

**Data Warehouse Project Report**

**Name: Hamza Sabih**

**Roll number: 22i-1948**

**Section: DS-5C**

**1. Project Overview**

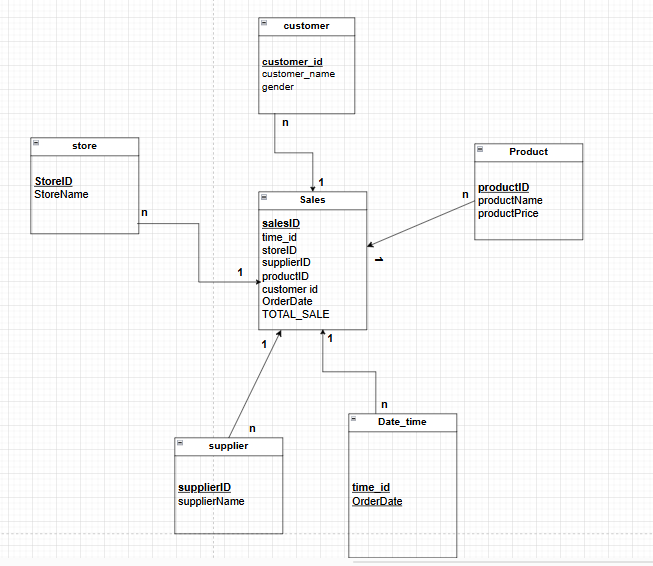
This project focuses on building a robust data warehouse (DW) tailored for analyzing a retail sales scenario. The project comprises four key tasks: designing a star schema for the DW, implementing a MESHJOIN algorithm for efficient data loading, performing multidimensional data analysis, and reflecting on the project’s challenges and learnings.

The primary goal is to enable effective business intelligence by structuring data to support slicing, dicing, and drill-down operations. The project also emphasizes optimizing data ingestion processes using the MESHJOIN algorithm.

**2. Star Schema for the Data Warehouse**

The star schema is central to the data warehouse design, providing a straightforward structure with a fact table and associated dimension tables. The schema ensures simplicity and efficiency in query execution.

**Star Schema**

The star schema consists of the following:

**Fact Table: Sales**

The fact table sales captures transactional data and serves as the core of the star schema. It includes the following attributes:

* **salesID**: A unique identifier for each transaction.
* **time\_id**: A foreign key linking to the Date\_time dimension for temporal analysis.
* **storeID**: A foreign key linking to the store dimension to analyze sales by store.
* **supplierID**: A foreign key linking to the supplier dimension for supplier-level sales.
* **productID**: A foreign key linking to the Product dimension for product-level sales insights.
* **customer\_id**: A foreign key linking to the customer dimension for customer segmentation.
* **OrderDate**: The date and time of the transaction.
* **TOTAL\_SALE**: The revenue generated from the transaction.

**Dimension Tables**

1. **customer**:
   * Captures demographic data about customers.
   * Attributes:
     + **customer\_id**: Primary key.
     + **customer\_name**: Name of the customer.
     + **gender**: Gender of the customer (Male/Female).
2. **Product**:
   * Stores information about products sold.
   * Attributes:
     + **productID**: Primary key.
     + **productName**: Name of the product.
     + **productPrice**: Price of the product.
     + **supplierID**: Links to the supplier dimension.
     + **supplierName**: Name of the supplier.
     + **storeID**: Links to the store dimension.
     + **storeName**: Name of the store.
3. **supplier**:
   * Provides details about suppliers.
   * Attributes:
     + **supplierID**: Primary key.
     + **supplierName**: Name of the supplier.
4. **store**:
   * Contains information about stores.
   * Attributes:
     + **storeID**: Primary key.
     + **storeName**: Name of the store.
5. **Date\_time**:
   * Represents the temporal aspect of sales transactions.
   * Attributes:
     + **time\_id**: Primary key.
     + **OrderDate**: The date and time of the order.

**3. Implementation of the MESHJOIN Algorithm**

**Code**

Java

package DWProject;

import java.io.BufferedReader;

import java.io.FileReader;

import java.io.IOException;

import java.sql.\*;

import java.text.ParseException;

import java.text.SimpleDateFormat;

import java.util.Date;

public class DWProject {

public static void main(String[] args) throws IOException {

Connection con = null;

PreparedStatement pstmt = null;

PreparedStatement productStmt = null;

PreparedStatement storeStmt = null;

PreparedStatement supplierStmt = null;

PreparedStatement timeStmt = null;

PreparedStatement customerStmt = null;

ResultSet rs = null;

BufferedReader reader = null;

