Using materialized views in Amazon Redshift

AWS Prescriptive Guidance



Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

AWS Prescriptive Guidance: Using materialized views in Amazon Redshift

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

Services or capabilities described in AWS documentation might vary by Region. To see the differences applicable to the AWS European Sovereign Cloud Region, see the AWS European Sovereign Cloud User Guide.

Table of Contents

Introduction **1**
Overview 1
Targeted business outcomes 1
Understanding materialized views **2**
Views 2
Materialized views 2
Comparison of view types 3
Creating materialized views **5**
Querying materialized views **7**
Refreshing materialized views **8**
Automatic query rewriting **10**
How query rewriting works 10
Advantages of materialized views **12**
Resources **13**
Document history **14**

Using materialized views in Amazon Redshift

Ethan Stark, Kelly Ragan, and Srinivasan Krishnasamy, Amazon Web Services (AWS)

December 2022 (document history)

Overview

Data warehouse applications often require you to perform complex queries on large tables. Processing these queries can be time-consuming and expensive. For example, when you run a query in Amazon Redshift, the leader node parses, validates, plans, optimizes, and runs your query. This process can increase your opportunity costs and actual costs by using up a significant amount of wall-clock time, CPU time, and memory.

This guide shows you how to use materialized views in Amazon Redshift to speed up queries, especially predictable and frequently repeated queries. Materialized views reduce query time by storing a precomputed result set, so that you don't have to directly access underlying base tables. Reduced query times, in turn, can reduce the cost of query processing so that you can cost-effectively scale your application. This guide is intended for data engineers, data architects, and data analysts.

Targeted business outcomes

You can use this guide to achieve the following business outcomes:

- Reduce the cost of processing Amazon Redshift queries
- Grow your application efficiently and cost-effectively
- Free up users to spend time on higher-value tasks

Understanding materialized views

Views

A view is a virtual table that's based on the result set of a SQL SELECT query. The virtual table contains the data retrieved from the query expression, but the result isn't stored on disk. When you use views, you always get the most up-to-date data because the query pulls the data from the original tables every single time that you run the query. You can create a view from one or more base tables or views. You can also query a view the same way that you query the original base tables.

The following example query shows how to create a view:

```
CREATE VIEW tickets_view
AS
   select e.eventname,
          sum(s.price) as total_sales
   from sales s
   join event e
       on e.eventid = s.eventid
   group by e.eventname;
```

The following example query shows how to query a view:

```
select eventname,
       total_sales
from ticket_view
where eventname = 'Gotterburg';
```

Materialized views

A materialized view is a database object that contains the results of a query. For example, a materialized view can be any of the following:

- A local copy of data located remotely
- A subset of the rows or columns of a table or join result

- A summary using an aggregate function

Comparison of view types

The following table summarizes the differences between a view and materialized view.

Key	View	Materialized view
Definition	A virtual table that doesn't store any data, but instead runs a defined SQL query to get data from one or more tables in a database	A virtual table that's defined by an editable SQL query, but the result of the query gets stored on disk
Storage	Result of the query expressio n is not stored on disk—only the query expression is stored on disk	Query expression and the result of the query expression both stored on disk
Run	The query that defines the view runs every time the view is referenced in a query	The result of the query is stored on disk, and the query expression doesn't run every time a user tries to fetch the data from a materialized view
Data recency	Always provides the latest updated value from the base tables	Doesn't provide the latest updated value if that value gets changed in the database
Cost	No storage cost	Has a storage cost
Design	To create a standard view, you must: · Have access to the underlyin g tables · Use a standard SELECT statement	To create a materialized view, you must: · Have access to the underlyin g tables · Use a standard SELECT statement

		Optionally, you can specify the following: · Whether the materialized view is included in automated and manual cluster snapshots , which are stored in Amazon Simple Storage Service (Amazon S3) · How the data in the materialized view is distribut ed and sorted · Whether the materialized view should be automatic ally refreshed with the latest changes from its base tables
Usage	When data is accessed or updated infrequently	When data is accessed or updated frequently

