game`. È possibile estrarre i dati da questi oggetti nella vista in modo semplice. La funzione index passa gli elenchi di oggetti alla vista per il rendering in formato HTML.

```
return render_template("index.html",
                       user=session["username"],
                       invites=inviteGames,
                       inprogress=inProgressGames,
                       finished=finishedGames)
```

L' Tic-Tac-Toeapplicazione definisce la `Game` classe principalmente per archiviare i dati di gioco recuperati da DynamoDB. Queste funzioni restituiscono elenchi di oggetti `Game` che permettono di isolare il resto dell'applicazione dal codice correlato agli elementi Amazon DynamoDB. Queste funzioni aiutano quindi a separare il codice dell'applicazione dai dettagli del livello datastore.

Il pattern architettonico qui descritto viene anche chiamato pattern di interfaccia utente model-view-controller (MVC). In questo caso, le istanze dell'oggetto `Game` (che rappresentano i dati) sono il modello e la pagina HTML è la vista. Il controller è diviso in due file. Il file `application.py` ha la logica del controller per il framework Flask e la logica di business è isolata nel file `gameController.py`. Ciò significa che l'applicazione archivia tutti i dati correlati all'SDK DynamoDB in un file distinto nella cartella `dynamodb`.

Esaminiamo le tre funzioni e come eseguono query sulla tabella Games usando gli indici secondari globali per recuperare i dati rilevanti.

Si usa getGameInvites per ottenere l'elenco degli inviti di gioco in sospeso

La funzione `getGameInvites` restituisce l'elenco dei 10 inviti in attesa più recenti. Queste partite sono state create dagli utenti, ma gli avversari non hanno accettato l'invito. Per queste partite, lo stato rimane `PENDING` fino a quando l'avversario non accetta l'invito. Se l'avversario rifiuta l'invito, l'applicazione rimuove l'item corrispondente dalla tabella.

La funzione specifica la query come segue:

- Specifica l'indice `OpponentId-StatusDate-index` da usare con i valori delle chiavi di indice e gli operatori di confronto seguenti:

- La chiave di partizione è `OpponentId` e accetta la chiave di indice *`user ID`*.
- La chiave di ordinamento è `StatusDate` e accetta l'operatore di confronto e il valore della chiave di indice `beginswith="PENDING_"`.

  È possibile utilizzare l'indice `OpponentId-StatusDate-index` per recuperare i giochi a cui è invitato l'utente connesso, ovvero dove l'utente connesso è l'avversario.
- La query prevede un limite di 10 item nel risultato.

```
gameInvitesIndex = self.cm.getGamesTable().query(
                                                Opponent__eq=user,
                                                StatusDate__beginswith="PENDING_",
                                                index="OpponentId-StatusDate-index",
                                                limit=10)
```

Nell'indice, per ogni valore `OpponentId` (la chiave di partizione) DynamoDB mantiene gli elementi ordinati in base a `StatusDate` (la chiave di ordinamento). Le partite restituite dalla query saranno quindi le 10 più recenti.

Utilizzo di getGamesWith Status per ottenere l'elenco dei giochi con uno stato specifico

Quando un avversario accetta un invito a una partita, lo stato diventa `IN_PROGRESS`. Al termine della partita, lo stato cambia in `FINISHED`.

Le query per trovare le partite in corso o terminate sono uguali, ad eccezione del valore di stato diverso. L'applicazione definisce quindi la funzione `getGamesWithStatus`, che accetta un valore di stato come parametro.

```
inProgressGames = controller.getGamesWithStatus(session["username"], "IN_PROGRESS")
finishedGames   = controller.getGamesWithStatus(session["username"], "FINISHED")
```

La sezione seguente si riferisce alle partite in corso, ma la stessa descrizione si applica alle partite terminate.

Un elenco delle partite in corso per un utente specifico include entrambi i tipi seguenti:

- Partite in corso ospitate dall'utente
- Partite in corso in cui l'utente è l'avversario

La funzione `getGamesWithStatus` esegue le due query seguenti, ogni volta usando l'indice secondario appropriato.

- La funzione esegue una query sulla tabella `Games` usando l'indice `HostId-StatusDate-index`. Per l'indice, la query specifica i valori della chiave primaria, sia la chiave di partizione (`HostId`) che la chiave di ordinamento (`StatusDate`), insieme agli operatori di confronto.

  ```
  hostGamesInProgress = self.cm.getGamesTable ().query(HostId__eq=user,
                                                       StatusDate__beginswith=status,
                                                       index="HostId-StatusDate-index",
                                                       limit=10)
  ```

  Ecco la sintassi Python per gli operatori di confronto:

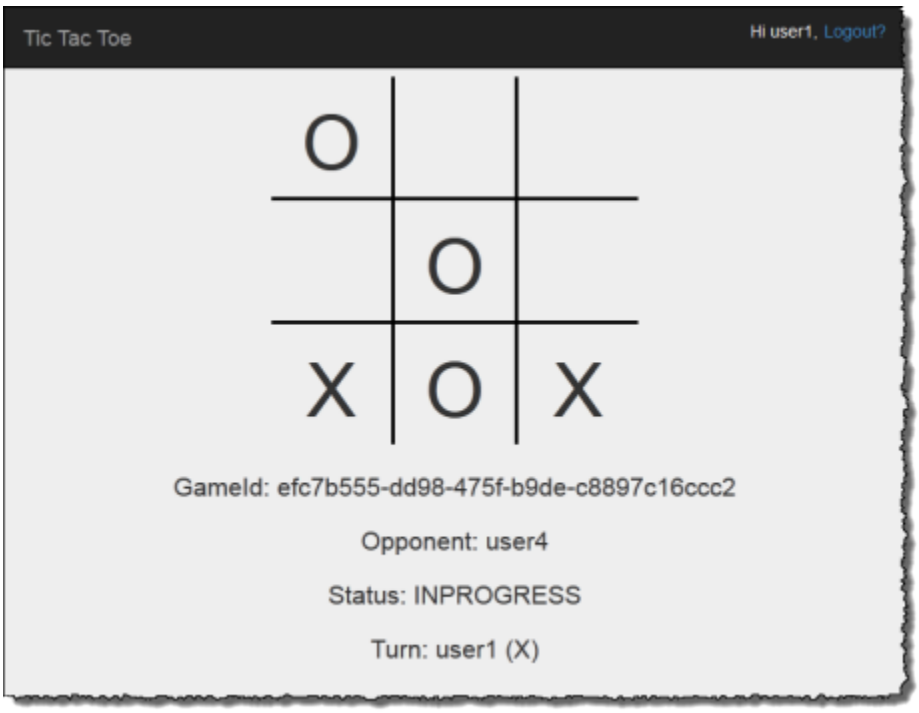
  - `HostId__eq=user` specifica l'operatore di confronto di uguaglianza.
  - `StatusDate__beginswith=status` specifica l'operatore di confronto `BEGINS_WITH`.
- La funzione esegue una query sulla tabella `Games` usando l'indice `OpponentId-StatusDate-index`.

  ```
  oppGamesInProgress = self.cm.getGamesTable().query(Opponent__eq=user,
                                                     StatusDate__beginswith=status,
                                                     index="OpponentId-StatusDate-index",
                                                     limit=10)
  ```
- La funzione combina quindi i due elenchi, li ordina, crea un elenco di oggetti `Game` per i primi da 0 a 10 item e restituisce l'elenco alla funzione chiamante, ovvero all'indice.

  ```
  games = self.mergeQueries(hostGamesInProgress,
                            oppGamesInProgress)
  return games
  ```

Pagina del gioco

La pagina del gioco è dove l'utente tic-tac-toe gioca. In questa pagina sono presenti la griglia di gioco e informazioni sulla partita. Lo screenshot seguente mostra una partita di esempio in corso:



L'applicazione visualizza la pagina del gioco nelle situazioni seguenti:

- L'utente crea una partita invitando un altro utente a giocare.

  In questo caso, la pagina mostra l'utente come host e lo stato della partita come `PENDING`, in attesa che l'avversario accetti l'invito.
- L'utente accetta uno degli inviti in attesa nella home page.

  In questo caso, la pagina mostra l'utente come avversario e lo stato della partita come `IN_PROGRESS`.

Quando l'utente effettua una selezione sulla griglia, viene generata una richiesta `POST` per l'applicazione. Flask chiama la funzione `selectSquare` (in `application.py`) con i dati in formato HTML. Questa funzione, a sua volta, chiama la funzione `updateBoardAndTurn` (in `gameController.py`) per aggiornare l'item di gioco come indicato di seguito:

- Aggiunge un nuovo attributo specifico della mossa.
- Aggiorna il valore dell'attributo `Turn` in base all'utente che deve giocare il turno successivo.

```
controller.updateBoardAndTurn(item, value, session["username"])
```

La funzione restituisce true se l'aggiornamento dell'item è stato eseguito correttamente, altrimenti restituisce false. Tieni presente quanto segue in relazione alla funzione `updateBoardAndTurn`:

- La funzione chiama la funzione `update_item` di SDK for Python per apportare un set specifico di aggiornamenti a un elemento esistente. La funzione è mappata all'operazione `UpdateItem` in DynamoDB. Per ulteriori informazioni, consulta UpdateItem.

> **Note**
>
> La differenza tra le operazioni `UpdateItem` e `PutItem` è legata al fatto che `PutItem` sostituisce l'intero item. Per ulteriori informazioni, consulta PutItem.

Per la chiamata di `update_item`, il codice identifica quanto segue:

- La chiave primaria della tabella `Games`, ovvero `ItemId`.

```
key = { "GameId" : { "S" : gameId } }
```

- Il nuovo attributo da aggiungere, specifico della mossa dell'utente corrente, e il relativo valore (ad esempio `TopLeft="X"`).

```
attributeUpdates = {
    position : {
        "Action" : "PUT",
        "Value" : { "S" : representation }
    }
}
```

- Le condizioni che devono essere soddisfatte affinché venga eseguito l'aggiornamento:
  - La partita deve essere in corso. Ciò significa che il valore dell'attributo `StatusDate` deve iniziare con `IN_PROGRESS`.
  - Il turno corrente deve essere un turno utente valido come specificato dall'attributo `Turn`.
  - Il quadrato della griglia scelto dall'utente deve essere disponibile. Ciò significa che l'attributo corrispondente al quadrato non deve esistere.

```
expectations = {"StatusDate" : {"AttributeValueList": [{"S" : "IN_PROGRESS_"}],
    "ComparisonOperator": "BEGINS_WITH"},
    "Turn" : {"Value" : {"S" : current_player}},
```

```
    position : {"Exists" : False}}
```

La funzione chiama ora `update_item` per aggiornare l'item.

```
self.cm.db.update_item("Games", key=key,
    attribute_updates=attributeUpdates,
    expected=expectations)
```

Dopo che la funzione restituisce il risultato, la funzione `selectSquare` chiama il metodo redirect, come illustrato nell'esempio seguente.

```
redirect("/game="+gameId)
```

Questa chiamata causa l'aggiornamento del browser. Nel corso di questo aggiornamento, l'applicazione verifica se la partita si è conclusa con una vittoria o un pareggio. In caso affermativo, l'applicazione aggiorna l'item di gioco di conseguenza.

Fase 3: distribuzione in produzione tramite il servizio DynamoDB

Argomenti

- 3.1: Creare un ruolo IAM per Amazon EC2
- 3.2: Creazione della tabella Giochi in Amazon DynamoDB
- 3.3: Raggruppa e distribuisci il codice dell' tic-tac-toeapplicazione
- 3.4: Configurazione dell'ambiente AWS Elastic Beanstalk

Nelle sezioni precedenti, hai distribuito e testato l' Tic-Tac-Toeapplicazione localmente sul tuo computer utilizzando DynamoDB local. L'applicazione viene ora distribuita nell'ambiente di produzione come illustrato di seguito:

- Implementate l'applicazione utilizzando AWS Elastic Beanstalk, un easy-to-use servizio per la distribuzione e la scalabilità di applicazioni e servizi Web. Per ulteriori informazioni, vedere Distribuzione di un'applicazione flask su. AWS Elastic Beanstalk

  Elastic Beanstalk avvia una o più istanze Amazon Elastic Compute Cloud (EC2Amazon), che configuri tramite Elastic Beanstalk, su cui verrà eseguita l'applicazione. Tic-Tac-Toe

- Usando il servizio Amazon DynamoDB, creare una tabella `Games` presente in AWS e non in locale nel computer.

Devi inoltre configurare le autorizzazioni. Tutte AWS le risorse create, ad esempio la `Games` tabella in DynamoDB, sono private per impostazione predefinita. Solo il proprietario della risorsa, ovvero l'account AWS che ha creato la tabella `Games`, può accedere alla tabella. Pertanto, per impostazione predefinita, l' Tic-Tac-Toeapplicazione non può aggiornare la `Games` tabella.

Per concedere le autorizzazioni necessarie, crei un ruolo AWS Identity and Access Management (IAM) e concedi a questo ruolo le autorizzazioni per accedere alla `Games` tabella. La tua EC2 istanza Amazon assume innanzitutto questo ruolo. In risposta, AWS restituisce credenziali di sicurezza temporanee che l' EC2 istanza Amazon può utilizzare per aggiornare la `Games` tabella per conto dell' Tic-Tac-Toeapplicazione. Quando configuri la tua applicazione Elastic Beanstalk, specifichi il ruolo IAM che l'istanza o le istanze EC2 Amazon possono assumere. Per ulteriori informazioni sui ruoli IAM, consulta IAM roles for amazon EC2 nella Amazon EC2 User Guide.

> **Note**
>
> Prima di creare EC2 istanze Amazon per l' Tic-Tac-Toeapplicazione, devi prima decidere la AWS regione in cui desideri che Elastic Beanstalk crei le istanze. Dopo aver creato l'applicazione Elastic Beanstalk, specificare il nome della stessa regione e l'endpoint in un file di configurazione. L' Tic-Tac-Toeapplicazione utilizza le informazioni contenute in questo file per creare la `Games` tabella e inviare le richieste successive in una regione specifica. AWS Sia la tabella `Games` DynamoDB che le istanze EC2 Amazon avviate da Elastic Beanstalk devono trovarsi nella stessa regione. Per un elenco delle regioni disponibili, vedi Amazon DynamoDB in Riferimenti generali di Amazon Web Services.

In sintesi, per distribuire l'applicazione in produzione, procedi come segue: Tic-Tac-Toe

1. Crea un ruolo o un utente IAM mediante il servizio IAM. Collegare una policy a questo ruolo per concedere le autorizzazioni per le operazioni di DynamoDB per l'accesso alla tabella `Games`.
2. Raggruppa il codice Tic-Tac-Toe dell'applicazione e un file di configurazione e crea un `.zip` file. Questo `.zip` file viene utilizzato per fornire il codice dell' Tic-Tac-Toeapplicazione a Elastic Beanstalk da inserire nei server. Per ulteriori informazioni sulla creazione di un bundle, consulta Creazione di un bundle di origine dell'applicazione nella Guida per gli sviluppatori di AWS Elastic Beanstalk .

Nel file di configurazione (`beanstalk.config`) è necessario specificare le informazioni relative a regione AWS ed endpoint. L' Tic-Tac-Toeapplicazione utilizza queste informazioni per determinare con quale regione DynamoDB parlare.

3. Configurare l'ambiente Elastic Beanstalk. Elastic Beanstalk avvia EC2 una o più istanze Tic-Tac-Toe Amazon e distribuisce su di esse il tuo pacchetto di applicazioni. Quando l'ambiente Elastic Beanstalk è pronto, sarà necessario specificare il nome del file di configurazione aggiungendo la variabile di ambiente `CONFIG_FILE`.
4. Crea la tabella DynamoDB. Utilizzando il servizio Amazon DynamoDB, crei `Games` la tabella AWS sul tuo computer anziché localmente. Tieni presente che questa tabella ha una chiave primaria semplice costituita dalla chiave di partizione `GameId` di tipo stringa.
5. Test del gioco in produzione.

3.1: Creare un ruolo IAM per Amazon EC2

La creazione di un ruolo IAM di EC2 tipo Amazon consente all' EC2istanza Amazon che esegue l' Tic-Tac-Toeapplicazione di assumere il ruolo corretto ed effettuare richieste applicative per accedere alla `Games` tabella. Quando crei il ruolo, scegli l'opzione Custom Policy (Policy personalizzata) e copia e incolla la policy seguente.

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/*"
         ]
      }
   ]
}
```

Per istruzioni dettagliate, consulta Creazione di un ruolo per un servizio AWS (Console di gestione AWS) nella Guida per l'utente di IAM.

3.2: Creazione della tabella Giochi in Amazon DynamoDB

La tabella `Games` memorizza i dati di gioco in DynamoDB. Se la tabella non esiste, viene creata dall'applicazione. In questo caso, lascia che l'applicazione crei la tabella `Games`.

3.3: Raggruppa e distribuisci il codice dell' tic-tac-toeapplicazione

Se hai seguito i passaggi di questo esempio, hai già scaricato l' Tic-Tac-Toeapplicazione. In caso contrario, scarica l'applicazione ed estrai tutti i file in una cartella nel computer locale. Per istruzioni, consulta Fase 1: distribuzione e test in locale.

Dopo avere estratto tutti i file, sarà presente una cartella `code`. Per distribuire questa cartella su Elastic Beanstalk, puoi raggruppare il contenuto di questa cartella come un file `.zip`. Prima di tutto, devi aggiungere un file di configurazione nella cartella. L'applicazione userà le informazioni relative a regione ed endpoint per creare una tabella DynamoDB nella regione specificata ed eseguirà le richieste successive di operazioni sulla tabella usando l'endpoint specificato.

1. Passa alla cartella in cui hai scaricato l' Tic-Tac-Toeapplicazione.
2. Nella cartella root dell'applicazione crea un file di testo denominato `beanstalk.config` con il contenuto seguente.

   ```
   [dynamodb]
   region=<AWS region>
   endpoint=<DynamoDB endpoint>
   ```

   Puoi ad esempio usare questo contenuto.

   ```
   [dynamodb]
   region=us-west-2
   endpoint=dynamodb.us-west-2.amazonaws.com
   ```

Per un elenco delle regioni disponibili, consulta Amazon DynamoDB in Riferimenti generali di Amazon Web Services.

> **Important**
>
> La regione specificata nel file di configurazione è la posizione in cui l' Tic-Tac-Toeapplicazione crea la `Games` tabella in DynamoDB. L'applicazione Elastic Beanstalk illustrata nella sezione successiva deve essere creata nella stessa regione.

> **Note**
>
> Quando viene creata l'applicazione Elastic Beanstalk, viene richiesto l'avvio di un ambiente in cui è possibile scegliere il tipo. Per testare l'applicazione di Tic-Tac-Toe esempio, puoi scegliere il tipo di ambiente Single Instance, saltare quanto segue e andare al passaggio successivo.
> Tuttavia, il tipo di ambiente Load balancing, autoscaling (Bilanciamento del carico, dimensionamento automatico) offre livelli elevati di disponibilità e scalabilità e deve essere preso in considerazione quando crei e distribuisci altre applicazioni. Se scegli questo tipo di ambiente, dovrai generare anche un UUID e aggiungerlo al file di configurazione come illustrato di seguito.
>
> ```
> [dynamodb]
> region=us-west-2
> endpoint=dynamodb.us-west-2.amazonaws.com
> [flask]
> secret_key= 284e784d-1a25-4a19-92bf-8eeb7a9example
> ```
>
> Nella comunicazione client-server, quando il server invia la risposta, per garantire la sicurezza invia un cookie firmato che il client rimanda al server nella richiesta successiva. Quando è presente un solo server, il server può generare in locale una chiave di crittografia all'avvio. Quando ci sono più server, devono conoscere tutti la stessa chiave di crittografia, altrimenti non potranno leggere i cookie impostati dai server peer. Aggiungendo `secret_key` al file di configurazione, indichi a tutti i server di usare questa chiave di crittografia.

3. Comprimere il contenuto della cartella principale dell'applicazione (che include il file `beanstalk.config`), ad esempio `TicTacToe.zip`.
4. Caricare il file `.zip` in un bucket Amazon Simple Storage Service (Amazon S3). Nella sezione successiva devi fornire questo file `.zip` a Elastic Beanstalk per il caricamento nei server.

   Per istruzioni su come caricare un file in un bucket Amazon S3, consulta Creazione di un bucket e Aggiunta di un oggetto a un bucket nella Guida per l'utente di Amazon Simple Storage Service.

3.4: Configurazione dell'ambiente AWS Elastic Beanstalk

In questa fase viene creata un'applicazione Elastic Beanstalk, che è una raccolta di componenti che includono gli ambienti. Per questo esempio, lanci un' EC2 istanza Amazon per distribuire ed eseguire la tua Tic-Tac-Toe applicazione.

1. Immetti il seguente URL personalizzato per impostare una console Elastic Beanstalk per configurare l'ambiente.

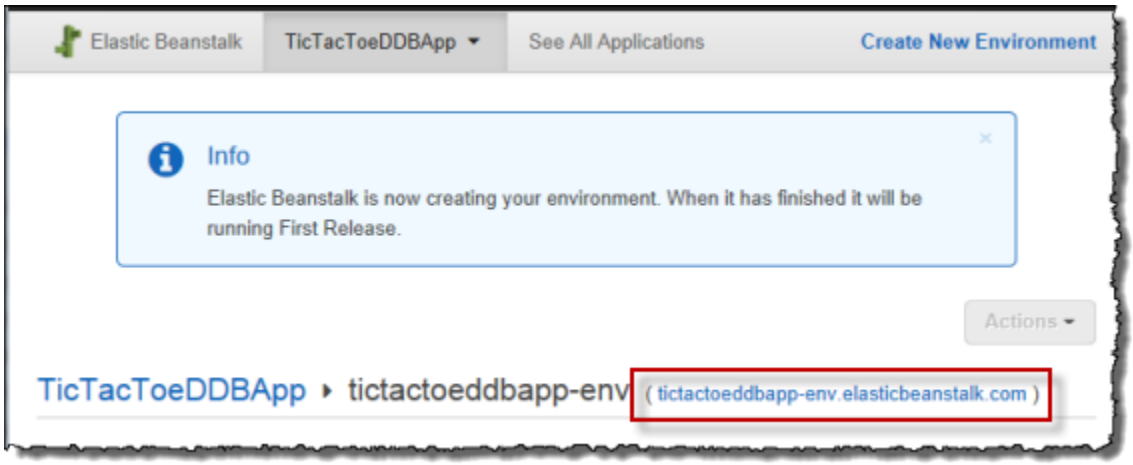
   ```
   https://console.aws.amazon.com/elasticbeanstalk/?region=<AWS-Region>#/
   newApplication
   ?applicationName=TicTacToeyour-name
   &solutionStackName=Python
   &sourceBundleUrl=https://s3.amazonaws.com/<bucket-name>/TicTacToe.zip
   &environmentType=SingleInstance
   &instanceType=t1.micro
   ```

   Per ulteriori informazioni sulla personalizzazione URLs, consulta Constructing a Launch Now URL nella AWS Elastic Beanstalk Developer Guide. Per quanto riguarda l'URL, tieni presente quanto segue:

   - Devi fornire un nome di AWS regione (lo stesso che hai fornito nel file di configurazione), un nome di bucket Amazon S3 e il nome dell'oggetto.
   - Per il test, l'URL richiede il tipo di SingleInstanceambiente e il tipo `t1.micro` di istanza.
   - Il nome dell'applicazione deve essere univoco. Nell'URL precedente ti consigliamo quindi di aggiungere il tuo nome come prefisso di `applicationName`.

   Viene visualizzata la console Elastic Beanstalk. In alcuni casi, potrebbe essere necessario eseguire l'accesso.

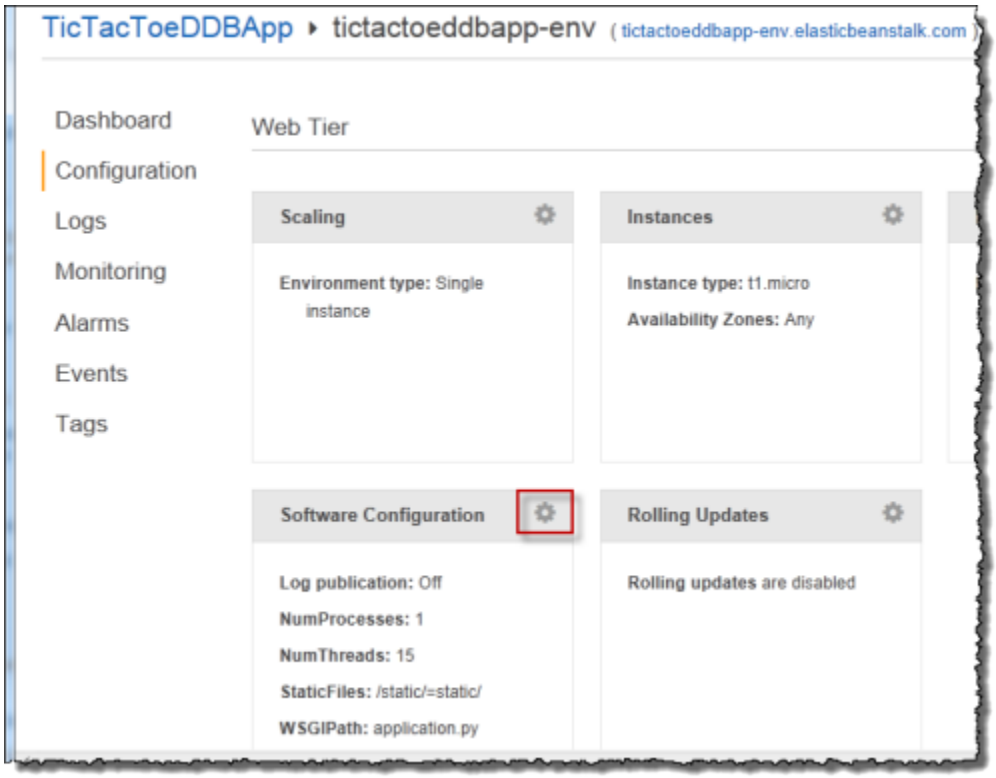
2. Nella console Elastic Beanstalk, scegli Verifica e avvia, quindi seleziona Avvia.
3. Prendi nota dell'URL per riferimento futuro. Questo URL apre la home page Tic-Tac-Toe dell'applicazione.



4. Configura l' Tic-Tac-Toeapplicazione in modo che conosca la posizione del file di configurazione.

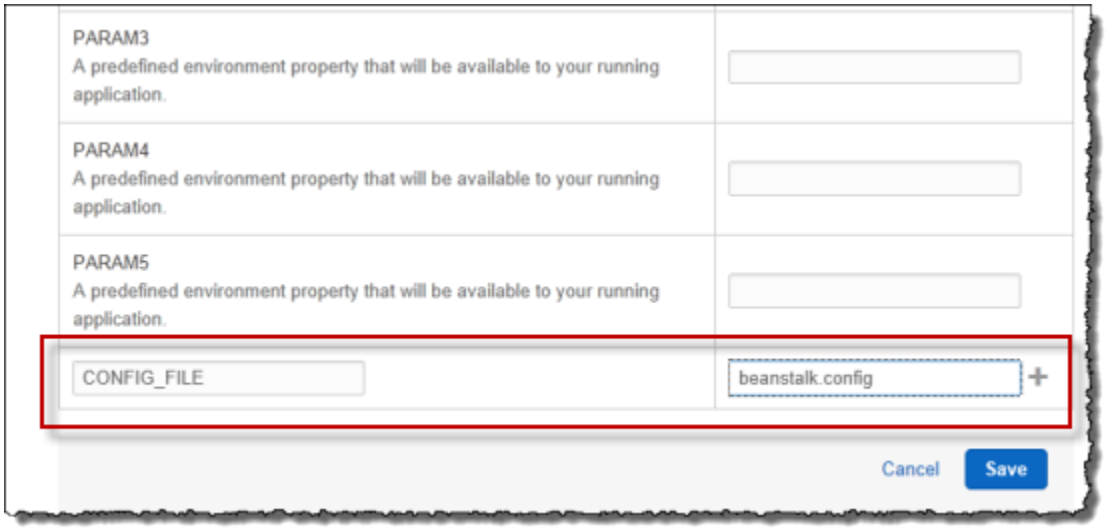
   Dopo che Elastic Beanstalk ha creato l'applicazione, scegliere Configurazione.

   a. Scegli l'icona di ingranaggio accanto a Software Configuration (Configurazione software), come illustrato nello screenshot seguente.



b. Alla fine della sezione Environment Properties (Proprietà ambiente) inserisci **`CONFIG_FILE`** e il relativo valore **`beanstalk.config`** e quindi scegli Save (Salva).

   Per il completamento dell'aggiornamento dell'ambiente potrebbero essere necessari alcuni minuti.

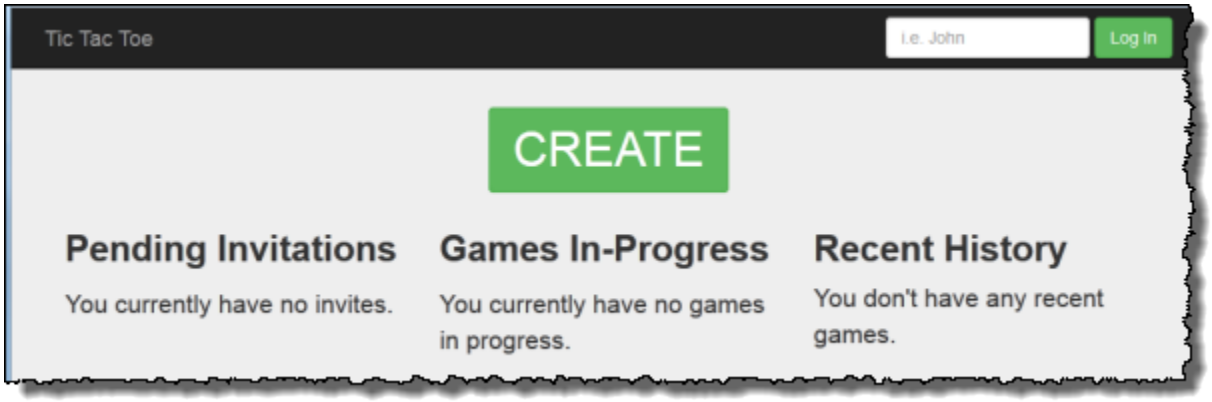


Una volta completato l'aggiornamento, puoi giocare.

5. Nel browser inserisci l'URL copiato nella fase precedente, come illustrato nell'esempio seguente.

```
http://<pen-name>.elasticbeanstalk.com
```

   In questo modo si apre la home page dell'applicazione.



6. Accedi come testuser1 e scegli CREATE per iniziare una nuova partita. tic-tac-toe
7. Inserisci **`testuser2`** nella casella Choose an Opponent (Scegli un avversario).

8. Apri un'altra finestra del browser.

   Assicurati di cancellare tutti i cookie nella finestra del browser, per non eseguire l'accesso con lo stesso utente.

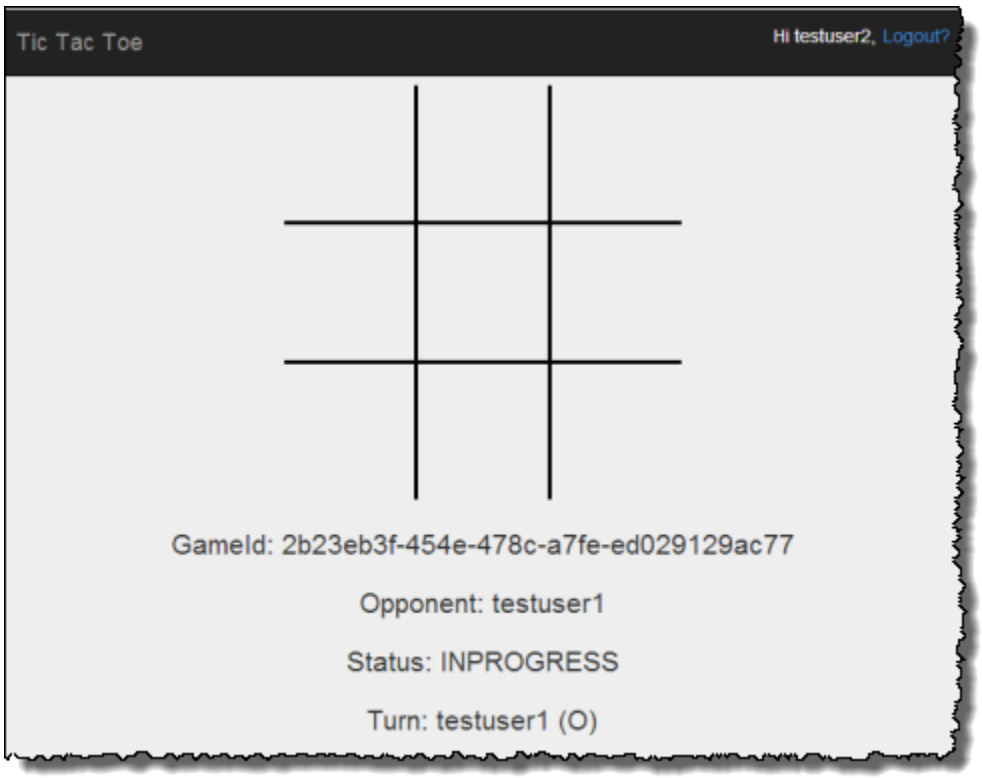
9. Inserisci lo stesso URL per aprire la home page dell'applicazione, come illustrato nell'esempio seguente.

   ```
   http://<env-name>.elasticbeanstalk.com
   ```

10. Accedi come testuser2.

11. Scegli accept (accetta) per l'invito di testuser1 nell'elenco di inviti in attesa.

12. Verrà visualizzata la pagina del gioco.



Sia testuser1 che testuser2 possono giocare. Per ogni mossa, l'applicazione salva lo spostamento nell'elemento corrispondente nella tabella `Games`.

Fase 4: Eliminazione delle risorse

Ora hai completato la distribuzione e il test Tic-Tac-Toe dell'applicazione. L'applicazione copre lo sviluppo di applicazioni end-to-end Web su Amazon DynamoDB, ad eccezione dell'autenticazione degli utenti. L'applicazione usa le informazioni di accesso nella home page solo per aggiungere il nome di un giocatore quando viene creata una partita. In un'applicazione di produzione, dovresti aggiungere il codice necessario per l'accesso e l'autenticazione degli utenti.

Una volta terminato il test, puoi rimuovere le risorse che hai creato per testare l' Tic-Tac-Toeapplicazione per evitare di incorrere in addebiti.

Per rimuovere le risorse create

1. Rimuovere la tabella `Games` creata in DynamoDB.
2. Termina l'ambiente Elastic Beanstalk per liberare le istanze Amazon. EC2
3. Eliminare il ruolo IAM creato.

4. Rimuovere l'oggetto creato in Amazon S3.

Parole riservate in DynamoDB

Le parole chiave seguenti sono riservate per l'uso da parte di DynamoDB. Non utilizzare queste parole come nomi di attributi nelle espressioni. Questo elenco non prevede la distinzione tra lettere maiuscole e minuscole.

Se è necessario scrivere un'espressione contenente un nome di attributo in conflitto con una parola riservata di DynamoDB, è possibile definire un nome di attributo di espressione da utilizzare al posto della parola riservata. Per ulteriori informazioni, consulta Nomi di attributi di espressione (alias) in 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
```

AWS Esempi di SDK for Java 1.x

In questa sezione è contenuto il codice di esempio per le applicazioni DAX che utilizzano SDK per Java 1.x.

Argomenti

- Uso di DAX con AWS SDK per Java 1.x
- Modifica dell'applicazione SDK per Java 1.x esistente per l'utilizzo di DAX
- Esecuzione di query su indici secondari globali con SDK per Java 1.x

Uso di DAX con AWS SDK per Java 1.x

Seguire questa procedura per eseguire l'applicazione di esempio Java per Amazon DynamoDB Accelerator (DAX) sull'istanza Amazon EC2.