String filePath = "C:\\Users\\Neelofar Wasi\\Desktop\\Semester 5\\Data Warehousing\\transactions.csv";

try {

// Establish the database connection

Class.forName("com.mysql.cj.jdbc.Driver");

con = DriverManager.getConnection("jdbc:mysql://localhost:3306/dw", "root", "mrskoochi1024");

System.out.println("Connection established successfully!");

// Prepare SQL statements

String salesInsertSQL = "INSERT INTO sales (time\_id, storeID, supplierID, productID, OrderID, customer\_id, OrderDate, TOTAL\_SALE) "

+ "VALUES (?, ?, ?, ?, ?, ?, ?, ?)";

pstmt = con.prepareStatement(salesInsertSQL);

String productQuerySQL = "SELECT productPrice, supplierID, storeID FROM Product WHERE productID = ?";

productStmt = con.prepareStatement(productQuerySQL);

String timeQuerySQL = "SELECT time\_id FROM Date\_time WHERE OrderDate = ?";

timeStmt = con.prepareStatement(timeQuerySQL);

String customerQuerySQL = "SELECT customer\_id FROM customer WHERE customer\_id = ?";

customerStmt = con.prepareStatement(customerQuerySQL);

// Read the CSV file

reader = new BufferedReader(new FileReader(filePath));

String line;

int rowCount = 0;

// Read and process each row in the CSV file

while ((line = reader.readLine()) != null) {

String[] data = line.split(",");

// Skip the header row

if (rowCount == 0) {

rowCount++;

continue;

}

try {

int orderID = Integer.parseInt(data[0]); // From CSV

String orderDateStr = data[1]; // From CSV

int productID = Integer.parseInt(data[2]); // From CSV

int quantityOrdered = Integer.parseInt(data[3]); // From CSV

int customerID = Integer.parseInt(data[4]); // From CSV

int timeID = Integer.parseInt(data[5]); // From CSV (already mapped to Date\_time)

// Parse the OrderDate field

SimpleDateFormat inputDateFormat = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

SimpleDateFormat outputDateFormat = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

Date orderDate = inputDateFormat.parse(orderDateStr);

String formattedOrderDate = outputDateFormat.format(orderDate);

// Retrieve productPrice, supplierID, and storeID from the Product table

productStmt.setInt(1, productID);

rs = productStmt.executeQuery();

if (rs.next()) {

double productPrice = rs.getDouble("productPrice");

int supplierID = rs.getInt("supplierID");

int storeID = rs.getInt("storeID");

// Validate customer\_id exists in the customer dimension

customerStmt.setInt(1, customerID);

ResultSet customerRs = customerStmt.executeQuery();

if (!customerRs.next()) {

System.out.println("CustomerID " + customerID + " not found. Skipping row: " + line);

continue;

}

// Calculate TOTAL\_SALE

double totalSale = quantityOrdered \* productPrice;

// Insert into the sales fact table

pstmt.setInt(1, timeID); // time\_id from CSV

pstmt.setInt(2, storeID); // storeID from Product table

pstmt.setInt(3, supplierID); // supplierID from Product table

pstmt.setInt(4, productID); // From CSV

pstmt.setInt(5, orderID); // From CSV

pstmt.setInt(6, customerID); // From CSV

pstmt.setString(7, formattedOrderDate); // Parsed order date

pstmt.setDouble(8, totalSale); // Computed TOTAL\_SALE

pstmt.executeUpdate();

System.out.println("Inserted row for OrderID: " + orderID);

} else {

System.out.println("ProductID " + productID + " not found in Product table. Skipping row: " + line);

}

} catch (NumberFormatException e) {

System.out.println("Skipping row due to invalid numeric data: " + line);

} catch (ParseException e) {

System.out.println("Skipping row due to invalid date format: " + line);

}

}

System.out.println("Data processing complete!");

} catch (Exception e) {

e.printStackTrace();

} finally {

// Close resources

try {

if (rs != null) rs.close();

if (reader != null) reader.close();

if (productStmt != null) productStmt.close();

if (timeStmt != null) timeStmt.close();

if (customerStmt != null) customerStmt.close();

if (pstmt != null) pstmt.close();

if (con != null) con.close();

System.out.println("Database connection closed.");

} catch (SQLException e) {

e.printStackTrace();

}

}

}

}

**Description**

The MESHJOIN algorithm is designed to optimize the process of loading transactional data into a data warehouse by joining it with the master data (MD) stored in dimension tables. This implementation ensures consistency, eliminates redundant data, and facilitates efficient querying.

