Data Warehouse Implementation Report

Author: Hari Prasad Reddy Etta
Course: Data Warehousing

1. Introduction

This report documents the design, implementation, and execution of a data warehouse for retail sales data using Snowflake and Google Cloud Storage (GCS). The project focuses on ETL (Extract, Transform, Load) processes, dimensional modeling, and analytical querying to derive actionable business insights.

Project Scope

- The goal is to integrate data from multiple CSV files into a Snowflake data warehouse.
- The project involves cleaning and structuring raw data to enable efficient analytics.
- Analytical queries help understand revenue trends, customer segmentation, and product profitability.

2. Technology Stack Used

- Cloud Platform: Google Cloud Storage (GCS)
- Data Warehouse: Snowflake
- ETL Methodology: SQL-based transformations
- Storage Integration: Snowflake External Stages for GCS
- Query Language: SQL

3. Data Sourcing

Dataset Overview

The data used in this project is sourced from a Superstore Retail Dataset on Kaggle. This dataset contains sales and customer transaction details for a furniture and office products store. The data is structured to enable efficient business intelligence analysis through a data warehouse approach.

3.1 Data Sources

The data files used in this project are in CSV format. Even though the information is divided into various tables, there was no clear relationship between few tables initially such as orders and customers , orders and products initially. Establishing relationships between these tables was one of the key challenges during data integration.

The project integrates multiple data sources to support dimensional modeling:

- Customers.csv: Contains customer demographic details.
- Products.csv: Includes product category and description.
- Orders.csv: Holds sales transactions (Order ID, Customer ID, Product ID, Sales, etc.).
- Dates.csv: Provides time-related data for sales trends.

3.2 Source Relationships & Key Additions

3.2.1 Source Relationships & Key Additions

Initially, no clear relationships existed between the tables. To integrate the datasets, the following keys were introduced:

- CustomerID connects Customers and Orders.
- ProductID connects Products and Orders.
- OrderDate links Orders with the Dates table.
- Primary Keys Added:
    - CustomerID in Customers Table
    - ProductID in Products Table
    - OrderID in Orders Table
- Data Cleaning & Missing Value Handling:
    - Missing PostalCode: Replaced with "Unknown"
    - Inconsistent OrderStatus Labels: Standardized to "Completed", "Pending", "Canceled"

3.3 Data Dictionary

Customers Table

Column Name	Data Type	Description
CustomerID	STRING	Unique identifier for each customer
CustomerName	STRING	Full name of the customer
Segment	STRING	Customer segment type (e.g., Corporate, Home Office, Consumer)
Country	STRING	Country where the customer resides
City	STRING	City of the customer
State	STRING	State of the customer
PostalCode	STRING	Postal code of the customer's address
Region	STRING	Geographical region of the customer
Age	INT	Age of the customer

Products Table

Column Name	Data Type	Description
ProductID	STRING	Unique identifier for each product
Category	STRING	Product category (e.g., Office Supplies, Furniture, Technology)
SubCategory	STRING	More specific classification within a category (e.g., Chairs, Laptops)
ProductName	STRING	Name of the product

Orders Table

Column Name	Data Type	Description
OrderID	STRING	Unique identifier for each order
CustomerID	STRING	References CustomerID from Customers Table
ProductID	STRING	References ProductID from Products Table
Sales	FLOAT	Total sales amount for the order
Quantity	INT	Number of units sold in the order
Discount	FLOAT	Discount applied to the order
Profit	FLOAT	Profit earned from the order
OrderDate	DATE	Date when the order was placed
ShipDate	DATE	Date when the order was shipped
ShipMode	STRING	Mode of shipment (e.g., Standard Class, First Class)
OrderStatus	STRING	Status of the order (e.g., Completed, Pending, Canceled)

Dates Table

Column Name	Data Type	Description
OrderDate	DATE	Date when the order was placed
ShipDate	DATE	Date when the order was shipped
Year	INT	Extracted year from OrderDate
Quarter	INT	Extracted quarter from OrderDate
Month	INT	Extracted month from OrderDate
Day	INT	Extracted day from OrderDate