> **Note**
>
> Queste istruzioni sono per le applicazioni che utilizzano AWS SDK per Java 1.x. Per le applicazioni che utilizzano AWS SDK for Java 2.x, vedere Java e DAX.

Come eseguire l'applicazione di esempio Java per DAX

1. Installa Java Development Kit (JDK).

```
sudo yum install -y java-devel
```

2. Scarica AWS SDK per Java (file `.zip`) ed estrailo.

```
wget http://sdk-for-java.amazonwebservices.com/latest/aws-java-sdk.zip

unzip aws-java-sdk.zip
```

3. Scaricare la versione più recente del client Java DAX (file `.jar`).

```
wget http://dax-sdk.s3-website-us-west-2.amazonaws.com/java/DaxJavaClient-
latest.jar
```

> **Note**
>
> Il client per l'SDK DAX per Java è disponibile su Apache Maven. Per ulteriori informazioni, consulta Utilizzo del client come una dipendenza Apache Maven.

4. Imposta la variabile `CLASSPATH`. In questo esempio, sostituire *`sdkVersion`* con il numero effettivo della versione di AWS SDK per Java (ad esempio, `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. Scarica il codice sorgente del programma di esempio (file `.zip`).

```
wget http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/samples/
TryDax.zip
```

Al termine del download, estrai i file di origine.

```
unzip TryDax.zip
```

6. Passare alla directory del codice Java e compilare il codice come riportato di seguito.

```
cd TryDax/java/
javac TryDax*.java
```

7. Esegui il programma.

```
java TryDax
```

Verrà visualizzato un output simile al seguente.

```
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.
```

Prendi nota delle informazioni sui tempi: il numero di millisecondi richiesto per i test `GetItem`, `Query` e `Scan`.

8. Nella fase precedente, il programma è stato eseguito sull'endpoint DynamoDB. Ora eseguire nuovamente il programma, ma questa volta le operazioni `GetItem`, `Query` e `Scan` vengono elaborate dal cluster DAX.

   Per determinare l'endpoint per il cluster DAX, scegli una delle seguenti opzioni:

   - Utilizzo della console DynamoDB: scegli il cluster DAX. L'endpoint del cluster viene visualizzato nella console, come nell'esempio seguente.

     ```
     dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
     ```

   - Utilizzo della AWS CLI: immettere il comando riportato di seguito.

     ```
     aws dax describe-clusters --query "Clusters[*].ClusterDiscoveryEndpoint"
     ```

     L'endpoint del cluster viene visualizzato nell'output, come nell'esempio seguente.

     ```
     {
         "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"
     }
     ```

   Ora esegui di nuovo il programma, ma questa volta specifica l'endpoint del cluster come parametro della riga di comando.

```
java TryDax dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
```

Guarda il resto dell'output e prendi nota delle informazioni sui tempi. I tempi trascorsi per `GetItem`, `Query` e `Scan` devono essere significativamente più bassi con DAX che con DynamoDB.

Per ulteriori informazioni sul programma, consulta le seguenti sezioni:

- TryDax.java
- TryDaxHelper.java
- TryDaxTests.java

Utilizzo del client come una dipendenza Apache Maven

Segui queste fasi per utilizzare il client per l'SDK DAX per Java nella tua applicazione come una dipendenza:

Per utilizzare il client come dipendenza Maven

1. Scarica e installa Apache Maven. Per ulteriori informazioni, consulta le pagine relative al download di Apache Maven e all'installazione di Apache Maven.
2. Aggiungi la dipendenza Maven client al file POM (Project Object Model) dell'applicazione. In questo esempio, sostituisci `x.x.x.x` con il numero effettivo della versione del client (ad esempio, `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

Il file `TryDax.java` contiene il metodo `main`. Se si esegue il programma senza parametri della riga di comando, viene creato un client Amazon DynamoDB che viene utilizzato per tutte le operazioni API. Se si specifica un endpoint del cluster DynamoDB Accelerator (DAX) sulla riga di comando, il programma crea anche un client DAX e lo usa per le operazioni `GetItem`, `Query` e `Scan`.

Puoi modificare il programma in diversi modi:

- Utilizza il client DAX anziché il client DynamoDB. Per ulteriori informazioni, consulta Java e DAX.
- Scegli un nome diverso per la tabella di test.
- Modifica il numero di item scritti cambiando i parametri `helper.writeData`. Il secondo parametro è il numero di chiavi di partizione e il terzo parametro è il numero di chiavi di ordinamento. Per impostazione predefinita, il programma utilizza 1-10 per i valori delle chiavi di partizione e 1-10 per i valori delle chiavi di ordinamento, per un totale di 100 elementi scritti nella tabella. Per ulteriori informazioni, consulta TryDaxHelper.java.
- Modifica il numero dei test `GetItem`, `Query` e `Scan` e modifica i relativi parametri.
- Imposta come commento le righe contenenti `helper.createTable` e `helper.deleteTable` (se non vuoi creare ed eliminare la tabella ogni volta che esegui il programma).

> **Note**
>
> Per eseguire questo programma, è possibile configurare Maven in modo che usi il client per l'SDK per Java DAX e AWS SDK per Java come dipendenze. Per ulteriori informazioni, consulta Utilizzo del client come una dipendenza Apache Maven.
> In alternativa, è possibile scaricare e includere sia il client Java DAX che AWS SDK per Java nella classpath. Consulta Java e DAX per un esempio dell'impostazione della variabile `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

Il file `TryDaxHelper.java` contiene i metodi di utilità.

I metodi `getDynamoDBClient` e `getDaxClient` forniscono i client Amazon DynamoDB e DynamoDB Accelerator (DAX). Per le operazioni del piano di controllo (`CreateTable`, `DeleteTable`) e le operazioni di scrittura, il programma usa il client DynamoDB. Se si specifica un endpoint del cluster DAX, il programma principale crea un client DAX per l'esecuzione delle operazioni di lettura (`GetItem`, `Query`, `Scan`).

Gli altri metodi `TryDaxHelper` (`createTable`, `writeData`, `deleteTable`) sono per la configurazione e la distruzione della tabella DynamoDB e dei relativi dati.

Puoi modificare il programma in diversi modi:

- Utilizza per la tabella diverse impostazioni del throughput assegnato.
- Modifica la dimensione di ogni item scritto (vedi la variabile `stringSize` nel metodo `writeData`).
- Modifica il numero dei test `GetItem`, `Query` e `Scan` e i relativi parametri.
- Imposta come commento le righe contenenti `helper.CreateTable` e `helper.DeleteTable` (se non vuoi creare ed eliminare la tabella ogni volta che esegui il programma).

> **Note**
>
> Per eseguire questo programma, è possibile configurare Maven in modo che usi il client per l'SDK per Java DAX e AWS SDK per Java come dipendenze. Per ulteriori informazioni, consulta Utilizzo del client come una dipendenza Apache Maven.
> Oppure, è possibile scaricare e includere sia il client Java DAX che AWS SDK per Java nella classpath. Consulta Java e DAX per un esempio dell'impostazione della variabile `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

Il file `TryDaxTests.java` contiene i metodi che eseguono operazioni di lettura su una tabella di test in Amazon DynamoDB. Questi metodi non riguardano il modo in cui accedono ai dati (utilizzando il client DynamoDB o il client DAX), quindi non è necessario modificare la logica dell'applicazione.

Puoi modificare il programma in diversi modi:

- Modifica il metodo `queryTest` in modo che utilizzi una differente `KeyConditionExpression`.
- Aggiungi un `ScanFilter` al metodo `scanTest` in modo che vengano restituite solo alcuni item.

> **Note**
>
> Per eseguire questo programma, è possibile configurare Maven in modo che usi il client per l'SDK per Java DAX e AWS SDK per Java come dipendenze. Per ulteriori informazioni, consulta Utilizzo del client come una dipendenza Apache Maven.
> Oppure, è possibile scaricare e includere sia il client Java DAX che AWS SDK per Java nella classpath. Consulta Java e DAX per un esempio dell'impostazione della variabile `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));
    }
}
```

Modifica dell'applicazione SDK per Java 1.x esistente per l'utilizzo di DAX

Se si dispone già di un'applicazione Java che utilizza Amazon DynamoDB, è necessario modificarla in modo che possa accedere al cluster DynamoDB Accelerator (DAX). Non è necessario riscrivere l'intera applicazione perché il client Java DAX è molto simile al client di basso livello DynamoDB incluso in AWS SDK per Java.

> **Note**
>
> Queste istruzioni sono per le applicazioni che utilizzano AWS SDK per Java 1.x. Per le applicazioni che utilizzano AWS SDK for Java 2.x, vedere Modifica di un'applicazione esistente per l'utilizzo con DAX.

Si supponga di avere una tabella DynamoDB denominata `Music`. La chiave di partizione della tabella è `Artist` e la chiave di ordinamento è `SongTitle`. Il programma seguente legge un item direttamente dalla tabella `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());
        }
    }
}
```

Per modificare il programma, sostituire il client DynamoDB con un client 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)
       ** ...
       */

    }
}
```

Utilizzo dell'API del documento DynamoDB

AWS SDK per Java fornisce un'interfaccia di documento per DynamoDB. L'API documento funge da wrapper per il client DynamoDB di basso livello. Per ulteriori informazioni, consulta l'argomento relativo alle interfacce di documento.

L'interfaccia di documento può essere utilizzata anche con il client DAX di basso livello, come mostrato nell'esempio seguente.

```
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());
        }
```

```
    }
}
```

Client asincrono DAX

`AmazonDaxClient` è sincrono. Per un'operazione API DAX a esecuzione prolungata, ad esempio un'operazione `Scan` su una tabella di grandi dimensioni, ciò potrebbe determinare il blocco dell'esecuzione del programma finché tale operazione non viene completata. Se il programma deve svolgere altre attività mentre un'operazione API DAX è in corso, è possibile utilizzare invece `ClusterDaxAsyncClient`.

Il programma seguente mostra come utilizzare `ClusterDaxAsyncClient`, in combinazione con Java `Future`, per implementare una soluzione senza blocchi.

```
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();

 }
}
```

Esecuzione di query su indici secondari globali con SDK per Java 1.x

È possibile utilizzare Amazon DynamoDB Accelerator (DAX) per eseguire query su indici secondari globali tramite le interfacce programmatiche di DynamoDB.

Nell'esempio seguente viene illustrato come utilizzare DAX per eseguire una query sull'indice secondario globale `CreateDateIndex` creato in Esempio: Indici secondari globali che utilizzano l'API del documento AWS SDK per Java.

La classe `DAXClient` crea l'istanza degli oggetti client necessari per interagire con le interfacce programmatiche di 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);
 }
```

```
}
```

È possibile eseguire query su un indice secondario globale nei seguenti modi:

- Utilizza il metodo `queryIndex` della classe `QueryIndexDax` definita nel seguente esempio. `QueryIndexDax` prende come parametro l'oggetto client restituito dal metodo `getDaxDocClient` sulla classe `DaxClient`.
- Se si sta utilizzando l'interfaccia di persistenza degli oggetti, utilizza il metodo `queryIndexMapper` sulla classe `QueryIndexDax` definito nel seguente esempio. `queryIndexMapper` prende come parametro l'oggetto client restituito dal metodo `getDaxMapperClient` definito sulla classe `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 definizione di classe qui di seguito rappresenta la tabella Issues ed è utilizzata nel metodo `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;
 }
}
```

AWS Esempi di SDK for Go 1.x

In questa sezione è contenuto il codice di esempio per le applicazioni DAX che utilizzano Go 1.x.

Argomenti

- SDK per Go di DAX

SDK per Go di DAX

Segui questa procedura per eseguire l'applicazione di esempio Amazon DynamoDB Accelerator (DAX) SDK for Go sulla tua istanza Amazon. EC2

Come eseguire l'esempio SDK per Go per DAX

1. Configura l'SDK for Go sulla tua istanza EC2 Amazon:

   a. Installa il linguaggio di programmazione Go (`Golang`).

   ```
   sudo yum install -y golang
   ```

   b. Verifica che Golang sia installata e funzioni correttamente.

   ```
   go version
   ```

   Dovrebbe apparire un messaggio come questo.

   ```
   go version go1.15.5 linux/amd64
   ```

   Le istruzioni rimanenti si basano sul supporto del modulo, che è diventato l'impostazione predefinita con Go versione 1.13.

2. Installa l'applicazione Golang di esempio.

   ```
   go get github.com/aws-samples/aws-dax-go-sample
   ```

3. Esegui i seguenti programmi Golang. Il primo programma crea una tabella DynamoDB denominata `TryDaxGoTable`. Il secondo programma scrive i dati nella tabella.

```
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. Esegui i seguenti programmi 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
```

   Prendi nota delle informazioni sui tempi: il numero di millisecondi richiesto per i test `GetItem`, `Query` e `Scan`.

5. Nella fase precedente, i programmi sono stati eseguiti sull'endpoint DynamoDB. A questo punto, eseguire nuovamente i programmi, ma questa volta le operazioni `GetItem`, `Query` e `Scan` saranno elaborate dal cluster DAX.

   Per determinare l'endpoint per il cluster DAX, scegli una delle seguenti opzioni:

   - Utilizzo della console DynamoDB: scegli il cluster DAX. L'endpoint del cluster viene visualizzato nella console, come nell'esempio seguente.

     ```
     dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
     ```

   - Utilizzando il AWS CLI — Inserisci il seguente comando.

     ```
     aws dax describe-clusters --query "Clusters[*].ClusterDiscoveryEndpoint"
     ```

     L'endpoint del cluster viene visualizzato nell'output, come nell'esempio seguente.

     ```
     {
     ```

```
    "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"
}
```

Ora esegui di nuovo i programmi, ma questa volta specifica l'endpoint del cluster come parametro della riga di comando.

```
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
```

Guarda il resto dell'output e prendi nota delle informazioni sui tempi. I tempi trascorsi per `GetItem`, `Query` e `Scan` devono essere significativamente più bassi con DAX che con DynamoDB.

6. Esegui il seguente programma Golang per eliminare `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
```

AWS SDK per esempi di Node.js 2.x

Questa sezione contiene codice di esempio per applicazioni DAX che utilizzano SDK per Node.js 2.x. AWS

Argomenti

- Node.js e DAX

Node.js e DAX

Segui questi passaggi per eseguire l'applicazione di esempio Node.js sulla tua EC2 istanza Amazon.

Come eseguire l'applicazione di esempio Node.js per DAX

1. Configura Node.js sulla tua EC2 istanza Amazon, come segue:

   a. Installa il gestore delle versioni del nodo (`nvm`).

   ```
   curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.35.3/install.sh | bash
   ```

   b. Usa nvm per installare Node.js.

   ```
   nvm install 12.16.3
   ```

   c. Verifica che Node.js sia installato e funzioni correttamente.

   ```
   node -e "console.log('Running Node.js ' + process.version)"
   ```

   Si dovrebbe visualizzare il seguente messaggio.

   ```
   Running Node.js v12.16.3
   ```

2. Installare il client Node.js per DAX usando il gestore di pacchetti del nodo (`npm`).

   ```
   npm install amazon-dax-client
   ```

3. Scarica il codice sorgente del programma di esempio (file `.zip`).

   ```
   wget http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/samples/
   TryDax.zip
   ```

   Al termine del download, estrai i file di origine.

   ```
   unzip TryDax.zip
   ```

4. Esegui i seguenti programmi Node.js. Il primo programma crea una tabella Amazon DynamoDB denominata `TryDaxTable`. Il secondo programma scrive i dati nella tabella.

   ```
   node 01-create-table.js
   ```

```
node 02-write-data.js
```

5. Esegui i seguenti programmi Node.js.

```
node 03-getitem-test.js
node 04-query-test.js
node 05-scan-test.js
```

Prendi nota delle informazioni sui tempi: il numero di millisecondi richiesto per i test `GetItem`, `Query` e `Scan`.

6. Nella fase precedente, i programmi sono stati eseguiti sull'endpoint DynamoDB. Eseguire nuovamente i programmi, ma questa volta le operazioni `GetItem`, `Query` e `Scan` saranno elaborate dal cluster DAX.

Per determinare l'endpoint per il cluster DAX, scegli una delle seguenti opzioni:

- Utilizzo della console DynamoDB: scegli il cluster DAX. L'endpoint del cluster viene visualizzato nella console, come nell'esempio seguente.

```
dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
```

- Utilizzando il comando AWS CLI—Enter il seguente comando.

```
aws dax describe-clusters --query "Clusters[*].ClusterDiscoveryEndpoint"
```

L'endpoint del cluster viene visualizzato nell'output, come nell'esempio seguente.

```
{
    "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"
}
```

Ora esegui di nuovo i programmi, ma questa volta specifica l'endpoint del cluster come parametro della riga di comando.

```
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
```

Guarda il resto dell'output e prendi nota delle informazioni sui tempi. I tempi trascorsi per `GetItem`, `Query` e `Scan` devono essere significativamente più bassi con DAX che con DynamoDB.

7. Esegui il seguente programma Node.js per eliminare `TryDaxTable`.

```
node 06-delete-table
```

Per ulteriori informazioni sui programmi, consulta le seguenti sezioni:

- 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

Il programma `01-create-table.js` crea una tabella (`TryDaxTable`). I restanti programmi Node.js in questa sezione dipendono da questa tabella.

```
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

Il programma `02-write-data.js` scrive i dati di test in `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

Il programma `03-getitem-test.js` esegue le operazioni `GetItem` in `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

Il programma `04-query-test.js` esegue le operazioni `Query` in `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

Il programma `05-scan-test.js` esegue le operazioni `Scan` in `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

Il programma `06-delete-table.js` elimina `TryDaxTable`. Esegui questo programma dopo aver completato i test.