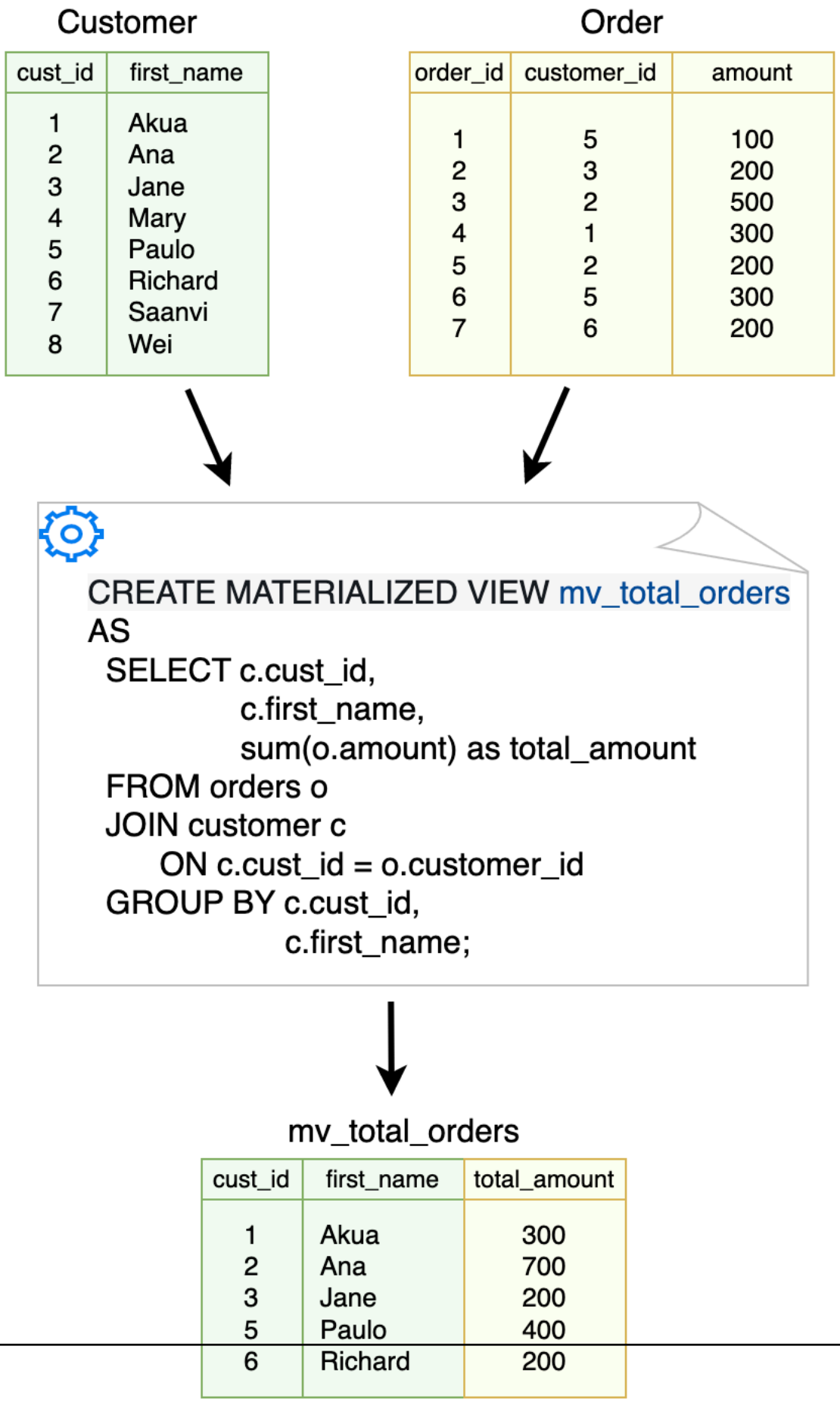
Creating materialized views

You can create a materialized view based on one or more of the following:

- Amazon Redshift tables
- External tables that you can create by using Amazon Redshift Spectrum or a federated query
- Other materialized views

When you create a materialized view, Amazon Redshift runs the user-specified SQL statement to gather the data from the base table or tables. Then, Amazon Redshift stores the result set.