4. Normalized Database

4.1 DDL Scripts for Creating Normalized Database

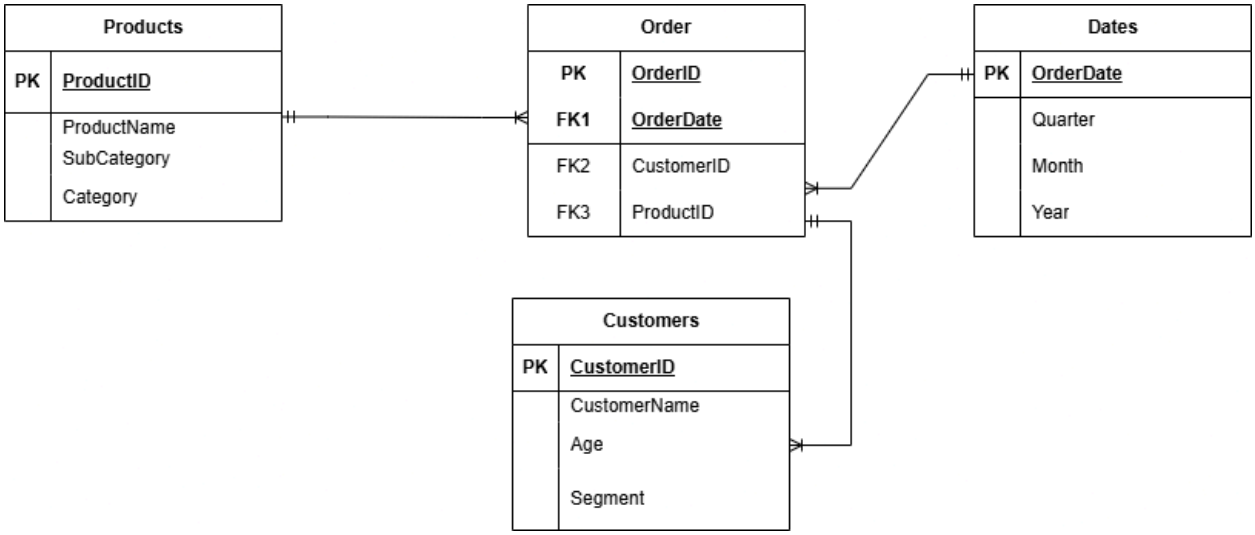
```
CREATE DATABASE IF NOT EXISTS RetailSales_DW;
CREATE SCHEMA IF NOT EXISTS Staging;
CREATE SCHEMA IF NOT EXISTS DataWarehouse;

CREATE OR REPLACE TABLE DataWarehouse.Products (
    ProductKey INT AUTOINCREMENT PRIMARY KEY,
    ProductID STRING UNIQUE,
    ProductName STRING,
    Category STRING,
    SubCategory STRING
);

CREATE OR REPLACE TABLE DataWarehouse.Customers (
    CustomerKey INT AUTOINCREMENT PRIMARY KEY,
    CustomerID STRING UNIQUE,
    CustomerName STRING,
    Segment STRING,
    Country STRING,
    City STRING,
    State STRING,
```

```
    PostalCode STRING,
    Region STRING,
    Age INT
);



CREATE OR REPLACE TABLE DataWarehouse.Orders (
    OrderKey INT AUTOINCREMENT PRIMARY KEY,
    OrderID STRING UNIQUE,
    CustomerID STRING,
    ProductID STRING,
    Sales FLOAT,
    Quantity INT,
    Discount FLOAT,
    Profit FLOAT,
    OrderDate DATE,
    ShipDate DATE,
    ShipMode STRING,
    OrderStatus STRING
);

CREATE OR REPLACE TABLE DataWarehouse.Dates (
    DateKey INT AUTOINCREMENT PRIMARY KEY,
    OrderDate DATE UNIQUE,
    ShipDate DATE,
    Year INT,
    Quarter INT,
    Month INT,
    Day INT
);
```

4.2 ER Diagram

- The normalized schema includes four main entities: Products, Customers, Orders, and Dates.
- Relationships:

- Orders table acts as the fact table, linking Products, Customers, and Dates.
- ProductID, CustomerID, and OrderDate serve as foreign keys in Orders.