```
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)
    );
  }
});
```

Cronologia dei documenti per DynamoDB

La tabella seguente descrive le importanti modifiche apportate a ogni versione della Guida per sviluppatori di DynamoDB dal 3 luglio 2018 in poi. Per ricevere notifiche sugli aggiornamenti della documentazione, puoi abbonarti al feed RSS (nell'angolo in alto a sinistra di questa pagina).

Modifica	Descrizione	Data
DAX ora supporta AWS PrivateLink nelle regioni commerciali e della Cina	Sono state aggiunte linee guida per l'utilizzo AWS PrivateLink con DynamoDB Accelerator (DAX) per aiutare gli utenti a stabilire una connettività privata e sicura agli endpoint del piano di controllo DAX all'interno dello stesso. Regione AWS Per ulteriori informazioni, consulta Generare codice di infrastruttura per l'utilizzo di Amazon DynamoDB Console-to-Code.	30 ottobre 2025
DynamoDB ora supporta il protocollo Internet versione 6 () IPv6	DynamoDB ora supporta gli indirizzi Internet Protocol versione 6 IPv6 () nel tuo Amazon VPC quando ti connetti a tabelle, stream e DAX di DynamoDB, inclusi gli endpoint Gateway e Interface . AWS PrivateLink È possibile utilizzarli IPv6 per semplificare lo stack di rete e soddisfare i requisiti di conformità con una rete che supporta sia E. IPv4 IPv6	3 ottobre 2025

Aggiunta documentazione per la nuova Console-to-Code funzionalità di DynamoDB	Console-to-Code trasforma le fasi manuali di creazione di tabelle DynamoDB in Console di gestione AWS codice di infrastruttura automatizzato e riproducibile in diversi formati come CDK e. AWS CloudFormation Per ulteriori informazioni, consulta Generare codice di infrastruttura per l'utilizzo di Amazon DynamoDB Console-to-Code.	23 settembre 2025
Lancio della versione 3.1.0 di DynamoDB locale 3.1.0	Questo aggiornamento migliora le prestazioni per le query PartiQL, inclusa la dipendenza Joda-time.	14 settembre 2025

DynamoDB ha migliorato le sue funzionalità di diagnostica della limitazione con una CloudWatch nuova modalità Contributor Insights, l'aggiunta di dettagli sulle eccezioni di limitazione e metriche mirate CloudWatch	DynamoDB offre una diagnostica avanzata del throttling con una `Throttled keys`nuova modalità unica CloudWatch in Contributor Insights che tiene traccia in modo specifico delle chiavi di partizione con più limitazioni, eccezioni di limitazione più dettagliate che includono campi `ThrottlingReason` strutturati, che forniscono codici di motivo e risorse specifici per aiutare a identificare esattamente quale ARNs risorsa viene limitata e perché, e 16 nuove metriche CloudWatch mirate che corrispondono a diversi scenari di throttling, per un monitoraggio preciso di eventi di throttling specifici . Per ulteriori informazioni, consulta Risoluzione dei problemi di limitazione (della larghezza di banda della rete) in DynamoDB.	14 agosto 2025

Aumento del limite per il passaggio da una modalità di capacità all'altra	È stato aumentato il limite per il passaggio di una tabella dalla modalità con capacità allocata a quella con capacità on demand da una volta ogni 24 ore a quattro volte ogni 24 ore. Questa modifica offre maggiore flessibilità per i carichi di lavoro che richiedono il caricamento di grandi volumi di dati più volte al giorno o devono far fronte ad aumenti imprevisti del traffico. Per ulteriori informazioni, consulta Considerazioni sul passaggio tra modalità di capacità in DynamoDB.	12 agosto 2025
Rilasciata la versione 3.0 di DynamoDB Local, che migra da SDK AWS for Java V1 a V2	Questo aggiornamento rimuove la dipendenza V1, migliora le prestazioni e si allinea alla struttura del pacchetto SDK V2. AWS	17 luglio 2025

Aggiunta del supporto per l'individuazione ottimizzata degli shard nei flussi DynamoDB	Il nuovo meccanismo di individuazione degli shard riduce i tempi di individuazione iniziali, migliora le prestazioni di elaborazione allo stato stazionario, offre visibilità sullo stato dell'elaborazione e mantiene una latenza p100 costante nelle operazioni su larga scala. Per ulteriori informazioni, consulta AWS PrivateLink for Amazon DynamoDB Streams.	17 luglio 2025
Aggiunta del supporto per l'elevata consistenza multi-Regione (MRSC) per le tabelle globali DynamoDB	DynamoDB ora supporta l'elevata consistenza multi-Regione (MRSC) per le tabelle globali con Regione testimone opzionale. La MRSC consente operazioni a elevata consistenza di lettura tra le Regioni con un RPO pari a zero durante i guasti a livello Regionale . Le tabelle globali MRSC devono essere implementate esattamente in tre Regioni, con tre repliche o due repliche e un testimone. Per ulteriori informazioni e le Regioni disponibili, consulta Elevata consistenza multi-Regione (MRSC).	30 giugno 2025

Adattatore Kinesis Client Library (KCL) per i flussi Amazon DynamoDB 2.x	DynamoDB ora supporta un adattatore per i flussi DynamoDB aggiornato compatibile con KCL 2.x e SDK AWS v2. Per ulteriori informazioni, consulta Migrazione da KCL1 .x a KCL 3.x.	11 giugno 2025
Aggiunto il supporto per l'esecuzione di DynamoDB in locale in AWS CloudShell	Ora è possibile eseguire DynamoDB locale direttamente nel browser tramite la Console di gestione AWS senza scaricare o installare software. Per ulteriori informazioni, consulta Distribuzione di DynamoDB in locale sul computer.	19 maggio 2025
Aggiunta del supporto per i tipi di istanze r7i in DAX	Sono state ampliate le opzioni di tipi di istanza per DAX per offrire maggiore flessibilità in termini di prestazioni e capacità. Per maggiori dettagli sui nuovi tipi di istanze r7i e sulle relative funzionalità, consulta Crittografia DAX dei dati inattivi.	29 aprile 2025

La nuova policy di accesso completa e ad ambito ridotto `AmazonDynamoDBFullAccess_v2` sostituisce `AmazonDynamoDBFullAccess` .	A partire dal 25 aprile 2025, DynamoDB ha reso obsoleta la policy `AmazonDynamoDBFullAccess` e l'ha sostituita con una policy ad ambito ridotto denominata `AmazonDynamoDBFullAccess_v2` . Per ulteriori informazioni, consulta la policy AWS gestita: AmazonDynamoDBFull access_v2.	28 aprile 2025
DynamoDB AWS PrivateLink introduce il supporto per DynamoDB Streams	AWS ha aggiunto AWS PrivateLink il supporto per DynamoDB Streams, che consente ai clienti di richiamare DynamoDB Streams dall'interno di APIs Amazon Virtual Private Cloud (VPC) utilizzando la connettività di rete privata. Questa funzionalità consente un accesso semplificato alla rete privata senza passare dalla rete Internet pubblica, supportando i requisiti di conformità e sicurezza per i carichi di lavoro DynamoDB. Per ulteriori informazioni, consulta AWS PrivateLink for Amazon DynamoDB Streams.	26 marzo 2025
SDK DAX per JS 3.x e Go 2.x ora disponibile	L'SDK DynamoDB Accelerator (DAX) per JS 3.x è ora disponibile ed è compatibile con l'SDK per Java 3.x. AWS	17 marzo 2025

Modalità di capacità on demand di NoSQL Workbench 3.13.5 per il lancio delle impostazioni di tabella predefinite	Quando si crea una tabella con impostazioni predefinite, DynamoDB crea una tabella che utilizza la modalità con capacità on demand anziché la modalità con capacità allocata.	24 febbraio 2025
Nuova best practice per configurare il client DAX	È stato pubblicato un nuovo argomento sulle best practice del prontuario DAX per la configurazione del client DAX. Per ulteriori informazioni, consulta Configurazione del client DAX.	17 febbraio 2025
Nuove best practice per le operazioni sui dati in blocco	Sono stati pubblicati nuovi argomenti relativi alle best practice per lavorare con modelli di dati complessi in DynamoDB. Per ulteriori informazioni, consulta Best practice per l'utilizzo di operazioni sui dati in blocco in DynamoDB.	9 gennaio 2025
Amazon DynamoDB ora supporta il ripristino point-in-time configurabile (PITR)	DynamoDB ora supporta periodi di recupero configura bili per il PITR. Ora è possibile impostare il periodo del PITR per ogni tabella su un valore compreso tra 1 e 35 giorni. Per ulteriori informazioni, consulta Point-in-time backup per DynamoDB.	7 gennaio 2025

È stato pubblicato un nuovo argomento sull'integrazione di Streaming gestito da Amazon per Apache Kafka con Amazon DynamoDB	Informazioni su come lo Streaming gestito da Amazon per Apache Kafka si integra con Amazon DynamoDB leggendo i dati dagli argomenti di Apache Kafka e archivian doli in DynamoDB. Per ulteriori informazioni, consulta Integrazione di DynamoDB con lo Streaming gestito da Amazon per Apache Kafka.	26 dicembre 2024
DynamoDB introduce il supporto per un'integrazione zero-ETL con AI Lakehouse SageMaker	DynamoDB introduce un'integrazione Zero-ETL che automatizza l'estrazi one e il caricamento dei dati da Amazon DynamoDB nel data lake del cliente. Per ulteriori informazioni, consulta Integrazione Zero-ETL di DynamoDB con AI Lakehouse . SageMaker	3 dicembre 2024
Le tabelle globali DynamoDB ora supportano l'elevata consistenza multi-Regione	Grazie all'elevata consisten za multi-Regione, è possibile creare applicazioni multi-Reg ione ad alta disponibilità con un RPO pari a zero, ottenendo il massimo livello di resilienz a. Per ulteriori informazioni, consulta Tabelle globali con elevata consistenza multi-Reg ione.	3 dicembre 2024

Aggiornamento della policy gestita da DynamoDB	Sono state aggiunte due nuove autorizzazioni alla policy gestita da `AmazonDynamoDBReadOnlyAccess` : `dynamodb:GetAbacStatus` e `dynamodb:UpdateAbacStatus` . Queste autorizzazioni ti consentono di visualizzare lo stato ABAC e abilitare ABAC per la tua regione corrente. Account AWS Per ulteriori informazioni, vedere la politica AWS gestita:. AmazonDynamoDBRead OnlyAccess	18 novembre 2024
DynamoDB introduce il supporto per il controllo degli accessi basato su attributi (ABAC)	ABAC è una strategia di autorizzazione che consente di definire le autorizzazioni di accesso in base ai tag allegati agli utenti, ruoli, e AWS risorse. ABAC utilizza condizioni basate su tag nelle politiche AWS Identity and Access Management (IAM) o in altre politiche per consentire o negare azioni specifiche sulle tabelle o sugli indici quando i tag dei principali IAM corrispondono ai tag delle tabelle. Per ulteriori informazioni, consulta Utilizzo del controllo degli accessi basato su attributi con DynamoDB.	18 novembre 2024

DynamoDB introduce un throughput a caldo per tabelle on demand e con provisioning	DynamoDB ora supporta il throughput a caldo. Il throughput a caldo offre visibilità sul numero di operazioni di lettura e scrittura che la tabella DynamoDB è in grado di supportare in modo istantaneo nonché la capacità di pre-risca ldare le tabelle DynamoDB. Per ulteriori informazioni, consulta Throughput a caldo di DynamoDB.	13 novembre 2024
Possibilità di utilizzare i record dei flussi Amazon DynamoDB con Apache Flink	Ora puoi utilizzare i record di Amazon DynamoDB Streams con Apache Flink e sfruttare Amazon Managed Service per Apache Flink per creare e gestire rapidamente applicazi oni di elaborazione di flussi. end-to-end Per ulteriori informazioni, consulta Flussi DynamoDB e Apache Flink.	12 novembre 2024
Nuovi argomenti di fatturazi one pubblicati per tabelle globali e backup	Sono stati pubblicati due nuovi argomenti sulla fatturazi one per tabelle globali e backup. Per ulteriori informazi oni, consulta Informazioni sulla fatturazione di Amazon DynamoDB per le tabelle globali e Informazioni sulla fatturazione di Amazon DynamoDB per i backup.	16 ottobre 2024

Integrazione Zero-ETL di Amazon DynamoDB con Amazon Redshift	L'integrazione Zero-ETL di Amazon DynamoDB con Amazon Redshift fornisce una pipeline ETL no code e completamente gestita con replica da DynamoDB ad Amazon Redshift. Per ulteriori informazioni, consulta Integrazione Zero-ETL di Amazon DynamoDB con Amazon Redshift.	15 ottobre 2024
Aggiornamento della sola documentazione per aggiungere un argomento sull'utilizzo dell'IA generativa con DynamoDB	È stato pubblicato un nuovo argomento che fornisce informazioni sull'utilizzo dell'IA generativa con DynamoDB, inclusi esempi di casi d'uso dell'IA di generazione per DynamoDB. Per ulteriori informazioni, consulta Utilizzo dell'IA generativa con DynamoDB.	11 ottobre 2024
AWS SDK ora supporta gli endpoint basati su account	È stata aggiunta la documenta zione per gli endpoint basati sugli account e l'impostazione `ACCOUNT_ID_ENDPOIN T_MODE` per i client SDK. Per ulteriori informazioni, consulta Supporto SDK per gli endpoint basati su account. AWS	3 settembre 2024

Riprogettazione delle nozioni di base	L'esperienza delle nozioni di base è stata riprogettata per consolidare le informazioni e velocizzare l'onboarding. Per ulteriori informazioni, consulta la pagina Nozioni di base su DynamoDB.	1° agosto 2024
Espansione di DAX in nuove Regioni per Spagna e Svezia	DAX è ora disponibile nelle Regioni di Spagna e Svezia. Per ulteriori informazioni, consulta Componenti del cluster DAX.	30 luglio 2024
Revisione e consolidamento della documentazione di backup e ripristino di DynamoDB	La Guida per gli sviluppatori di DynamoDB ha una nuova struttura per il backup e il ripristino. Per ulteriori informazioni, consulta Backup e ripristino per DynamoDB.	2 luglio 2024
Riscrittura dell'argomento "Che cos'è Amazon DynamoDB?"	È stata pubblicata una versione rivista e aggiornata dell'argomento "Che cos'è Amazon DynamoDB?". Per ulteriori informazioni, consulta Che cos'è Amazon DynamoDB?.	21 giugno 2024
Integra DynamoDB Streams con EventBridge	Pubblicato un nuovo argomento sull'integrazione di DynamoDB Streams con. EventBridge Per ulteriori informazioni, consulta Integrazione con. EventBridge	21 giugno 2024

Prontuario DAX	È stato pubblicato un nuovo argomento sulle best practice che fornisce informazioni complete su come utilizzar e DynamoDB Accelerator in modo efficace. Questo argomento tratta l'ottimiz zazione delle prestazioni, la gestione dei costi e le best practice operative. Per ulteriori informazioni, consulta il Prontuario DAX.	3 giugno 2024
Migrazione di una tabella DynamoDB da un account a un altro	È stato aggiunto un nuovo argomento sulla migrazion e di tabelle DynamoDB da un account a un altro. Per ulteriori informazioni, consulta Migrazione di una tabella DynamoDB da un account a un altro.	29 maggio 2024
Revisione e consolidamento della documentazione su monitoraggio e registrazione di log di DynamoDB	Una nuova struttura per il monitoraggio e la registrazione di log in DynamoDB include tre capitoli concisi per metriche, operazioni di registrazione di log e Contributor Insights.	3 maggio 2024

Revisione e consolidamento della documentazione sulla modalità di capacità di DynamoDB	La guida di DynamoDB ora include un nuovo capitolo che contiene tutte le informazioni sulle modalità di capacità di DynamoDB: on demand e con provisioning. Con questo aggiornamento, l'argomento Considerazioni sulla modifica read/write della modalità di capacità è stato spostato all'interno del capitolo sulle migliori pratiche. Questo argomento è ora rinominato Considerazioni sul passaggio tra modalità di capacità e include informazioni elaborate sulle best practice per il passaggio da una modalità di capacità all'altra. Inoltre, la guida contiene ora un nuovo capitolo che include tutte le informazioni sulle letture e le scritture in DynamoDB e sui consumi delle unità di capacità per le operazioni di lettura e scrittura. Per ulteriori informazioni, consulta Capacità di throughput di DynamoDB, Considerazioni sulla commutazione delle modalità di capacità e Letture e scritture di DynamoDB.	1º maggio 2024

Numero massimo di richieste on demand	È ora possibile specifica re il numero massimo di richieste on demand che una singola tabella, un indice o entrambi possono eseguire. Specificare il throughpu t massimo on demand consente di mantenere limitati l'utilizzo e i costi a livello di tabella e di proteggere da un aumento involontario delle risorse utilizzate. Per ulteriori informazioni, consulta Throughput massimo per le tabelle on demand.	1º maggio 2024
Miglioramenti a Operation builder di NoSQL Workbench	NoSQL Workbench ora include il supporto nativo per la modalità scura. Sono state migliorate le operazion i su tabelle ed elementi in Operation builder. I risultati degli elementi e le informazi oni sulle richieste di Operation builder sono disponibili in formato JSON. Per ulteriori informazioni, consulta Operation builder di NoSQL Workbench.	24 aprile 2024

Policy basate su risorse per risorse Amazon DynamoDB	DynamoDB ora supporta le policy basate su risorse per tabelle, indici e flussi. Le policy basate su risorse consentono di definire le autorizzazioni di accesso specificando chi ha accesso a ciascuna risorsa e le azioni eseguibili su ciascuna risorsa. Per ulteriori informazioni, consulta Utilizzo di policy basate su risorse per DynamoDB.	20 marzo 2024
Aggiornamento della policy gestita da DynamoDB	È stata aggiunta una nuova autorizzazione `dynamodb:GetResourcePolicy` alla policy gestita da `AmazonDynamoDBReadOnlyAccess`. Questa autorizzazione fornisce l'accesso alle policy in lettura basate su risorse collegate alle risorse DynamoDB. Per ulteriori informazioni, consulta la policy gestita:.AWS AmazonDynamo DBRead OnlyAccess	20 marzo 2024

AWS PrivateLink per Amazon DynamoDB	Amazon DynamoDB ora supporta. AWS PrivateLink Con AWS PrivateLink, puoi semplificare la connettività di rete privata tra cloud privati virtuali (VPCs), DynamoDB e i data center locali utilizzando endpoint VPC di interfaccia e indirizzi IP privati. Per ulteriori informazioni, consulta AWS PrivateLink per DynamoDB.	19 marzo 2024
Programmazione con guida JavaScript	Amazon DynamoDB presenta una guida alla programma zione per. AWS SDK per JavaScript Informazioni sull' AWS SDK per JavaScrip t, i livelli di astrazione, la configurazione delle connessio ni, la gestione degli errori, la definizione delle policy di ripetizione dei tentativi, la gestione del keep-alive e altro ancora. Per ulteriori informazi oni, consulta Programmazione con. JavaScript	6 marzo 2024

Programmazione con AWS SDK for Java 2.x guida	È stata creata una nuova guida alla programmazione che approfondisce le interfacce di alto, basso livello e documentali, i client HTTP e la loro configurazione, la gestione degli errori e affronta le impostazioni di configurazione più comuni da considerare quando si utilizza l'SDK per Java 2.x. Per ulteriori informazioni, consulta Programming Amazon DynamoDB with AWS SDK for Java 2.x.	5 marzo 2024
Clonazione di tabelle con NoSQL Workbench	Opportunità per gli sviluppatori di utilizzare NoSQL Workbench per copiare o clonare tabelle tra ambienti e Regioni di sviluppo (DynamoDB locale e DynamoDB web). Per ulteriori informazioni, consulta Clonazione di tabelle con NoSQL Workbench.	26 febbraio 2024
Guida alla programmazione con Python	È stata creata una nuova guida che approfondisce le librerie di alto e basso livello e affronta le impostazioni di configurazione più comuni da considerare quando si utilizza l'SDK per Python. Per ulteriori informazioni, consulta Programmazione con Python.	5 gennaio 2024

Riscrittura dell'argomento Time to Live (TTL)	Riscrittura completa della sezione TTL della guida. La nuova guida ti aiuta a iniziare a usare TTL fornendo frammenti di ready-to-use codice lungo il percorso. Gli attuali frammenti di codice forniti sono in Python e Javascript. Per ulteriori informazioni, consulta TTL.	20 dicembre 2023
Le migliori pratiche per comprendere i report di fatturazione e AWS utilizzo	È stata aggiunta una nuova sezione che chiarisce i vari tipi di utilizzo e i relativi costi in DynamoDB. Per ulteriori informazioni, consulta Report di fatturazione e utilizzo.	15 dicembre 2023
Integrazione zero-ETL di Amazon DynamoDB con Amazon Service OpenSearch	Amazon DynamoDB ora supporta l'integrazione zero-ETL con OpenSearch Amazon Service, che consente di eseguire una ricerca sui dati DynamoDB replicandoli e trasformandoli automatic amente senza codice o infrastruttura personalizzati. Per ulteriori informazioni, consulta Integrazione Zero-ETL di DynamoDB con Amazon Service. OpenSearch	28 novembre 2023
Migrazione da un database relazionale a DynamoDB	È stata creata una guida alla migrazione per aiutare gli utenti a capire come migrare a DynamoDB da un database relazionale.	27 novembre 2023

Generazione di dati di esempio con NoSQL Workbench	NoSQL Workbench per Amazon DynamoDB ora supporta la creazione di modelli di dati direttamente da modelli di dati di esempio per aiutarti a progettare schemi di dati per i carichi di lavoro. Puoi utilizzare questa funzionalità per acquisire familiarità con le best practice di modellazione dei dati NoSQL durante la creazione di applicazioni su DynamoDB.	28 settembre 2023
Esportazione incrementale su S3	Ora puoi esportare i dati che sono stati inseriti, aggiornati o eliminati, in piccoli incrementi. Con l'esportazione incrementale, puoi esportare dati modificati che vanno da pochi megabyte a terabyte con pochi clic nella Console di AWS gestione, una chiamata API o l'interfaccia a riga di comando. AWS	26 settembre 2023
Modellazione dei dati per tabelle DynamoDB	Ora puoi saperne di più sulla modellazione dei dati con esempi di DynamoDB incentrati su casi d'uso specifici, sui relativi modelli di accesso step-by-step e sulle linee guida per realizzarli.	14 luglio 2023

Sezione Risoluzione dei problemi	Ora sono disponibili i contenuti per la risoluzione dei problemi di latenza e limitazione (della larghezza di banda della rete) che potrebbero verificarsi nelle tabelle DynamoDB.	13 marzo 2023
Protezione da eliminazione per Amazon DynamoDB	La protezione da eliminazione è ora disponibile per le tabelle Amazon DynamoDB in tutte le regioni AWS . DynamoDB ora consente di proteggere le tabelle dall'eliminazione accidentale durante le normali operazioni di gestione delle tabelle.	8 marzo 2023
CloudFormation supporto per KDSD nelle tabelle globali	Amazon Kinesis Data Streams for DynamoDB CloudForm ation ora supporta le tabelle globali DynamoDB, il che significa che puoi abilitare lo streaming su un Amazon Kinesis Data Streams sulle tabelle globali DynamoDB con modelli. CloudFormation	15 febbraio 2023
DynamoDB locale supporta 100 operazioni per transazione	Ora puoi eseguire fino a 100 operazioni in una singola transazione su DynamoDB locale.	9 febbraio 2023

Utilizzo di DynamoDB Well-Architected Lens per ottimizzare il carico di lavoro di DynamoDB	Ora puoi usare DynamoDB Well-Architected Lens, una raccolta di principi di progettazione e linee guida per la progettazione di carichi di lavoro DynamoDB correttamente strutturati.	3 febbraio 2023
Disponibilità PartiQL GovGloud	PARTIQL, un linguaggio di query compatibile con SQL per Amazon DynamoDB, è ora supportato negli Stati Uniti orientali e negli Stati Uniti occidentali. AWS GovCloud AWS GovCloud	21 dicembre 2022
Suite di installazione unica per NoSQL Workbench e DynamoDB locale	NoSQL Workbench per DynamoDB ora include un processo di installazione guidato per DynamoDB locale per semplificare la configurazione dell'ambiente di sviluppo di DynamoDB locale.	6 dicembre 2022
Importazione in blocco da S3	Amazon DynamoDB ora semplifica la migrazione e il caricamento dei dati in nuove tabelle DynamoDB supportando le importazioni in blocco di dati da Amazon S3.	18 agosto 2022

Integrazione avanzata con Service Quotas	Service Quotas ora ti consente di gestire in modo proattivo le quote a livello di account e tabelle. È possibile visualizz are i valori correnti, impostare allarmi in caso di superamen to di una soglia configurabile da parte della quota e altro ancora.	15 giugno 2022
NoSQL Workbench aggiunge il supporto per tabelle e GSI	È ora possibile utilizzar e NoSQL Workbench per operazioni del piano di controllo su tabelle e indici secondari globali (GSI) come, e. CreateTable UpdateTable DeleteTable	2 giugno 2022
Classe di tabella Standard (accesso infrequente) ora disponibile in Cina	La classe di tabella Standard (accesso infrequente) Amazon DynamoDB ora disponibile nelle Regioni della Cina. Riduzione dei costi di DynamoDB fino al 60%, utilizzando questa nuova classe di tabella per le tabelle che memorizzano i dati ad accesso infrequente.	18 aprile 2022
Aumento delle Service Quotas predefinite e le operazioni di gestione delle tabelle	DynamoDB ha aumentato la quota predefinita per il numero di tabelle per ogni account e regione da 256 a 2.500 tabelle e ha aumentato il numero di operazioni di gestione simultanea delle tabelle da 50 a 500.	9 marzo 2022

Limitazione facoltativa di elementi con PartiQL per DynamoDB	DynamoDB può limitare il numero di elementi elaborati in PartiQL per le operazioni DynamoDB come parametro opzionale su ogni richiesta.	8 marzo 2022
AWS Backup integrazione disponibile nelle regioni della Cina (Pechino e Ningxia)	AWS Backup si integra ora con DynamoDB nelle regioni Cina (Pechino e Ningxia). È possibile soddisfare più facilmente i requisiti di conformità e continuità aziendale attraverso funzionalità di backup avanzate in AWS Backup, come i backup tra account e più regioni.	26 gennaio 2022
Informazioni sulla capacità effettiva di trasmissione tramite chiamate API PartiQL	DynamoDB può restituire la capacità di throughput consumata dalle chiamate API PartiQL per aiutarti a ottimizzare le query e i costi di throughput.	18 gennaio 2022
AWS Backup integration	DynamoDB ora consente di soddisfare più facilmente i requisiti di conformità e continuità aziendale grazie a funzionalità di backup avanzate in AWS Backup, ad esempio backup tra account e tra regioni.	24 novembre 2021

Set di dati NoSQL Workbench import/export in formato CSV	NoSQL Workbench per Amazon DynamoDB ora consente di importare e popolare automaticamente i dati di esempio per creare e visualizzare i modelli di dati.	11 ottobre 2021
Filtra e recupera l'attività del piano dati di Amazon DynamoDB Streams con AWS CloudTrail	Amazon DynamoDB ora offre un controllo più granulare della registrazione degli audit consentendoti difiltrare l'attività dell'API del piano dati Streams in AWS CloudTrail.	22 settembre 2021
Console aggiornata	La console DynamoDB ora è la console predefinita per aiutarti a gestire i dati più facilmente, semplificare le attività comuni e offrirti un accesso più rapido alle risorse e alle funzionalità.	25 agosto 2021
DAX SDK per Java 2.x è ora disponibile	L'SDK DynamoDB Accelerator (DAX) per Java 2.x è ora disponibile ed è compatibile con l'SDK per Java 2.x. AWS Puoi beneficiare delle funzioni più recenti, tra cui l'I/O senza blocchi.	29 luglio 2021
Aggiornamenti delle funzionalità NoSQL Workbench, incluse le operazioni del piano di controllo (control plane)	NoSQL Workbench per Amazon DynamoDB ora consente di eseguire più facilmente operazioni frequenti per modificare e accedere ai dati della tabella.	28 luglio 2021

Le tabelle globali DynamoDB sono ora disponibili nella regione Asia Pacifico	Le Tabelle globali DynamoDB sono ora disponibili nella regione Asia Pacifico (Osaka). Replica le tabelle Amazon DynamoDB in modo automatico nelle 22 regioni AWS che hai scelto.	28 luglio 2021
DAX è ora disponibile in Cina	DynamoDB Accelerator (DAX) è ora disponibile nella regione Cina (Pechino), gestita da Sinnet.	28 luglio 2021
Crittografia DAX in transito	DynamoDB Accelerator (DAX) ora supporta la crittografia durante il transito dei dati tra le applicazioni e i cluster DAX e tra i nodi all'interno di un cluster DAX.	24 luglio 2021
CloudFormation e integrazione CloudTrail	Integrazione AWS CloudFormation e miglioramenti della sicurezza con la registrazione del piano CloudFormation dati.	18 giugno 2021
CloudFormation ora supportato per le tabelle globali	Le tabelle globali di Amazon DynamoDB ora AWS CloudFormationsupportano, il che significa che puoi creare tabelle globali e gestirne le impostazioni con modelli. CloudFormation	14 maggio 2021

Supporto locale Amazon DynamoDB per Java 2.x	Puoi ora utilizzare laAWS SDK per Java 2.x con DynamoDB Local, la versione scaricabile di Amazon DynamoDB. Con DynamoDB Local, è possibile sviluppare e testare applicazioni utilizzando una versione di DynamoDB in esecuzione nell'ambiente di sviluppo locale senza sostenere costi aggiuntivi.	3 maggio 2021
NoSQL Workbench ora supporta AWS CloudFormation	NoSQL Workbench for Amazon DynamoDB AWS CloudFormationora supporta, quindi puoi gestire e modificare i modelli di dati DynamoDB con modelli. CloudFormation Inoltre, ora è possibile configurare le impostazioni di capacità della tabella in NoSQL Workbench.	22 aprile 2021
DynamoDB e ora l'integrazione delle funzionalità AWS Amplify	AWS Amplify ora gestisce più aggiornamenti globali dell'indice secondario globale DynamoDB in una singola distribuzione.	20 aprile 2021
AWS CloudTrail per registrare il piano dati di Amazon DynamoDB Streams APIs	Ora puoi usare AWS CloudTrail per registrare l'attività API del piano dati di Amazon DynamoDB Streams e monitorare e analizzare le modifiche a livello di elemento nelle tabelle DynamoDB.	20 aprile 2021

Amazon Kinesis Data Streams per Amazon DynamoDB ora supporta AWS CloudFormation	Amazon Kinesis Data Streams per Amazon DynamoDB AWS CloudFormation ora supporta, il che significa che puoi abilitare lo streaming su un flusso di dati Amazon Kinesis sulle tue tabelle DynamoDB con modelli. CloudFormation Trasmettendo le modifiche ai dati di DynamoDB in un flusso di dati Kinesis, puoi creare applicazioni di streaming avanzate con i servizi Kinesis. AAmazon	12 Aprile 2021
Amazon Keyspaces ora offre endpoint conformi a FIPS 140-2	Amazon Keyspaces (per Apache Cassandra) offre ora gli endpoint conformi alla norma Federal Information Processing Standard (FIPS 140-2) per aiutarti a gestire più facilmente carichi di lavoro altamente regolamentati. Il Federal Information Processing Standard (FIPS) Publication 140-2 è uno standard di sicurezza del governo degli Stati Uniti e del Canada che specifica i requisiti di sicurezza previsti per i moduli crittografici che proteggono le informazioni sensibili.	8 aprile 2021

Istanze Amazon EC2 T3 per DAX	DAX ora supporta i tipi di istanze Amazon EC2 T3, che forniscono un livello base di prestazioni della CPU con la possibilità di superare la linea di base quando necessario.	15 febbraio 2021
Supporto NoSQL Workbench per Amazon DynamoDB per PartiQL	Ora puoi utilizzare NoSQL Workbench per DynamoDB per generare istruzioni PartiQL per DynamoDB.	4 dicembre 2020
PartiQL per DynamoDB	Ora puoi usare PartiQL for DynamoDB, un linguaggio di query compatibile con SQL, per interagire con le tabelle DynamoDB ed eseguire query ad hoc utilizzando, e DynamoDB for PartiQL. Console di gestione AWS AWS Command Line Interface APIs	23 novembre 2020
Amazon Kinesis Data Streams per Amazon DynamoDB	Ora puoi utilizzare Amazon Kinesis Data Streams per Amazon DynamoDB con le tabelle DynamoDB per acquisire modifiche a livello di elemento e replicarle in un Kinesis Data Streams.	23 novembre 2020

Esportazione di tabelle DynamoDB	Ora puoi esportare le tue tabelle DynamoDB in Amazon S3, consentendoti di eseguire analisi e query complesse sui tuoi dati con servizi come Athena e Lake Formation. AWS Glue	9 novembre 2020
Supporto per valori vuoti	DynamoDB ora supporta valori vuoti per gli attributi String e Binary non chiave nelle tabelle DynamoDB. Il supporto dei valori vuoti offre una maggiore flessibilità nell'utilizzo degli attributi per un insieme più ampio di casi d'uso perché evita di dover trasformare tali attributi prima di inviarli a DynamoDB. I tipi di dati List, Map e Set supportano anche valori String e Binary vuoti.	18 maggio 2020
Supporto NoSQL Workbench per Amazon DynamoDB per Linux	NoSQL Workbench for Amazon DynamoDB è ora supportato su Linux- Ubuntu, Fedora e Debian.	4 maggio 2020

CloudWatch Informazioni sui contributori per DynamoDB — GA	CloudWatchContributor Insights for DynamoDB è disponibile a livello generale. CloudWatch Contributor Insights for DynamoDB è uno strumento diagnosti co che fornisce una at-a-glance visualizzazione delle tendenze del traffico della tabella DynamoDB e aiuta a identificare le chiavi a cui si accede più frequentemente nella tabella (note anche come tasti di scelta rapida).	2 aprile 2020
Aggiornamento delle tabelle globali	È ora possibile aggiornar e le tabelle globali dalla versione 2017.11.29 all'ultima versione delle tabelle globali (2019.11.21), con pochi clic nella console DynamoDB. Aggiornando la versione delle tabelle globali, puoi aumentare facilmente la disponibilità delle tabelle DynamoDB estendend o le tabelle esistenti in AWS regioni aggiuntive, senza dover ricostruire le tabelle.	16 marzo 2020
NoSQL Workbench per Amazon DynamoDB - GA	NoSQL Workbench per Amazon DynamoDB è disponibile al pubblico. Utilizza NoSQL Workbench per progettare, creare, interrogare e gestire tabelle DynamoDB.	2 marzo 2020

Parametri del cluster di cache DAX	Supporto DAX per nuove CloudWatch metriche, che consentono di comprende re meglio le prestazioni del cluster DAX.	6 febbraio 2020
CloudWatch Informazioni sui collaboratori per DynamoDB — Anteprima	CloudWatchContributor Insights for DynamoDB è uno strumento diagnosti co che fornisce una at-a-glance visualizzazione delle tendenze del traffico della tabella DynamoDB e aiuta a identificare le chiavi a cui si accede più frequentemente nella tabella (note anche come tasti di scelta rapida).	26 novembre 2019
Supporto della capacità adattiva per carichi di lavoro sbilanciati	La capacità adattiva di Amazon DynamoDB adesso gestisce meglio i carichi di lavoro sbilanciati isolando automaticamente gli elementi a cui si accede di frequente . Se l'applicazione instrada un traffico elevato in modo sproporzionato verso uno o più elementi, DynamoDB eseguirà un bilanciamento delle partizioni, affinché gli elementi ad accesso frequente non siano ubicati all'interno della stessa partizione.	26 novembre 2019

Supporto per le chiavi gestite dal cliente	DynamoDB ora supporta le chiavi gestite dal cliente, il che significa che puoi avere il pieno controllo su come criptare e gestire la sicurezza dei tuoi dati DynamoDB.	25 novembre 2019
Supporto NoSQL Workbench per la versione locale di DynamoDB (versione scaricabile)	NoSQL Workbench ora supporta la connessione alla versione locale di DynamoDB (versione scaricabile) per progettare, creare, eseguire query e gestire tabelle DynamoDB.	8 Novembre 2019
NoSQL Workbench - Anteprima	Questa è la versione iniziale di NoSQL Workbench per DynamoDB. Utilizza NoSQL Workbench per progettare, creare, interrogare e gestire tabelle DynamoDB. Per ulteriori informazioni, consulta NoSQL Workbench per Amazon DynamoDB (Anteprima).	16 settembre 2019
DAX aggiunge il supporto per le operazioni transazionali con Python e .NET	DAX supporta `TransactWriteItems` e `TransactGetItems` APIs per le applicazioni scritte in Go, Java, .NET, Node.js e Python. Per ulteriori informazioni, consulta Accelerazione in memoria con DAX.	14 febbraio 2019

Aggiornamenti della versione locale di Amazon DynamoDB (versione scaricabile)	DynamoDB local (versione scaricabile) ora supporta la capacità APIs transazionale read/write su richiesta, il reporting della capacità per le operazioni di lettura e scrittura e 20 indici secondari globali. Per ulteriori informazioni, consulta Differenze tra DynamoDB scaricabile e il servizio Web DynamoDB.	4 febbraio 2019
Amazon DynamoDB On-Demand	DynamoDB on demand è un'opzione di fatturazione flessibile in grado di gestire migliaia di richieste al secondo senza alcuna pianificazione della capacità. DynamoDB on-demand pay-per-request offre prezzi per le richieste di lettura e scrittura in modo da pagare solo per ciò che si utilizza. Per ulteriori informazioni, consulta Capacità di throughput di DynamoDB.	28 novembre 2018
Amazon DynamoDB Transactions	Le transazioni DynamoDB apportano modifiche coordinate e coordinate all-or-nothing a più elementi sia all'interno che tra tabelle, fornendo atomicità, coerenza, isolamento e durabilità (ACID) in DynamoDB. Per ulteriori informazioni, consulta Amazon DynamoDB Transactions.	27 novembre 2018

Amazon DynamoDB crittografa tutti i dati inattivi dei clienti	La crittografia dei dati inattivi di DynamoDB fornisce un livello aggiuntivo di protezione dei dati nella tabella crittografata, che include chiave primaria, indici secondari locali e globali, flussi, tabelle globali, backup e cluster DAX, ogni volta che i dati vengono archiviati in supporti durevoli. Per ulteriori informazioni, consulta la sezione relativa alla crittografia dei dati inattivi di Amazon DynamoDB.	15 novembre 2018
Utilizza la versione locale di Amazon DynamoDB in modo più semplice con la nuova immagine Docker	Ora è ancora più semplice utilizzare la versione locale di DynamoDB, la versione scaricabile di DynamoDB, per sviluppare e testare le applicazioni DynamoDB grazie alla nuova immagine Docker della versione locale di DynamoDB. Per ulteriori informazioni, consulta DynamoDB (versione scaricabile) e Docker.	22 agosto 2018

DynamoDB Accelerator (DAX) aggiunge il supporto per la crittografia a riposo	DynamoDB Accelerator (DAX) ora supporta la crittografia dei dati a riposo per i nuovi cluster DAX in modo da accelerare le letture dalle tabelle Amazon DynamoDB in applicazioni sicure soggette a rigorosi requisiti di conformità e regolamentazione. Per ulteriori informazioni, consulta l'articolo sulla crittografia DAX dei dati inattivi.	9 agosto 2018
point-in-timeDynamoDB recovery (PITR) aggiunge il supporto per il ripristino delle tabelle eliminate	Se si elimina una tabella con point-in-time il ripristino abilitato, viene creato automaticamente un backup di sistema che viene conservato per 35 giorni (senza costi aggiuntivi). Per ulteriori informazioni, consulta Prima di iniziare a utilizzare il ripristino point-in-time.	7 agosto 2018
Aggiornamenti ora disponibili tramite RSS	È ora possibile abbonarsi al feed RSS (nell'angolo in alto a sinistra di questa pagina) per ricevere notifiche sugli aggiornamenti della Guida per sviluppatori di Amazon DynamoDB.	3 luglio 2018

Aggiornamenti precedenti

Nella tabella seguente sono descritte le modifiche importanti apportate alla Guida per gli sviluppatori di DynamoDB prima del 3 luglio 2018.

Modifica	Descrizione	Data della modifica
Supporto Go per DAX	È ora possibile abilitare le prestazioni di lettura in microsecondi per le tabelle Amazon DynamoDB nelle applicazioni scritte nel linguaggio di programmazione Go usando il nuovo SDK per Go di DynamoDB Accelerator (DAX). Per ulteriori informazioni, consulta SDK per Go di DAX.	26 giugno 2018
DynamoDB annuncia un contratto sul livello di servizio	DynamoDB ha rilasciato un contratto sul livello di servizio di disponibilità pubblica. Per ulteriori informazioni, consulta Contratto sul livello di servizio Amazon DynamoDB.	19 giugno 2018
Backup Point-In-Time e ripristino continui con DynamoDB (PITR)	Point-in-time il ripristino aiuta a proteggere le tabelle Amazon DynamoDB da operazioni di scrittura o eliminazione accidentali. Grazie al ripristino point-in-time, non dovrai preoccuparti di creare, gestire o programmare i backup on-demand. Ad esempio, supponi che uno script di prova scriva accidentalmente su una tabella DynamoDB di produzione. Con point-in-time il ripristino, puoi ripristinare la tabella in qualsiasi momento negli ultimi 35 giorni.	25 aprile 2018

Modifica	Descrizione	Data della modifica
	DynamoDB conserva i backup incrementali della tabella. Per ulteriori informazioni, consulta Backup point-in-time per DynamoDB.	
Crittografia a riposo per DynamoDB	La crittografia dei dati a riposo di DynamoDB, disponibile per le nuove tabelle DynamoDB, aiuta a proteggere i dati delle applicazioni nelle tabelle Amazon DynamoDB usando le chiavi di crittografia gestite da AWS archiviate in AWS Key Management Service. Per ulteriori informazioni, consulta Crittografia a riposo per DynamoDB.	8 febbraio 2018
Backup e ripristino di DynamoDB	On-Demand Backup consente di creare backup completi dei dati delle tabelle DynamoDB per l'archiviazione dei dati, consentendo di soddisfare i requisiti normativi aziendali e governativi. È possibile eseguire il backup di tabelle da pochi megabyte a centinaia di terabyte di dati senza nessun impatto sulle performance e sulla disponibilità delle applicazioni di produzione. Per ulteriori informazioni, consulta Backup e ripristino per DynamoDB.	29 novembre 2017

Modifica	Descrizione	Data della modifica
Tabelle globali DynamoDB	Global Tables sfrutta l'impatto globale di DynamoDB per offrirti un database completamente gestito, multi-regionale e multi-attivo che offre prestazioni di lettura e scrittura rapide e locali per applicazioni globali altamente dimensionate. Global Tables replica automaticamente le tue tabelle Amazon DynamoDB nelle regioni che preferisci. AWS Per ulteriori informazioni, consulta Tabelle globali: replica multi-attiva e multi-Regione.	29 novembre 2017
Supporto Node.js per DAX	Gli sviluppatori di Node.js possono utilizzare DynamoDB Accelerator (DAX) con il client DAX per Node.js. Per ulteriori informazioni, consulta Accelerazione in memoria con DynamoDB Accelerator (DAX).	5 ottobre 2017

Modifica	Descrizione	Data della modifica
Endpoint VPC per DynamoDB	Gli endpoint DynamoDB consentono alle EC2 istanze Amazon nel tuo Amazon VPC di accedere a DynamoDB, senza esposizione alla rete Internet pubblica. Il traffico di rete tra il tuo VPC e DynamoDB non esce dalla rete Amazon. Per ulteriori informazioni, consulta Utilizzo di endpoint Amazon VPC per accedere a DynamoDB.	16 agosto 2017

Modifica	Descrizione	Data della modifica
Auto Scaling per DynamoDB	L'Auto Scaling di DynamoDB elimina la necessità di definire o regolare manualmente le impostazioni del throughput assegnato. DynamoDB Auto Scaling infatti adegua dinamicamente la capacità di throughput in lettura e scrittura in risposta a modelli di traffico effettivi. In tal modo una tabella o un indice secondario globale può aumentare la capacità di lettura e scrittura allocata per gestire improvvisi aumenti di traffico, senza limitazione (della larghezza di banda della rete). Quando il carico di lavoro diminuisce, l'Auto Scaling di DynamoDB riduce la capacità assegnata. Per ulteriori informazioni, consulta Gestione automatica della capacità effettiva di trasmissione con il dimensionamento automatico di DynamoDB.	14 giugno 2017

Modifica	Descrizione	Data della modifica
DynamoDB Accelerator (DAX)	DynamoDB Accelerator (DAX) è un sistema di caching in memoria completamente gestito e ad elevata disponibilità per DynamoDB che migliora fino a 10 volte le prestazioni anche in presenza di milioni di richieste al secondo, consentendo di ottenere risposte in microsecondi invece che in millisecondi. Per ulteriori informazioni, consulta Accelerazione in memoria con DynamoDB Accelerator (DAX).	19 aprile 2017
DynamoDB ora supporta la scadenza automatica degli elementi con Time-To-Live (TTL)	La durata (TTL, Time to Live) di Amazon DynamoDB consente di eliminare automaticamente gli elementi scaduti dalle tabelle senza costi aggiuntivi. Per ulteriori informazioni, consulta Utilizzo del Time to Live in DynamoDB.	27 febbraio 2017
DynamoDB ora supporta i tag dell'allocazione dei costi	Ora puoi aggiungere i tag alle tabelle di Amazon DynamoDB per una migliore categorizzazione dell'utilizzo e un reporting dei costi più dettagliato. Per ulteriori informazioni, consulta Aggiunta di tag ed etichette alle risorse in DynamoDB.	19 gennaio 2017

Modifica	Descrizione	Data della modifica
Nuova API `DescribeLimits` di DynamoDB	L'`DescribeLimits` API restituisce i limiti di capacità attualmente assegnati per il tuo AWS account in una regione, sia per l'intera regione che per ogni tabella DynamoDB che crei lì. Ti consente di determinare quali sono i limiti correnti del tuo account in modo da poterli confrontare con la capacità assegnata che stai utilizzando attualmente e avere tutto il tempo per richiedere un aumento prima di raggiungere un limite. Per ulteriori informazioni, consulta Quote in Amazon DynamoDB e consulta l'DescribeLimitsAmazon DynamoDB API Reference.	1 marzo 2016

Modifica	Descrizione	Data della modifica
Aggiornamento della console DynamoDB e nuova terminolo gia per gli attributi della chiave primaria	La console di gestione DynamoDB è stata riprogett ata per essere più intuitiva e facile da usare. Come parte di questo aggiornamento, viene introdotta una nuova terminolo gia per gli attributi della chiave primaria: • Chiave di partizione: nota anche come attributo hash. • Chiave di ordinamento: nota anche come attributo di intervallo. Solo i nomi sono cambiati, la funzionalità rimane la stessa. Quando crei una tabella o un indice secondario, puoi scegliere una chiave primaria semplice (solo chiave di partizione) o una chiave primaria composita (chiave di partizione e chiave di ordinamento). La documenta zione di DynamoDB è stata aggiornata in modo da includere queste modifiche.	12 novembre 2015

Modifica	Descrizione	Data della modifica
Amazon DynamoDB Storage Back-end per Titan	DynamoDB Storage Backend for Titan è un back-end di archiviazione per il database grafico Titan implement ato sulla base di Amazon DynamoDB. Quando si utilizza DynamoDB Storage Backend for Titan, i dati traggono vantaggio dalla protezione di DynamoDB, che funziona attraverso i data center ad alta disponibilità di Amazon. Il plug-in è disponibile per Titan versione 0.4.4 (fondamen talmente per compatibilità con le applicazioni esistenti) e Titan versione 0.5.4 (consigli ato per le nuove applicazi oni). Come altri back-end di storage per Titan, questo plug-in supporta lo stack Tinkerpop (versioni 2.4 e 2.5), inclusa l'API Blueprints e la shell Gremlin.	20 agosto 2015

Modifica	Descrizione	Data della modifica
DynamoDB Streams, replica tra regioni e scansione con letture fortemente consistenti	DynamoDB Streams acquisisce una sequenza cronologica di modifiche a livello di elemento in una tabella DynamoDB e le memorizza in un log per un massimo di 24 ore. Le applicazioni possono accedere a questo log e visualizzare gli elementi di dati nel rispettivo stato prima e dopo la modifica, praticamente in tempo reale. Per ulteriori informazioni, consulta Acquisizione dei dati di modifica per DynamoDB Streams e la Documentazione di riferimento delle API di DynamoDB Streams. La replica interregionale di DynamoDB è una soluzione lato client per mantenere copie identiche delle tabelle DynamoDB in diverse regioni, quasi in tempo reale. AWS Puoi utilizzare la replica tra regioni per eseguire il backup delle tabelle DynamoDB o per fornire l'accesso a bassa latenza ai dati in cui gli utenti sono distribuiti geografic amente. L'operazione `Scan` di DynamoDB usa le letture a consistenza finale per	16 luglio 2015

Modifica	Descrizione	Data della modifica