The following illustration provides an overview of a materialized view called `mv_total_orders`. This view is defined by a SQL query that uses two base tables: customer and order.


Customer
cust_id
first_name
1 Akua
2 Ana
3 Jane
4 Mary
5 Paulo
6 Richard
7 Saanvi
8 Wei
Order
order_id
customer_id
amount
1 5 100
2 3 200
3 2 500
4 1 300
5 2 200
6 5 300
7 6 200
CREATE MATERIALIZED VIEW mv_total_orders
AS
SELECT c.cust_id,
c.first_name,
sum(o.amount) as total_amount
FROM orders o
JOIN customer c
ON c.cust_id = o.customer_id
GROUP BY c.cust_id,
c.first_name;
mv_total_orders
cust_id
first_name
total_amount
1 Akua 300
2 Ana 700
3 Jane 200
5 Paulo 400
6 Richard 200

Querying materialized views

When you query a materialized view, you directly access the precomputed data in the materialized view. You can use a materialized view in any SQL query by referencing the materialized view name as the data source, as in a table or standard view.

For example, consider the `mv_total_orders` materialized view example illustration from the Creating materialized views section of this guide. If you want to build a query for `mv_total_orders` (which returns a list of customers who have orders totaling more than $500), then you could run the following standard query:

```
statement. SELECT c.cust_id,
       c.first_name,
       sum(o.amount) as total_amount
FROM orders o
JOIN customer c
  ON c.cust_id = o.customer_id
GROUP BY c.cust_id,
         c.first_name
HAVING sum(o.amount) > 500;
```

However, the preceding query isn't optimized for speed. We recommend that you run the following query instead:

```
SELECT cust_id,
       first_name,
       total_amount
FROM mv_total_orders
WHERE total_amount > 500;
```

The recommended query runs much faster because the query results are precomputed, and there's no need to access the underlying tables (customer and order). Amazon Redshift can return the results directly from `mv_total_orders`.

> **Important**
>
> When a query accesses a materialized view, the query sees only the data stored in the materialized view as of its most recent refresh. Therefore, the query might not see all the latest changes from the corresponding base tables of the materialized view.

Refreshing materialized views

A materialized view contains a snapshot of the query result. Materialized views are not updated periodically, unless you configure Amazon Redshift to make periodic updates. To manually refresh and update the data in a materialized view, you can use the `REFRESH MATERIALIZED VIEW` statement at any time. This command identifies changes that take place in the base tables and applies those changes to the materialized view.

There are two ways to refresh a materialized view: a manual refresh and an automatic refresh (called autorefreshing). The following example query shows how to manually refresh a materialized view:

```
REFRESH MATERIALIZED VIEW mv_total_orders;
```

To autorefresh a materialized view, add the `AUTO REFRESH YES` clause to the `CREATE MATERIALIZED VIEW` statement as the following example demonstrates:

```
CREATE MATERIALIZED VIEW mv_total_orders
AUTO REFRESH YES -- Add this clause to auto refresh the MV
AS
 SELECT c.cust_id,
        c.first_name,
        sum(o.amount) as total_amount
 FROM orders o
 JOIN customer c
    ON c.cust_id = o.customer_id
 GROUP BY c.cust_id,
          c.first_name;
```

Amazon Redshift autorefreshes materialized views as soon as possible after a base table changes. To minimize the impact of active workloads in your cluster when processing the refresh, Amazon Redshift considers the following factors:

- Current system load
- The resources required for a refresh
- Available cluster resources
- How often the materialized views are used

Amazon Redshift prioritizes your workloads over autorefresh and could stop autorefreshing to preserve the performance of the user workload. Keep in mind that this approach can delay the refreshes of some materialized views. For refresh status, you can check the SVL_MV_REFRESH_STATUS view. This view records queries that are user-initiated or autorefreshed.