4.3 Loading Data into Normalized Tables

INSERT INTO DataWarehouse.Products (ProductID, ProductName, Category, SubCategory)
SELECT DISTINCT ProductID, ProductName, Category, SubCategory FROM Staging.Products;

INSERT INTO DataWarehouse.Customers (CustomerID, CustomerName, Segment, Country, City, State, PostalCode, Region, Age)
SELECT DISTINCT CustomerID, CustomerName, Segment, Country, City, State, PostalCode, Region, Age FROM Staging.Customers;

INSERT INTO DataWarehouse.Dates (OrderDate, ShipDate, Year, Quarter, Month, Day)
SELECT DISTINCT OrderDate, ShipDate, EXTRACT(YEAR FROM OrderDate), EXTRACT(QUARTER FROM OrderDate), EXTRACT(MONTH FROM OrderDate), EXTRACT(DAY FROM OrderDate) FROM Staging.Orders;

This ensures that:

1. Raw data from staging is successfully transferred into the normalized database.
2. Referential integrity is maintained by ensuring foreign keys properly reference dimension tables.
3. Date inconsistencies and missing values are handled through type casting and default values.

5. ETL Implementation

5.1 ETL Workflow Overview

The ETL pipeline in this project consists of three main stages:

1. Extract

- Data is extracted from Google Cloud Storage (GCS) into Snowflake Staging Tables.
- COPY INTO is used to ingest data into Snowflake.
- Creating Storage Integration for GCS:

```
CREATE OR REPLACE STORAGE INTEGRATION gcs_integration
TYPE = EXTERNAL_STAGE
STORAGE_PROVIDER = 'GCS'
ENABLED = TRUE
STORAGE_ALLOWED_LOCATIONS = ('gcs://hari_dw/');
```

- Creating External Stage for Data Ingestion:

```
CREATE OR REPLACE STAGE gcs_stage
STORAGE_INTEGRATION = gcs_integration
URL = 'gcs://hari_dw/';
```

- Loading Data into Staging Tables:

```
COPY INTO Staging.Customers FROM @gcs_stage/Customers.csv FILE_FORMAT = (TYPE =
CSV, SKIP_HEADER = 1, FIELD_OPTIONALLY_ENCLOSED_BY='"') ON_ERROR = CONTINUE;
COPY INTO Staging.Products FROM @gcs_stage/Products.csv FILE_FORMAT = (TYPE = CSV,
SKIP_HEADER = 1, FIELD_OPTIONALLY_ENCLOSED_BY='"') ON_ERROR = CONTINUE;
COPY INTO Staging.Orders FROM @gcs_stage/Orders.csv FILE_FORMAT = (TYPE = CSV,
SKIP_HEADER = 1, FIELD_OPTIONALLY_ENCLOSED_BY='"') ON_ERROR = CONTINUE;
COPY INTO Staging.Dates FROM @gcs_stage/Dates.csv FILE_FORMAT = (TYPE = CSV,
SKIP_HEADER = 1, FIELD_OPTIONALLY_ENCLOSED_BY='"') ON_ERROR = CONTINUE;
```

- Handling missing values (e.g., using COALESCE for NULLs).
- Extracting derived columns (e.g., Year, Quarter, Month, Day from OrderDate).
- Creating surrogate keys for optimized joins in fact tables. The ETL pipeline in this project consists of three main stages:

1. Extract

- Data is extracted from Google Cloud Storage (GCS) into Snowflake Staging Tables.
- COPY INTO is used to ingest data into Snowflake.

2. Transform

- Handling missing values (e.g., using COALESCE for NULLs).
- Extracting derived columns (e.g., Year, Quarter, Month, Day from OrderDate).
- Creating surrogate keys for optimized joins in fact tables.

3. Load

- The transformed data is structured into:
    - Dim_Customers: Customer details.
    - Dim_Products: Product information.
    - Dim_Dates: Time-based insights.
    - Fact_Orders: Sales transactions.

5.2 Handling ETL Challenges

Issue	Solution
Missing PostalCode	Assigned 'Unknown' as default
Negative Age values	Set to NULL to avoid incorrect data
OrderStatus inconsistencies	Standardized to 'Completed', 'Pending', 'Canceled'
Orders referencing non-existing customers	Ensured CustomerID in Fact_Orders only references existing records in Dim_Customers