	impostazione predefini ta. Puoi utilizzare letture fortemente consistenti invece impostando il parametro `ConsistentRead` su true. Per ulteriori informazi oni consulta Consistenza di lettura per la scansione e Scan nella Documentazione di riferimento alle API di Amazon DynamoDB.	
AWS CloudTrail supporto per Amazon DynamoDB	DynamoDB è ora integrato con. CloudTrail CloudTrai l acquisisce le chiamate API effettuate dalla console DynamoDB o dall'API DynamoDB e le tiene traccia nei file di registro. Per ulteriori informazioni, consulta Registrazione delle operazion i di DynamoDB con AWS CloudTrail e la Guida per l'utente di AWS CloudTrail.	28 maggio 2015

Modifica	Descrizione	Data della modifica
Supporto migliorato per le espressioni di query	In questa versione è stato aggiunto un nuovo parametro `KeyConditionExpression` all'API `Query`. Una `Query` legge gli elementi di una tabella o un indice usando i valori delle chiavi primarie. Il parametro `KeyConditionExpression` è una stringa che identifica i nomi delle chiavi primarie e le condizioni da applicare ai valori delle chiavi. La `Query` recupera solo gli elementi che soddisfano l'espressione. La sintassi di `KeyConditionExpression` è simile a quella di altri parametri di espressione in DynamoDB e consente di definire le variabili di sostituzione per nomi e valori all'interno dell'espressione. Per ulteriori informazioni, consulta Esecuzione di query in DynamoDB.	27 Aprile 2015

Modifica	Descrizione	Data della modifica
Nuove funzioni di confronto per le scritture condizionali	In DynamoDB, il parametro `ConditionExpression` determina se `PutItem`, `UpdateItem` o `DeleteItem` ha esito positivo. L'elemento è scritto solo se la condizione restituisce true. In questa versione sono state aggiunte due nuove funzioni, `attribute_type` e `size`, da usare con `ConditionExpression`. Queste funzioni consentono di eseguire le scritture condizionali in base al tipo di dati o alle dimensioni di un attributo in una tabella. Per ulteriori informazioni, consulta Esempio di CLI di espressione condizionale in DynamoDB.	27 Aprile 2015

Modifica	Descrizione	Data della modifica
API di scansione per indici secondari	In DynamoDB, un'operazione `Scan` legge tutti gli elementi in una tabella, applica i criteri di filtro definiti dall'utente e restituisce gli elementi di dati selezionati all'applicazione. Questa stessa funzionalità è ora disponibile anche per gli indici secondari. Per eseguire la scansione di un indice secondario locale o di un indice secondario globale, specifichi il nome dell'indice e il nome della relativa tabella padre. Per impostazione predefinita, uno `Scan` di indice restituisce tutti i dati nell'indice. Puoi utilizzare un'espressione di filtro per limitare i risultati restituiti all'applicazione. Per ulteriori informazioni, consulta Scansione di tabelle in DynamoDB.	10 febbraio 2015

Modifica	Descrizione	Data della modifica
Operazioni online per indici secondari globali	L'indicizzazione online ti consente di aggiungere o rimuovere indici secondari globali su tabelle esistenti. Con l'indicizzazione online, non è necessario definire tutti gli indici di una tabella quando crei una tabella, puoi invece aggiungere un nuovo indice in qualsiasi momento. Analogamente, se decidi che non hai più bisogno di un indice, puoi rimuoverlo in qualsiasi momento. Le operazioni di indicizzazione online sono non bloccanti, in modo che la tabella rimanga disponibile per l'attività di lettura e scrittura mentre vengono aggiunti o rimossi gli indici. Per ulteriori informazioni, consulta Gestione degli indici secondari globali in DynamoDB.	27 gennaio 2015

Modifica	Descrizione	Data della modifica
Supporto per modelli di documenti con JSON	DynamoDB consente di memorizzare e recuperare i documenti con il supporto completo per i modelli di documenti. I nuovi tipi di dati sono completamente compatibili con lo standard JSON e consentono di nidificare gli elementi del documento l'uno dentro l'altro. Puoi utilizzare gli operatori di deferenziazione del percorso del documento per leggere e scrivere i singoli elementi, senza dover recuperare l'intero documento. In questa versione sono stati introdotti anche nuovi parametri di espressione per specificare proiezioni, condizioni e operazioni di aggiornamento durante la lettura o la scrittura di item di dati. Per ulteriori informazioni sul supporto per modelli di documenti con JSON, consulta Tipi di dati e Utilizzo di espressioni in DynamoDB.	7 Ottobre 2014

Modifica	Descrizione	Data della modifica
Dimensionamento flessibile	Per le tabelle e gli indici secondari globali, puoi aumentare la capacità di throughput assegnata in lettura e scrittura di qualsiasi importo, a condizione che si rimanga entro i limiti per tabella e per account. Per ulteriori informazioni, consulta Quote in Amazon DynamoDB.	7 Ottobre 2014
Dimensioni di item grandi	La dimensione massima degli elementi in DynamoDB è aumentata da 64 KB a 400 KB. Per ulteriori informazioni, consulta Quote in Amazon DynamoDB.	7 Ottobre 2014

Modifica	Descrizione	Data della modifica
Espressioni condizionali migliorate	DynamoDB espande gli operatori disponibili per le espressioni condizionali, offrendo una maggiore flessibilità per operazioni condizionali, aggiornamenti ed eliminazioni. Gli operatori disponibili di recente consentono di verificare se un attributo esiste o meno, se è maggiore o minore di un determinato valore, se è compreso tra due valori, se inizia con determinati caratteri e molto altro. DynamoDB offre anche un operatore OR opzionale per la valutazione di più condizioni. Per impostazione predefinita, più condizioni in un'espressione sono unite insieme da AND, quindi l'espressione è true solo se tutte le sue condizioni sono true. Se invece si specifica OR, l'espressione è true se una o più condizioni sono true. Per ulteriori informazioni, consulta Utilizzo di elementi e attributi in DynamoDB.	24 Aprile 2014

Modifica	Descrizione	Data della modifica
Filtro query	L'API `Query` di DynamoDB supporta una nuova opzione `QueryFilter` . Per impostazione predefinita, una `Query` trova gli elementi che corrispondono a un valore di chiave di partizione specifico e una condizione di chiave di ordinamento opzionale . Un filtro `Query` applica le espressioni condizionali ad altri attributi non di chiave. Se un filtro `Query` è presente, gli elementi che non corrispon dono alle condizioni del filtro vengono scartati prima che i risultati della `Query` vengano restituiti all'applicazione. Per ulteriori informazioni, consulta Esecuzione di query in DynamoDB.	24 Aprile 2014

Modifica	Descrizione	Data della modifica
Esportazione e importazione dei dati utilizzando il Console di gestione AWS	La console DynamoDB è stata migliorata per semplific are le esportazioni e le importazioni di dati nelle tabelle DynamoDB. Con pochi clic, puoi configurare un cluster Amazon Elastic AWS Data Pipeline per orchestrare il flusso di lavoro e un MapReduce cluster Amazon Elastic per copiare i dati dalle tabelle DynamoDB in un bucket Amazon S3 o viceversa. Puoi eseguire un'esportazione o un'importazione solo una volta o impostare un lavoro di esportazione giornaliero. È anche possibile eseguire esportazioni e importazioni tra regioni, copiando i dati DynamoDB da una tabella in una regione a una tabella in un'altra AWS regione. AWS	6 marzo 2014

Modifica	Descrizione	Data della modifica
Documentazione riorganizzata per le API di livello superiore	Le informazioni su quanto segue ora sono più facili da APIs trovare: • Java: Dynamo DBMappper • .NET: modello di documento e modello di persistenza dell'oggetto Questi livelli superiori APIs sono ora documentati qui:. Interfacce di programmazione di livello superiore per DynamoDB	20 gennaio 2014

Modifica	Descrizione	Data della modifica
Indici secondari globali	In DynamoDB è stato aggiunto il supporto per indici secondari globali. Come con un indice secondario locale, puoi definire un indice secondario globale utilizzando una chiave alternativa da una tabella e quindi emettendo le richieste Query sull'indice. A differenza di un indice secondario locale, la chiave di partizione per l'indice secondario globale non deve essere uguale a quella della tabella, ma può essere qualsiasi attributo scalare dalla tabella. La chiave di ordinamento è facoltativa e può anche essere un attributo della tabella scalare. Un indice secondario globale ha anche le proprie impostazioni del throughput assegnato che sono separate da quelle della tabella padre. Per ulteriori informazioni, consultare Miglioramento dell'accesso ai dati con gli indici secondari in DynamoDB e Utilizzo degli indici secondari globali in DynamoDB.	12 dicembre 2013

Modifica	Descrizione	Data della modifica
Controllo granulare degli accessi	In DynamoDB è stato aggiunto il supporto per il controllo granulare degli accessi. Questa caratteristica consente ai clienti di specificare quali principal (utenti, gruppi o ruoli) possono accedere a singoli elementi e attributi in una tabella DynamoDB o in un indice secondario. Le applicazioni possono anche utilizzare la federazione delle identità Web per demandare l'autenticazione dell'utente a un provider di identità di terze parti, come Facebook, Google o Login with Amazon. In questo modo, le applicazioni (comprese le applicazioni per dispositivi mobili) possono gestire un numero molto elevato di utenti, garantendo nel contempo che nessuno acceda agli elementi di dati di DynamoDB a meno che non sia autorizzato a farlo. Per ulteriori informazioni, consulta Utilizzo di condizioni di policy IAM per il controllo granulare degli accessi.	29 Ottobre 2013

Modifica	Descrizione	Data della modifica
Dimensione di 4 KB dell'unità di capacità in lettura	La dimensione dell'unità di capacità in lettura è stata aumentata da 1 KB a 4 KB. Questo miglioramento può ridurre il numero delle unità di capacità in lettura assegnate richieste per molte applicazioni. Ad esempio, prima di questa versione, la lettura di un item da 10 KB consumerebbe 10 unità di capacità in lettura, ora quello stesso item da 10 KB letto consumerebbe solo 3 unità (10 KB / 4 KB, arrotondati al successivo limite di 4 KB). Per ulteriori informazioni, consulta Capacità di throughput di DynamoDB.	14 maggio 2013

Modifica	Descrizione	Data della modifica
Scansioni parallele	In DynamoDB è stato aggiunto il supporto per le operazioni di scansione parallela. Le applicazioni ora possono dividere una tabella in segmenti logici e scansionare tutti i segmenti contemporaneamente. Questa caratteristica consente di ridurre il tempo richiesto per il completamento di una scansione e di utilizzare completamente la capacità di lettura assegnata dalla tabella. Per ulteriori informazioni, consulta Scansione di tabelle in DynamoDB.	14 maggio 2013
Indici secondari locali	DynamoDB aggiunge supporto per indici secondari locali. Puoi definire gli indici delle chiavi di ordinamento negli attributi non di chiave e quindi utilizzare questi indici nelle richieste di query. Con le applicazioni degli indici secondari locali, si possono recuperare gli elementi di dati in modo efficiente in più dimensioni. Per ulteriori informazioni, consulta Indici secondari locali.	18 Aprile 2013

Modifica	Descrizione	Data della modifica
Nuova versione dell'API	Con questa versione in DynamoDB è stata introdotta una nuova versione dell'API (2012-08-10). La versione precedente dell'API (2011-12-05) è comunque supportata per compatibilità con le versioni precedenti delle applicazioni esistenti. Le nuove applicazioni devono utilizzare la nuova versione dell'API 2012-08-10. Consigliamo di eseguire la migrazione delle applicazioni esistenti alla versione dell'API 2012-08-10 poiché le nuove caratteristiche DynamoDB (ad esempio, gli indici secondari locali) non saranno sottoposte al backport per la precedente versione dell'API. Per ulteriori informazioni sulla versione dell'API 2012-08-10, consulta la Documentazione di riferimento delle API di Amazon DynamoDB.	18 Aprile 2013

Modifica	Descrizione	Data della modifica
Supporto delle variabili di policy IAM	La sintassi delle policy di accesso IAM supporta ora le variabili. Quando una policy viene valutata, eventuali variabili di policy vengono sostituite con valori forniti in base a informazioni basate sul contesto relative alla sessione dell'utente autenticato. Si possono utilizzare variabili di policy per definire policy generali senza elencare esplicitamente tutte le componenti della policy. Per ulteriori informazioni sulle variabili di policy, passa a Variabili di policy nella guida AWS Identity and Access Management con IAM. Per esempi di variabili di policy in DynamoDB, consulta Identity and Access Management per Amazon DynamoDB.	4 Aprile 2013

Modifica	Descrizione	Data della modifica
Esempi di codice PHP aggiornati per la versione 2 AWS SDK per PHP	La versione 2 di AWS SDK per PHP è ora disponibile. Gli esempi di codice PHP riportati nella Guida per gli sviluppatori di Amazon DynamoDB sono stati aggiornati in modo da utilizzare questo nuovo SDK. Per maggiori informazioni sulla versione 2 dell'SDK, consulta AWS SDK per PHP.	23 gennaio 2013
Nuovo endpoint	DynamoDB si espande nella regione (Stati Uniti occidentali AWS GovCloud). Per l'elenco corrente dei protocolli e degli endpoint del servizio, consulta Regioni ed endpoint.	3 dicembre 2012
Nuovo endpoint	DynamoDB si espande nella regione Sud America (San Paolo). Per l'elenco corrente degli endpoint supportati, consulta Regioni ed endpoint.	3 dicembre 2012
Nuovo endpoint	DynamoDB si espande nella regione Asia Pacifico (Sydney). Per l'elenco corrente degli endpoint supportati, consulta Regioni ed endpoint.	13 Novembre 2012

Modifica	Descrizione	Data della modifica
DynamoDB implementa il supporto CRC32 per i checksum, supporta risultati in batch estremamente coerenti e rimuove le restrizioni sugli aggiornamenti simultanei delle tabelle.	• DynamoDB calcola CRC32 un checksum del payload HTTP e lo restituisce in una nuova intestazione,. `x-amz-crc32` Per ulteriori informazioni, consulta API DynamoDB di basso livello. • Per impostazione predefinita, le operazioni di lettura eseguite dall'API `BatchGetItem` sono consistenti finali. Un nuovo parametro `ConsistentRead` in `BatchGetItem` ti consente di scegli la lettura consistente per qualsiasi tabella della richiesta. Per ulteriori informazioni, consulta Descrizione. • In questa versione sono state rimosse alcune restrizioni quando si aggiornano più tabelle contemporaneamente. Il numero totale di tabelle che possono essere aggiornate contemporaneamente è ancora 10. Tuttavia, lo stato di queste tabelle può ora essere qualsiasi combinazione di `CREATING`, `UPDATING` or `DELETING`. Inoltre, non esiste più	2 Novembre 2012

Modifica	Descrizione	Data della modifica
	un importo minimo per aumentare o ridurre l'or per una tabella. ReadCapacityUnitsWriteCapacityUnits Per ulteriori informazioni, consulta Quote in Amazon DynamoDB.	
Documentazione delle best practice	Nella Guida per gli sviluppatori di Amazon DynamoDB sono riportate le best practice per l'utilizzo delle tabelle e degli elementi, insieme ai suggerimenti per le operazioni di query e scansione.	28 settembre 2012

Modifica	Descrizione	Data della modifica
Supporto per il tipo di dati binario	Oltre ai tipi numero e stringa, DynamoDB ora supporta il tipo di dati binario. Prima di questa versione, per memorizzare i dati binari i dati binari venivano convertiti in formato stringa e poi venivano archiviati in DynamoDB. Oltre al lavoro di conversio ne richiesto sul lato client, la conversione spesso ha aumentato la dimensione dell'item di dati che richiedeva più storage e potenzialmente una capacità di throughput assegnata aggiuntiva. Con gli attributi di tipo binario puoi ora memorizzare qualsiasi tipo di dati binari, ad esempio dati compressi, dati crittografati e immagini. Per ulteriori informazioni, consulta Tipi di dati. Per esempi pratici di gestione di dati di tipo binario utilizzando il AWS SDKs, consultate le seguenti sezioni: • Esempio: gestione degli attributi di tipo binario utilizzando l'API del documento AWS SDK per Java	21 agosto 2012

Modifica	Descrizione	Data della modifica
	• Esempio: gestione degli attributi di tipo binario utilizzando l'API di basso livello AWS SDK per .NET Per il supporto dei tipi di dati binari aggiunto in AWS SDKs, è necessario scaricare la versione più recente SDKs e potrebbe essere necessari o aggiornare le applicazioni esistenti. Per ulteriori informazi oni sul download di AWS SDKs, consulta Esempi di codice .NET.	
Le voci della tabella DynamoDB possono essere aggiornate e copiate usando la console DynamoDB	Gli utenti di DynamoDB ora possono non solo aggiunger e ed eliminare gli elementi di una tabella, ma anche aggiornarli e copiarli usando la console DynamoDB. Questa nuova funzionalità semplific a le modifiche apportate ai singoli item attraverso la console.	14 agosto 2012

Modifica	Descrizione	Data della modifica
DynamoDB riduce i requisiti minimi di throughput della tabella	DynamoDB ora supporta requisiti minimi di throughput della tabella più bassi, in particolare 1 unità di capacità in scrittura e 1 unità di capacità in lettura. Per ulteriori informazioni, consulta l'argomento Quote in Amazon DynamoDBnella Guida per gli sviluppatori di Amazon DynamoDB.	9 agosto 2012
Supporto Signature Version 4	DynamoDB ora supporta Signature Version 4 per l'autenticazione delle richieste.	5 luglio 2012
Supporto di esplorazione della tabella nella console DynamoDB	La console DynamoDB ora supporta una funzione di esplorazione delle tabelle che consente di sfogliare ed eseguire query sui dati delle tabelle. Puoi anche inserire nuovi item o eliminare item esistenti. Le sezioni `SampleData` e Utilizzo della console sono state aggiornate per includere queste caratteristiche.	22 maggio 2012

Modifica	Descrizione	Data della modifica
Nuovi endpoint	La disponibilità di DynamoDB si espande con nuovi endpoint nella regione Stati Uniti occidentali (California), Stati Uniti occidentali (Oregon) e Asia Pacifico (Singapore). Per l'elenco corrente degli endpoint supportati, vai all'argomento relativo a regioni ed endpoint.	24 Aprile 2012
BatchWriteItem Supporto API	DynamoDB ora supporta un'API di scrittura in batch che ti consente di inserire ed eliminare più elementi da una o più tabelle in una singola chiamata API. Per ulteriori informazioni sull'API di scrittura in batch di DynamoDB, consulta BatchWriteItem. Per informazioni sull'utilizzo degli elementi e sull'utilizzo della funzionalità di scrittura in batch AWS SDKs, vedere Utilizzo di elementi e attributi in DynamoDB eEsempi di codice .NET.	19 Aprile 2012
Altri codici di errore documentati	Per ulteriori informazioni, consulta Gestione degli errori con DynamoDB.	5 Aprile 2012

Modifica	Descrizione	Data della modifica
Nuovo endpoint	DynamoDB si espande nella regione Asia Pacifico (Tokyo). Per l'elenco corrente degli endpoint supportati, consulta Regioni ed endpoint.	29 febbraio 2012
Parametro `ReturnedItemCount` aggiunto	Una nuova metrica`ReturnedItemCount`, fornisce il numero di elementi restituiti nella risposta di un'operazione di interrogazione o scansione per DynamoDB ed è disponibile per il monitoraggio. CloudWatch	24 febbraio 2012
Aggiunti esempi per l'incremento dei valori	DynamoDB supporta l'aumento e la diminuzione dei valori numerici esistenti. Gli esempi mostrano l'aggiunta ai valori esistenti nella sezione "Aggiornamento di un item" in: Utilizzo degli elementi: Java. Uso di elementi: .NET.	25 gennaio 2012
Versione iniziale del prodotto	DynamoDB è stato introdotto come nuovo servizio nella versione beta.	18 gennaio 2012

Funzionalità legacy di DynamoDB

I seguenti argomenti sono funzionalità legacy che DynamoDB supporta ancora. Non viene effettuato alcuno sviluppo attivo su queste funzionalità.

Argomenti

- Tabelle globali versione 2017.11.29 (Legacy)
- Versione precedente dell'API DynamoDB di basso livello (2011-12-05)
- Parametri condizionali legacy di DynamoDB

Tabelle globali versione 2017.11.29 (Legacy)

> **Important**
>
> Questa documentazione riguarda la versione 2017.11.29 (legacy) delle tabelle globali, che dovrebbe essere evitata per le nuove tabelle globali. Ove possibile, i clienti devono utilizzare le Tabelle globali versione 2019.11.21 (Corrente) poiché offrono maggiore flessibilità, maggiore efficienza e utilizzano meno capacità di scrittura rispetto alla versione 2017.11.29 (Legacy).
> Per determinare la versione in uso, consultare Determinare la versione di una tabella globale.
> Per aggiornare le tabelle globali esistenti dalla versione 2017.11.29 (legacy) alla versione 2019.11.21 (corrente), consultare Versioni delle tabelle globali DynamoDB.

Argomenti

- Tabelle globali: come funzionano
- Best practice e requisiti per la gestione delle tabelle globali
- Creazione di una tabella globale
- Monitoraggio delle tabelle globali
- Uso di IAM con le tabelle globali

Tabelle globali: come funzionano

> **Important**
>
> Questa documentazione riguarda la versione 2017.11.29 (legacy) delle tabelle globali, che dovrebbe essere evitata per le nuove tabelle globali. Ove possibile, i clienti devono utilizzare le Tabelle globali versione 2019.11.21 (Corrente) poiché offrono maggiore flessibilità, maggiore efficienza e utilizzano meno capacità di scrittura rispetto alla versione 2017.11.29 (Legacy).
> Per determinare la versione in uso, consultare Determinare la versione di una tabella globale.
> Per aggiornare le tabelle globali esistenti dalla versione 2017.11.29 (legacy) alla versione 2019.11.21 (corrente), consultare Versioni delle tabelle globali DynamoDB.

Nelle sezioni seguenti sono riportate ulteriori informazioni sui concetti e il comportamento delle tabelle globali in Amazon DynamoDB.

Concetti relativi alle tabelle globali per la versione 2017.11.29 (legacy)

Una tabella globale è una raccolta di una o più tabelle di replica, tutte di proprietà di un singolo account AWS.

Una tabella di replica (o replica, in breve) è una singola tabella DynamoDB che funziona come parte di una tabella globale. Ogni replica memorizza lo stesso set di elementi di dati. Una tabella globale specificata può avere una sola tabella di replica per regione AWS.

Di seguito è riportata una panoramica concettuale della modalità di creazione di una tabella globale.

1. Crea una tabella DynamoDB ordinaria, con DynamoDB Streams abilitato, in una regione AWS.
2. Ripeti il passaggio 1 per ogni altra regione in cui si desidera replicare i dati.
3. Definire una tabella globale DynamoDB in base alle tabelle create.

La Console di gestione AWS consente di automatizzare queste attività in modo da poter creare una tabella globale più rapidamente e semplicemente. Per ulteriori informazioni, consulta Creazione di una tabella globale.

La tabella globale DynamoDB risultante è costituita da più tabelle di replica (una per regione) che DynamoDB considera come una singola unità. Ogni replica ha lo stesso nome di tabella e lo stesso schema di chiave primaria. Quando un'applicazione scrive dati in una tabella di replica in una

regione, DynamoDB propaga automaticamente la scrittura alle altre tabelle di replica nelle altre regioni AWS.

> **Important**
>
> Per mantenere sincronizzati i dati della tabella, le tabelle globali creano automaticamente i seguenti attributi per ogni elemento:
>
> - `aws:rep:deleting`
> - `aws:rep:updatetime`
> - `aws:rep:updateregion`
>
> Non modificare questi attributi e non creare attributi con lo stesso nome.

È possibile aggiungere tabelle di replica alla tabella globale in modo che possa essere disponibile in altre regioni. (Per fare ciò, la tabella globale deve essere vuota. In altre parole, nessuna delle tabelle di replica può contenere alcun dato.)

È inoltre possibile rimuovere una tabella di replica da una tabella globale. In questo caso, la tabella viene completamente dissociata dalla tabella globale. Questa nuova tabella indipendente non interagisce più con la tabella globale e i dati non vengono più propagati da o verso la tabella globale.

> **Warning**
>
> È necessario tenere presente che rimuovere una replica non è un processo atomico. Per garantire un comportamento coerente e uno stato noto, è consigliabile indirizzare il traffico di scrittura dell'applicazione fuori dalla replica che deve essere rimossa in anticipo. Dopo la rimozione, attendi che tutti gli endpoint della Regione della replica mostrino la replica come dissociata prima di effettuare ulteriori scritture come tabella delle Regioni isolata.

Attività comuni

Le attività comuni per le tabelle globali funzionano come segue.

È possibile eliminare la tabella di replica di una tabella globale allo stesso modo di una tabella normale. Ciò interromperà la replica in tale regione ed eliminerà la copia della tabella conservata

nella regione. Non è possibile interrompere la replica e fare in modo che le copie della tabella esistano come entità indipendenti.

> **Note**
>
> Una tabella di origine non può essere eliminata finché non sono trascorse almeno 24 ore da quando è stata utilizzata per avviare una nuova regione. Se si tenta di eliminarla troppo presto, verrà restituito un errore.

Se le applicazioni aggiornano lo stesso elemento in regioni diverse quasi nello stesso momento è possibile che si verifichino dei conflitti. Per garantire la consistenza finale, le tabelle globali DynamoDB utilizzano una riconciliazione di tipo "last writer wins" per aggiornamenti contestuali. Tutte le repliche concorderanno sull'aggiornamento più recente e convergeranno verso uno stato in cui tutte hanno dati identici.

> **Note**
>
> Sono disponibili diversi modi per evitare i conflitti, inclusi i seguenti:
>
> - Utilizzo di una policy IAM per consentire solo le scritture sulla tabella in una Regione.
> - Utilizzo di una policy IAM per instradare gli utenti verso una sola Regione e mantenimento dell'altra in standby inattiva, oppure, in alternativa, instradamento degli utenti dispari verso una Regione e gli utenti pari verso un'altra Regione.
> - Evitando l'uso di aggiornamenti non idempotenti come Bookmark = Bookmark + 1, a favore di aggiornamenti statici come Bookmark=25.

Monitoraggio delle tabelle globali

Puoi usare CloudWatch per osservare la metrica `ReplicationLatency`. Questa metrica tiene traccia del tempo trascorso tra la visualizzazione di un elemento aggiornato nel flusso DynamoDB per una tabella di replica e la sua visualizzazione in un'altra replica nella tabella globale. `ReplicationLatency` è espresso in millisecondi e viene emesso per ogni coppia di Regioni di origine e di destinazione. Questa è l'unica metrica di CloudWatch fornita da Tabelle globali v2.

Le latenze osservate dipendono dalla distanza tra le regioni scelte, nonché altre variabili. Le latenze comprese tra 0,5 e 2,5 secondi per le regioni possono essere comuni all'interno della stessa area geografica.

Time to live (TTL)

È possibile utilizzare Time to live (TTL) per specificare un nome di attributo il cui valore indica l'ora di scadenza dell'elemento. Questo valore è specificato come un numero in secondi dall'inizio dell'epoca Unix.

Con la versione legacy delle tabelle globali, le eliminazioni TTL non vengono replicate automaticamente su altre repliche. Quando un elemento viene eliminato tramite una regola TTL, tale attività viene eseguita senza utilizzare unità di scrittura.

Tieni presente che se le tabelle di origine e di destinazione hanno una capacità in scrittura con provisioning molto bassa, ciò potrebbe causare una limitazione (della larghezza di banda della rete) poiché le eliminazioni TTL richiedono capacità di scrittura.

Flussi e transazioni con le tabelle globali

Ogni tabella globale produce un flusso indipendente basato su tutte le relative scritture, a prescindere dal punto di origine di tali scritture. Puoi scegliere di utilizzare questo flusso DynamoDB in una regione o in tutte le regioni in modo indipendente.

Se desideri scritture locali elaborate ma non scritture replicate, puoi aggiungere l'attributo di regione a ciascun elemento. Quindi puoi utilizzare un filtro di eventi Lambda per richiamare solo la Lambda per le scritture nella regione locale.

Le operazioni transazionali forniscono garanzie ACID (atomicità, coerenza, isolamento e durabilità) SOLO all'interno della Regione in cui è stata effettuata originariamente l'operazione di scrittura. Le transazioni non sono supportate tra le Regioni nelle tabelle globali.

Ad esempio, se hai una tabella globale con repliche nelle regioni Stati Uniti orientali (Ohio) e Stati Uniti occidentali (Oregon) ed esegui un'operazione TransactWriteItems nella regione Stati Uniti orientali (Ohio), potrai osservare transazioni parzialmente completate nella regione Stati Uniti occidentali (Oregon) man mano che le modifiche vengono replicate. Le modifiche vengono replicate in altre regioni solo dopo essere state confermate nella regione di origine.

> **Note**
>
> - Le tabelle globali eseguono la “scrittura diretta” (write around) DynamoDB Accelerator aggiornando direttamente DynamoDB. Di conseguenza, DAX non saprà che contiene dati obsoleti. La cache DAX verrà aggiornata solo alla scadenza del TTL della cache.
> - I tag nelle tabelle globali non vengono propagati automaticamente.

Velocità di trasmissione effettiva di lettura e di scrittura

Le tabelle globali gestiscono la velocità di trasmissione effettiva di lettura e di scrittura nei seguenti modi.

- La capacità di scrittura deve essere la stessa su tutte le istanze di tabella tra regioni.
- Con la versione 2019.11.21 (Corrente), se la tabella è impostata per supportare il dimensionamento automatico o è in modalità on demand, la capacità di scrittura viene automaticamente mantenuta sincronizzata. La quantità attuale di capacità di scrittura allocata in ciascuna Regione aumenterà e diminuirà indipendentemente all’interno delle impostazioni di dimensionamento automatico sincronizzate. Se viene attivata la modalità on demand della tabella, tale modalità verrà sincronizzata con le altre repliche.
- La capacità di lettura può variare tra le regioni perché le letture potrebbero essere diverse. Quando si aggiunge una replica globale a una tabella, la capacità della regione di origine viene propagata. Dopo la creazione, è possibile regolare la capacità di lettura di una replica e questa nuova impostazione non viene trasferita sull’altro lato.

Coerenza e risoluzione dei conflitti

Tutte le modifiche apportate a qualsiasi elemento in una tabella di replica vengono replicate a tutte le altre repliche all'interno della stessa tabella globale. In una tabella globale, un elemento appena scritto viene in genere propagato a tutte le tabelle di replica in pochi secondi.

Con una tabella globale, ogni tabella di replica memorizza lo stesso set di elementi di dati. DynamoDB non supporta la replica parziale soltanto di alcuni elementi.

Un'applicazione può leggere e scrivere dati in qualsiasi tabella di replica. DynamoDB supporta letture a consistenza finale tra regioni, ma non supporta elevate consistenze di lettura tra regioni. Se l'applicazione utilizza solo letture a consistenza finale ed emette le letture solo per una regione AWS,

funzionerà senza che sia necessaria alcuna modifica. Tuttavia, se l'applicazione richiede elevate consistenze di lettura, dovrà eseguire tutte le elevate consistenze di lettura e le scritture nella stessa regione. In caso contrario, se si scrive in una regione e si legge da un'altra, la risposta di lettura potrebbe includere dati non aggiornati che non riflettono i risultati delle scritture completate di recente nell'altra regione.

Se le applicazioni aggiornano lo stesso elemento in regioni diverse quasi nello stesso momento è possibile che si verifichino dei conflitti. Per garantire la consistenza finale, le tabelle globali DynamoDB utilizzano una riconciliazione di tipo l'ultimo che scrive vince tra gli aggiornamenti simultanei, in cui DynamoDB fa il massimo sforzo per determinare l'ultima operazione di scrittura. Con questo meccanismo di risoluzione dei conflitti, tutte le repliche concorderanno sull'aggiornamento più recente e convergeranno verso uno stato in cui tutte hanno dati identici.

Disponibilità e durabilità

Se una singola regione AWS viene isolata o degradata, l'applicazione può essere reindirizzata a una regione diversa ed eseguire letture e scritture su una tabella di replica diversa. È possibile applicare la logica di business personalizzata per determinare quando reindirizzare le richieste ad altre regioni.

Se una regione viene isolata o degradata, DynamoDB terrà traccia delle scritture eseguite ma non ancora propagate a tutte le tabelle di replica. Quando la regione torna online, DynamoDB riprenderà la propagazione di eventuali scritture in sospeso da tale regione alle tabelle di replica in altre regioni. Riprende inoltre la propagazione delle scritture da altre tabelle di replica alla regione che è ora di nuovo online. Tutte le scritture precedenti riuscite verranno propagate, a prescindere dalla durata dell'isolamento della regione.

Best practice e requisiti per la gestione delle tabelle globali

> **Important**
>
> Questa documentazione riguarda la versione 2017.11.29 (legacy) delle tabelle globali, che dovrebbe essere evitata per le nuove tabelle globali. Ove possibile, i clienti devono utilizzare le Tabelle globali versione 2019.11.21 (Corrente) poiché offrono maggiore flessibilità, maggiore efficienza e utilizzano meno capacità di scrittura rispetto alla versione 2017.11.29 (Legacy).
> Per determinare la versione in uso, consultare Determinare la versione di una tabella globale.
> Per aggiornare le tabelle globali esistenti dalla versione 2017.11.29 (legacy) alla versione 2019.11.21 (corrente), consultare Versioni delle tabelle globali DynamoDB.

Grazie alle tabelle globali di Amazon DynamoDB, è possibile replicare i dati delle tabelle tra regioni AWS. Per garantire la corretta replica dei dati è necessario che le tabelle di replica e gli indici secondari nella tabella globale abbiano impostazioni di capacità di scrittura identiche.

Argomenti

- Versione per tabelle globali
- Requisiti per l'aggiunta di una nuova tabella di replica
- Best practice e requisiti per la gestione della capacità

Versione per tabelle globali

Sono disponibili due versioni delle tabelle globali DynamoDB: Tabelle globali versione 2019.11.21 (Corrente) e Tabelle globali versione 2017.11.29 (Legacy). Ove possibile, i clienti devono utilizzare le Tabelle globali versione 2019.11.21 (Corrente) poiché offrono maggiore flessibilità, maggiore efficienza e utilizzano meno capacità di scrittura rispetto alla versione 2017.11.29 (Legacy).

Per determinare la versione in uso, consultare Determinare la versione di una tabella globale. Per aggiornare le tabelle globali esistenti dalla versione 2017.11.29 (legacy) alla versione 2019.11.21 (corrente), consultare Aggiornamento delle tabelle globali.

Requisiti per l'aggiunta di una nuova tabella di replica

Se si desidera aggiungere una nuova tabella di replica a una tabella globale, ciascuna delle seguenti condizioni deve essere true:

- La tabella deve avere la stessa chiave di partizione di tutte le altre repliche.
- La tabella deve avere le stesse impostazioni di gestione della capacità di scrittura specificate.
- La tabella deve avere lo stesso nome di tutte le altre repliche.
- La tabella deve avere DynamoDB Streams abilitato, con il flusso contenente sia le immagini nuove dell'elemento che le vecchie.
- Nessuna delle tabelle di replica nuove o esistenti nella tabella globale può contenere dati.

Se sono stati specificati indici secondari globali, è necessario che siano soddisfatte anche le condizioni indicate di seguito:

- Gli indici secondari globali devono avere lo stesso nome.

- Gli indici secondari globali devono avere la stessa chiave di partizione e la stessa chiave di ordinamento (se presente).

> **Important**
>
> Le impostazioni della capacità di scrittura devono essere impostate in modo coerente in tutte le tabelle di replica delle tabelle globali e negli indici secondari corrispondenti. Per aggiornare le impostazioni della capacità di scrittura per la tabella globale, si consiglia vivamente di utilizzare la console DynamoDB o l'operazione API `UpdateGlobalTableSettings`. `UpdateGlobalTableSettings` applica automaticamente le modifiche alle impostazioni della capacità di scrittura a tutte le tabelle di replica e agli indici secondari corrispondenti in una tabella globale. Se si utilizzano le operazioni `UpdateTable`, `RegisterScalableTarget` o `PutScalingPolicy`, è necessario applicare la modifica a ogni tabella di replica e indice secondario corrispondente singolarmente. Per ulteriori informazioni, consulta UpdateGlobalTableSettings nella Documentazione di riferimento delle API di Amazon DynamoDB.
>
> È consigliabile abilitare la scalabilità automatica per gestire le impostazioni della capacità in scrittura assegnata. Se si preferisce gestire manualmente le impostazioni della capacità di scrittura, assegnare lo stesso numero di unità di capacità di scrittura replicata a tutte le tabelle di replica. È inoltre possibile assegnare lo stesso numero di unità di capacità di scrittura replicata per gli indici secondari corrispondenti nella tabella globale.
>
> È inoltre necessario disporre delle autorizzazioni AWS Identity and Access Management(IAM) appropriate. Per ulteriori informazioni, consulta Uso di IAM con le tabelle globali.

Best practice e requisiti per la gestione della capacità

Quando si gestiscono le impostazioni di capacità per le tabelle di replica in DynamoDB, tenere presente quanto segue.

Utilizzo del dimensionamento automatico di DynamoDB

L'utilizzo della scalabilità automatica DynamoDB è il modo consigliato per gestire le impostazioni della capacità di velocità effettiva per le tabelle di replica che utilizzano la modalità assegnata. La scalabilità automatica di DynamoDB regola automaticamente le unità di capacità di lettura (RCU) e le unità di capacità di scrittura (WCU) per ogni tabella di replica in base al carico di lavoro effettivo

dell'applicazione. Per ulteriori informazioni, consulta Gestione automatica della capacità effettiva di trasmissione con il dimensionamento automatico di DynamoDB.

Se le tabelle di replica vengono create dalla Console di gestione AWS, la scalabilità automatica è abilitata per impostazione predefinita per ogni tabella di replica, con impostazioni di scalabilità automatica predefinite per la gestione delle unità di capacità di lettura e delle unità di capacità di scrittura.

Le modifiche apportate alle impostazioni di scalabilità automatica per una tabella di replica o un indice secondario effettuate tramite la console DynamoDB o utilizzando la chiamata `UpdateGlobalTableSettings` vengono applicate automaticamente a tutte le tabelle di replica e gli indici secondari corrispondenti nella tabella globale. Queste modifiche sovrascrivono eventuali impostazioni di scalabilità automatica esistenti. Ciò garantisce che le impostazioni della capacità di scrittura assegnata siano coerenti tra le tabelle di replica e gli indici secondari della tabella globale. Se si utilizzano le chiamate `UpdateTable`, `RegisterScalableTarget` o `PutScalingPolicy`, è necessario applicare la modifica a ogni tabella di replica e indice secondario corrispondente singolarmente.

> **Note**
>
> Se la scalabilità automatica non soddisfa le modifiche di capacità dell'applicazione (carico di lavoro imprevedibile) o se non si desidera configurarne le impostazioni (impostazioni di destinazione per soglia minima, massima o soglia di utilizzo), per gestire la capacità delle tabelle globali è possibile utilizzare la modalità on demand. Per ulteriori informazioni, consulta Modalità on demand.
>
> Se si abilita la modalità on demand su una tabella globale, il consumo delle unità di richiesta di scrittura replicata (rWCU) sarà coerente con il modo in cui le rWCU vengono assegnate. Ad esempio, se si eseguono 10 scritture su una tabella locale replicata in due regioni aggiuntive, si consumeranno 60 unità di richiesta di scrittura (10 + 10 + 10 = 30; 30 x 2 = 60). Le 60 unità di richiesta di scrittura utilizzate includono le richieste di scrittura extra utilizzate dalle tabelle globali versione 2017.11.29 (legacy) per aggiornare gli attributi `aws:rep:deleting`, `aws:rep:updatetime` e `aws:rep:updateregion`.

Gestione manuale della capacità

Se si decide di non utilizzare la scalabilità automatica di DynamoDB, sarà necessario impostare manualmente le impostazioni di capacità di lettura e scrittura su ogni tabella di replica e indice secondario.

Le unità di capacità di scrittura replicata (rWCU, Replicated Write Capacity Unit) assegnata in ogni tabella di replica devono essere impostate sul numero totale di rWCU necessarie per le scritture delle applicazioni in tutte le regioni moltiplicate per due. Ciò supporta le scritture delle applicazioni che si verificano nella regione locale e le scritture delle applicazioni replicate provenienti da altre regioni. Si supponga, ad esempio, di prevedere 5 scritture al secondo nella tabella di replica in Ohio e 5 scritture al secondo nella tabella di replica in Virginia settentrionale. In questo caso, è necessario allocare 20 rWCU per ogni tabella di replica (5 + 5 = 10; 10 x 2 = 20).

Per aggiornare le impostazioni della capacità di scrittura per la tabella globale, si consiglia vivamente di utilizzare la console DynamoDB o l'operazione API `UpdateGlobalTableSettings`. `UpdateGlobalTableSettings` applica automaticamente le modifiche alle impostazioni della capacità di scrittura a tutte le tabelle di replica e agli indici secondari corrispondenti in una tabella globale. Se si utilizzano le operazioni `UpdateTable`, `RegisterScalableTarget` o `PutScalingPolicy`, è necessario applicare la modifica a ogni tabella di replica e indice secondario corrispondente singolarmente. Per ulteriori informazioni, consulta la Documentazione di riferimento delle API di Amazon DynamoDB.

> **Note**
>
> Per aggiornare le impostazioni (`UpdateGlobalTableSettings`) per una tabella globale in DynamoDB, è necessario disporre delle autorizzazioni `dynamodb:UpdateGlobalTable`, `dynamodb:DescribeLimits`, `application-autoscaling:DeleteScalingPolicy` e `application-autoscaling:DeregisterScalableTarget`. Per ulteriori informazioni, consulta Uso di IAM con le tabelle globali.

Creazione di una tabella globale

> **Important**
>
> Questa documentazione riguarda la versione 2017.11.29 (legacy) delle tabelle globali, che dovrebbe essere evitata per le nuove tabelle globali. Ove possibile, i clienti devono utilizzare

le Tabelle globali versione 2019.11.21 (Corrente) poiché offrono maggiore flessibilità, maggiore efficienza e utilizzano meno capacità di scrittura rispetto alla versione 2017.11.29 (Legacy).
Per determinare la versione in uso, consultare Determinare la versione di una tabella globale.
Per aggiornare le tabelle globali esistenti dalla versione 2017.11.29 (legacy) alla versione 2019.11.21 (corrente), consultare Versioni delle tabelle globali DynamoDB.

In questa sezione viene descritto come creare una tabella globale utilizzando la console Amazon DynamoDB o AWS Command Line Interface (AWS CLI).

Argomenti

- Creazione di una tabella globale (console)
- Creazione di una tabella globale (AWS CLI)

Creazione di una tabella globale (console)

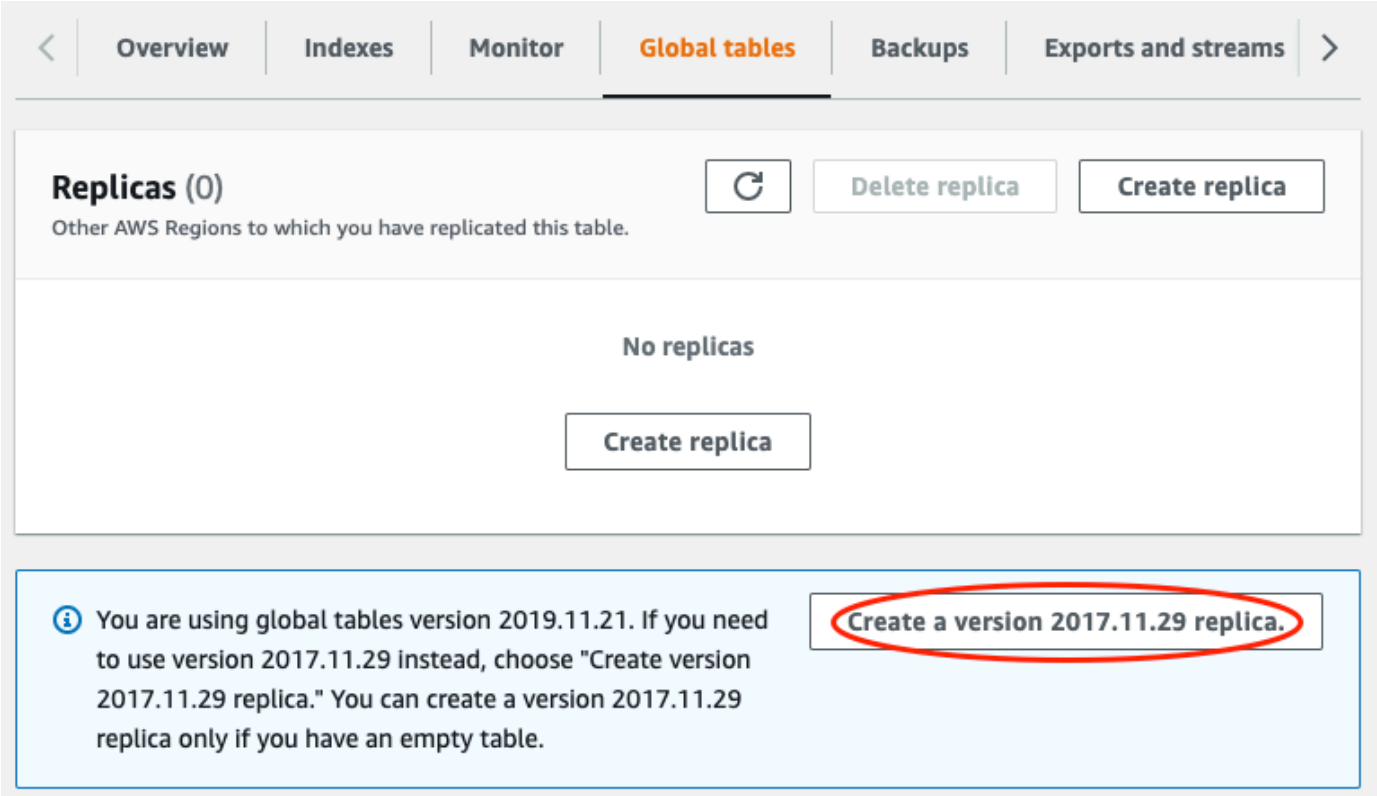
Segui queste fasi per creare una tabella globale usando la console. Il seguente esempio consente di creare una tabella globale con le tabelle di replica negli Stati Uniti e in Europa.

1. Apri la console DynamoDB all'indirizzo https://console.aws.amazon.com/dynamodb/home. Per questo esempio, scegli la regione us-east-2 (Stati Uniti orientali Ohio).
2. Nel riquadro di navigazione sul lato sinistro della console scegli Tables (Tabelle).
3. Scegliere Create Table (Crea tabella).

   Nel campo Table name (Nome tabella) immetti **`Music`**.

   In Primary key (Chiave primaria), immetti **`Artist`**. Scegli Add sort key (Aggiungi chiave di ordinamento) e immetti **`SongTitle`** (**`Artist`** e **`SongTitle`** devono essere entrambi stringhe)

   Per creare la tabella scegli Create (Crea). Questa tabella funge da prima tabella di replica in una nuova tabella globale. È il prototipo per altre tabelle di replica che verranno aggiunte in seguito.
4. Scegli la scheda Tabelle globali, quindi seleziona Crea una replica della versione 2017.11.29 (legacy).



5. Dal menu a discesa Available replication Regions (Regioni di replica disponibili), scegli Stati Uniti occidentali (Oregon).

   La console verifica che non esista una tabella con lo stesso nome nella regione selezionata. Se esiste una tabella con lo stesso nome, è necessario eliminare la tabella esistente prima di poter creare una nuova tabella di replica in quella regione.

6. Scegli Create replica (Crea replica). Viene avviato il processo di creazione della tabella nella regione Stati Uniti occidentali (Oregon).

   La scheda Global Table (Tabella globale) per la tabella selezionata (e per qualsiasi altra tabella di replica) mostra che la tabella è replicata in più regioni.

7. Ora aggiungi un'altra regione in modo che la tua tabella globale venga replicata e sincronizzata negli Stati Uniti e in Europa. A tale scopo, ripeti il passaggio 5, ma questa volta specifica Europa (Francoforte) al posto di Stati Uniti occidentali (Oregon)

8. Devi comunque utilizzare la Console di gestione AWS nella regione Stati Uniti orientali (Ohio). Seleziona Items (Elementi) dal menu di navigazione a sinistra, seleziona la tabella Musica, quindi scegli Create item (Crea elemento).

   a. In Artist, digita **`item_1`**.

b. Per SongTitle, inserisci **`Song Value 1`**.

c. Per scrivere l'elemento, scegli Create item (Crea elemento).

9. Dopo un breve periodo, l'item viene replicato in tutte e tre le regioni della tabella globale. Per verificarlo, nella console o con il selettore della regione in alto a destra, scegli Europa (Francoforte). La tabella `Music` nella regione Europa (Francoforte) deve contenere il nuovo elemento.
10. Ripeti il passaggio 9 e scegli Stati Uniti occidentali (Oregon) per verificare la replica in quella regione.

Creazione di una tabella globale (AWS CLI)

Completa questa procedura per creare una tabella globale `Music` utilizzando AWS CLI. Il seguente esempio consente di creare una tabella globale con le tabelle di replica negli Stati Uniti e in Europa.

1. Crea una nuova tabella (`Music`) nella regione Stati Uniti orientali (Ohio), con DynamoDB Streams abilitato (`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. Crea una tabella `Music` identica nella regione Stati Uniti orientali (Virginia settentrionale).

