🚀 Project Title:

Query Performance Optimization using Amazon Redshift Materialized Views on Olist E-Commerce Dataset

🧩 Problem Statement

An e-commerce platform is running **complex analytical queries** on large datasets stored in Amazon Redshift. These queries often involve **multiple joins across large tables** (orders, products, customers) and **aggregation operations**, making them **slow and resource-heavy**, especially when used repeatedly for business intelligence dashboards and reporting.

🎯 Objective

To **improve performance** of frequently executed, complex join queries using **Amazon Redshift's materialized views**, ensuring faster execution times and reduced computational cost.

📦 Dataset Used

Olist Brazilian E-Commerce Dataset

- Source: Kaggle - Olist E-Commerce Dataset
- Files used:
  - `olist_customers_dataset.csv`
  - `olist_orders_dataset.csv`
  - `olist_order_items_dataset.csv`
  - `olist_products_dataset.csv`

🛠️ Implementation Steps

1️⃣ Prepare Amazon S3 and Redshift Serverless Environment

- Uploaded CSV datasets to Amazon S3 bucket: `s3://olist-ecommerce-dataset1/`
- Created Redshift Serverless workgroup and namespace

- Created and attached IAM Role `olist-ecommerce-data` to allow Redshift to read from S3

2️⃣ Create Schema and Load Data into Redshift

```
CREATE SCHEMA IF NOT EXISTS olist;

CREATE TABLE olist.customers (
  customer_id VARCHAR,
  customer_unique_id VARCHAR,
  customer_zip_code_prefix VARCHAR,
  customer_city VARCHAR,
  customer_state VARCHAR
);

-- Similarly created tables for orders, order_items, products

COPY olist.customers
FROM 's3://olist-ecommerce-dataset1/olist_customers_dataset.csv'
IAM_ROLE 'arn:aws:iam::535002864770:role/olist-ecommerce-data'
FORMAT AS CSV
IGNOREHEADER 1;
```

✅ All data successfully loaded into Redshift `olist` schema.

3️⃣ Execute Complex Join Query (Without Materialized View)

```
SELECT
    c.customer_city,
    p.product_category_name,
    ROUND(SUM(oi.price + oi.freight_value), 2) AS total_revenue,
    COUNT(DISTINCT o.order_id) AS total_orders
FROM
    olist.orders o
JOIN
    olist.customers c ON o.customer_id = c.customer_id
JOIN
    olist.order_items oi ON o.order_id = oi.order_id
JOIN
    olist.products p ON oi.product_id = p.product_id
WHERE
    o.order_status = 'delivered'
GROUP BY
    c.customer_city,
    p.product_category_name
ORDER BY
```

```
    total_revenue DESC
LIMIT 20;
```

⏱️ **Execution Time**: ~138ms

4️⃣ Create Materialized View

```
CREATE MATERIALIZED VIEW olist.mv_city_product_sales AS
SELECT
    c.customer_city,
    p.product_category_name,
    ROUND(SUM(oi.price + oi.freight_value), 2) AS total_revenue,
    COUNT(DISTINCT o.order_id) AS total_orders
FROM
    olist.orders o
JOIN
    olist.customers c ON o.customer_id = c.customer_id
JOIN
    olist.order_items oi ON o.order_id = oi.order_id
JOIN
    olist.products p ON oi.product_id = p.product_id
WHERE
    o.order_status = 'delivered'
GROUP BY
    c.customer_city,
    p.product_category_name;
```

⚠️ Note: This view is not incrementally refreshable because of the use of `ROUND()` and `SUM()` aggregation.

5️⃣ Query Materialized View

```
SELECT *
FROM olist.mv_city_product_sales
ORDER BY total_revenue DESC
LIMIT 20;
```

⏱️ **Execution Time**: ~8ms ⚡ (Significantly faster than raw join query)

6️⃣ Performance Comparison

Query Type	Execution Time
Raw Join Query	~138ms
Materialized View	~8ms

✅ Performance Improved: **~17x faster** query execution using materialized views.