5.3 Creating Fact and Dimension Tables

Creating Dimension Tables

```
CREATE OR REPLACE TABLE DataWarehouse.Dim_Customers (
    CustomerKey INT AUTOINCREMENT PRIMARY KEY,
    CustomerID STRING UNIQUE,
    CustomerName STRING,
    Segment STRING,
    Country STRING,
```

```
    City STRING,
    State STRING,
    PostalCode STRING,
    Region STRING,
    Age INT
);

CREATE OR REPLACE TABLE DataWarehouse.Dim_Products (
    ProductKey INT AUTOINCREMENT PRIMARY KEY,
    ProductID STRING UNIQUE,
    Category STRING,
    SubCategory STRING,
    ProductName STRING
);

CREATE OR REPLACE TABLE DataWarehouse.Dim_Dates (
    DateKey INT AUTOINCREMENT PRIMARY KEY,
    OrderDate DATE UNIQUE,
    ShipDate DATE,
    Year INT,
    Quarter INT,
    Month INT,
    Day INT
);
```

Creating Fact Table

```
CREATE OR REPLACE TABLE DataWarehouse.Fact_Orders (
    OrderKey INT AUTOINCREMENT PRIMARY KEY,
    OrderID STRING UNIQUE,
    CustomerKey INT REFERENCES DataWarehouse.Dim_Customers(CustomerKey),
    ProductKey INT REFERENCES DataWarehouse.Dim_Products(ProductKey),
    DateKey INT REFERENCES DataWarehouse.Dim_Dates(DateKey),
    Sales FLOAT,
    Quantity INT,
    Discount FLOAT,
    Profit FLOAT,
    OrderStatus STRING
```

);

Loading Data into Dimension Tables

INSERT INTO DataWarehouse.Dim_Customers (CustomerID, CustomerName, Segment, Country, City, State, PostalCode, Region, Age)
SELECT DISTINCT CustomerID, CustomerName, Segment, Country, City, State, PostalCode, Region, Age FROM Staging.Customers;

INSERT INTO DataWarehouse.Dim_Products (ProductID, Category, SubCategory, ProductName)
SELECT DISTINCT ProductID, Category, SubCategory, ProductName FROM Staging.Products;

INSERT INTO DataWarehouse.Dim_Dates (OrderDate, ShipDate, Year, Quarter, Month, Day)
SELECT DISTINCT OrderDate, ShipDate, EXTRACT(YEAR FROM OrderDate), EXTRACT(QUARTER FROM OrderDate), EXTRACT(MONTH FROM OrderDate), EXTRACT(DAY FROM OrderDate)
FROM Staging.Dates;

Loading Data into Fact Table

INSERT INTO DataWarehouse.Fact_Orders (OrderID, CustomerKey, ProductKey, DateKey, Sales, Quantity, Discount, Profit, OrderStatus)
SELECT o.OrderID, c.CustomerKey, p.ProductKey, d.DateKey, o.Sales, o.Quantity, o.Discount, o.Profit, o.OrderStatus
FROM Staging.Orders o
LEFT JOIN DataWarehouse.Dim_Customers c ON o.CustomerID = c.CustomerID
LEFT JOIN DataWarehouse.Dim_Products p ON o.ProductID = p.ProductID
LEFT JOIN DataWarehouse.Dim_Dates d ON o.OrderDate = d.OrderDate;

5.4 Slowly Changing Dimensions (SCD)

- SCD Type 1 (Overwrite): The latest records overwrite old values in Dim_Customers and Dim_Products.
- SCD Type 2 (History Tracking): If implemented, a versioning column (EffectiveDate, EndDate) would track history.
- SCD Type 1 (Overwrite): The latest records overwrite old values in Dim_Customers and Dim_Products.
- SCD Type 2 (History Tracking): If implemented, a versioning column (EffectiveDate, EndDate) would track history.

6. Dimensional Modeling

6.1 Star Schema Design

This project follows the Star Schema approach, where a central Fact Table (Fact_Orders) is connected to multiple Dimension Tables (Dim_Customers, Dim_Products, Dim_Dates).