```
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. Crea una tabella globale (`Music`) costituita da tabelle di replica nelle regioni `us-east-2` e `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**
>
> Il nome della tabella globale (`Music`) deve corrispondere al nome di ciascuna delle tabelle di replica (`Music`). Per ulteriori informazioni, consulta Best practice per le tabelle globali.

4. Crea un'altra tabella nella regione Europa (Irlanda) con le stesse impostazioni definite nei passaggi 1 e 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
```

Dopo aver eseguito questo passaggio, aggiungere la nuova tabella alla tabella globale `Music`.

```
aws dynamodb update-global-table \
    --global-table-name Music \
    --replica-updates 'Create={RegionName=eu-west-1}' \
```

```
    --region us-east-2
```

5. Per verificare che la replica sta funzionando come dovrebbe, aggiungere un nuovo elemento alla tabella `Music` nella regione Stati Uniti orientali (Ohio).

```
aws dynamodb put-item \
    --table-name Music \
    --item '{"Artist": {"S":"item_1"},"SongTitle": {"S":"Song Value 1"}}' \
    --region us-east-2
```

6. Attendere alcuni secondi, quindi verificare se l'elemento è stato replicato correttamente nelle regioni Stati Uniti orientali (Virginia settentrionale) e 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
```

Monitoraggio delle tabelle globali

> **Important**
>
> Questa documentazione riguarda la versione 2017.11.29 (legacy) delle tabelle globali, che dovrebbe essere evitata per le nuove tabelle globali. Ove possibile, i clienti devono utilizzare le Tabelle globali versione 2019.11.21 (Corrente) poiché offrono maggiore flessibilità, maggiore efficienza e utilizzano meno capacità di scrittura rispetto alla versione 2017.11.29 (Legacy).
> Per determinare la versione in uso, consultare Determinare la versione di una tabella globale. Per aggiornare le tabelle globali esistenti dalla versione 2017.11.29 (legacy) alla versione 2019.11.21 (corrente), consultare Versioni delle tabelle globali DynamoDB.

È possibile utilizzare Amazon CloudWatch per monitorare il comportamento e le prestazioni di una tabella globale. Amazon DynamoDB pubblica i parametri `ReplicationLatency` e `PendingReplicationCount` per ogni replica nella tabella globale.

- **`ReplicationLatency`**: il tempo trascorso tra la visualizzazione di un elemento aggiornato nel flusso DynamoDB per una tabella di replica e la sua visualizzazione in un'altra replica nella tabella globale. `ReplicationLatency` è espresso in millisecondi e viene emesso per ogni coppia di regioni di origine e di destinazione.

  Durante il normale funzionamento, `ReplicationLatency` deve essere abbastanza costante. Un valore elevato per `ReplicationLatency` potrebbe indicare che gli aggiornamenti da una replica non si propagano ad altre tabelle di replica in modo tempestivo. Con il passare del tempo, ciò potrebbe tradursi in altre tabelle di replica in ritardo poiché non ricevono più aggiornamenti in modo coerente. In questo caso, devi verificare che le unità di capacità in lettura (RCU) e le unità di capacità in scrittura (WCU) siano identiche per ciascuna delle tabelle di replica. Inoltre, durante la scelta delle impostazioni WCU, segui le raccomandazioni in Versione per tabelle globali.

  `ReplicationLatency` può aumentare se una regione AWS diventa degradata e si ha una tabella di replica in tale regione. In questo caso, è possibile reindirizzare temporaneamente l'attività di lettura e scrittura dell'applicazione in una regione AWS diversa.

- **`PendingReplicationCount`**: il numero di aggiornamenti degli elementi scritti in una tabella di replica ma che non sono stati ancora scritti in un'altra replica nella tabella globale. `PendingReplicationCount` è espresso in numero di elementi e viene emesso per ogni coppia di origine e di destinazione.

  Durante il normale funzionamento, il valore di `PendingReplicationCount` dovrebbe essere molto basso. Se `PendingReplicationCount` aumenta per periodi prolungati, verificare se le impostazioni di capacità di scrittura assegnata delle tabelle di replica sono sufficienti per il carico di lavoro corrente.

  `PendingReplicationCount` può aumentare se una regione AWS diventa degradata e si ha una tabella di replica in tale regione. In questo caso, è possibile reindirizzare temporaneamente l'attività di lettura e scrittura dell'applicazione in una regione AWS diversa.

Per ulteriori informazioni, consulta Parametri e dimensioni di DynamoDB.

Uso di IAM con le tabelle globali

> **Important**
>
> Questa documentazione riguarda la versione 2017.11.29 (legacy) delle tabelle globali, che dovrebbe essere evitata per le nuove tabelle globali. Ove possibile, i clienti devono utilizzare le Tabelle globali versione 2019.11.21 (Corrente) poiché offrono maggiore flessibilità, maggiore efficienza e utilizzano meno capacità di scrittura rispetto alla versione 2017.11.29 (Legacy).
> Per determinare la versione in uso, consultare Determinare la versione di una tabella globale.
> Per aggiornare le tabelle globali esistenti dalla versione 2017.11.29 (legacy) alla versione 2019.11.21 (corrente), consultare Versioni delle tabelle globali DynamoDB.

Quando si crea una tabella globale per la prima volta, Amazon DynamoDB crea automaticamente un ruolo collegato al servizio AWS Identity and Access Management (IAM) per conto tuo. Questo ruolo è denominato e permette a DynamoDB di gestire per conto tuo la replica tra regioni per le tabelle globali. Non eliminare questo ruolo collegato al servizio. Se si fa, tutte le tabelle globali non funzioneranno più.

Per ulteriori informazioni sui ruoli collegati al servizio, consulta Utilizzo dei ruoli collegati al servizio nella Guida per l'utente IAM.

Per creare e gestire tabelle globali in DynamoDB, è necessario disporre dell'autorizzazione `dynamodb:CreateGlobalTable` per accedere a ciascuno dei seguenti elementi:

- La tabella di replica che si desidera aggiungere.
- Ogni replica esistente che fa già parte della tabella globale.
- La tabella globale stessa.

Per aggiornare le impostazioni (`UpdateGlobalTableSettings`) per una tabella globale in DynamoDB, è necessario disporre delle autorizzazioni `dynamodb:UpdateGlobalTable`, `dynamodb:DescribeLimits`, `application-autoscaling:DeleteScalingPolicy` e `application-autoscaling:DeregisterScalableTarget`.

Le autorizzazioni `application-autoscaling:DeleteScalingPolicy` e `application-autoscaling:DeregisterScalableTarget` sono necessarie quando si aggiorna una policy di

dimensionamento esistente. In questo modo il servizio delle tabelle globali può rimuovere la vecchia policy di dimensionamento prima di allegare la nuova policy alla tabella o all'indice secondario.

Se per gestire l'accesso a una tabella di replica si utilizza una policy IAM, è necessario applicare una policy identica a tutte le altre repliche all'interno di tale tabella globale. Questa procedura consente di mantenere un modello di autorizzazioni coerente in tutte le tabelle di replica.

L'uso di policy IAM identiche su tutte le repliche in una tabella globale consente inoltre di evitare di dover concedere l'accesso non intenzionale in lettura e scrittura ai dati delle tabelle globali. Si consideri, ad esempio, un utente che ha accesso a una sola replica in una tabella globale. Se tale utente può scrivere su questa replica, DynamoDB propaga la scrittura a tutte le altre tabelle di replica. In effetti, l'utente può (indirettamente) scrivere su tutte le altre repliche nella tabella globale. Questo scenario può essere evitato utilizzando policy IAM coerenti su tutte le tabelle di replica.

Esempio: consentire l' CreateGlobalTableazione

Prima di poter aggiungere una replica a una tabella globale, è necessario disporre dell'autorizzazione `dynamodb:CreateGlobalTable`per la tabella globale e per ciascuna delle relative tabelle di replica.

La seguente policy IAM concede le autorizzazioni per consentire l'operazione `CreateGlobalTable` su tutte le tabelle.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["dynamodb:CreateGlobalTable"],
            "Resource": "*"
        }
    ]
}
```

Esempio: consentire le azioni UpdateGlobalTable, DescribeLimits, application-autoscaling: e application-autoscaling: DeleteScalingPolicy DeregisterScalableTarget

Per aggiornare le impostazioni (`UpdateGlobalTableSettings`) per una tabella globale in DynamoDB, è necessario disporre delle autorizzazioni `dynamodb:UpdateGlobalTable`, `dynamodb:DescribeLimits`, `application-autoscaling:DeleteScalingPolicy` e `application-autoscaling:DeregisterScalableTarget`.

La seguente policy IAM concede le autorizzazioni per consentire l'operazione `UpdateGlobalTableSettings` su tutte le tabelle.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:UpdateGlobalTable",
                "dynamodb:DescribeLimits",
                "application-autoscaling:DeleteScalingPolicy",
                "application-autoscaling:DeregisterScalableTarget"
            ],
            "Resource": "*"
        }
    ]
}
```

Esempio: consente l' CreateGlobalTable azione per uno specifico nome di tabella globale con repliche consentite solo in determinate aree

La seguente policy IAM concede le autorizzazioni per consentire l'operazione `CreateGlobalTable` per creare una tabella globale denominata `Customers` con repliche in due regioni.

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"
            ]
        }
    ]
}
```

Versione precedente dell'API DynamoDB di basso livello (2011-12-05)

In questa sezione vengono riportate le operazioni disponibili nella versione dell'API di basso livello DynamoDB precedente (2011-12-05). Questa versione dell'API di basso livello viene mantenuta per compatibilità con le versioni precedenti delle applicazioni esistenti.

Le nuove applicazioni devono utilizzare la versione corrente dell'API (2012-08-10). Per ulteriori informazioni, consulta Documentazione di riferimento dell'API DynamoDB.

> **Note**
>
> Consigliamo di eseguire la migrazione delle applicazioni alla versione più recente dell'API (2012-08-10) poiché le nuove caratteristiche DynamoDB non saranno sottoposte al backport per la precedente versione dell'API.

Argomenti

- BatchGetItem
- BatchWriteItem
- CreateTable
- DeleteItem
- DeleteTable
- DescribeTables

- GetItem
- ListTables
- PutItem
- Query
- Scan
- UpdateItem
- UpdateTable

BatchGetItem

> **Important**
>
> ***In questa sezione si fa riferimento alla versione API 2011-12-05, che è obsoleta e non deve essere utilizzata per le nuove applicazioni.***
> Per informazioni sull'API di basso livello corrente, consulta la Amazon DynamoDB API Reference.

Descrizione

L'operazione `BatchGetItem` restituisce gli attributi per più elementi da più tabelle utilizzando le loro chiavi primarie. Il numero massimo di elementi che possono essere recuperati per una singola operazione è 100. Inoltre, il numero di elementi recuperati è vincolato da un limite di 1 MB. Se il limite delle dimensioni della risposta viene superato o viene restituito un risultato parziale perché viene superata la velocità effettiva assegnata della tabella oppure a causa di un errore di elaborazione interno, DynamoDB restituisce un valore `UnprocessedKeys` in modo da poter riprovare l'operazione iniziando con l'elemento successivo da ottenere. per applicare questo limite DynamoDB regola automaticamente il numero di elementi restituiti per pagina. Ad esempio, anche se si chiede di recuperare 100 elementi, ma ogni singolo elemento ha una dimensione di 50 KB, il sistema restituisce 20 elementi e un valore `UnprocessedKeys` appropriato in modo da visualizzare la pagina di risultati successiva. Se lo desidera, l'applicazione può includere la propria logica per assemblare le pagine dei risultati in un unico set.

Se non è possibile elaborare gli elementi a causa della velocità effettiva assegnata insufficiente su ciascuna delle tabelle coinvolte nella richiesta, DynamoDB restituisce un errore `ProvisionedThroughputExceededException`.

> **Note**
>
> Per impostazione predefinita,`BatchGetItem` esegue letture a consistenza finale su ogni tabella nella richiesta. È possibile impostare il parametro `ConsistentRead` su `true`, su base per tabella, se desideri invece avere letture consistenti.
> `BatchGetItem` recupera gli elementi in parallelo per ridurre al minimo la latenza delle risposte.
> Durante la progettazione dell'applicazione, tenere presente che DynamoDB non garantisce come gli attributi vengono ordinati nella risposta restituita. Includere i valori delle chiavi primarie in `AttributesToGet` per gli elementi nella richiesta per analizzare la risposta in base all'elemento.
> Se gli elementi richiesti non esistono, non viene restituito nulla nella risposta per tali elementi. Le richieste di elementi inesistenti consumano le unità di capacità di lettura minima in base al tipo di lettura. Per ulteriori informazioni, consulta Dimensioni e formati degli elementi di DynamoDB.

Richieste

Sintassi

```
// This header is abbreviated. For a sample of a complete header, see API DynamoDB di
 basso livello.
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"]
        }
    }
}
```

Nome	Descrizione	Richiesto
`RequestItems`	Un container del nome della tabella e degli elementi corrispondenti da ottenere in base alla chiave primaria. Durante la richiesta di elementi, ogni nome di tabella può essere richiamato una sola volta per operazione. Tipo: string Impostazione predefinita: nessuna	Sì
`Table`	Il nome della tabella che contiene gli elementi da ricevere. La voce è semplicem ente una stringa che specifica una tabella esistente senza etichetta. Tipo: string Impostazione predefinita: nessuna	Sì
`Table:Keys`	I valori della chiave primaria che definiscono gli elementi nella tabella specificata. Per ulteriori informazioni sulle chiavi primarie, vedere Chiave primaria.	Sì

Nome	Descrizione	Richiesto
	Tipo: chiavi	
`Table:AttributesToGet`	Matrice di nomi di attributi all'interno della tabella specificata. Se i nomi degli attributi non sono specifica ti, verranno restituiti tutti gli attributi. Se alcuni attributi non vengono trovati, non verranno visualizzati nel risultato. Tipo: Array	No
`Table:ConsistentRead`	Se impostato su `true`, viene emessa una lettura consisten te, altrimenti viene utilizzata la consistenza finale. Tipo: Booleano	No

Risposte

Sintassi

```
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"]}
        }
}
```

Nome	Descrizione
`Responses`	I nomi delle tabelle e i rispettivi attributi degli elementi delle tabelle. Tipo: Map
`Table`	Il nome della tabella che contiene gli elementi. La voce è semplicemente una stringa che specifica la tabella senza etichetta. Tipo: string

Nome	Descrizione
`Items`	Container per i nomi e i valori degli attributi che soddisfano i parametri dell'operazione. Tipo: mappa di nomi degli attributi e dei relativi tipi di dati e valori.
`ConsumedCapacityUnits`	Il numero di unità di capacità di lettura utilizzate per ogni tabella. Questo valore mostra il numero applicato alla velocità effettiva assegnata. Le richieste di elementi inesistenti consumano le unità di capacità di lettura minima in base al tipo di lettura. Per ulteriori informazioni, consulta Modalità con capacità allocata di DynamoDB. Tipo: numero
`UnprocessedKeys`	Contiene una matrice di tabelle e le rispettive chiavi che non sono state elaborate con la risposta corrente, probabilmente a causa del raggiungimento di un limite sulla dimensione della risposta. Il valore `UnprocessedKeys` è nella stessa forma di un parametro `RequestItems` (quindi il valore può essere fornito direttamente a un'operazione `BatchGetItem` successiva). Per ulteriori informazioni, consulta la sezione precedente sul parametro `RequestItems`. Tipo: Array

Nome	Descrizione
`UnprocessedKeys` : `Table`: `Keys`	I valori degli attributi della chiave primaria che definiscono gli elementi e gli attributi associati agli elementi. Per ulteriori informazioni sulle chiavi primarie, vedere Chiave primaria. Tipo: matrice di coppie nome-valore.
`UnprocessedKeys` : `Table`: `AttributesToGet`	I nomi degli attributi all'interno della tabella specificata. Se i nomi degli attributi non sono specificati, verranno restituiti tutti gli attributi . Se alcuni attributi non vengono trovati, non verranno visualizzati nel risultato. Tipo: matrice di nomi di attributi.
`UnprocessedKeys` : `Table`: `ConsistentRead`	Se impostato su`true`, viene utilizzata una lettura consistente per la tabella specifica ta, altrimenti viene utilizzata una lettura a consistenza finale. Tipo: Boolean

Errori speciali

Errore	Descrizione
`ProvisionedThroughputExceededException`	È stata superata la velocità effettiva assegnata massima consentita.

Esempi

Gli esempi seguenti mostrano una richiesta HTTP POST e una risposta tramite l'operazione BatchGetItem. Per gli esempi di utilizzo dell'SDK AWS, consulta Utilizzo di elementi e attributi in DynamoDB.

Richiesta di esempio

L'esempio seguente richiede gli attributi da due tabelle differenti.

```
// This header is abbreviated.
// For a sample of a complete header, see API DynamoDB di basso livello.
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"]}
    }
}
```

Risposta di esempio

Il seguente esempio è la risposta.

```
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**
>
> ***In questa sezione si fa riferimento alla versione API 2011-12-05, che è obsoleta e non deve essere utilizzata per le nuove applicazioni.***
> Per informazioni sull'API di basso livello corrente, consulta la Amazon DynamoDB API Reference.

Descrizione

Questa operazione consente di inserire ed eliminare più elementi da una o più tabelle con un'unica chiamata.

Per caricare un elemento è possibile utilizzare `PutItem` mentre per eliminare un elemento è possibile utilizzare `DeleteItem`. Tuttavia, se desideri caricare o eliminare grandi quantità di dati, ad esempio caricare grandi quantità di dati da Amazon EMR (Amazon EMR) o migrare dati da un altro database in DynamoDB, `BatchWriteItem` rappresenta un'alternativa efficiente.

Se si utilizzano linguaggi come Java, è possibile utilizzare i thread per caricare gli elementi in parallelo. Ciò aggiunge complessità all'applicazione per gestire i thread. Altri linguaggi non supportano il threading. Ad esempio, se si utilizza PHP, è necessario caricare o eliminare gli elementi uno alla volta. In entrambe le situazioni, `BatchWriteItem` fornisce un'alternativa in cui le operazioni put ed delete specificate vengono elaborate in parallelo, offrendo così la potenza dell'approccio del pool di thread senza dover introdurre complessità nell'applicazione.

Si noti che ogni singola operazione put ed delete specificata in un'operazione `BatchWriteItem` costa uguale in termini di unità di capacità consumate. Tuttavia, perché `BatchWriteItem` esegue le operazioni specificate in parallelo, si ottiene una latenza inferiore. Le operazioni di eliminazione su elementi inesistenti consumano 1 unità di capacità di scrittura. Per ulteriori informazioni sulle unità di capacità consumate, consulta Utilizzo di tabelle e dati in DynamoDB.

Quando si utilizza `BatchWriteItem`, tenere presente le limitazioni seguenti:

- Operazioni massime in una singola richiesta: è possibile specificare un totale massimo di 25 operazioni put o delete; tuttavia, la dimensione totale della richiesta non può superare 1 MB (il payload HTTP).
- È possibile utilizzare l'operazione `BatchWriteItem` solo per inserire ed eliminare elementi. Non è possibile utilizzarla per aggiornare gli elementi esistenti.
- Non è un'operazione atomica: le singole operazioni specificate in `BatchWriteItem` sono atomiche; tuttavia `BatchWriteItem`nel suo complesso è un'operazione "migliore sforzo" e non un'operazione atomica. In altre parole, in una richiesta `BatchWriteItem` alcune operazioni potrebbero avere esito positivo e altre potrebbero non riuscire. Le operazioni non riuscite vengono restituite in un campo `UnprocessedItems` nella risposta. Alcuni di questi errori potrebbero essere dovuti al superamento della velocità effettiva assegnata configurata per la tabella o a un errore temporaneo, ad esempio un errore di rete. È possibile esaminare e inviare nuovamente le richieste. In genere `BatchWriteItem` viene richiamato in un loop e in ogni iterazione vengono controllati gli elementi non elaborati, quindi si invia una nuova richiesta `BatchWriteItem` con gli elementi non elaborati.
- Non restituisce alcun elemento: `BatchWriteItem` è progettato per caricare grandi quantità di dati in modo efficiente. Non fornisce alcune delle sofisticazioni offerte da `PutItem` e `DeleteItem`. Ad esempio, `DeleteItem` supporta il campo `ReturnValues` nel corpo della richiesta per richiedere l'elemento eliminato nella risposta. L'operazione `BatchWriteItem` non restituisce alcun elemento nella risposta.
- A differenza di `PutItem` e `DeleteItem`, `BatchWriteItem` non consente di specificare le condizioni sulle singole richieste di scrittura nell'operazione.
- I valori degli attributi non devono essere nulli; gli attributi di tipo stringa e binario devono avere lunghezze maggiori di zero e gli attributi di tipo set non devono essere vuoti. Le richieste con valori vuoti verranno rifiutate con una`ValidationException`.

DynamoDB rifiuta l'intera operazione di scrittura in batch se si verifica una delle seguenti condizioni:

- Se una o più tabelle specificate nella richiesta `BatchWriteItem` non esiste.
- Se gli attributi della chiave primaria specificati su un elemento nella richiesta non corrispondono allo schema di chiave primaria della tabella corrispondente.

- Se si prova a eseguire più operazioni sullo stesso elemento nella stessa richiesta `BatchWriteItem`. Ad esempio, non è possibile inserire ed eliminare lo stesso elemento nella stessa richiesta `BatchWriteItem`.
- Se la dimensione totale della richiesta supera il limite di 1 MB (il payload HTTP).
- Se un singolo elemento in un batch supera il limite di dimensioni degli elementi di 64 KB.

Richieste

Sintassi

```
// This header is abbreviated. For a sample of a complete header, see API DynamoDB di
 basso livello.
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", ... ]
}
```

Nel corpo della richiesta, l'oggetto JSON `RequestItems` descrive le operazioni che desideri eseguire. Le operazioni sono raggruppate per tabelle. È possibile utilizzare `BatchWriteItem` per aggiornare o eliminare più elementi in più tabelle. Per ogni richiesta di scrittura specifica, è necessario identificare il tipo di richiesta (`PutItem`, `DeleteItem`) seguito da informazioni dettagliate sull'operazione.

- Per un `PutRequest`, si fornisce l'elemento, ovvero un elenco di attributi e i relativi valori.
- Per un `DeleteRequest`, si forniscono il nome e il valore della chiave primaria.

Risposte

Sintassi

Di seguito è riportata la sintassi del corpo JSON restituito nella risposta.

```
{
  "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.
```

Errori speciali

Nessun errore specifico per questa operazione.

Esempi

Il seguente esempio mostra una richiesta HTTP POST e la risposta di un'operazione `BatchWriteItem`. La richiesta specifica le seguenti operazioni nelle tabelle Reply e Thread:

- Inserimento ed eliminazione di un elemento dalla tabella Reply.
- Inserisci un elemento nella tabella Thread

Per gli esempi di utilizzo dell'SDK AWS, consulta Utilizzo di elementi e attributi in DynamoDB.

