14-Materialized View

Author: Vincent Lau

Learning Objectives

- Understand the purpose of using Materialized View.
- Understand the ways of implementing Materialized View in PostgreSQL.
- Understand the difference between View and Materialized View.

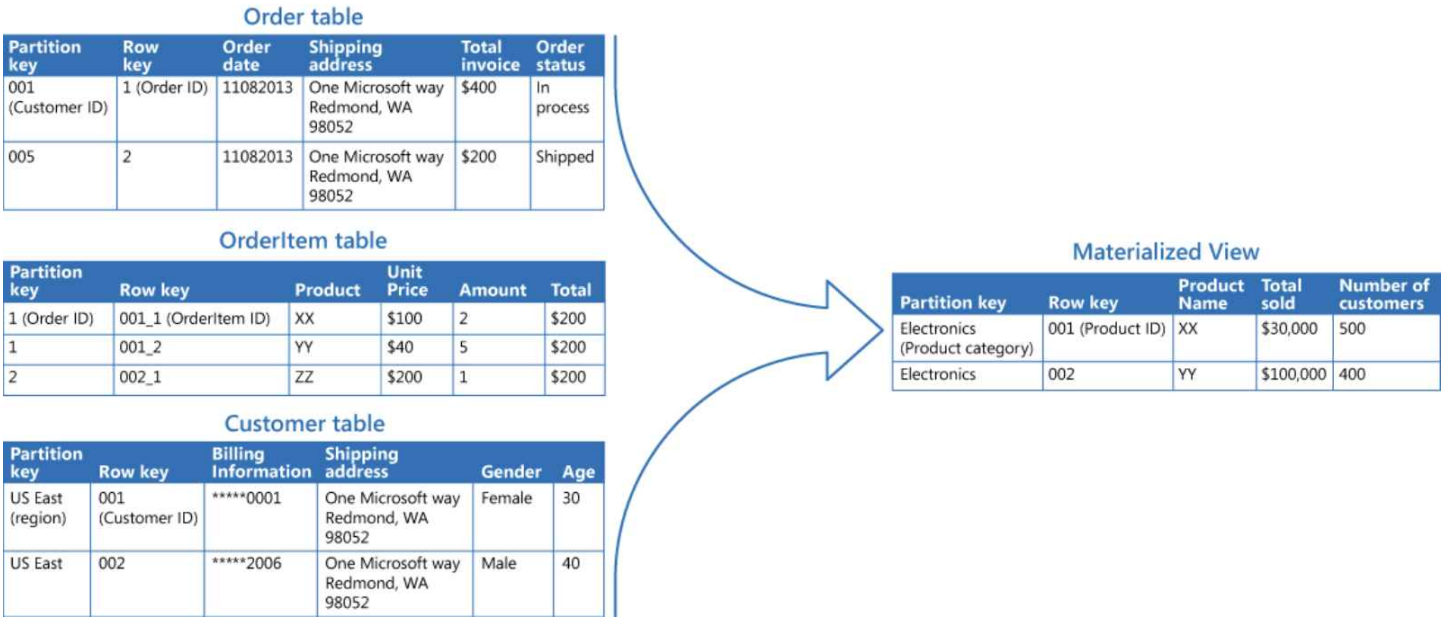
Introduction

A materialized view is a database object that stores the result of a query as a **physical table**. Unlike regular views, which simply provide a **saved query** that is executed each time it's accessed, a materialized view **stores the actual data resulting from the query**. This can be particularly useful for improving query performance, as it allows you to **precompute and store the results of complex or resource-intensive queries**.

Here's a basic example of creating and using a materialized view in PostgreSQL:

Let's assume we have a table named `orders` with columns "order_id", "customer_id", "order_date", and "total_amount". We want to create a materialized view that shows the total revenue generated by each customer:

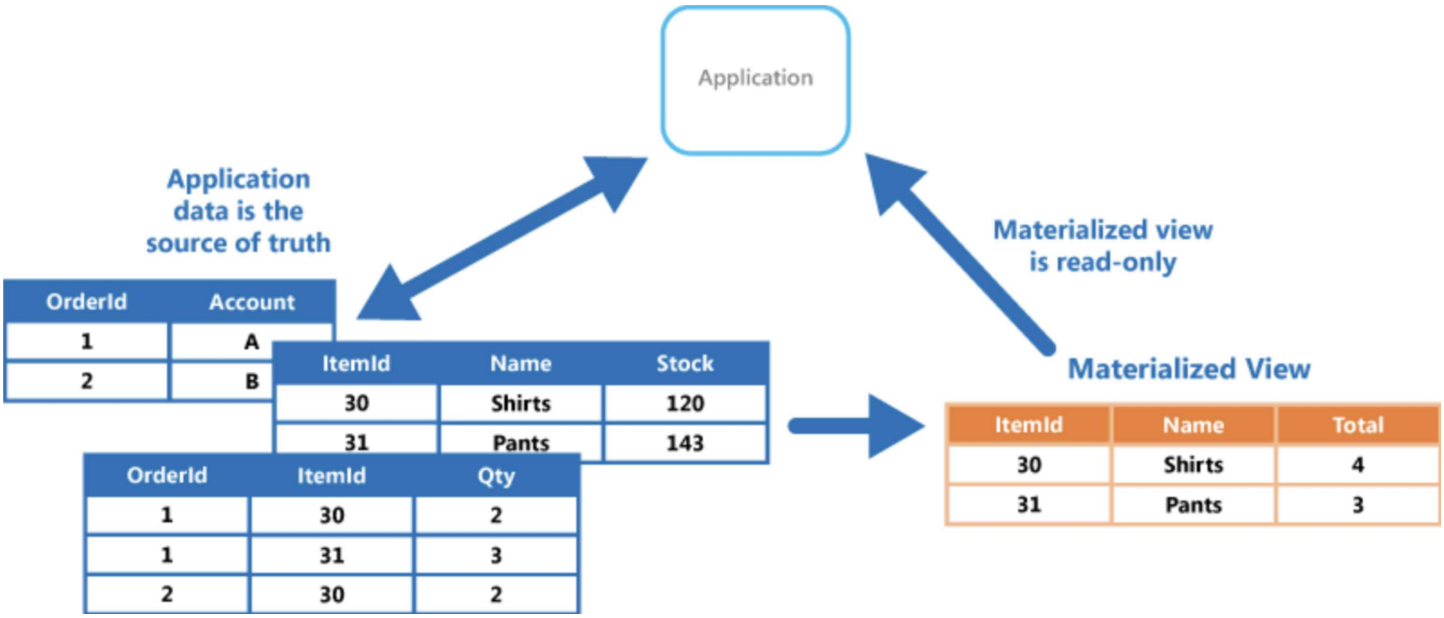
Create Materialized View



In this example, the materialized view "customer_revenue_summary" is created by aggregating the "orders" table to calculate the total revenue for each customer.

```
CREATE MATERIALIZED VIEW customer_revenue_summary
AS
SELECT customer_id,
SUM(total_amount) AS total_revenue
FROM orders
GROUP BY customer_id;
```

Refresh Materialized View



The `REFRESH MATERIALIZED VIEW` command updates the data in the materialized view to reflect any changes in the underlying "orders" table. Finally, you can query the materialized view as you would query a regular table.

It's important to note that materialized views introduce a **trade-off between improved query performance and potential data staleness**. Since the data in the materialized view needs to be **refreshed explicitly** or automatically on a scheduled basis, there might be a **delay between changes in the source tables and updates to the materialized view**.

```
1 REFRESH MATERIALIZED VIEW customer_revenue_summary;
```

Query the materialized view

```
1 SELECT * FROM customer_revenue_summary;
```

You can also specify options like indexes and refresh methods when creating a materialized view to further optimize its behavior for your specific use case.

Keep in mind that materialized views are available in PostgreSQL, but not all database systems support them in the same way or with the same capabilities (There is no materialized view in MySQL).

Why using Materialized View

Advantages of using MV

Improved Query Performance

Materialized views can significantly enhance query performance by storing precomputed results. Queries that involve complex calculations or aggregations can be much faster when using materialized views.

Reduced Load on the Database

By storing precomputed results, materialized views reduce the need for expensive computations during query execution, which can offload processing from the main database.

Data Aggregation and Reporting

Materialized views are particularly beneficial for **data warehousing and reporting scenarios** where large datasets are aggregated, summarized, and queried frequently.

Offline Analysis

Materialized views allow for analytical queries to be performed on the cached data **without impacting the operational database (offline version of original data)**.

Query Optimization

Database systems often optimize queries involving materialized views, leading to more efficient execution plans.

Query Consistency

Materialized views can be refreshed at specific intervals or on-demand, ensuring that the cached data remains consistent.

Downsides of using MV

Data Staleness

The data in materialized views can become stale if the underlying data changes frequently and the materialized view is not refreshed accordingly.

Maintenance Overhead

Maintaining materialized views involves periodically updating them, which adds maintenance overhead and may affect system performance during refresh operations.

Storage Usage

Materialized views consume storage space, which can become a concern if you're dealing with large datasets or if you need to create multiple materialized views.

Complexity

Managing multiple materialized views across different parts of a database schema can lead to increased complexity. Impact analysis should be performed and kept update-to-date after table updates.

Real-Time Data

Materialized views are not suitable for real-time data scenarios, as they involve a delay between data updates and the corresponding refresh of the materialized view.

Resource Intensive Refresh: Refreshing materialized views can be resource-intensive, especially for large datasets, which might impact other operations.

Query Patterns

Materialized views are most effective for frequently executed queries. If query patterns change frequently, maintaining relevant materialized views can be challenging.

Materialized Views & Views

Materialized Views

Data Storage

A materialized view stores the actual data of the query result as a physical table in the database. It's a snapshot of the data at the time of creation or last refresh.

Data Freshness

The data in a materialized view can become stale if not regularly refreshed. Refreshing involves recalculating and updating the data in the materialized view.

Performance

Materialized views can significantly enhance performance for complex queries by allowing precomputed data to be used, especially for aggregated or summarized data.

Storage

Materialized views consume storage space proportional to the size of the data they store. This can be a concern for large datasets or multiple materialized views.

Usage

Materialized views are used to optimize query performance, aggregate data for reporting, and offload processing from the main database.

Updates

Depending on the database system and the nature of the materialized view, updates might be possible if certain conditions are met.

Views

Data Storage

A regular view does not store any data itself. It's a saved SQL query that's executed dynamically each time the view is queried. The data is fetched from the underlying tables in real-time.

Data Freshness

The data in a regular view is always up-to-date with the underlying tables because it's generated on the fly whenever the view is queried.

Performance

Regular views can impact performance because they involve executing the query every time the view is accessed. Performance depends on the complexity of the view's query and the size of the underlying tables.

Storage

Regular views do not consume additional storage space beyond the saved query definition.

Usage

Regular views are used for simplifying complex queries, abstracting data for users, and providing a logical representation of data.

Updates

Regular views are not directly updatable if they involve multiple tables or complex queries.

Summary

Materialized views are a powerful tool for optimizing query performance and simplifying complex calculations, especially in scenarios where certain queries are executed frequently with similar patterns. However, their effectiveness depends on the nature of your data, the frequency of updates, and the resources available for maintaining and refreshing them. Careful consideration should be given to choosing the right queries for materialized views and balancing the trade-offs they bring.