6.2 Fact and Dimension Tables

Creating Dimension Tables

```
CREATE OR REPLACE TABLE DataWarehouse.Dim_Customers (
    CustomerKey INT AUTOINCREMENT PRIMARY KEY,
    CustomerID STRING UNIQUE,
    CustomerName STRING,
    Segment STRING,
    Country STRING,
    City STRING,
    State STRING,
    PostalCode STRING,
    Region STRING,
    Age INT
);

CREATE OR REPLACE TABLE DataWarehouse.Dim_Products (
    ProductKey INT AUTOINCREMENT PRIMARY KEY,
    ProductID STRING UNIQUE,
    Category STRING,
    SubCategory STRING,
    ProductName STRING
);

CREATE OR REPLACE TABLE DataWarehouse.Dim_Dates (
    DateKey INT AUTOINCREMENT PRIMARY KEY,
    OrderDate DATE UNIQUE,
    ShipDate DATE,
    Year INT,
    Quarter INT,
```

```
    Month INT,
    Day INT
);
```

Creating Fact Table

```
CREATE OR REPLACE TABLE DataWarehouse.Fact_Orders (
    OrderKey INT AUTOINCREMENT PRIMARY KEY,
    OrderID STRING UNIQUE,
    CustomerKey INT REFERENCES DataWarehouse.Dim_Customers(CustomerKey),
    ProductKey INT REFERENCES DataWarehouse.Dim_Products(ProductKey),
    DateKey INT REFERENCES DataWarehouse.Dim_Dates(DateKey),
    Sales FLOAT,
    Quantity INT,
    Discount FLOAT,
    Profit FLOAT,
    OrderStatus STRING
);
```

6.3 Key Relationships & Constraints

- Fact_Orders is the central transaction table, containing references to dimension tables.
- CustomerKey, ProductKey, and DateKey act as foreign keys, ensuring referential integrity.
- Each Dimension Table uses surrogate keys for efficient indexing and storage.

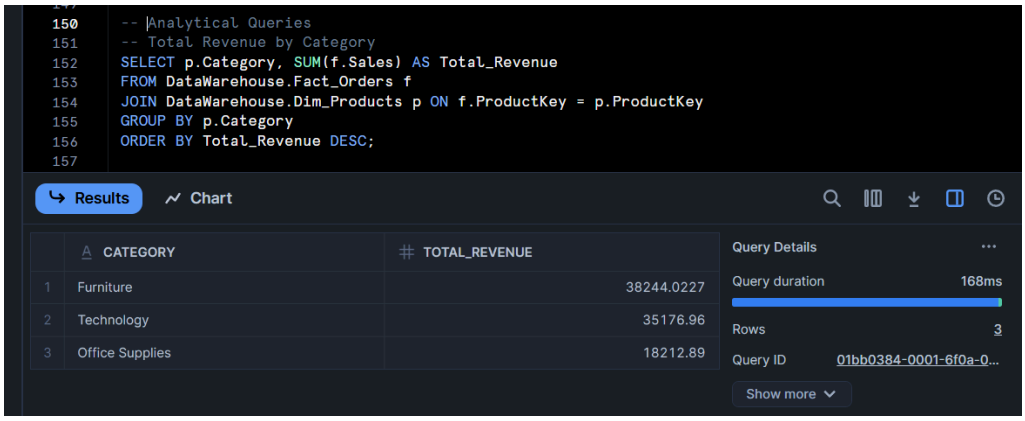
6.4 Slowly Changing Dimensions (SCD)

- SCD Type 1 (Overwrite): The latest records overwrite old values in Dim_Customers and Dim_Products.
- SCD Type 2 (History Tracking): If implemented, a versioning column (EffectiveDate, EndDate) would track history.

Part 7: Analytical Querying

```
-- Total Revenue by Category
SELECT p.Category, SUM(f.Sales) AS Total_Revenue
FROM DataWarehouse.Fact_Orders f
JOIN DataWarehouse.Dim_Products p ON f.ProductKey = p.ProductKey
```

GROUP BY p.Category
ORDER BY Total_Revenue DESC;