Richiesta di esempio

```
// This header is abbreviated. For a sample of a complete header, see API DynamoDB di
 basso livello.
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"
            }
          }
        }
      }
    ]
  }
}
```

Risposta di esempio

La risposta di esempio seguente mostra un'operazione put nelle tabelle Thread e Reply completata e un'operazione di eliminazione nella tabella Reply non riuscita (per motivi quali la limitazione causata quando si supera la velocità effettiva assegnata nella tabella). Nella risposta JSON, tenere presente quanto segue:

- L'oggetto `Responses` mostra che un'unità di capacità è stata utilizzata su entrambe le tabelle `Thread` e `Reply` come risultato dell'operazione put riuscita su ciascuna di queste tabelle.
- L'oggetto `UnprocessedItems` mostra l'operazione di eliminazione non riuscita sulla tabella `Reply`. È quindi possibile emettere una nuova chiamata `BatchWriteItem` per rispondere a queste richieste non elaborate.

```
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**
>
> ***In questa sezione si fa riferimento alla versione API 2011-12-05, che è obsoleta e non deve essere utilizzata per le nuove applicazioni.***
>
> Per informazioni sull'API di basso livello corrente, consulta la Amazon DynamoDB API Reference.

Descrizione

L'operazione `CreateTable` aggiunge una nuova tabella all'account.

Il nome della tabella deve essere univoco tra quelli associati all'account AWS che effettua la richiesta e alla regione AWS che riceve la richiesta (ad esempio dynamodb.us-west-2.amazonaws.com). Ogni endpoint DynamoDB è completamente indipendente. Ad esempio, se sono presenti due tabelle chiamate "MyTable", una in dynamodb.us-west-2.amazonaws.com e una in dynamodb.us-west-1.amazonaws.com, queste tabelle sono completamente indipendenti e non condividono alcun dato.

L'operazione `CreateTable` attiva un flusso di lavoro asincrono per iniziare a creare la tabella. DynamoDB restituisce immediatamente lo stato della tabella (`CREATING`) fino a quando la tabella si trova nello stato `ACTIVE`. Una volta che la tabella si trova nello stato `ACTIVE`, è possibile eseguire le operazioni del piano dati.

Utilizza l'operazione DescribeTables per visualizzare lo stato della tabella.

Richieste

Sintassi

```
// This header is abbreviated.
// For a sample of a complete header, see API DynamoDB di basso livello.
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}
}
```

Nome	Descrizione	Richiesto
`TableName`	Nome della tabella da creare.	Sì

Nome	Descrizione	Richiesto
	I caratteri consentiti sono a-z, A-Z, 0-9, '_' (sottolineatura), '-' (trattino) e '.' (punto). I nomi possono contenere un numero di caratteri compreso tra 3 e 255 caratteri. Tipo: string	

Nome	Descrizione	Richiesto
`KeySchema`	La struttura della chiave primaria (semplice o composita) per la tabella. Una coppia nome-valore per `HashKeyElement` è obbligatoria mentre una coppia nome-valore per `RangeKeyElement` è facoltativa (richiesta solo per le chiavi primarie composite). Per ulteriori informazioni sulle chiavi primarie, vedere Chiave primaria. I nomi degli elementi della chiave primaria possono contenere un massimo di 255 caratteri, senza alcuna limitazione. I valori possibili per Attribute Type sono "S" (stringa), "N" (numerico) o "B" (binario). Tipo: mappa di `HashKeyElement` , oppure `HashKeyElement` e `RangeKeyElement` per una chiave primaria composita.	Sì

Nome	Descrizione	Richiesto
`ProvisionedThroughput`	La nuova velocità effettiva per la tabella specificata, costituita dai valori per `ReadCapacityUnits` e `WriteCapacityUnits`. Per informazioni dettagliate, consultare Modalità con capacità allocata di DynamoDB. **Note** Per i valori massimi e minimi correnti, consulta Quote in Amazon DynamoDB. Tipo: Array	Sì

Nome	Descrizione	Richiesto
`ProvisionedThroughput : ReadCapacityUnits`	Imposta il numero minimo di `ReadCapacityUnits` con consistenza consumate al secondo per la tabella specificata prima che DynamoDB bilanci il carico con altre operazioni. Le operazioni di lettura a consistenza finale richiedono meno sforzo rispetto a un'operazione di lettura consistente, quindi un'impostazione di 50 `ReadCapacityUnits` consistente al secondo fornisce 100 `ReadCapacityUnits` a consistenza finale al secondo. Tipo: numero	Sì
`ProvisionedThroughput : WriteCapacityUnits`	Imposta il numero minimo di `WriteCapacityUnits` consumate al secondo per la tabella specificata prima che DynamoDB bilanci il carico con altre operazioni. Tipo: numero	Sì

Risposte

Sintassi

```
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"
    }
}
```

Nome	Descrizione
`TableDescription`	Un container per le proprietà della tabella.
`CreationDateTime`	La data in cui la tabella è stata creata in formato orario epoch UNIX. Tipo: numero
`KeySchema`	La struttura della chiave primaria (semplice o composita) per la tabella. Una coppia nome-valore per `HashKeyElement` è obbligatoria mentre una coppia nome-valore per `RangeKeyElement` è facoltativa (richiesta solo per le chiavi primarie composite). Per ulteriori informazioni sulle chiavi primarie, vedere Chiave primaria. Tipo: mappa di `HashKeyElement`, oppure `HashKeyElement` e `RangeKeyElement` per una chiave primaria composita.
`ProvisionedThroughput`	Velocità effettiva per la tabella specificata, costituita dai valori per `ReadCapacityUnits`

Nome	Descrizione
	e `WriteCapacityUnits` . Consultare Modalità con capacità allocata di DynamoDB. Tipo: Array
`ProvisionedThroughput :ReadCapacityUnits`	Il numero minimo di `ReadCapacityUnits` consumate al secondo prima che DynamoDB bilanci il carico con altre operazioni Tipo: numero
`ProvisionedThroughput :WriteCapacityUnits`	Il numero minimo di `ReadCapacityUnits` consumate al secondo prima che `WriteCapacityUnits` bilanci il carico con altre operazioni Tipo: numero
`TableName`	Il nome della tabella creata. Tipo: string
`TableStatus`	Lo stato corrente della tabella (`CREATING`). Una volta che la tabella si trova nello stato `ACTIVE`, è possibile inserire i dati. Utilizza l'API DescribeTables per controllare lo stato della tabella. Tipo: string

Errori speciali

Errore	Descrizione
`ResourceInUseException`	Tentativo di creare nuovamente una tabella già esistente.
`LimitExceededException`	Il numero di richieste di tabella simultane e (numero cumulativo di tabelle nello stato `CREATING`, `DELETING` o `UPDATING`) supera il massimo consentito. **Note** Per i valori massimi e minimi correnti, consulta Quote in Amazon DynamoDB. .

Esempi

Nell'esempio seguente viene creata una tabella con una chiave primaria composita contenente una stringa e un numero. Per gli esempi di utilizzo dell'SDK AWS, consulta Utilizzo di tabelle e dati in DynamoDB.

Richiesta di esempio

```
// This header is abbreviated.
// For a sample of a complete header, see API DynamoDB di basso livello.
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}
}
```

Risposta di esempio

```
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"
    }
}
```

Operazioni correlate

- DescribeTables
- DeleteTable

DeleteItem

> **Important**
>
> ***In questa sezione si fa riferimento alla versione API 2011-12-05, che è obsoleta e non deve essere utilizzata per le nuove applicazioni.***
>
> Per informazioni sull'API di basso livello corrente, consulta la Amazon DynamoDB API Reference.

Descrizione

Elimina un singolo elemento in una tabella in base alla chiave primaria. È possibile eseguire un'operazione di eliminazione condizionale che elimina l'elemento se esiste o se ha un valore di attributo previsto.

> **Note**
>
> Se si specifica `DeleteItem` senza attributi o valori, tutti gli attributi per l'elemento saranno eliminati.
> A meno che non si specifichino delle condizioni, `DeleteItem` è un'operazione idempotente; eseguendola più volte sullo stesso elemento o attributo non si ha una risposta di errore.
> Le eliminazioni condizionali sono utili per eliminare elementi e attributi solo quando sono soddisfatte condizioni specifiche. Se le condizioni sono soddisfatte, DynamoDB esegue l'eliminazione. In caso contrario, l'elemento non viene eliminato.
> È possibile eseguire il controllo condizionale previsto su un attributo per operazione.

Richieste

Sintassi

```
// This header is abbreviated.
// For a sample of a complete header, see API DynamoDB di basso livello.
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"}
}
```

Nome	Descrizione	Richiesto
`TableName`	Il nome della tabella che contiene l'elemento da eliminare. Tipo: string	Sì
`Key`	La chiave primaria che definisce l'elemento. Per ulteriori informazioni sulle chiavi primarie, vedere Chiave primaria. Tipo: mappa di `HashKeyElement` sul suo valore e `RangeKeyElement` sul suo valore.	Sì
`Expected`	Designa un attributo per un'eliminazione condizionale. Il parametro `Expected` consente di fornire un nome di attributo e se DynamoDB deve controllare o meno se l'attributo ha un valore particolare prima di eliminarlo. Tipo: mappa dei nomi degli attributi.	No
`Expected:AttributeName`	Il nome dell'attributo per il put condizionale. Tipo: string	No
`Expected:AttributeName: ExpectedAttributeValue`	Utilizza questo parametro per specificare se esiste già un	No

Nome	Descrizione	Richiesto

	valore per la coppia nome-valore dell'attributo.
	
	La seguente notazione JSON elimina l'elemento se l'attributo "Colore" non esiste per quell'elemento:
	
	`"Expected" : {"Color":{"Exists":false}}`
	
	La seguente notazione JSON controlla se l'attributo con nome "Colore" ha un valore esistente pari a "Giallo" prima di eliminare l'elemento:
	
	`"Expected" : {"Color":{"Exists":true},{"Value": {"S":"Yellow"}}}`
	
	Per impostazione predefinita, se si utilizza il parametro `Expected` e si fornisce un `Value`, DynamoDB presuppone che l'attributo esista e abbia un valore corrente da sostituire. Quindi non è necessario specificare `{"Exists":true}` perché è implicito. È possibile abbreviare la richiesta in:
	
	`"Expected" :`	

Nome	Descrizione	Richiesto
	`{"Color":{"Value":{"S":"Yellow"}}}` **Note** Se si specifica `{"Exists":true}` senza un valore di attributo da controllare, DynamoDB restituisce un errore.	
`ReturnValues`	Utilizza questo parametro se desideri ottenere le coppie nome-valore dell'attributo prima che vengano eliminate. I valori possibili dei parametri sono `NONE` (predefinito) o `ALL_OLD`. Se viene specificato `ALL_OLD`, viene restituito il contenuto del vecchio elemento. Se questo parametro non viene fornito o è `NONE`, non viene restituito nulla. Tipo: string	No

Risposte

Sintassi

```
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
}
```

Nome	Descrizione
`Attributes`	Se il parametro `ReturnValues` viene fornito come `ALL_OLD` nella richiesta, DynamoDB restituisce una matrice di coppie nome-valore dell'attributo (essenzialmente, l'elemento eliminato). In caso contrario, la risposta contiene un set vuoto. Tipo: matrice di coppie nome-valore.
`ConsumedCapacityUnits`	Il numero di unità di capacità di scrittura utilizzate dall'operazione. Questo valore mostra il numero applicato alla velocità effettiva assegnata. Le richieste di eliminazione su elementi inesistenti consumano 1 unità di capacità di scrittura. Per ulteriori informazioni, consulta Modalità con capacità allocata di DynamoDB. Tipo: numero

Errori speciali

Errore	Descrizione
`ConditionalCheckFailedException`	Controllo condizionale non riuscito. Il valore di attributo previsto non è stato trovato.

Esempi

Richiesta di esempio

```
// This header is abbreviated.
// For a sample of a complete header, see API DynamoDB di basso livello.
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"
}
```

Risposta di esempio

```
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
```

```
}
```

Operazioni correlate

- PutItem

DeleteTable

> **Important**
>
> ***In questa sezione si fa riferimento alla versione API 2011-12-05, che è obsoleta e non deve essere utilizzata per le nuove applicazioni.***
> Per informazioni sull'API di basso livello corrente, consulta la Amazon DynamoDB API Reference.

Descrizione

L'operazione `DeleteTable` elimina una tabella e tutti i suoi elementi. Dopo una richiesta `DeleteTable`, la tabella specificata si trova nello stato `DELETING` finché DynamoDB non completa l'eliminazione. Se la tabella si trova nello stato `ACTIVE`, è possibile eliminarlo. Se una tabella si trova nello `CREATING` o `UPDATING`, allora DynamoDB restituisce un errore `ResourceInUseException`. Se la tabella specificata non esiste, DynamoDB restituisce un `ResourceNotFoundException`. Se la tabella si trova già nello stato `DELETING`, non viene restituito alcun errore.

> **Note**
>
> DynamoDB potrebbe continuare ad accettare le richieste di operazioni del piano dati, ad esempio `GetItem` e`PutItem`, su una tabella nello stato `DELETING` fino al completamento dell'eliminazione della tabella.

Le tabelle sono univoche tra quelle associate all'account AWS che effettua la richiesta e alla regione AWS che riceve la richiesta (ad esempio dynamodb.us-west-2.amazonaws.com). Ogni endpoint DynamoDB è completamente indipendente. Ad esempio, se sono presenti due tabelle chiamate "MyTable", una in dynamodb.us-west-2.amazonaws.com e una in dynamodb.us-west-1.amazonaws.com, queste tabelle sono completamente indipendenti e non condividono alcun dato; l'eliminazione di una non implica l'eliminazione dell'altra.

Utilizza l'operazione DescribeTables per visualizzare lo stato della tabella.

Richieste

Sintassi

```
// This header is abbreviated.
// For a sample of a complete header, see API DynamoDB di basso livello.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.DeleteTable
content-type: application/x-amz-json-1.0

{"TableName":"Table1"}
```

Nome	Descrizione	Richiesto
`TableName`	Il nome della tabella da eliminare. Tipo: string	Sì

Risposte

Sintassi

```
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"
    }
```

```
}
```

Nome	Descrizione
`TableDescription`	Un container per le proprietà della tabella.
`CreationDateTime`	La data in cui è stata creata la tabella. Tipo: numero
`KeySchema`	La struttura della chiave primaria (semplice o composita) per la tabella. Una coppia nome-valore per `HashKeyElement` è obbligatoria mentre una coppia nome-valore per `RangeKeyElement` è facoltativa (richiesta solo per le chiavi primarie composite). Per ulteriori informazioni sulle chiavi primarie, vedere Chiave primaria. Tipo: mappa di `HashKeyElement` , oppure `HashKeyElement` e `RangeKeyElement` per una chiave primaria composita.
`ProvisionedThroughput`	Velocità effettiva per la tabella specificata, costituita dai valori per `ReadCapacityUnits` e `WriteCapacityUnits` . Consultare Modalità con capacità allocata di DynamoDB.
`ProvisionedThroughput` : `ReadCapacityUnits`	Il numero minimo di `ReadCapacityUnits` consumate al secondo per la tabella specificata prima che DynamoDB bilanci il carico con altre operazioni. Tipo: numero
`ProvisionedThroughput` : `WriteCapacityUnits`	Il numero minimo di `WriteCapacityUnits` consumate al secondo per la tabella specificata

Nome	Descrizione
	prima che DynamoDB bilanci il carico con altre operazioni. Tipo: numero
`TableName`	Il nome della tabella eliminata. Tipo: string
`TableStatus`	Lo stato corrente della tabella (`DELETING`). Una volta che la tabella viene eliminata, le richieste successive per la tabella restituiscono `resource not found` . Utilizza l'operazione DescribeTables per visualizzare lo stato della tabella. Tipo: string

Errori speciali

Errore	Descrizione
`ResourceInUseException`	La tabella è nello stato `CREATING` o `UPDATING` e non può essere eliminata.

Esempi

Richiesta di esempio

```
// This header is abbreviated. For a sample of a complete header, see API DynamoDB di
 basso livello.
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"}
```

Risposta di esempio

```
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"
}
```

Operazioni correlate

- CreateTable
- DescribeTables

DescribeTables

> **Important**
>
> ***In questa sezione si fa riferimento alla versione API 2011-12-05, che è obsoleta e non deve essere utilizzata per le nuove applicazioni.***
> Per informazioni sull'API di basso livello corrente, consulta la Amazon DynamoDB API Reference.

Descrizione

Restituisce informazioni sulla tabella, inclusi lo stato corrente della tabella, lo schema della chiave primaria e la data di creazione della tabella. I risultati di DescribeTable sono a consistenza finale. Se si utilizza DescribeTable troppo presto nel processo di creazione di una tabella, DynamoDB restituisce una `ResourceNotFoundException`. Se si utilizza DescribeTable troppo presto nel

processo di aggiornamento di una tabella, i nuovi valori potrebbero non essere immediatamente disponibili.

Richieste

Sintassi

```
// This header is abbreviated.
// For a sample of a complete header, see API DynamoDB di basso livello.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.DescribeTable
content-type: application/x-amz-json-1.0

{"TableName":"Table1"}
```

Nome	Descrizione	Richiesto
`TableName`	Il nome della tabella da descrivere. Tipo: string	Sì

Risposte

Sintassi

```
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"
    }
}
```

Nome	Descrizione
`Table`	Un container per la tabella descritta. Tipo: string
`CreationDateTime`	La data in cui la tabella è stata creata in formato orario epoch UNIX.
`ItemCount`	Il numero di elementi nella tabella specificata. DynamoDB aggiorna questo valore ogni sei ore circa. Le modifiche recenti potrebbero non essere riflesse in questo valore. Tipo: numero
`KeySchema`	La struttura della chiave primaria (semplice o composita) per la tabella. Una coppia nome-valore per `HashKeyElement` è obbligatoria mentre una coppia nome-valore per `RangeKeyElement` è facoltativa (richiesta solo per le chiavi primarie composite). La dimensione massima della chiave hash è 2048 byte. La dimensione massima della chiave di intervallo è 1024 byte. Entrambi i limiti sono applicati separatamente (cioè è possibile avere una chiave hash + intervallo combinato, ovvero 2048+1024). Per ulteriori informazioni sulle chiavi primarie, vedere Chiave primaria.
`ProvisionedThroughput`	Velocità effettiva assegnata, costituita dai valori per `LastIncreaseDateTime` (se applicabile), `LastDecreaseDateTime` (se applicabile),`ReadCapacityUnits` e `WriteCapa`

Nome	Descrizione
	`cityUnits` . Se la velocità effettiva della tabella non è mai stata aumentata o ridotta, DynamoDB non restituisce valori per tali elementi. Consultare Modalità con capacità allocata di DynamoDB. Tipo: Array
`TableName`	Il nome della tabella richiesta. Tipo: string
`TableSizeBytes`	La dimensione totale della tabella specificata in byte. DynamoDB aggiorna questo valore ogni sei ore circa. Le modifiche recenti potrebbero non essere riflesse in questo valore. Tipo: numero
`TableStatus`	Lo stato corrente della tabella (`CREATING`, `ACTIVE`, `DELETING` o `UPDATING`). Una volta che la tabella si trova nello stato `ACTIVE`, è possibile aggiungere i dati.

Errori speciali

Non esiste alcun errore specifico per questa operazione.

Esempi

Gli esempi seguenti mostrano una richiesta POST HTTP e una risposta tramite l'operazione DescribeTable per una tabella denominata "comp-table". La tabella dispone di una chiave primaria composita.

Richiesta di esempio

```
// This header is abbreviated.
```

```
// For a sample of a complete header, see API DynamoDB di basso livello.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.DescribeTable
content-type: application/x-amz-json-1.0

{"TableName":"users"}
```

Risposta di esempio

```
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"
    }
}
```

Operazioni correlate

- CreateTable
- DeleteTable
- ListTables

GetItem

> **Important**
>
> ***In questa sezione si fa riferimento alla versione API 2011-12-05, che è obsoleta e non deve essere utilizzata per le nuove applicazioni.***

Per informazioni sull'API di basso livello corrente, consulta la Amazon DynamoDB API Reference.

Descrizione

L'operazione `GetItem`restituisce un set di `Attributes` per un elemento che corrisponde alla chiave primaria. Se non viene trovato un elemento corrispondente, `GetItem` non restituisce alcun dato.

Per impostazione predefinita, l'operazione `GetItem` usa le letture a consistenza finali. Se le letture a consistenza finale non sono accettabili per l'applicazione, utilizza `ConsistentRead`. Sebbene questa operazione possa richiedere più tempo di una lettura standard, restituisce sempre l'ultimo valore aggiornato. Per ulteriori informazioni, consulta Coerenza di lettura di DynamoDB.

Richieste

Sintassi

```
// This header is abbreviated.
// For a sample of a complete header, see API DynamoDB di basso livello.
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
}
```

Nome	Descrizione	Richiesto
`TableName`	Il nome della tabella che contiene gli elementi richiesti. Tipo: string	Sì

Nome	Descrizione	Richiesto
`Key`	I valori della chiave primaria che definiscono l'elemento. Per ulteriori informazioni sulle chiavi primarie, vedere Chiave primaria. Tipo: mappa di `HashKeyElement` sul suo valore e `RangeKeyElement` sul suo valore.	Sì
`AttributesToGet`	Matrice di nomi di attributi. Se i nomi degli attributi non sono specificati, verranno restituit i tutti gli attributi. Se alcuni attributi non vengono trovati, non verranno visualizzati nel risultato. Tipo: Array	No
`ConsistentRead`	Se impostato su `true`, viene emessa una lettura consisten te, altrimenti viene utilizzata la consistenza finale. Tipo: Booleano	No

Risposte

Sintassi

```
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
}
```

Nome	Descrizione
`Item`	Contiene gli attributi richiesti. Tipo: mappa di coppie nome-valore dell'attr ibuto.
`ConsumedCapacityUnits`	Il numero di unità di capacità di scrittura utilizzate dall'operazione. Questo valore mostra il numero applicato alla velocità effettiva assegnata. Le richieste di elementi inesisten ti consumano le unità di capacità di lettura minima in base al tipo di lettura. Per ulteriori informazioni, consulta Modalità con capacità allocata di DynamoDB. Tipo: numero

Errori speciali

Nessun errore specifico per questa operazione.

Esempi

Per gli esempi di utilizzo dell'SDK AWS, consulta Utilizzo di elementi e attributi in DynamoDB.

Richiesta di esempio

```
// This header is abbreviated.
// For a sample of a complete header, see API DynamoDB di basso livello.
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
}
```

Risposta di esempio

Si noti che il valore di ConsumedCapacityUnits è 1, poiché il parametro facoltativo `ConsistentRead` è impostato su `true`. Se `ConsistentRead` è impostato su `false` (o se non è specificato) per la stessa richiesta, la risposta è a consistenza finale e il valore ConsumedCapacityUnits sarebbe 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**
>
> ***In questa sezione si fa riferimento alla versione API 2011-12-05, che è obsoleta e non deve essere utilizzata per le nuove applicazioni.***
> Per informazioni sull'API di basso livello corrente, consulta la Amazon DynamoDB API Reference.

Descrizione

Restituisce una matrice di tutte le tabelle associate all'account e all'endpoint correnti.

Ogni endpoint DynamoDB è completamente indipendente. Ad esempio, se sono presenti due tabelle chiamate "MyTable", una in dynamodb.us-west-2.amazonaws.com e una in dynamodb.us-east-1.amazonaws.com, queste tabelle sono completamente indipendenti e non condividono alcun dato. L'operazione ListTables restituisce tutti i nomi di tabella associati all'account che effettua la richiesta, per l'endpoint che riceve la richiesta.

Richieste

Sintassi

```
// This header is abbreviated.
// For a sample of a complete header, see API DynamoDB di basso livello.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.ListTables
content-type: application/x-amz-json-1.0

{"ExclusiveStartTableName":"Table1","Limit":3}
```

Per impostazione predefinita, l'operazione ListTables richiede tutti i nomi delle tabelle associati all'account che effettua la richiesta per l'endpoint che riceve la richiesta.

Nome	Descrizione	Richiesto
`Limit`	Un numero massimo di nomi di tabella da restituire. Tipo: integer	No
`ExclusiveStartTableName`	Il nome della tabella che inizia l'elenco. Se hai già eseguito un'operazione ListTables e hai ricevuto un valore `LastEvaluatedTableName` nella risposta, utilizza questo valore qui per continuare l'elenco. Tipo: string	No

Risposte

Sintassi

```
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"}
```

Nome	Descrizione
`TableNames`	I nomi delle tabelle associate al conto corrente nell'endpoint corrente. Tipo: Array
`LastEvaluatedTableName`	Il nome dell'ultima tabella dell'elenco corrente, solo se alcune tabelle per l'account e l'endpoin t non sono state restituite. Questo valore non esiste in una risposta se sono già stati restituit i tutti i nomi delle tabelle. Usare questo valore come `ExclusiveStartTableName` in una nuova richiesta per continuare l'elenco fino a quando non vengono restituiti tutti i nomi delle tabelle. Tipo: string

Errori speciali

Non esiste alcun errore specifico per questa operazione.

Esempi

Gli esempi seguenti mostrano una richiesta HTTP POST e una risposta tramite l'operazione ListTables.

Richiesta di esempio

```
// This header is abbreviated.
// For a sample of a complete header, see API DynamoDB di basso livello.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.ListTables
content-type: application/x-amz-json-1.0

{"ExclusiveStartTableName":"comp2","Limit":3}
```

Risposta di esempio

```
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"]}
```

Operazioni correlate

- DescribeTables
- CreateTable
- DeleteTable

PutItem

> **Important**
>
> ***In questa sezione si fa riferimento alla versione API 2011-12-05, che è obsoleta e non deve essere utilizzata per le nuove applicazioni.***
>
> Per informazioni sull'API di basso livello corrente, consulta la Amazon DynamoDB API Reference.

Descrizione

Crea un nuovo elemento o sostituisce un vecchio elemento con un nuovo elemento (includendo tutti gli attributi). Se nella tabella specificata esiste già un elemento con la stessa chiave primaria, il nuovo elemento sostituisce completamente l'elemento esistente. È possibile eseguire un put condizionale (inserire un nuovo elemento se non ne esiste uno con la chiave primaria specificata) o sostituire un elemento esistente se dispone di determinati valori di attributo.

I valori degli attributi non possono essere nulli; gli attributi stringa e di tipo binario devono avere lunghezze maggiori di zero e gli attributi di tipo impostato non devono essere vuoti. Le richieste con valori vuoti verranno rifiutate con una eccezione `ValidationException`.

> **Note**
>
> Per essere certi che un nuovo elemento non sostituisca un elemento esistente, utilizza un'operazione put condizionale con `Exists` impostato su `false`per l'attributo o gli attributi della chiave primaria.

Per ulteriori informazioni sull'utilizzo di `PutItem`, consultare Utilizzo di elementi e attributi in DynamoDB.

Richieste

Sintassi

```
// This header is abbreviated.
// For a sample of a complete header, see API DynamoDB di basso livello.
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"}
```

Nome	Descrizione	Richiesto
`TableName`	Il nome della tabella che deve contenere l'elemento da eliminare. Tipo: string	Sì
`Item`	Una mappa degli attributi per l'elemento e deve includere i valori della chiave primaria che definiscono l'element o. È possibile fornire altre coppie nome-valore dell'attr ibuto per l'elemento. Per ulteriori informazioni sulle chiavi primarie, vedere Chiave primaria. Tipo: mappa dei nomi degli attributi ai valori degli attributi.	Sì
`Expected`	Designa un attributo per un put condizionale. Il parametro `Expected` consente di fornire un nome di attributo e se DynamoDB deve controllare o meno se il valore dell'attr ibuto esiste già o se esiste e ha un valore particolare prima di modificarlo. Tipo: mappa dei nomi di un attributo a un valore di attributo e se esiste.	No
`Expected:AttributeName`	Il nome dell'attributo per il put condizionale.	No

Nome	Descrizione	Richiesto
	Tipo: string	

Nome	Descrizione	Richiesto

`Expected:AttributeName: ExpectedAttributeValue`	Utilizza questo parametro per specificare se esiste già un valore per la coppia nome-valore dell'attributo.
	
	La seguente notazione JSON sostituisce l'elemento se l'attributo "Colore" non esiste già per quell'elemento:
	
	`"Expected" : {"Color":{"Exists":false}}`
	
	La seguente notazione JSON controlla se l'attributo con nome "Colore" ha un valore esistente pari a "Giallo" prima di sostituire l'elemento:
	
	`"Expected" : {"Color":{"Exists":true,{"Value":{"S":"Yellow"}}}}`
	
	Per impostazione predefinita, se si utilizza il parametro `Expected` e si fornisce un `Value`, DynamoDB presuppone che l'attributo esista e abbia un valore corrente da sostituire. Quindi non è necessario specificare `{"Exists":true}` perché è implicito. È possibile abbreviare la richiesta in:
		No

Nome	Descrizione	Richiesto
	`"Expected" : {"Color":{"Value":{"S":"Yellow"}}}` **Note** Se si specifica `{"Exists":true}` senza un valore di attributo da controllare, DynamoDB restituisce un errore.	
`ReturnValues`	Utilizza questo parametro se desideri ottenere le coppie nome-valore dell'attributo prima che vengano aggiornate con la richiesta `PutItem`. I valori possibili dei parametri sono `NONE` (predefinito) o `ALL_OLD`. Se è stato specificato `ALL_OLD` e `PutItem` ha sovrascritto una coppia nome-valore dell'attributo, viene restituito il contenuto del vecchio elemento. Se questo parametro non viene fornito o è `NONE`, non viene restituito nulla. Tipo: string	No

Risposte

Sintassi

Nella sintassi di esempio seguente si presuppone che la richiesta abbia specificato un parametro `ReturnValues` di `ALL_OLD`; in caso contrario, la risposta ha solo l'elemento `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
}
```

Nome	Descrizione
`Attributes`	I valori degli attributi prima dell'operazione put, ma solo se la proprietà `ReturnValues` viene specificata come `ALL_OLD` nella richiesta. Tipo: mappa di coppie nome-valore dell'attr ibuto.
`ConsumedCapacityUnits`	Il numero di unità di capacità di scrittura utilizzate dall'operazione. Questo valore mostra il numero applicato alla velocità effettiva assegnata. Per ulteriori informazioni, consulta Modalità con capacità allocata di DynamoDB. Tipo: numero

Errori speciali

Errore	Descrizione
`ConditionalCheckFailedException`	Controllo condizionale non riuscito. Il valore di attributo previsto non è stato trovato.
`ResourceNotFoundException`	L'elemento o l'attributo specificati non sono stati trovati.

Esempi

Per gli esempi di utilizzo dell'SDK AWS, consulta Utilizzo di elementi e attributi in DynamoDB.

Richiesta di esempio

```
// This header is abbreviated. For a sample of a complete header, see API DynamoDB di
 basso livello.
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"
}
```

Risposta di esempio

```
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
}
```

Operazioni correlate

- UpdateItem
- DeleteItem
- GetItem
- BatchGetItem

Query

> **Important**
>
> ***In questa sezione si fa riferimento alla versione API 2011-12-05, che è obsoleta e non deve essere utilizzata per le nuove applicazioni.***
> Per informazioni sull'API di basso livello corrente, consulta la Amazon DynamoDB API Reference.

Descrizione

Un'operazione `Query` ottiene i valori di uno o più elementi e i relativi attributi in base alla chiave primaria (`Query` è disponibile solo per le tabelle di chiavi primarie hash e di intervallo). È necessario fornire un `HashKeyValue` specifico ed è possibile restringere l'ambito della query utilizzando gli operatori di confronto sul `RangeKeyValue` della chiave primaria. Utilizza il parametro `ScanIndexForward` per ottenere risultati in ordine in avanti o all'indietro in base alla chiave di intervallo.

Le query che non restituiscono risultati consumano il numero minimo di unità di capacità di lettura in base al tipo di lettura.

> **Note**
>
> Se il numero totale di elementi che soddisfano i parametri della query supera il limite di 1 MB, la query si interrompe e i risultati vengono restituiti all'utente con un `LastEvaluatedKey` per continuare la query in un'operazione successiva. A differenza di un'operazione Scan, un'operazione Query non restituisce mai un set di risultati vuoto e un `LastEvaluatedKey`. `LastEvaluatedKey` viene fornito solo se i risultati superano 1 MB o se è stato utilizzato il parametro `Limit`.
> Il risultato può essere impostato per una lettura consistente utilizzando il parametro `ConsistentRead`.

Richieste

Sintassi

```
// This header is abbreviated.
// For a sample of a complete header, see API DynamoDB di basso livello.
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"]},
}
```

Nome	Descrizione	Richiesto
`TableName`	Il nome della tabella che contiene gli elementi richiesti.	Sì

Nome	Descrizione	Richiesto
	Tipo: string	
`AttributesToGet`	Matrice di nomi di attributi. Se i nomi degli attributi non sono specificati, verranno restituit i tutti gli attributi. Se alcuni attributi non vengono trovati, non verranno visualizzati nel risultato. Tipo: Array	No

Nome	Descrizione	Richiesto
`Limit`	Il numero massimo di elementi da restituire (non necessariamente il numero di elementi corrispondenti). Se DynamoDB elabora il numero di elementi fino al limite durante la query sulla tabella, interrompe la query e restituisce i valori corrispondenti fino a quel punto e un `LastEvaluatedKey` da applicare in un'operazione successiva per continuare la query. Inoltre, se la dimensione del set di risultati supera 1 MB prima che DynamoDB raggiunga questo limite, interrompe la query e restituisce i valori corrispondenti e un `LastEvaluatedKey` da applicare in un'operazione successiva per continuare la query. Tipo: numero	No
`ConsistentRead`	Se impostato su `true`, viene emessa una lettura consistente, altrimenti viene utilizzata la consistenza finale. Tipo: Booleano	No

Nome	Descrizione	Richiesto
`Count`	Se impostato su `true`, DynamoDB restituisce un numero totale di elementi che corrispondono ai parametri della query e non un elenco degli elementi corrispondenti e dei relativi attributi. È possibile applicare il parametro `Limit` alle query di solo conteggio. Non impostare `Count` su `true` mentre si fornisce un elenco di `AttributesToGet` ; in caso contrario , DynamoDB restituirà un errore di convalida. Per ulteriori informazioni, consulta Conteggio degli elementi nei risultati. Tipo: Booleano	No
`HashKeyValue`	Valore attributo del componente hash della chiave primaria composita. Tipo: String, Number o Binary	Sì

Nome	Descrizione	Richiesto
`RangeKeyCondition`	Un container per i valori degli attributi e gli operatori di confronto da utilizzare per la query. Una richiesta di query non richiede un `RangeKeyCondition` . Se si fornisce solo `HashKeyValue` , DynamoDB restituisce tutti gli elementi con il valore dell'elemento della chiave hash specificato. Tipo: Map	No
`RangeKeyCondition` : `AttributeValueList`	I valori degli attributi da valutare per i parametri della query. `AttributeValueList` contiene un valore di attributo, a meno che non sia specificato un confronto `BETWEEN`. Per il confronto `BETWEEN`, `AttributeValueList` contiene due valori di attributo. Tipo: una mappa di `AttributeValue` ad un `ComparisonOperator` .	No

Nome	Descrizione	Richiesto
`RangeKeyCondition : ComparisonOperator`	I criteri per valutare gli attributi forniti, come uguale, maggiore di, ecc. Di seguito sono riportati operatori di confronto validi per un'operazione Query. **Note** I confronti di valori String per maggiore di, uguale o minore di si basano sui valori del codice di caratteri ASCII. Ad esempio, `a` è maggiore di `A` e `aa` è maggiore di `B`. Per l'elenco dei valori di codifica, vedi http://en.wikipedia.org/wiki/ASCII#ASCII_printable_characters. Per Binary, DynamoDB tratta ciascun byte dei dati binari come non firmato quando confronta i valori binari, ad esempio quando valuta le espressioni di query. Tipo: String o Binary	No

Nome	Descrizione	Richiesto
	`EQ`: uguale. Per `EQ`, `AttributeValueList` può contenere solo un `AttributeValue` di tipo String, Number o Binary (non un set). Se un elemento contiene un `AttributeValue` di un tipo diverso da quello specificato nella richiesta, il valore non corrisponde. Ad esempio, `{"S":"6"}` non è uguale a `{"N":"6"}`. Inoltre, `{"N":"6"}` non è uguale a `{"NS":["6", "2", "1"]}`.	
	`LE`: minore o uguale a. Per `LE`, `AttributeValueList` può contenere solo un `AttributeValue` di tipo String, Number o Binary (non un set). Se un elemento contiene un `AttributeValue` di un tipo diverso da quello specificato nella richiesta, il valore non corrisponde. Ad esempio, `{"S":"6"}` non è uguale a `{"N":"6"}`. Inoltre, `{"N":"6"}` non si confronta con `{"NS":["6", "2", "1"]}`.	

Nome	Descrizione	Richiesto
	`LT`: minore di. Per `LT`, `AttributeValueList` può contenere solo un `AttributeValue` di tipo String, Number o Binary (non un set). Se un elemento contiene un `AttributeValue` di un tipo diverso da quello specificato nella richiesta, il valore non corrisponde. Ad esempio, `{"S":"6"}` non è uguale a `{"N":"6"}`. Inoltre, `{"N":"6"}` non si confronta con `{"NS":["6", "2", "1"]}`.	
	`GE`: maggiore o uguale a. Per `GE`, `AttributeValueList` può contenere solo un `AttributeValue` di tipo String, Number o Binary (non un set). Se un elemento contiene un `AttributeValue` di un tipo diverso da quello specificato nella richiesta, il valore non corrisponde. Ad esempio, `{"S":"6"}` non è uguale a `{"N":"6"}`. Inoltre, `{"N":"6"}` non si confronta con `{"NS":["6", "2", "1"]}`.	

Nome	Descrizione	Richiesto
	`GT`: maggiore di. Per `GT`, `AttributeValueList` può contenere solo un `AttributeValue` di tipo String, Number o Binary (non un set). Se un elemento contiene un `AttributeValue` di un tipo diverso da quello specificato nella richiesta, il valore non corrisponde. Ad esempio, `{"S":"6"}` non è uguale a `{"N":"6"}`. Inoltre, `{"N":"6"}` non si confronta con `{"NS":["6", "2", "1"]}`.	
	`BEGINS_WITH`: controlla la presenza di un prefisso. Per `BEGINS_WITH`, `AttributeValueList` può contenere solo un `AttributeValue` di tipo String o Binary (non Number o Set). L'attributo di destinazione del confronto deve essere String o Binary (non Number o un set).	

Nome	Descrizione	Richiesto
	`BETWEEN`: maggiore di o uguale a primo valore e minore di o uguale a secondo valore. Per `BETWEEN`, `AttributeValueList` deve contenere due elementi `AttributeValue` dello stesso tipo, String, Number o Binary (non un set). Un attributo di destinazione corrisponde se il valore di destinazione è maggiore o uguale al primo elemento e minore o uguale al secondo elemento. Se un elemento contiene un `AttributeValue` di un tipo diverso da quello specificato nella richiesta, il valore non corrisponde. Ad esempio, `{"S":"6"}` non si confronta con `{"N":"6"}` . Inoltre, `{"N":"6"}` non si confronta con `{"NS":["6", "2", "1"]}`.	