Automatic query rewriting

Amazon Redshift can automatically rewrite SQL queries that don't explicitly reference existing materialized views that are used for performance improvements. Directly querying a materialized view reduces the query processing time, but the query statement must be modified. If you use materialized views to accelerate queries, it's important to consider how to systematically and automatically use materialized views to respond to queries. You can use transparent rewriting to add or delete materialized views in the same way as an index, without affecting existing SQL statements. Amazon Redshift uses only up-to-date materialized views when it automatically rewrites queries.

How query rewriting works

Consider the following SQL query. The query joins two tables: customer (10+ million rows) and order (10+ billion rows):

```
SELECT c.cust_id,
       c.first_name,
       sum(o.amount) as total_amount
FROM orders o
JOIN customer c
  ON c.cust_id = o.customer_id
GROUP BY c.cust_id,
         c.first_name
HAVING sum(o.amount) > 500;
```

This query joins two large tables, applies a sum aggregation on the `o.amount` column, and then filters the results to display only the customers who ordered more than $500. This query could consume a lot of resources.

As an example, consider the `mv_total_orders` materialized view from the Creating materialized views section of this guide. After this view is created, then Amazon Redshift automatically rewrites the preceding query as follows:

```
SELECT cust_id,
       first_name,
       total_amount
FROM mv_total_orders
```

```
WHERE total_amount > 500;
```

This new query runs much faster because it uses a materialized view instead of joining two large tables.

Query rewriting has led to the wider adoption of materialized views for the following reasons:

1. Materialized views support the transparent rewriting of queries. The query rewrite transforms a SQL statement expressed in terms of tables into a statement accessing one or more materialized views. This transformation is transparent to the end user, requiring no intervention and no reference to the materialized view in the SQL statement.
2. Materialized views simplify the application of cached common result sets and support cross-query optimizations such as precomputing.
3. In data warehouses and data integration scenarios, materialized views can materialize foreign table results to hide the differences among multiple data sources, implementing local replicas or read/write splitting.
4. The query rewriting feature combined with automatic materialized view creation accelerates database autonomy.

Advantages of materialized views

There are several advantages to using materialized views:

- **Fewer updates** – A standard view isn't physically materialized, which means that the query that defines a standard view runs every time the view is referenced in a query. In contrast, a materialized view is precomputed and stored on disk (similar to an object) as a result of a query expression like regular views. Unlike standard views, materialized views are not updated each time they are used.
- **Faster response times** – A materialized view responds faster in comparison to a view. This is because the materialized view is precomputed and therefore doesn't waste time resolving the query or joins in the query that create the materialized view.
- **Stored SQL statement** – You can use a rollup, or aggregation, table instead of a materialized view. Rollup tables are precomputed and stored on disk (similar to materialized views), but they don't store their SQL statements in the database. Materialized views do store their SQL statements.
- **Easy to refresh** – Materialized views are easy to refresh. Simply run the `REFRESH MATERIALIZED VIEW` command.
- **Automatic query rewriting** – The query optimizer can rewrite your SQL statement to fetch data from an existing materialized view, even if the materialized view isn't explicitly used in your SQL statement.

Resources

References

- Creating materialized views in Amazon Redshift
- Querying a materialized view
- Automatic query rewriting to use materialized views
- Refreshing a materialized view

Tools

- Amazon Redshift Query Editor v2.0

Document history

The following table describes significant changes to this guide. If you want to be notified about future updates, you can subscribe to an RSS feed.

Change	Description	Date
Initial publication	—	December 15, 2022