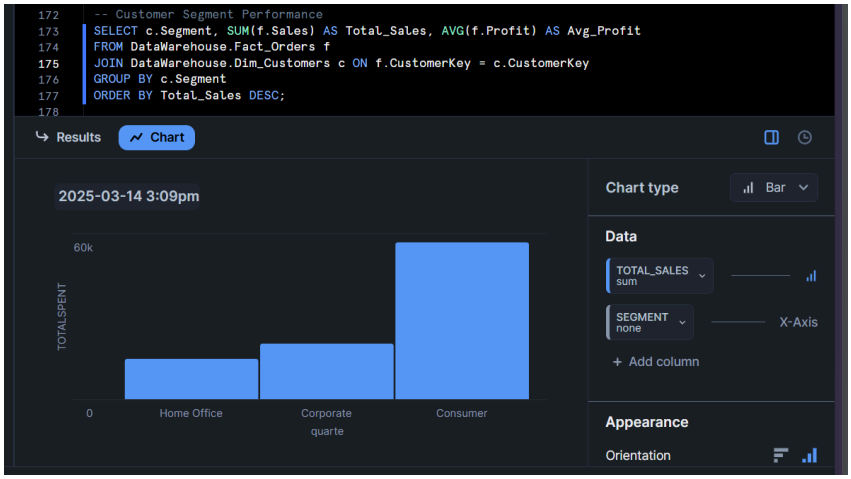


-- Top 5 Customers by Purchases
SELECT c.CustomerName, SUM(f.Sales) AS TotalSpent
FROM DataWarehouse.Fact_Orders f
JOIN DataWarehouse.Dim_Customers c ON f.CustomerKey = c.CustomerKey
GROUP BY c.CustomerName
ORDER BY TotalSpent DESC
LIMIT 8;

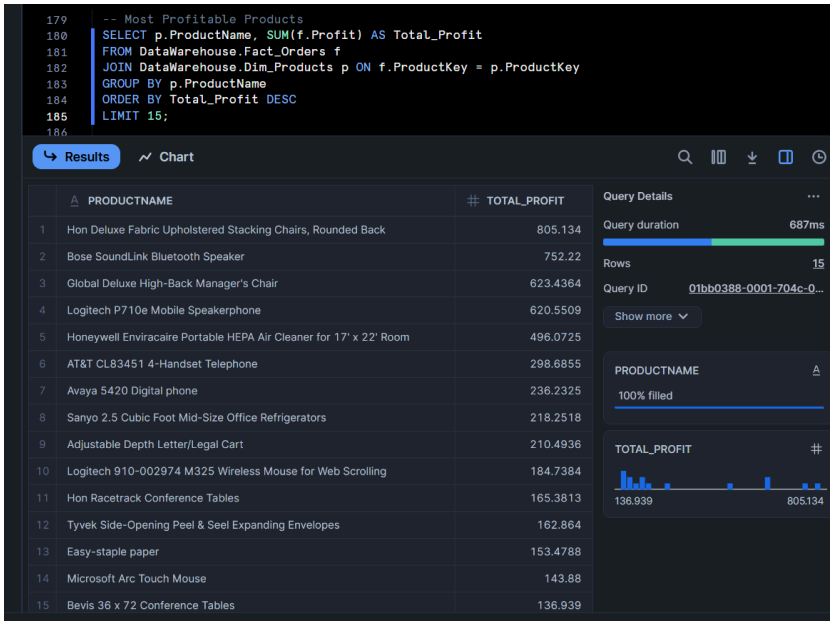
-- Quarterly Sales Trends
SELECT EXTRACT(YEAR FROM f.OrderDate) AS Year, EXTRACT(QUARTER FROM f.OrderDate) AS Quarter, SUM(f.Sales) AS Quarterly_Revenue
FROM DataWarehouse.Fact_Orders f

GROUP BY Year, Quarter
ORDER BY Year, Quarter;

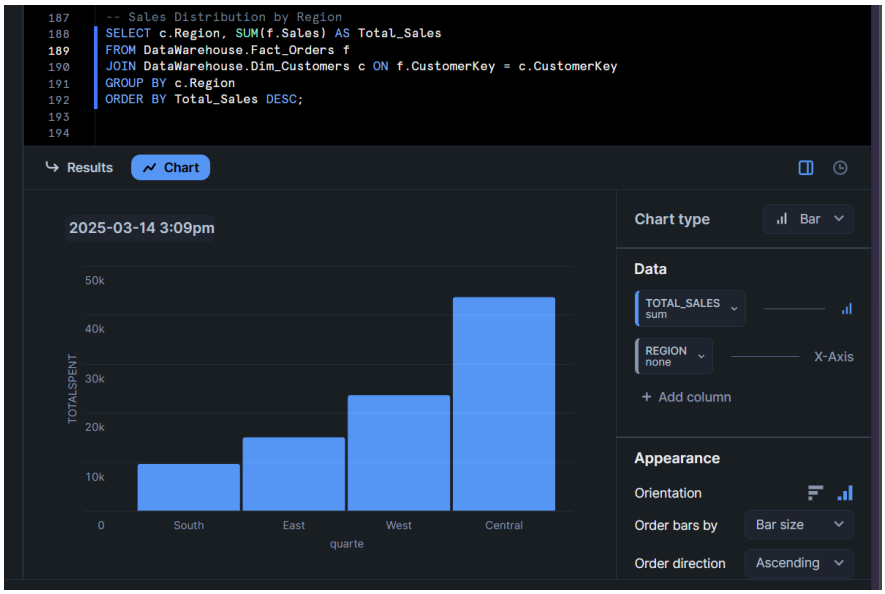
-- Customer Segment Performance
SELECT c.Segment, SUM(f.Sales) AS Total_Sales, AVG(f.Profit) AS Avg_Profit
FROM DataWarehouse.Fact_Orders f
JOIN DataWarehouse.Dim_Customers c ON f.CustomerKey = c.CustomerKey
GROUP BY c.Segment
ORDER BY Total_Sales DESC;