Nome	Descrizione	Richiesto
`ScanIndexForward`	Specifica l'attraversamento ascendente o discendente dell'indice. DynamoDB restituisce risultati che riflettono l'ordine richiesto determinato dalla chiave di intervallo: se il tipo di dati è Number, i risultati vengono restituiti in ordine numerico. In caso contrario, l'attraversamento si basa sui valori del codice di caratteri ASCII. Tipo: Booleano L'impostazione predefinita è `true` (ascendente).	No

Nome	Descrizione	Richiesto
`ExclusiveStartKey`	La chiave primaria dell'elemento da cui continuare una query precedente. Una query precedente potrebbe fornire questo valore come `LastEvaluatedKey` se l'operazione di query è stata interrotta prima di completare la query a causa della dimensione del set di risultati o del parametro `Limit`. `LastEvaluatedKey` può essere re-inviato in una nuova richiesta di query per continuare l'operazione da quel punto. Tipo: `HashKeyElement` , `HashKeyElement` e `RangeKeyElement` per una chiave primaria composita.	No

Risposte

Sintassi

```
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
}
```

Nome	Descrizione
`Items`	Attributi dell'elemento che soddisfano i parametri della query. Tipo: mappa di nomi degli attributi e dei relativi tipi di dati e valori.
`Count`	Il numero di elementi nella risposta. Per ulteriori informazioni, consulta Conteggio degli elementi nei risultati. Tipo: numero
`LastEvaluatedKey`	La chiave primaria dell'elemento per cui l'operazione di query è stata interrotta, incluso il set di risultati precedente. Utilizza questo valore per avviare una nuova operazione escludendo questo valore nella nuova richiesta. `LastEvaluatedKey` è `null` quando l'intero set di risultati della query è completo (cioè l'operazione ha elaborato l'ultima pagina). Tipo: `HashKeyElement` o `HashKeyElement` e `RangeKeyElement` per una chiave primaria composita.
`ConsumedCapacityUnits`	Il numero di unità di capacità di scrittura utilizzate dall'operazione. Questo valore mostra

Nome	Descrizione
	il numero applicato alla velocità effettiva assegnata. Per ulteriori informazioni, consulta Modalità con capacità allocata di DynamoDB. Tipo: numero

Errori speciali

Errore	Descrizione
`ResourceNotFoundException`	La tabella specificata non è stata trovata.

Esempi

Per gli esempi di utilizzo dell'SDK AWS, consulta Esecuzione di query in DynamoDB.

Richiesta di esempio

```
// This header is abbreviated. For a sample of a complete header, see API DynamoDB di
 basso livello.
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"}
 }
}
```

Risposta di esempio

```
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
}
```

Richiesta di esempio

```
// This header is abbreviated. For a sample of a complete header, see API DynamoDB di
 basso livello.
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}
```

Risposta di esempio

```
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
}
```

Operazioni correlate

- Scan

Scan

> **Important**
>
> ***In questa sezione si fa riferimento alla versione API 2011-12-05, che è obsoleta e non deve essere utilizzata per le nuove applicazioni.***
> Per informazioni sull'API di basso livello corrente, consulta la Amazon DynamoDB API Reference.

Descrizione

L'operazione `Scan` restituisce uno o più elementi e i relativi attributi eseguendo una scansione completa di una tabella. Fornire un `ScanFilter` per ottenere risultati più specifici.

> **Note**
>
> Se il numero totale di elementi sottoposti a scansione supera il limite di 1 MB, la scansione viene interrotta e i risultati vengono restituiti all'utente con un `LastEvaluatedKey` per continuare la scansione in un'operazione successiva. I risultati includono anche il numero di elementi che superano il limite. Una scansione può comportare l'assenza di dati di tabella che soddisfano i criteri di filtro.
> Il set di risultati è a consistenza finale.

Richieste

Sintassi

```
// This header is abbreviated.
// For a sample of a complete header, see API DynamoDB di basso livello.
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"]},
}
```

Nome	Descrizione	Richiesto
`TableName`	Il nome della tabella che contiene gli elementi richiesti. Tipo: string	Sì
`AttributesToGet`	Matrice di nomi di attributi. Se i nomi degli attributi non sono specificati, verranno restituit i tutti gli attributi. Se alcuni attributi non vengono trovati, non verranno visualizzati nel risultato. Tipo: Array	No

Nome	Descrizione	Richiesto
`Limit`	Il numero massimo di item da valutare (non necessariamente il numero di item corrispondenti). Se DynamoDB elabora il numero di elementi fino al limite durante l'elaborazione dei risultati, si arresta e restituisce i valori corrispondenti fino a quel punto e un `LastEvaluatedKey` da applicare in un'operazione successiva per continuare a recuperare gli elementi. Inoltre, se la dimensione del set di dati sottoposti a scansione supera 1 MB prima che DynamoDB raggiunga questo limite, interrompe la scansione e restituisce i valori corrispondenti fino al limite e un `LastEvaluatedKey` da applicare in un'operazione successiva per continuare la scansione. Tipo: numero	No

Nome	Descrizione	Richiesto
`Count`	Se impostato su `true`, DynamoDB restituisce un numero totale di elementi per l'operazione di scansione, anche se l'operazione non contiene elementi corrispon denti per il filtro assegnato . È possibile applicare il parametro Limit alle scansioni di solo conteggio. Non impostare `Count` su `true` mentre si fornisce un elenco di `Attribute sToGet` ; in caso contrario , DynamoDB restituirà un errore di convalida. Per ulteriori informazioni, consulta Conteggio degli elementi nei risultati. Tipo: Booleano	No
`ScanFilter`	Valuta i risultati della scansione e restituisce solo i valori desiderati. Le condizion i multiple sono trattate come operazioni "AND": per essere incluse nei risultati tutte le condizioni devono essere soddisfatte. Tipo: una mappa dei nomi degli attributi ai valori con operatori di confronto.	No

Nome	Descrizione	Richiesto
`ScanFilter` :`AttributeValueList`	I valori e le condizioni per valutare i risultati della scansione per il filtro. Tipo: una mappa di `AttributeValue` ad un `Condition` .	No

Nome	Descrizione	Richiesto
`ScanFilter : ComparisonOperator`	I criteri per valutare gli attributi forniti, come uguale, maggiore di, ecc. Di seguito sono riportati operatori di confronto validi per un'operazione di scansione. **Note** I confronti di valori String per maggiore di, uguale o minore di si basano sui valori del codice di caratteri ASCII. Ad esempio, `a` è maggiore di `A` e `aa` è maggiore di `B`. Per l'elenco dei valori di codifica, vedi http://en.wikipedia.org/wiki/ASCII#ASCII_printable_characters. Per Binary, DynamoDB tratta ciascun byte dei dati binari come non firmato quando confronta i valori binari, ad esempio quando valuta le espressioni di query. Tipo: String o Binary	No

Nome	Descrizione	Richiesto
	`EQ`: uguale. Per `EQ`, `AttributeValueList` può contenere solo un `AttributeValue` di tipo String, Number o Binary (non un set). Se un elemento contiene un `AttributeValue` di un tipo diverso da quello specificato nella richiesta, il valore non corrisponde. Ad esempio, `{"S":"6"}` non è uguale a `{"N":"6"}`. Inoltre, `{"N":"6"}` non è uguale a `{"NS":["6", "2", "1"]}`.	
	`NE`: non uguale. Per `NE`, `AttributeValueList` può contenere solo un `AttributeValue` di tipo String, Number o Binary (non un set). Se un elemento contiene un `AttributeValue` di un tipo diverso da quello specificato nella richiesta, il valore non corrisponde. Ad esempio, `{"S":"6"}` non è uguale a `{"N":"6"}`. Inoltre, `{"N":"6"}` non è uguale a `{"NS":["6", "2", "1"]}`.	

Nome	Descrizione	Richiesto
	`LE`: minore o uguale a. Per `LE`, `AttributeValueList` può contenere solo un `AttributeValue` di tipo String, Number o Binary (non un set). Se un elemento contiene un `AttributeValue` di un tipo diverso da quello specificato nella richiesta, il valore non corrisponde. Ad esempio, `{"S":"6"}` non è uguale a `{"N":"6"}`. Inoltre, `{"N":"6"}` non si confronta con `{"NS":["6", "2", "1"]}`.	
	`LT`: minore di. Per `LT`, `AttributeValueList` può contenere solo un `AttributeValue` di tipo String, Number o Binary (non un set). Se un elemento contiene un `AttributeValue` di un tipo diverso da quello specificato nella richiesta, il valore non corrisponde. Ad esempio, `{"S":"6"}` non è uguale a `{"N":"6"}`. Inoltre, `{"N":"6"}` non si confronta con `{"NS":["6", "2", "1"]}`.	

Nome	Descrizione	Richiesto
	`GE`: maggiore o uguale a. Per `GE`, `AttributeValueList` può contenere solo un `AttributeValue` di tipo String, Number o Binary (non un set). Se un elemento contiene un `AttributeValue` di un tipo diverso da quello specificato nella richiesta, il valore non corrisponde. Ad esempio, `{"S":"6"}` non è uguale a `{"N":"6"}`. Inoltre, `{"N":"6"}` non si confronta con `{"NS":["6", "2", "1"]}`.	
	`GT`: maggiore di. Per `GT`, `AttributeValueList` può contenere solo un `AttributeValue` di tipo String, Number o Binary (non un set). Se un elemento contiene un `AttributeValue` di un tipo diverso da quello specificato nella richiesta, il valore non corrisponde. Ad esempio, `{"S":"6"}` non è uguale a `{"N":"6"}`. Inoltre, `{"N":"6"}` non si confronta con `{"NS":["6", "2", "1"]}`.	

Nome	Descrizione	Richiesto
	`NOT_NULL`: l'attributo esiste.	
	`NULL`: l'attributo non esiste.	
	`CONTAINS`: controlla la presenza di una sottosequenza o di un valore in un set. Per `CONTAINS`, `AttributeValueList` può contenere solo un `AttributeValue` di tipo String, Number o Binary (non un set). Se l'attributo di destinazione del confronto è String, l'operazione verifica la presenza di una corrispondenza di sottostringa. Se l'attributo di destinazione del confronto è Binary, l'operazione cerca una sottosequenza della destinazione che corrisponde all'input. Se l'attributo di destinazione del confronto è un set ("SS", "NS" o "BS"), l'operazione verifica la presenza di un membro del set (non come sottostringa).	

Nome	Descrizione	Richiesto
	`NOT_CONTAINS` : controlla l'assenza di una sottosequenza o l'assenza di un valore in un set. Per `NOT_CONTAINS` , `AttributeValueList` può contenere solo un `AttributeValue` di tipo String, Number o Binary (non un set). Se l'attributo di destinazione del confronto è String, l'operazione verifica l'assenza di una corrispondenza di sottostringa. Se l'attributo di destinazione del confronto è Binary, l'operazione verifica l'assenza di una sottosequenza della destinazione che corrisponde all'input . Se l'attributo di destinazione del confronto è un set ("SS", "NS" o "BS"), l'operazione verifica l'assenza di un membro del set (non come sottostringa).	

Nome	Descrizione	Richiesto
	`BEGINS_WITH` : controlla la presenza di un prefisso. Per `BEGINS_WITH` , `AttributeValueList` può contenere solo un `AttributeValue` di tipo String o Binary (non Number o Set). L'attributo di destinazione del confronto deve essere String o Binary (non Number o un set).	
	`IN`: controlla le corrispondenze esatte. Per `IN`, `AttributeValueList` può contenere più di un `AttributeValue` di tipo String, Number o Binary (non un set). Per corrispondere, l'attributo di destinazione del confronto deve essere dello stesso tipo e valore esatto. Una valore stringa non corrisponde mai a un set di stringhe.	

Nome	Descrizione	Richiesto
	`BETWEEN`: maggiore di o uguale a primo valore e minore di o uguale a secondo valore. Per `BETWEEN`, `AttributeValueList` deve contenere due elementi `AttributeValue` dello stesso tipo, String, Number o Binary (non un set). Un attributo di destinazione corrisponde se il valore di destinazione è maggiore o uguale al primo elemento e minore o uguale al secondo elemento. Se un elemento contiene un `AttributeValue` di un tipo diverso da quello specificato nella richiesta, il valore non corrisponde. Ad esempio, `{"S":"6"}` non si confronta con `{"N":"6"}` . Inoltre, `{"N":"6"}` non si confronta con `{"NS":["6", "2", "1"]}`.	

Nome	Descrizione	Richiesto
`ExclusiveStartKey`	La chiave primaria dell'elemento da cui continuare una scansione precedente. Una scansione precedente potrebbe fornire questo valore se l'operazione di scansione è stata interrotta prima della scansione dell'intera tabella a causa della dimensione del set di risultati o del parametro `Limit`. `LastEvaluatedKey` può essere re-inviato in una nuova richiesta di scansione per continuare l'operazione da quel punto. Tipo: `HashKeyElement` o `HashKeyElement` e `RangeKeyElement` per una chiave primaria composita.	No

Risposte

Sintassi

```
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}
}
```

Nome	Descrizione
`Items`	Container per gli attributi che soddisfano i parametri dell'operazione. Tipo: mappa di nomi degli attributi e dei relativi tipi di dati e valori.
`Count`	Il numero di elementi nella risposta. Per ulteriori informazioni, consulta Conteggio degli elementi nei risultati. Tipo: numero
`ScannedCount`	Il numero di elementi nella scansione completa prima dell'applicazione di eventuali filtri. Un valore `ScannedCount` elevato con pochi o nessun risultato `Count` indica un'operazione Scan inefficiente. Per ulteriori informazioni, consulta Conteggio degli elementi nei risultati. Tipo: numero
`LastEvaluatedKey`	La chiave primaria dell'elemento su cui si è interrotta l'operazione di scansione. Fornire questo valore in una operazione di scansione successiva per continuare l'operazione da quel punto.

Nome	Descrizione
	`LastEvaluatedKey` è `null` quando l'intero set di risultati della scansione è completo (cioè l'operazione ha elaborato l'ultima pagina).
`ConsumedCapacityUnits`	Il numero di unità di capacità di scrittura utilizzate dall'operazione. Questo valore mostra il numero applicato alla velocità effettiva assegnata. Per ulteriori informazioni, consulta Modalità con capacità allocata di DynamoDB. Tipo: numero

Errori speciali

Errore	Descrizione
`ResourceNotFoundException`	La tabella specificata non è stata trovata.

Esempi

Per gli esempi di utilizzo dell'SDK AWS, consulta Scansione di tabelle in DynamoDB.

Richiesta di esempio

```
// This header is abbreviated. For a sample of a complete header, see API DynamoDB di
 basso livello.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.Scan
content-type: application/x-amz-json-1.0

{"TableName":"1-hash-rangetable","ScanFilter":{}}
```

Risposta di esempio

```
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}
```

Richiesta di esempio

```
// This header is abbreviated. For a sample of a complete header, see API DynamoDB di
 basso livello.
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"}
 }
}
```

Risposta di esempio

```
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
}
```

Richiesta di esempio

```
// This header is abbreviated. For a sample of a complete header, see API DynamoDB di
 basso livello.
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"}}
}
```

Risposta di esempio

```
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
}
```

Operazioni correlate

- Query
- BatchGetItem

UpdateItem

> **Important**
>
> ***In questa sezione si fa riferimento alla versione API 2011-12-05, che è obsoleta e non deve essere utilizzata per le nuove applicazioni.***
>
> Per informazioni sull'API di basso livello corrente, consulta la Amazon DynamoDB API Reference.

Descrizione

Modifica gli attributi di un elemento esistente. È possibile eseguire un aggiornamento condizionale (inserire una nuova coppia nome-valore attributo se non esiste o sostituire una coppia nome-valore esistente se ha determinati valori di attributo previsti).

> **Note**
>
> Non è possibile aggiornare gli attributi della chiave primaria utilizzando UpdateItem. Eliminare invece l'elemento e utilizzare PutItem per creare un nuovo elemento con nuovi attributi.

L'operazione UpdateItem include un parametro `Action`che definisce come eseguire l'aggiornamento. È possibile inserire, eliminare o aggiungere valori di attributo.

I valori degli attributi non possono essere nulli; gli attributi stringa e di tipo binario devono avere lunghezze maggiori di zero e gli attributi di tipo impostato non devono essere vuoti. Le richieste con valori vuoti verranno rifiutate con una eccezione `ValidationException`.

Se un elemento esistente ha una chiave primaria specificata:

- PUT: aggiunge l'attributo specificato. Se l'attributo esiste, viene sostituito dal nuovo valore.
- DELETE: se non viene specificato alcun valore, rimuove l'attributo e il relativo valore. Se viene specificato un set di valori, i valori nel set specificato vengono rimossi dal set precedente. Quindi, se il valore dell'attributo contiene [a, b, c] e l'azione di eliminazione contiene [a, c], il valore dell'attributo finale sarà [b]. Il tipo di valore specificato deve corrispondere al tipo di valore esistente. La specifica di un set vuoto non è consentita.
- ADD: utilizzare l'azione di aggiunta solo per i numeri o se l'attributo di destinazione è un set (inclusi i set di stringhe). ADD non funziona se l'attributo di destinazione è un singolo valore stringa o un valore binario scalare. Il valore specificato viene aggiunto a un valore numerico (incrementando o diminuendo il valore numerico esistente) o aggiunto come valore aggiuntivo in un set di stringhe. Se viene specificato un set di valori, i valori vengono aggiunti al set esistente. Ad esempio, se il set originale è [1,2] e il valore fornito è [3], dopo l'operazione di aggiunta l'insieme sarà [1,2,3] e non [4,5]. Se viene specificata un'operazione ADD per un attributo set e il tipo di attributo specificato non corrisponde al tipo di set esistente, si verifica un errore.

  Se si utilizza ADD per un attributo che non esiste, l'attributo e i relativi valori verranno aggiunti all'elemento.

Se nessun elemento corrisponde alla chiave primaria specificata:

- PUT: crea un nuovo elemento con la chiave primaria specificata. Quindi aggiunge l'attributo specificato.
- DELETE: non succede nulla.

- ADD: viene creato un elemento con la chiave primaria e il numero (o insieme di numeri) forniti per il valore dell'attributo. Non valido per un tipo stringa o binario.

> **Note**
>
> Se si utilizza ADD per aumentare o diminuire un valore numerico per un elemento che non esiste prima dell'aggiornamento, DynamoDB utilizzerà `0` come valore iniziale. Inoltre, se si aggiorna un elemento tramite ADD per aumentare o diminuire un valore numerico per un attributo che non esiste prima dell'aggiornamento, DynamoDB utilizzerà `0` come valore iniziale. Ad esempio, si supponga di utilizzare ADD per aggiungere `+3` a un attributo che non esisteva prima dell'aggiornamento. DynamoDB utilizzerà `0` come valore iniziale e il valore dopo l'aggiornamento sarà `3`.

Per ulteriori informazioni sull'utilizzo di questa operazione, consulta Utilizzo di elementi e attributi in DynamoDB.

Richieste

Sintassi

```
// This header is abbreviated.
// For a sample of a complete header, see API DynamoDB di basso livello.
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"
}
```

Nome	Descrizione	Richiesto
`TableName`	Il nome della tabella che contiene l'elemento da aggiornare. Tipo: string	Sì
`Key`	La chiave primaria che definisce l'elemento. Per ulteriori informazioni sulle chiavi primarie, vedere Chiave primaria. Tipo: mappa di `HashKeyElement` sul suo valore e `RangeKeyElement` sul suo valore.	Sì
`AttributeUpdates`	Mappa del nome dell'attributo al nuovo valore e operazione per l'aggiornamento. I nomi degli attributi specificano gli attributi da modificare e non possono contenere attributi di chiave primaria. Tipo: mappa del nome dell'attributo, del valore e dell'operazione per l'aggiornamento dell'attributo.	
`AttributeUpdates :Action`	Specifica come eseguire l'aggiornamento. Valori possibili `PUT` (predefinito), `ADD` o `DELETE`. La sintassi è spiegata nella descrizione di UpdateItem.	No

Nome	Descrizione	Richiesto
	Tipo: string Impostazione predefinita: PUT	
`Expected`	Designa un attributo per un aggiornamento condizion ale. Il parametro `Expected` consente di fornire un nome di attributo e se DynamoDB deve controllare o meno se il valore dell'attributo esiste già o se esiste e ha un valore particola re prima di modificarlo. Tipo: mappa dei nomi degli attributi.	No
`Expected:AttributeName`	Il nome dell'attributo per il put condizionale. Tipo: string	No

Nome	Descrizione	Richiesto

`Expected:AttributeName: ExpectedAttributeValue`	Utilizza questo parametro per specificare se esiste già un valore per la coppia nome-valore dell'attributo.
	
	La seguente notazione JSON aggiorna l'elemento se l'attributo "Colore" non esiste già per quell'elemento:
	
	`"Expected" : {"Color":{"Exists":false}}`
	
	La seguente notazione JSON controlla se l'attributo con nome "Colore" ha un valore esistente pari a "Giallo" prima di aggiornare l'elemento:
	
	`"Expected" : {"Color":{"Exists":true},{"Value": {"S":"Yellow"}}}`
	
	Per impostazione predefinita, se si utilizza il parametro `Expected` e si fornisce un `Value`, DynamoDB presuppone che l'attributo esista e abbia un valore corrente da sostituire. Quindi non è necessario specificare`{"Exists":true}` perché è implicito. È possibile abbreviare la richiesta in:
		No

Nome	Descrizione	Richiesto
	`"Expected" : {"Color":{"Value":{"S":"Yellow"}}}` **Note** Se si specifica `{"Exists":true}` senza un valore di attributo da controllare, DynamoDB restituisce un errore.	

Nome	Descrizione	Richiesto
`ReturnValues`	Utilizza questo parametro se desideri ottenere le coppie nome-valore dell'attributo prima che vengano aggiornate con la richiesta `UpdateItem`. I valori dei parametri possibili sono `NONE` (predefinito) o `ALL_OLD`, `UPDATED_OLD`, `ALL_NEW` o `UPDATED_NEW`. Se è stato specificato `ALL_OLD` e `UpdateItem` ha sovrascritto una coppia nome-valore dell'attributo, viene restituito il contenuto del vecchio elemento. Se questo parametro non viene fornito o è `NONE`, non viene restituito nulla. Se viene specificato `ALL_NEW`, vengono restituiti tutti gli attributi della nuova versione dell'elemento. Se viene specificato `UPDATED_NEW`, vengono restituite tutte le versioni dei soli attributi aggiornati. Tipo: string	No

Risposte

Sintassi

Nella sintassi di esempio seguente si presuppone che la richiesta abbia specificato un parametro `ReturnValues` di `ALL_OLD`; in caso contrario, la risposta ha solo l'elemento `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
}
```

Nome	Descrizione
`Attributes`	Una mappa di coppie nome-valore attributo, ma solo se il parametro `ReturnValues` viene specificato come qualcosa di diverso da `NONE` nella richiesta. Tipo: mappa di coppie nome-valore dell'attr ibuto.
`ConsumedCapacityUnits`	Il numero di unità di capacità di scrittura utilizzate dall'operazione. Questo valore mostra il numero applicato alla velocità effettiva assegnata. Per ulteriori informazioni, consulta Modalità con capacità allocata di DynamoDB. Tipo: numero

Errori speciali

Errore	Descrizione
`ConditionalCheckFailedException`	Controllo condizionale non riuscito. Il valore dell'attributo ("+ name +") è ("+ value +") ma era previsto ("+ expValue +")
`ResourceNotFoundExceptions`	L'elemento o l'attributo specificati non sono stati trovati.

Esempi

Per gli esempi di utilizzo dell'SDK AWS, consulta Utilizzo di elementi e attributi in DynamoDB.

Richiesta di esempio

```
// This header is abbreviated. For a sample of a complete header, see API DynamoDB di
 basso livello.
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"
}
```

Risposta di esempio

```
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
}
```

Operazioni correlate

- PutItem
- DeleteItem

UpdateTable

> **Important**
>
> ***In questa sezione si fa riferimento alla versione API 2011-12-05, che è obsoleta e non deve essere utilizzata per le nuove applicazioni.***
> Per informazioni sull'API di basso livello corrente, consulta la Amazon DynamoDB API Reference.

Descrizione

Aggiorna la velocità effettiva assegnata per la tabella specificata. L'impostazione della velocità effettiva per una tabella consente di gestire le prestazioni e fa parte della caratteristica di velocità effettiva assegnata di DynamoDB. Per ulteriori informazioni, consulta Modalità con capacità allocata di DynamoDB.

I valori della velocità effettiva assegnata possono essere aggiornati o declassati in base ai valori massimi e minimi elencati in Quote in Amazon DynamoDB.

Perché questa operazione abbia esito positivo, la tabella deve trovarsi nello stato `ACTIVE`. UpdateTable è un'operazione asincrona; durante l'esecuzione dell'operazione, la tabella si trova nello stato `UPDATING`. Mentre la tabella si trova nello stato `UPDATING`, la tabella ha ancora la velocità effettiva assegnata prima della chiamata. La nuova impostazione di velocità effettiva assegnata è valida solo quando la tabella ritorna allo stato `ACTIVE`dopo l'operazione UpdateTable.

Richieste

Sintassi

```
// This header is abbreviated.
// For a sample of a complete header, see API DynamoDB di basso livello.
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}
}
```

Nome	Descrizione	Richiesto
`TableName`	Il nome della tabella da aggiornare. Tipo: string	Sì
`ProvisionedThroughput`	La nuova velocità effettiva per la tabella specificata, costituita dai valori per `ReadCapacityUnits` e `WriteCapacityUnits`. Consultare Modalità con capacità allocata di DynamoDB. Tipo: Array	Sì
`ProvisionedThroughput :ReadCapacityUnits`	Imposta il numero minimo di `ReadCapacityUnits` con consistenza consumate al secondo per la tabella specificata prima che DynamoDB bilanci il carico con altre operazioni.	Sì

Nome	Descrizione	Richiesto
	Le operazioni di lettura a consistenza finale richiedono meno sforzo rispetto a un'operazione di lettura consistente, quindi un'impostazione di 50 `ReadCapacityUnits` consistente al secondo fornisce 100 `ReadCapacityUnits` a consistenza finale al secondo. Tipo: numero	
`ProvisionedThroughput:WriteCapacityUnits`	Imposta il numero minimo di `WriteCapacityUnits` consumate al secondo per la tabella specificata prima che DynamoDB bilanci il carico con altre operazioni. Tipo: numero	Sì

Risposte

Sintassi

```
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"}}
```

Nome	Descrizione
`CreationDateTime`	La data in cui è stata creata la tabella. Tipo: numero
`KeySchema`	La struttura della chiave primaria (semplice o composita) per la tabella. Una coppia nome-valore per `HashKeyElement` è obbligatoria mentre una coppia nome-valore per `RangeKeyElement` è facoltativa (richiesta solo per le chiavi primarie composite). La dimensione massima della chiave hash è 2048 byte. La dimensione massima della chiave di intervallo è 1024 byte. Entrambi i limiti sono applicati separatamente (cioè è possibile avere una chiave hash + intervallo combinato, ovvero 2048+1024). Per ulteriori informazioni sulle chiavi primarie, vedere Chiave primaria. Tipo: mappa di `HashKeyElement` , oppure `HashKeyElement` e `RangeKeyElement` per una chiave primaria composita.
`ProvisionedThroughput`	Impostazioni del throughput corrente per la tabella specificata, inclusi i valori per `LastIncreaseDateTime` (se applicabile), `LastDecreaseDateTime` (se applicabile), Tipo: Array

Nome	Descrizione
`TableName`	Il nome della tabella aggiornata. Tipo: string
`TableStatus`	Lo stato corrente della tabella (`CREATING`, `ACTIVE`, `DELETING` o `UPDATING`), che dovrebbe essere `UPDATING`. Utilizza l'operazione DescribeTables per controllare lo stato della tabella. Tipo: string

Errori speciali

Errore	Descrizione
`ResourceNotFoundException`	La tabella specificata non è stata trovata.
`ResourceInUseException`	La tabella non si trova nello stato `ACTIVE`.

Esempi

Richiesta di esempio

```
// This header is abbreviated.
// For a sample of a complete header, see API DynamoDB di basso livello.
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}
}
```

Risposta di esempio

```
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"}
}
```

Operazioni correlate

- CreateTable
- DescribeTables
- DeleteTable

Parametri condizionali legacy di DynamoDB

In questo documento si fornisce una panoramica dei parametri condizionali legacy in DynamoDB e si consiglia di utilizzare invece i nuovi parametri di espressione. Vengono forniti dettagli su parametri come AttributesToGet, AttributeUpdates, ConditionalOperator, Expected, KeyConditions, QueryFilter e ScanFilter, nonché esempi di come utilizzare i nuovi parametri di espressione come sostituti.

> **Important**
>
> Consigliamo di utilizzare i nuovi parametri di espressione piuttosto che i parametri precedenti, se possibile. Per ulteriori informazioni, consulta Utilizzo di espressioni in DynamoDB.

Inoltre, DynamoDB non consente l'uso misto di parametri condizionali legacy e parametri di espressione in una singola chiamata. Ad esempio, il richiamo dell'operazione `Query` con `AttributesToGet` e `ConditionExpression` restituirà un errore.

Nella tabella seguente vengono illustrate le operazioni API DynamoDB che supportano ancora questi parametri legacy e i parametri di espressione da utilizzare al loro posto. Questa tabella può essere utile se si sta valutando l'aggiornamento delle applicazioni in modo che utilizzino invece i parametri di espressione.

Se utilizzi questa operazione API…	Con questi parametri legacy...	Utilizza invece questo parametro di espressione
`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`
	`Expected`	`ConditionExpression`

Nelle seguenti sezioni vengono fornite ulteriori informazioni sui parametri condizionali legacy.

Argomenti

- AttributesToGet (legacy)
- AttributeUpdates (legacy)
- ConditionalOperator (legacy)
- Expected (legacy)
- KeyConditions (legacy)
- QueryFilter (legacy)
- ScanFilter (legacy)
- Scrittura di condizioni con parametri legacy

AttributesToGet (legacy)

> **Note**
>
> Consigliamo di utilizzare i nuovi parametri di espressione piuttosto che i parametri precedenti, se possibile. Per ulteriori informazioni, consulta Utilizzo di espressioni in DynamoDB. Per informazioni specifiche sul nuovo parametro che sostituisce questo, consulta Utilizza invece ProjectionExpression.

Il parametro condizionale legacy `AttributesToGet` è un array di uno o più attributi da recuperare da DynamoDB. Se non viene specificato alcun nome di attributo, verranno restituiti tutti gli attributi. Se alcuni degli attributi richiesti non vengono trovati, non appariranno nel risultato.

`AttributesToGet` consente di recuperare gli attributi di tipo List o Map, tuttavia, non può recuperare singoli elementi.

Tenere presente che `AttributesToGet` non ha alcun effetto sul consumo di velocità effettiva assegnata. DynamoDB determina le unità di capacità consumate in base alla dimensione dell'elemento, non alla quantità di dati restituiti a un'applicazione.

Utilizza invece ProjectionExpression: esempio

Si supponga di voler recuperare un elemento dalla tabella Music, ma che siano restituiti solo alcuni degli attributi. È possibile utilizzare una richiesta `GetItem` con un parametro `AttributesToGet`, come in questo esempio 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"}
    }'
```

Puoi invece utilizzare `ProjectionExpression`:

```
aws dynamodb get-item \
    --table-name Music \
    --projection-expression "Artist, Genre" \
    --key '{
        "Artist": {"S":"No One You Know"},
        "SongTitle": {"S":"Call Me Today"}
    }'
```

AttributeUpdates (legacy)

> **Note**
>
> Consigliamo di utilizzare i nuovi parametri di espressione piuttosto che i parametri precedenti, se possibile. Per ulteriori informazioni, consulta Utilizzo di espressioni in DynamoDB. Per informazioni specifiche sul nuovo parametro che sostituisce questo, consulta Utilizza invece UpdateExpression..

In un'operazione `UpdateItem`, parametro condizionale legacy `AttributeUpdates` contiene i nomi degli attributi da modificare, l'operazione da eseguire su ognuno di essi e il nuovo valore per ciascuno. Se si sta aggiornando un attributo che è un attributo chiave di indice per qualsiasi indice in tale tabella, il tipo di attributo deve corrispondere al tipo di chiave di indice definito in `AttributesDefinition` della descrizione della tabella. È possibile utilizzare `UpdateItem` per aggiornare eventuali attributi non chiave.

I valori degli attributi non possono essere nulli. Gli attributi di tipo String e Binary devono avere lunghezze maggiori di zero. Gli attributi di tipo Set non devono essere vuoti. Le richieste con valori vuoti verranno rifiutate con una eccezione `ValidationException`.

Ogni elemento `AttributeUpdates` è costituito da un nome di attributo da modificare, insieme a quanto segue:

- `Value`: il nuovo valore, se applicabile, per questo attributo.
- `Action`: un valore che specifica come eseguire l'aggiornamento. Questa operazione è valida solo per un attributo esistente il cui tipo di dati è Number o Set; non utilizzare `ADD` per altri tipi di dati.

  Se nella tabella viene trovato un elemento con la chiave primaria specificata, i valori seguenti eseguono le operazioni riportate:

  - `PUT`: aggiunge l'attributo specificato all'elemento. Se l'attributo esiste già, viene sostituito dal nuovo valore.
  - `DELETE`: rimuove l'attributo e il relativo valore se non viene specificato alcun valore per `DELETE`. Il tipo di dati per il valore specificato deve corrispondere al tipo di dati del valore esistente.

    Se viene specificato un set di valori, tali valori saranno sottratti dal vecchio set. Ad esempio, se il valore dell'attributo era il set `[a,b,c]` e l'operazione `DELETE` specifica `[a,c]`, il valore dell'attributo finale sarà `[b]`. La specifica di un set vuoto restituisce un errore.
  - `ADD`: aggiunge il valore specificato all'elemento, se l'attributo non esiste già. Se l'attributo esiste, il comportamento di `ADD` dipende dal tipo di dati dell'attributo:
    - Se l'attributo è un numero e anche `Value` è un numero, allora `Value` viene aggiunto matematicamente all'attributo esistente. Se `Value` è un numero negativo, viene sottratto dall'attributo esistente.

      > **Note**
      >
      > Se si utilizza `ADD` per aumentare o diminuire un valore numerico per un elemento che non esiste prima dell'aggiornamento, DynamoDB utilizzerà 0 come valore iniziale. Allo stesso modo, se si utilizza `ADD` per un elemento esistente per aumentare o diminuire un valore di attributo che non esiste prima dell'aggiornamento, DynamoDB utilizzerà `0` come valore iniziale. Si supponga, ad esempio, che l'elemento che desideri aggiornare non abbia un attributo denominato itemcount, ma decidi comunque di `ADD` il numero `3` a questo attributo. DynamoDB creerà l'attributo itemcount, imposterà il suo valore iniziale su `0` e infine aggiungerà `3` ad esso. Il risultato sarà un nuovo attributo itemcount, con un valore `3`.
    - Se il tipo di dati esistente è un set e se anche `Value` è un set, allora `Value` viene aggiunto al set esistente. Ad esempio, se il valore dell'attributo è il set `[1,2]` e l'operazione `ADD` specifica `[3]`, il valore dell'attributo finale sarà `[1,2,3]`. Se viene specificata un'operazione `ADD` per un

attributo Set e il tipo di attributo specificato non corrisponde al tipo di set esistente, si verifica un errore.

Entrambi i set devono avere lo stesso tipo di dati primitivi. Ad esempio, se il tipo di dati esistente è un set di stringhe, anche `Value` deve essere un set di stringhe.

Se nella tabella non viene trovato alcun elemento con la chiave specificata, i valori seguenti completano le operazioni riportate:

- `PUT`: consente a DynamoDB di creare un nuovo elemento con la chiave primaria specificata e quindi aggiunge l'attributo.
- `DELETE`: non succede nulla, perché gli attributi non possono essere eliminati da un elemento inesistente. L'operazione ha esito positivo, ma DynamoDB non crea un nuovo elemento.
- `ADD`: fa in modo che DynamoDB crei un elemento con la chiave primaria fornita e il numero (o set di numeri) per il valore dell'attributo. Gli unici tipi di dati consentiti sono Number e Number Set.

Se si forniscono attributi che fanno parte di una chiave di indice, i tipi di dati per tali attributi devono corrispondere a quelli dello schema nella definizione dell'attributo della tabella.

Utilizza invece UpdateExpression: esempio

Si supponga di voler modificare un elemento nella tabella Music. È possibile utilizzare una richiesta `UpdateItem` con un parametro `AttributeUpdates`, come in questo esempio 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"}
        }
    }'
```

Puoi invece utilizzare `UpdateExpression`:

```
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 (legacy)

> **Note**
>
> Consigliamo di utilizzare i nuovi parametri di espressione piuttosto che i parametri precedenti, se possibile. Per ulteriori informazioni, consulta Utilizzo di espressioni in DynamoDB.