Optional Maintenance Step

If the source tables change and the materialized view needs to be updated:

```
REFRESH MATERIALIZED VIEW olist.mv_city_product_sales;
```

Conclusion

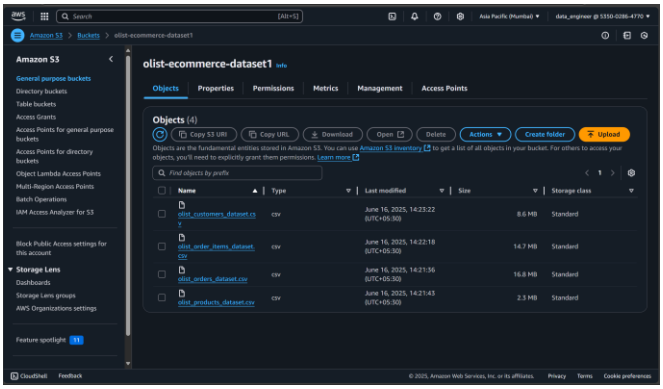
Using Amazon Redshift's **materialized views** significantly improves performance for repeated, resource-intensive analytical queries. This enhancement supports faster dashboards, better user experience, and reduced compute load.

Skills and Tools Used

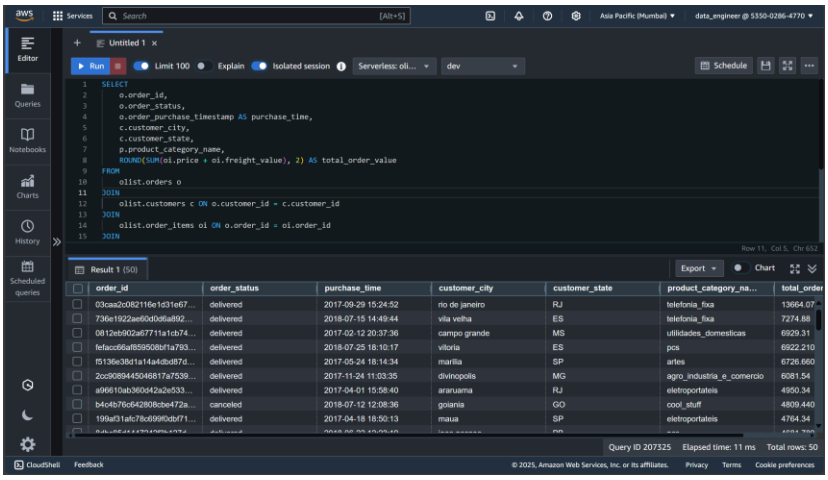
- Amazon Redshift Serverless
- Amazon S3
- Redshift Query Editor v2
- IAM Role Management
- SQL (joins, aggregates, materialized views)
- Performance Optimization
- Real-time Data Engineering Practices

Workspace Output (Paste Your Screenshots, Logs, and Outputs Below)

`[S3 Bucket Upload Screenshot]`



[Redshift Query Output Screenshots]



[Performance Timings Before vs After View Creation]

customer_city	product_category_na...	total_revenue	total_orders
sao paulo	beleza_saude	207702.9	1500
sao paulo	cama_mesa_banho	196578.9	1623
sao paulo	relogios_presentes	169652.84	760
sao paulo	esporte_lazer	160224.11000000002	1229
sao paulo	informatica_acessorios	156232.01	980
sao paulo	utilidades_domesticas	123038.68000000001	1083
sao paulo	moveis_decoracao	116219.35	974
rio de janeiro	beleza_saude	93574.81	504
rio de janeiro	relogios_presentes	93415.51	419
rio de janeiro	cama_mesa_banho	91471.81999999999	704

Query ID 206820 Elapsed time: 137 ms Total rows: 20

customer_city	product_category_na...	total_revenue	total_orders
sao paulo	beleza_saude	207702.9	1500
sao paulo	cama_mesa_banho	196578.9	1623
sao paulo	relogios_presentes	169652.84	760
sao paulo	esporte_lazer	160224.11000000002	1229
sao paulo	informatica_acessorios	156232.01	980
sao paulo	utilidades_domesticas	123038.68000000001	1083
sao paulo	moveis_decoracao	116219.35	974
rio de janeiro	beleza_saude	93574.81	504
rio de janeiro	relogios_presentes	93415.51	419
rio de janeiro	cama_mesa_banho	91471.81999999999	704

Query ID 206861 Elapsed time: 9 ms Total rows: 20

Tags

#AmazonRedshift #MaterializedViews #SQLPerformance #RealTimeAnalytics #DataEngineeringProject