-- Most Profitable Products
SELECT p.ProductName, SUM(f.Profit) AS Total_Profit
FROM DataWarehouse.Fact_Orders f
JOIN DataWarehouse.Dim_Products p ON f.ProductKey = p.ProductKey
GROUP BY p.ProductName
ORDER BY Total_Profit DESC
LIMIT 20;



-- Sales Distribution by Region
SELECT c.Region, SUM(f.Sales) AS Total_Sales
FROM DataWarehouse.Fact_Orders f
JOIN DataWarehouse.Dim_Customers c ON f.CustomerKey = c.CustomerKey
GROUP BY c.Region
ORDER BY Total_Sales DESC;



Summary

This project aimed to build a Retail Sales Data Warehouse in Snowflake, leveraging Google Cloud Storage (GCS) for data ingestion and implementing a dimensional model for analytical insights. The key steps involved database setup, schema design, data integration, ETL processes, and analytical query execution.

Project Approach

1. Database and Schema Setup:
    - Created a RetailSales_DW database with Staging and DataWarehouse schemas.
    - The Staging schema temporarily held raw data, while the DataWarehouse schema stored transformed data in fact and dimension tables.
2. Google Cloud Storage Integration:
    - Configured external storage integration in Snowflake to connect with GCS.
    - Created an external stage (gcs_stage) to load data directly from GCS into Snowflake.
3. Staging Table Creation and Data Ingestion:
    - Defined staging tables (Staging.Dates, Staging.Customers, Staging.Products, and Staging.Orders).
    - Used COPY INTO commands to load CSV files from GCS into Snowflake.
4. Dimensional Model Implementation:
    - Constructed dimension tables (Dim_Dates, Dim_Customers, and Dim_Products) to store unique attributes of dates, customers, and products.

- Created a fact table (Fact_Orders) to store sales transactions, linking it with dimension tables via foreign keys.
- Used Auto Increment Primary Keys in dimension tables for efficient join performance.
- Ensured data integrity with referential constraints and default values for missing data.

5. Data Transformation & Loading:
    - Inserted distinct records from staging tables into respective dimension tables.
    - Transformed raw sales data into aggregated, structured form to support analysis.
    - Linked the fact table with customer and product dimensions using JOINs.

Challenges and Solutions

- Data Quality Issues: Some records had missing values. Solution: Used COALESCE to assign default values.
- Data Duplication: Applied DISTINCT during insert operations to remove duplicates.
- Slow Query Performance: Optimized joins and indexing to improve query speed.

Effectiveness of the Dimensional Model

- The star schema design simplified data retrieval and improved query performance.
- Fact and dimension tables provided better data organization for analytical processing.
- The model supported efficient aggregation of sales trends, customer behavior, and profitability analysis.

Key Analytical Insights

- Total Revenue by Category: Identified the most profitable product categories.
- Top Customers by Purchases: Ranked customers based on total sales.
- Quarterly Sales Trends: Analyzed seasonal revenue patterns.
- Customer Segment Performance: Evaluated sales and profit by customer segments.
- Most Profitable Products: Identified top-selling products based on total profit.
- Sales by Region: Compared revenue distribution across different regions.

Recommendations for Future Improvements

- Automate Data Pipelines: Implement scheduled ETL jobs using Snowflake Tasks.
- Enhance Data Validation: Apply data quality checks before loading into the warehouse.
- Expand Analytical Capabilities: Integrate with BI tools like Tableau or Power BI for visualization.