Il parametro condizionale legacy `ConditionalOperator` è un operatore logico utilizzato per applicare le condizioni in una mappa `Expected`, `QueryFilter` o `ScanFilter`:

- AND: se tutte le condizioni restituiscono true, l'intera mappa restituisce true.
- OR: se almeno una delle condizioni restituisce true, l'intera mappa restituisce true.

Se si omette `ConditionalOperator`, allora AND è il valore predefinito.

L'operazione avrà esito positivo solo se l'intera mappa restituisce true.

> **Note**
>
> Questo parametro non supporta gli attributi di tipo List o Map.

Expected (legacy)

> **Note**
>
> Consigliamo di utilizzare i nuovi parametri di espressione piuttosto che i parametri precedenti, se possibile. Per ulteriori informazioni, consulta Utilizzo di espressioni in DynamoDB. Per informazioni specifiche sul nuovo parametro che sostituisce questo, consulta Utilizza invece ConditionExpression..

Il parametro condizionale legacy `Expected` è un blocco condizionale per un'operazione `UpdateItem`. `Expected` è una mappa di coppie attributo/condizione. Ogni elemento della mappa è costituito da un nome di attributo, un operatore di confronto e uno o più valori. DynamoDB confronta l'attributo con i valori forniti tramite l'operatore di confronto. Per ogni elemento `Expected`, il risultato della valutazione è true o false.

Se si specifica più di un elemento nella mappa `Expected`, allora, per impostazione predefinita, tutte le condizioni devono restituire true. In altre parole, le condizioni vengono combinate utilizzando l'operatore `AND`. È possibile utilizzare il parametro `ConditionalOperator` per applicare l'operatore OR alle condizioni. In questo caso, almeno una delle condizioni deve restituire true, se non tutte.

Se la mappa `Expected` restituisce true, l'operazione condizionale ha esito positivo, altrimenti ha esito negativo.

`Expected` contiene i seguenti dati:

- `AttributeValueList`: uno o più valori da valutare rispetto all'attributo fornito. Il numero di valori nell'elenco dipende dal `ComparisonOperator` utilizzato.

  Per il tipo Number, i confronti dei valori sono numerici.

  I confronti di valori String per maggiore di, uguale o minore di sono basati su Unicode con codifica binaria UTF-8. Ad esempio, `a` è maggiore di `A` e `a` è maggiore di `B`.

  Per il tipo Binary, quando confronta i valori binari DynamoDB tratta ogni byte dei dati binari come non firmato.
- `ComparisonOperator`: un comparatore per valutare gli attributi in `AttributeValueList`. Quando si esegue il confronto, DynamoDB utilizza letture fortemente consistenti.

  Sono disponibili i seguenti operatori di confronto:

```
EQ | NE | LE | LT | GE | GT | NOT_NULL | NULL | CONTAINS | NOT_CONTAINS |
BEGINS_WITH | IN | BETWEEN
```

Di seguito sono riportate le descrizioni di ogni operatore di confronto.

- EQ: uguale a. EQ è supportato per tutti i tipi di dati, inclusi elenchi e mappe.

  AttributeValueList può contenere solo un elemento AttributeValue di tipo String, Number, Binary, String Set, Number Set o Binary Set. Se un elemento contiene un elemento AttributeValue di un tipo diverso da quello specificato nella richiesta, il valore non corrisponde. Ad esempio, {"S":"6"} non è uguale a {"N":"6"}. Inoltre, {"N":"6"} non è uguale a {"NS":["6", "2", "1"]}.

- NE: diverso da. NE è supportato per tutti i tipi di dati, inclusi elenchi e mappe.

  AttributeValueList può contenere solo un elemento AttributeValue di tipo String, Number, Binary, String Set, Number Set o Binary Set. Se un elemento contiene un AttributeValue di un tipo diverso da quello specificato nella richiesta, il valore non corrisponde. Ad esempio, {"S":"6"} non è uguale a {"N":"6"}. Inoltre, {"N":"6"} non è uguale a {"NS":["6", "2", "1"]}.

- LE: minore o uguale a.

  AttributeValueList può contenere solo un elemento AttributeValue di tipo String, Number o Binary (non un tipo Set). Se un elemento contiene un elemento AttributeValue di un tipo diverso da quello specificato nella richiesta, il valore non corrisponde. Ad esempio, {"S":"6"} non è uguale a {"N":"6"}. Inoltre, {"N":"6"} non si confronta con {"NS":["6", "2", "1"]}.

- LT: minore di.

  AttributeValueList può contenere solo un AttributeValue di tipo String, Number o Binary (non un tipo Set). Se un elemento contiene un elemento AttributeValue di un tipo diverso da quello specificato nella richiesta, il valore non corrisponde. Ad esempio, {"S":"6"} non è uguale a {"N":"6"}. Inoltre, {"N":"6"} non si confronta con {"NS":["6", "2", "1"]}.

- GE: maggiore o uguale a.

  AttributeValueList può contenere solo un elemento AttributeValue di tipo String, Number o Binary (non un tipo Set). Se un elemento contiene un elemento AttributeValue di un tipo diverso da quello specificato nella richiesta, il valore non corrisponde. Ad esempio,

`{"S":"6"}` non è uguale a `{"N":"6"}`. Inoltre, `{"N":"6"}` non si confronta con `{"NS": ["6", "2", "1"]}`.

- `GT`: maggiore di.

  `AttributeValueList` può contenere solo un elemento `AttributeValue` di tipo String, Number o Binary (non un tipo Set). Se un elemento contiene un elemento `AttributeValue` di un tipo diverso da quello specificato nella richiesta, il valore non corrisponde. Ad esempio, `{"S":"6"}` non è uguale a `{"N":"6"}`. Inoltre, `{"N":"6"}` non si confronta con `{"NS": ["6", "2", "1"]}`.

- `NOT_NULL`: l'attributo esiste. `NOT_NULL` è supportato per tutti i tipi di dati, inclusi elenchi e mappe.

  > **Note**
  >
  > Questo operatore verifica l'esistenza di un attributo, non il suo tipo di dati. Se il tipo di dati dell'attributo "a" è nullo e lo si valuta usando `NOT_NULL`, il risultato è un `true` booleano. Questo risultato è perché l'attributo "a" esiste; il suo tipo di dati non è rilevante per l'operatore di confronto `NOT_NULL`.

- `NULL`: l'attributo non esiste. `NULL` è supportato per tutti i tipi di dati, inclusi elenchi e mappe.

  > **Note**
  >
  > Questo operatore verifica la non esistenza di un attributo, non il relativo tipo di dati. Se il tipo di dati dell'attributo "a" è nullo e lo si valuta usando `NULL`, il risultato è un `false` booleano. Questo è perché l'attributo "a" esiste; il suo tipo di dati non è rilevante per l'operatore di confronto `NULL`.

- `CONTAINS`: controlla la presenza di una sottosequenza o di un valore in un set.

  `AttributeValueList` può contenere solo un elemento `AttributeValue` di tipo String, Number o Binary (non un tipo Set). Se l'attributo di destinazione del confronto è di tipo String, l'operatore verifica la presenza di una corrispondenza di sottostringa. Se l'attributo di destinazione del confronto è di tipo Binary, l'operatore cerca una sottosequenza della destinazione che corrisponde all'input. Se l'attributo di destinazione del confronto è un set "SS", "NS" o "BS"), l'operatore restituisce true se trova una corrispondenza esatta con qualsiasi membro del set.

CONTAINES è supportato per gli elenchi: quando si valuta "a CONTAINS b", "a" può essere un List; tuttavia "b" non può essere set, Map o List.

- NOT_CONTAINS: controlla l'assenza di una sottosequenza o l'assenza di un valore in un set.

  AttributeValueList può contenere solo un elemento AttributeValue di tipo String, Number o Binary (non un tipo Set). Se l'attributo di destinazione del confronto è String, l'operatore verifica l'assenza di una corrispondenza di sottostringa. Se l'attributo di destinazione del confronto è Binary, l'operatore verifica l'assenza di una sottosequenza della destinazione che corrisponde all'input. Se l'attributo di destinazione del confronto è un set ("SS", "NS" o "BS"), l'operatore restituisce true se does not trova una corrispondenza esatta con qualsiasi membro del set.

  NOT_CONTAINS è supportato per gli elenchi: quando si valuta "a NOT CONTAINS b", "a" può essere un List; tuttavia "b" non può essere set, Map o List.

- BEGINS_WITH: controlla la presenza di un prefisso.

  AttributeValueList può contenere solo un AttributeValue di tipo String o Binary (non un tipo Number o Set). L'attributo di destinazione del confronto deve essere String o Binary (non un tipo Number o Set).

- IN: verifica la presenza di elementi corrispondenti all'interno di due set.

  AttributeValueList può contenere uno o più elementi AttributeValue di tipo String, Number o Binary (non un tipo Set). Questi attributi vengono confrontati con un attributo di tipo Set esistente di un elemento. Se nell'attributo dell'elemento sono presenti elementi del set di input, l'espressione restituisce true.

- BETWEEN: maggiore di o uguale a primo valore e minore di o uguale a secondo valore.

  AttributeValueList deve contenere due elementiAttributeValue dello stesso tipo, String, Number o Binary (non un tipo Set). Un attributo di destinazione corrisponde se il valore di destinazione è maggiore o uguale al primo elemento e minore o uguale al secondo elemento. Se un elemento contiene un elemento AttributeValue di un tipo diverso da quello specificato nella richiesta, il valore non corrisponde. Ad esempio, {"S":"6"} non si confronta con {"N":"6"}. Inoltre, {"N":"6"} non si confronta con {"NS":["6", "2", "1"]}

I seguenti parametri possono essere usati al posto di AttributeValueList e ComparisonOperator:

- `Value`: un valore per DynamoDB da confrontare con un attributo.
- `Exists`: un valore booleano che fa sì che DynamoDB valuti il valore prima di provare l'operazione condizionale:
  - Se `Exists` è `true`, DynamoDB controllerà se tale valore dell'attributo esiste già nella tabella. Se viene trovato, allora la condizione restituisce true, altrimenti restituisce false.
  - Se `Exists` è `false`, DynamoDB presuppone che il valore dell'attributo `not` esiste nella tabella. Se davvero il valore non esiste, allora l'ipotesi è valida e la condizione restituisce true. Se il valore viene trovato nonostante il presupposto che non esiste, la condizione restituisce false.

  Si noti che il valore predefinito per `Exists` è `true`.

I parametri `Value` e `Exists` sono incompatibili con `AttributeValueList` e `ComparisonOperator`. Si noti che se si utilizzano entrambi i set di parametri contemporaneamente, DynamoDB restituirà una eccezione `ValidationException`.

> **Note**
>
> Questo parametro non supporta gli attributi di tipo List o Map.

Utilizza invece ConditionExpression: esempio

Si supponga di voler modificare un elemento nella tabella Music, ma solo se una certa condizione è vera. È possibile utilizzare una richiesta `UpdateItem` con un parametro `Expected`, come in questo esempio 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"} ]
        }
    }'
```

Puoi invece utilizzare `ConditionExpression`:

```
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 (legacy)

> **Note**
>
> Consigliamo di utilizzare i nuovi parametri di espressione piuttosto che i parametri precedenti, se possibile. Per ulteriori informazioni, consulta Utilizzo di espressioni in DynamoDB. Per informazioni specifiche sul nuovo parametro che sostituisce questo, consulta Utilizza invece KeyConditionExpression..

Il parametro condizionale legacy `KeyConditions` contiene i criteri di selezione per un'operazione `Query`. Per una query su una tabella, è possibile disporre di condizioni solo sugli attributi della chiave primaria della tabella. È necessario specificare il nome e il valore della chiave di partizione come condizione `EQ`. Facoltativamente, è possibile fornire una seconda condizione che fa riferimento alla chiave di ordinamento.

> **Note**
>
> Se non si specifica una condizione della chiave di ordinamento, verranno recuperati tutti gli elementi che corrispondono alla chiave di partizione. Se è presente `FilterExpression` o `QueryFilter`, verrà applicato dopo che gli elementi sono stati recuperati.

Per una query su un indice, è possibile disporre di condizioni solo sugli attributi della chiave di indice. È necessario specificare il nome e il valore della chiave di partizione dell'indice come condizione `EQ`. Facoltativamente, è possibile fornire una seconda condizione che fa riferimento alla chiave di ordinamento dell'indice.

Ogni elemento `KeyConditions` è costituito da un nome di attributo da confrontare, insieme a quanto segue:

- `AttributeValueList`: uno o più valori da valutare rispetto all'attributo fornito. Il numero di valori nell'elenco dipende dal `ComparisonOperator` utilizzato.

  Per il tipo Number, i confronti dei valori sono numerici.

  I confronti di valori String per maggiore di, uguale o minore di sono basati su Unicode con codifica binaria UTF-8. Ad esempio, `a` è maggiore di `A` e `a` è maggiore di `B`.

  Per il tipo Binary, quando confronta i valori binari DynamoDB tratta ogni byte dei dati binari come non firmato.
- `ComparisonOperator`: un comparatore per valutare gli attributi. Ad esempio, uguale a, maggiore di e minore di.

  Per `KeyConditions`, sono supportati solo i seguenti operatori di confronto:

  `EQ | LE | LT | GE | GT | BEGINS_WITH | BETWEEN`

  Di seguito sono riportate le descrizioni di questi operatori di confronto.

  - `EQ`: uguale.

    `AttributeValueList` può contenere solo un `AttributeValue` di tipo String, Number o Binary (non un tipo Set). Se un elemento contiene un elemento `AttributeValue` di un tipo diverso da quello specificato nella richiesta, il valore non corrisponde. Ad esempio, `{"S":"6"}` non è uguale a `{"N":"6"}`. Inoltre, `{"N":"6"}` non è uguale a `{"NS":["6", "2", "1"]}`.

- `LE`: minore o uguale a.

  `AttributeValueList` può contenere solo un elemento `AttributeValue` di tipo String, Number o Binary (non un tipo Set). Se un elemento contiene un elemento `AttributeValue` di un tipo diverso da quello specificato nella richiesta, il valore non corrisponde. Ad esempio, `{"S":"6"}` non è uguale a `{"N":"6"}`. Inoltre, `{"N":"6"}` non si confronta con `{"NS":["6", "2", "1"]}`.

- `LT`: minore di.

  `AttributeValueList` può contenere solo un `AttributeValue` di tipo String, Number o Binary (non un tipo Set). Se un elemento contiene un elemento `AttributeValue` di un tipo diverso da quello specificato nella richiesta, il valore non corrisponde. Ad esempio, `{"S":"6"}` non è uguale a `{"N":"6"}`. Inoltre, `{"N":"6"}` non si confronta con `{"NS":["6", "2", "1"]}`.

- `GE`: maggiore o uguale a.

  `AttributeValueList` può contenere solo un elemento `AttributeValue` di tipo String, Number o Binary (non un tipo Set). Se un elemento contiene un elemento `AttributeValue` di un tipo diverso da quello specificato nella richiesta, il valore non corrisponde. Ad esempio, `{"S":"6"}` non è uguale a `{"N":"6"}`. Inoltre, `{"N":"6"}` non si confronta con `{"NS":["6", "2", "1"]}`.

- `GT`: maggiore di.

  `AttributeValueList` può contenere solo un elemento `AttributeValue` di tipo String, Number o Binary (non un tipo Set). Se un elemento contiene un elemento `AttributeValue` di un tipo diverso da quello specificato nella richiesta, il valore non corrisponde. Ad esempio, `{"S":"6"}` non è uguale a `{"N":"6"}`. Inoltre, `{"N":"6"}` non si confronta con `{"NS":["6", "2", "1"]}`.

- `BEGINS_WITH`: controlla la presenza di un prefisso.

  `AttributeValueList` può contenere solo un `AttributeValue` di tipo String o Binary (non un tipo Number o Set). L'attributo di destinazione del confronto deve essere String o Binary (non un tipo Number o Set).

- `BETWEEN`: maggiore di o uguale a primo valore e minore di o uguale a secondo valore.

  `AttributeValueList` deve contenere due elementi`AttributeValue` dello stesso tipo, String, Number o Binary (non un tipo Set). Un attributo di destinazione corrisponde se il valore

di destinazione è maggiore o uguale al primo elemento e minore o uguale al secondo elemento. Se un elemento contiene un elemento `AttributeValue` di un tipo diverso da quello specificato nella richiesta, il valore non corrisponde. Ad esempio, `{"S":"6"}` non si confronta con `{"N":"6"}`. Inoltre, `{"N":"6"}` non si confronta con `{"NS":["6", "2", "1"]}`.

Utilizza invece KeyConditionExpression: esempio

Si supponga di voler recuperare diversi elementi con la stessa chiave di partizione dalla tabella Music. È possibile utilizzare una richiesta `Query` con un parametro `KeyConditions`, come in questo esempio 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"} ]
        }
    }'
```

Puoi invece utilizzare `KeyConditionExpression`:

```
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 (legacy)

> **Note**
>
> Consigliamo di utilizzare i nuovi parametri di espressione piuttosto che i parametri precedenti, se possibile. Per ulteriori informazioni, consulta Utilizzo di espressioni in DynamoDB. Per informazioni specifiche sul nuovo parametro che sostituisce questo, consulta Utilizza invece FilterExpression..

In un'operazione `Query`, parametro condizionale legacy `QueryFilter` è una condizione che valuta i risultati delle query dopo la lettura degli elementi e restituisce solo i valori desiderati.

Questo parametro non supporta gli attributi di tipo List o Map.

> **Note**
>
> `QueryFilter` viene applicato dopo che gli elementi sono già stati letti; il processo di filtraggio non consuma alcuna unità di capacità di lettura aggiuntiva.

Se si fornisce più di una condizione nella mappa `QueryFilter`, allora, per impostazione predefinita, tutte le condizioni devono restituire true. In altre parole, le condizioni vengono combinate utilizzando l'operatore `AND`. È possibile utilizzare il parametro ConditionalOperator (legacy) per applicare l'operatore OR alle condizioni. In questo caso, almeno una delle condizioni deve restituire true, se non tutte.

Tenere presente che `QueryFilter` non consente gli attributi chiave. Non è possibile definire una condizione di filtro su una chiave di partizione o una chiave di ordinamento.

Ogni elemento `QueryFilter` è costituito da un nome di attributo da confrontare, insieme a quanto segue:

- `AttributeValueList`: uno o più valori da valutare rispetto all'attributo fornito. Il numero di valori in List dipende dall'operatore specificato in `ComparisonOperator`.

  Per il tipo Number, i confronti dei valori sono numerici.

  I confronti di valori String per maggiore di, uguale o minore di sono basati sulla codifica binaria UTF-8. Ad esempio, `a` è maggiore di `A` e `a` è maggiore di `B`.

Per il tipo Binary, quando confronta i valori binari DynamoDB tratta ogni byte dei dati binari come non firmato.

Per informazioni su come specificare i tipi di dati in JSON, consulta API DynamoDB di basso livello.

- `ComparisonOperator`: un comparatore per valutare gli attributi. Ad esempio, uguale a, maggiore di e minore di.

  Sono disponibili i seguenti operatori di confronto:

  `EQ | NE | LE | LT | GE | GT | NOT_NULL | NULL | CONTAINS | NOT_CONTAINS | BEGINS_WITH | IN | BETWEEN`

Utilizza invece FilterExpression: esempio

Si supponga di voler eseguire una query sulla tabella Music e di applicare una condizione agli elementi corrispondenti. È possibile utilizzare una richiesta `Query` con un parametro `QueryFilter`, come in questo esempio 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"} ]
        }
    }'
```

Puoi invece utilizzare `FilterExpression`:

```
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 (legacy)

> **Note**
>
> Consigliamo di utilizzare i nuovi parametri di espressione piuttosto che i parametri precedenti, se possibile. Per ulteriori informazioni, consulta Utilizzo di espressioni in DynamoDB. Per informazioni specifiche sul nuovo parametro che sostituisce questo, consulta Utilizza invece FilterExpression..

In una operazione `Scan`, il parametro condizionale legacy `ScanFilter` è una condizione che valuta i risultati della scansione e restituisce solo i valori desiderati.

> **Note**
>
> Questo parametro non supporta gli attributi di tipo List o Map.

Se si specifica più di una condizione nella mappa `ScanFilter`, allora, per impostazione predefinita, tutte le condizioni devono restituire true. In altre parole, le condizioni sono unite da AND. È possibile utilizzare il parametro ConditionalOperator (legacy) per applicare l'operatore OR alle condizioni. In questo caso, almeno una delle condizioni deve restituire true, se non tutte.

Ogni elemento `ScanFilter` è costituito da un nome di attributo da confrontare, insieme a quanto segue:

- `AttributeValueList`: uno o più valori da valutare rispetto all'attributo fornito. Il numero di valori in List dipende dall'operatore specificato in `ComparisonOperator`.

  Per il tipo Number, i confronti dei valori sono numerici.

  I confronti di valori String per maggiore di, uguale o minore di sono basati sulla codifica binaria UTF-8. Ad esempio, `a` è maggiore di `A` e `a` è maggiore di `B`.

  Per il tipo Binary, quando confronta i valori binari DynamoDB tratta ogni byte dei dati binari come non firmato.

Per informazioni su come specificare i tipi di dati in JSON, consulta API DynamoDB di basso livello.

- `ComparisonOperator`: un comparatore per valutare gli attributi. Ad esempio, uguale a, maggiore di e minore di.

  Sono disponibili i seguenti operatori di confronto:

  `EQ | NE | LE | LT | GE | GT | NOT_NULL | NULL | CONTAINS | NOT_CONTAINS | BEGINS_WITH | IN | BETWEEN`

Utilizza invece FilterExpression: esempio

Supponiamo di voler eseguire la scansione della tabella Music e di applicare una condizione agli elementi corrispondenti. È possibile utilizzare una richiesta `Scan` con un parametro `ScanFilter`, come in questo esempio AWS CLI:

```
aws dynamodb scan \
    --table-name Music \
    --scan-filter '{
        "Genre":{
            "AttributeValueList":[ {"S":"Rock"} ],
            "ComparisonOperator": "EQ"
        }
    }'
```

Puoi invece utilizzare `FilterExpression`:

```
aws dynamodb scan \
    --table-name Music \
    --filter-expression 'Genre = :g' \
    --expression-attribute-values '{
        ":g": {"S":"Rock"}
    }'
```

Scrittura di condizioni con parametri legacy

> **Note**
>
> Consigliamo di utilizzare i nuovi parametri di espressione piuttosto che i parametri precedenti, se possibile. Per ulteriori informazioni, consulta Utilizzo di espressioni in DynamoDB.

Nella sezione seguente viene descritto come scrivere le condizioni da utilizzare con i parametri legacy, ad esempio `Expected`, `QueryFilter` e `ScanFilter`.

> **Note**
>
> Le nuove applicazioni dovrebbero invece utilizzare i parametri di espressione. Per ulteriori informazioni, consulta Utilizzo di espressioni in DynamoDB.

Condizioni semplici

Con i valori degli attributi, è possibile scrivere condizioni per i confronti rispetto agli attributi della tabella. Una condizione restituisce sempre true o false ed è costituita da:

- `ComparisonOperator`: maggiore di, minore di, uguale a e così via.
- `AttributeValueList` (facoltativo): i valori degli attributi da confrontare. A seconda di `ComparisonOperator` utilizzato, `AttributeValueList` potrebbe contenere uno, due o più valori; o potrebbe non essere affatto presente.

Nelle sezioni seguenti sono descritti i vari operatori di confronto e sono riportati gli esempi di come utilizzarli nelle condizioni.

Operatori di confronto senza valori di attributo

- `NOT_NULL`: true se esiste un attributo.
- `NULL`: true se non esiste un attributo.

Utilizza questi operatori per verificare se un attributo esiste o non esiste. Poiché non vi è alcun valore da confrontare, non specificare `AttributeValueList`.

Esempio

La seguente espressione restituisce true se l'attributo Dimensions esiste.

```
...
    "Dimensions": {
         ComparisonOperator: "NOT_NULL"
    }
...
```

Operatori di confronto con un valore di attributo

- `EQ`: true se un attributo è uguale a un valore.

  `AttributeValueList` può contenere solo un valore di tipo String, Number, Binary, String Set, Number Set o Binary Set. Se un elemento contiene un valore di un tipo diverso da quello specificato nella richiesta, il valore non corrisponde. Ad esempio, la stringa `"3"` non è uguale al numero `3`. Inoltre, il numero `3` non è uguale al set di numeri `[3, 2, 1]`.
- `NE`: true se un attributo non è uguale a un valore.

  `AttributeValueList` può contenere solo un valore di tipo String, Number, Binary, String Set, Number Set o Binary Set. Se un elemento contiene un valore di un tipo diverso da quello specificato nella richiesta, il valore non corrisponde.
- `LE`: true se un attributo è minore o uguale a un valore.

  `AttributeValueList` può contenere solo un valore di tipo String, Number o Binary (non un tipo Set). Se un elemento contiene un `AttributeValue` di un tipo diverso da quello specificato nella richiesta, il valore non corrisponde.
- `LT`: true se un attributo è minore di un valore.

  `AttributeValueList` può contenere solo un valore di tipo String, Number o Binary (non un tipo Set). Se un elemento contiene un valore di un tipo diverso da quello specificato nella richiesta, il valore non corrisponde.
- `GE`: true se un attributo è maggiore o uguale a un valore.

  `AttributeValueList` può contenere solo un valore di tipo String, Number o Binary (non un tipo Set). Se un elemento contiene un valore di un tipo diverso da quello specificato nella richiesta, il valore non corrisponde.
- `GT`: true se un attributo è maggiore di un valore.

`AttributeValueList` può contenere solo un valore di tipo String, Number o Binary (non un tipo Set). Se un elemento contiene un valore di un tipo diverso da quello specificato nella richiesta, il valore non corrisponde.

- `CONTAINS`: true se un valore è presente all'interno di un set o se un valore ne contiene un altro.

  `AttributeValueList` può contenere solo un valore di tipo String, Number o Binary (non un tipo Set). Se l'attributo di destinazione del confronto è di tipo String, l'operatore controlla la presenza di una corrispondenza di sottostringa. Se l'attributo di destinazione del confronto è di tipo Binary, l'operatore cerca una sottosequenza della destinazione che corrisponde all'input. Se l'attributo di destinazione del confronto è di tipo Set, l'operatore restituisce true se trova una corrispondenza esatta con qualsiasi membro del set.

- `NOT_CONTAINS`: true se un valore non è presente all'interno di un set o se un valore ne contiene un altro.

  `AttributeValueList` può contenere solo un valore di tipo String, Number o Binary (non un tipo Set). Se l'attributo di destinazione del confronto è String, l'operatore verifica l'assenza di una corrispondenza di sottostringa. Se l'attributo di destinazione del confronto è Binary, l'operatore verifica l'assenza di una sottosequenza della destinazione che corrisponde all'input. Se l'attributo di destinazione del confronto è di tipo Set, l'operatore restituisce true se non trova una corrispondenza esatta con qualsiasi membro del set.

- `BEGINS_WITH`: true se i primi caratteri di un attributo corrispondono al valore fornito. Non utilizzare questo operatore per confrontare i numeri.

  `AttributeValueList` può contenere solo un valore di tipo String o Binary (non un tipo Number o Set). L'attributo di destinazione del confronto deve essere String o Binary (non Number o un set).

Utilizza questi operatori per confrontare un attributo con un valore. È necessario specificare un `AttributeValueList` costituito da un valore singolo. Per la maggior parte degli operatori, questo valore deve essere uno scalare; tuttavia, gli operatori `EQ` e `NE` supportano anche i set.

Examples (Esempi)

Le seguenti espressioni restituiscono true se:

- Il prezzo di un prodotto è maggiore di 100.

```
...
```

```
    "Price": {
        ComparisonOperator: "GT",
        AttributeValueList: [ {"N":"100"} ]
    }
...
```

- Una categoria di prodotti inizia con "Bo".

```
...
    "ProductCategory": {
        ComparisonOperator: "BEGINS_WITH",
        AttributeValueList: [ {"S":"Bo"} ]
    }
...
```

- Un prodotto è disponibile in rosso, verde o nero:

```
...
    "Color": {
        ComparisonOperator: "EQ",
        AttributeValueList: [
            [ {"S":"Black"}, {"S":"Red"}, {"S":"Green"} ]
        ]
    }
...
```

> **Note**
>
> Quando si confrontano i tipi di dati Set, l'ordine degli elementi non ha importanza. DynamoDB restituirà solo gli elementi con lo stesso set di valori, indipendentemente dall'ordine in cui vengono specificati nella richiesta.

Operatori di confronto con due valori di attributo

- `BETWEEN`: true se un valore è compreso tra un limite inferiore e un limite superiore, endpoint inclusi.

  `AttributeValueList` deve contenere due elementi dello stesso tipo, String, Number o Binary (non un tipo Set). Un attributo di destinazione corrisponde se il valore di destinazione è maggiore

o uguale al primo elemento e minore o uguale al secondo elemento. Se un elemento contiene un valore di un tipo diverso da quello specificato nella richiesta, il valore non corrisponde.

Utilizza questo operatore per determinare se un valore di attributo è compreso in un intervallo. `AttributeValueList` deve contenere due elementi scalari dello stesso tipo: String, Number o Binary.

Esempio

L'espressione seguente restituisce true se il prezzo di un prodotto è compreso tra 100 e 200.

```
...
    "Price": {
        ComparisonOperator: "BETWEEN",
        AttributeValueList: [ {"N":"100"}, {"N":"200"} ]
    }
...
```

Operatori di confronto con N valori di attributo

- `IN`: true se un valore è uguale a uno qualsiasi dei valori in un elenco enumerato. Nell'elenco sono supportati solo i valori scalari, non i set. Per corrispondere, l'attributo di destinazione deve essere dello stesso tipo e valore esatto.

  `AttributeValueList` può contenere uno o più elementi di tipo String, Number o Binary (non un tipo Set). Questi attributi vengono confrontati con un attributo di tipo non Set esistente di un elemento. Se nell'attributo dell'elemento è presente qualsiasi elemento del set di input, l'espressione restituisce true.

  `AttributeValueList` può contenere uno o più valori di tipo String, Number o Binary (non un tipo Set). Per corrispondere, l'attributo di destinazione del confronto deve essere dello stesso tipo e valore esatto. Una valore stringa non corrisponde mai a un set di stringhe.

Utilizza questo operatore per determinare se il valore fornito è incluso in un elenco numerato. È possibile specificare un numero qualsiasi di valori scalari in `AttributeValueList`, ma devono essere tutti dello stesso tipo di dati.

Esempio

L'espressione seguente restituisce true se il valore per Id è 201, 203 o 205.

```
...
    "Id": {
        ComparisonOperator: "IN",
        AttributeValueList: [ {"N":"201"}, {"N":"203"}, {"N":"205"} ]
    }
...
```

Utilizzo di più condizioni

DynamoDB consente di combinare più condizioni per formare espressioni complesse. Questa operazione viene eseguita fornendo almeno due espressioni, con un ConditionalOperator (legacy) facoltativo.

Per impostazione predefinita, quando si specifica più di una condizione, tutte le condizioni devono restituire true affinché l'intera espressione restituisca true. In altre parole, si verifica un'operazione AND implicita.

Esempio

L'espressione seguente restituisce true se un prodotto è un libro che ha almeno 600 pagine. Entrambe le condizioni devono restituire true, poiché sono implicitamente collegate con un operatore AND.

```
...
    "ProductCategory": {
        ComparisonOperator: "EQ",
        AttributeValueList: [ {"S":"Book"} ]
    },
    "PageCount": {
        ComparisonOperator: "GE",
        AttributeValueList: [ {"N":600} ]
    }
...
```

È possibile utilizzare ConditionalOperator (legacy) per chiarire che avrà luogo un'operazione AND. L'esempio seguente si comporta allo stesso modo del precedente.

```
...
    "ConditionalOperator" : "AND",
    "ProductCategory": {
```

```
        "ComparisonOperator": "EQ",
        "AttributeValueList": [ {"N":"Book"} ]
    },
    "PageCount": {
        "ComparisonOperator": "GE",
        "AttributeValueList": [ {"N":600"} ]
    }
...
```

È possibile impostare `ConditionalOperator` anche su OR, il che significa che almeno una delle condizioni deve restituire true.

Esempio

La seguente espressione restituisce true se un prodotto è una mountain bike, se si tratta di un marchio particolare o se il suo prezzo è maggiore di 100.

```
...
    ConditionalOperator : "OR",
    "BicycleType": {
        "ComparisonOperator": "EQ",
        "AttributeValueList": [ {"S":"Mountain" ]
    },
    "Brand": {
        "ComparisonOperator": "EQ",
        "AttributeValueList": [ {"S":"Brand-Company A" ]
    },
    "Price": {
        "ComparisonOperator": "GT",
        "AttributeValueList": [ {"N":"100"} ]
    }
...
```

> **Note**
>
> In un'espressione complessa, le condizioni vengono elaborate in ordine, dalla prima all'ultima condizione.
>
> Non è possibile utilizzare sia AND che OR in una singola espressione.

Altri operatori condizionali

Nelle versioni precedenti di DynamoDB, il parametro `Expected` si comportava in modo diverso per le scritture condizionali. Ogni elemento nella mappa `Expected` rappresentava un nome di attributo per il controllo di DynamoDB, insieme a quanto segue:

- `Value`: un valore da confrontare con l'attributo.
- `Exists`: determina se il valore esiste prima di provare l'operazione.

Le opzioni `Value` e `Exists` continuano a essere supportate in DynamoDB; tuttavia, consentono solo di verificare una condizione di uguaglianza o se esiste un attributo. Consigliamo di utilizzare invece `ComparisonOperator` e `AttributeValueList`, perché queste opzioni consentono di costruire una gamma molto più ampia di condizioni.

Example

`DeleteItem` può verificare se un libro non è più in pubblicazione ed eliminarlo solo se questa condizione è true. Ecco un esempio AWS CLI che utilizza una condizione legacy:

```
aws dynamodb delete-item \
    --table-name ProductCatalog \
    --key '{
        "Id": {"N":"600"}
    }' \
    --expected '{
        "InPublication": {
            "Exists": true,
            "Value": {"BOOL":false}
        }
    }'
```

L'esempio seguente fa la stessa cosa, ma non usa una condizione legacy:

```
aws dynamodb delete-item \
    --table-name ProductCatalog \
    --key '{
        "Id": {"N":"600"}
    }' \
```

```
    --expected '{
        "InPublication": {
            "ComparisonOperator": "EQ",
            "AttributeValueList": [ {"BOOL":false} ]
        }
    }'
```

Example

Una operazione `PutItem` può proteggere dalla sovrascrittura di un elemento esistente con gli stessi attributi della chiave primaria. Ecco un esempio che utilizza una condizione legacy:

```
aws dynamodb put-item \
    --table-name ProductCatalog \
    --item '{
      "Id": {"N":"500"},
        "Title": {"S":"Book 500 Title"}
    }' \
    --expected '{
        "Id": { "Exists": false }
    }'
```

L'esempio seguente fa la stessa cosa, ma non usa una condizione legacy:

```
aws dynamodb put-item \
    --table-name ProductCatalog \
    --item '{
      "Id": {"N":"500"},
        "Title": {"S":"Book 500 Title"}
    }' \
    --expected '{
        "Id": { "ComparisonOperator": "NULL" }
    }'
```

> **Note**
>
> Per le condizioni nella mappa `Expected`, non utilizzare le opzioni `Value` e `Exists` legacy insieme a `ComparisonOperator` e `AttributeValueList`. Se si utilizzano, la scrittura condizionale non avrà esito positivo.

Le traduzioni sono generate tramite traduzione automatica. In caso di conflitto tra il contenuto di una traduzione e la versione originale in Inglese, quest'ultima prevarrà.