**Steps in the Implementation**

1. **Establish Database Connection**:
   * A connection is established to the data warehouse using JDBC, ensuring the seamless interaction between the transactional data and the dimension tables.
2. **Data Parsing and Preprocessing**:
   * The transactional data is read line by line from the CSV file, skipping the header row.
   * Each row is parsed into individual fields, including orderID, orderDate, productID, quantityOrdered, and customerID.
3. **Dimension Table Lookup**:
   * The algorithm retrieves data from dimension tables to enrich the transactional data:
     + Product table for productPrice, supplierID, and storeID.
     + Date\_time table for mapping timeID.
     + customer table for validating customerID.
   * Missing data or invalid references are flagged, and those rows are skipped.
4. **Calculate Derived Metrics**:
   * The total sale value (TOTAL\_SALE) is computed as the product of quantityOrdered and productPrice.
5. **Fact Table Insertion**:
   * The enriched and validated data is inserted into the sales fact table with relevant foreign keys and derived metrics.
6. **Error Handling**:
   * The implementation incorporates robust error handling for issues like invalid numeric data or date formats.

**4. Analysis on the Data Warehouse**

Multidimensional analysis was conducted using slicing, dicing, drill-down, and materializing view techniques. The following queries illustrate these operations:

1. **Slicing**:
   * Example: Filtered sales data for a specific store and year.
2. **Dicing**:
   * Example: Analyzed monthly revenue trends across multiple stores and suppliers.
3. **Drill-Down**:
   * Example: Examined product-level sales within a supplier for detailed insights.
4. **Materialized View**:
   * Example: Created a STORE\_QUARTERLY\_SALES view to optimize quarterly trend queries.

Each operation demonstrated DW’s capability to deliver granular insights quickly, aiding in decision-making.

**5. Shortcomings of the MESHJOIN Algorithm**

While MESHJOIN enhances the ETL process, it has certain limitations:

1. **Dependency on Memory**: The algorithm’s performance is highly dependent on available memory for caching master data segments.
2. **Complexity in Implementation**: Efficient partitioning and indexing require careful design, increasing development time.
3. **Limited Parallelism**: MESHJOIN can be challenging to parallelize effectively, especially with large-scale data.
4. **Single-Threaded Processing**: The current implementation processes data sequentially, which may lead to inefficiencies for large datasets.
5. **Limited Error Correction**: While invalid rows are flagged, no corrective measures are taken, leading to potential data loss.
6. **Scalability Constraints**: The reliance on real-time lookups for dimension tables may introduce performance bottlenecks as data volume increases.

**6. Learnings from the Project**

This project provided significant insights into data warehousing and ETL processes. Key takeaways include:

* The importance of a well-structured star schema in enabling fast and intuitive querying.
* The practical challenges and benefits of implementing algorithms like MESHJOIN.
* The value of multidimensional analysis in uncovering actionable business intelligence.
* Different perspective of SQL queries, From OLTP To OLAP.
* Learned how OLAP queries look and how they are different from everyday queries.

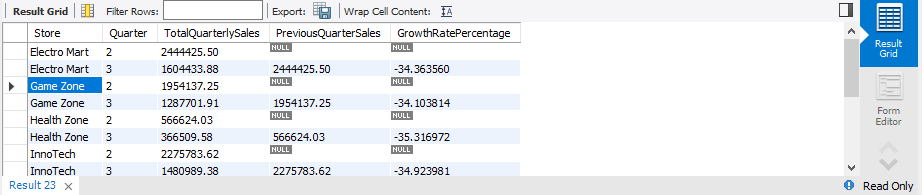
**OLAP Queries**

Q1. Top Revenue-Generating Products on Weekdays and Weekends with Monthly Drill-Down. Find the top 5 products that generated the highest revenue, separated by weekday and weekend sales, with results grouped by month for a specified year.

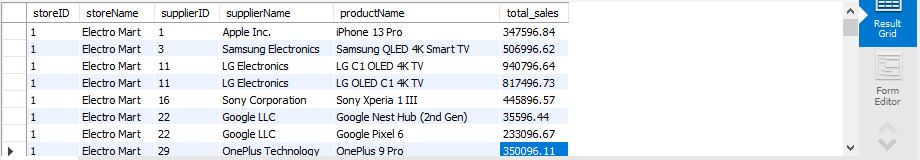
A screenshot of a computer

Description automatically generated

Q2. Trend Analysis of Store Revenue Growth Rate Quarterly for 2017. Calculate the revenue growth rate for each store on a quarterly basis for 2017.



Q3. Detailed Supplier Sales Contribution by Store and Product Name. For each store, show the total sales contribution of each supplier broken down by product name. The output should group results by store, then supplier, and then product name under each supplier.

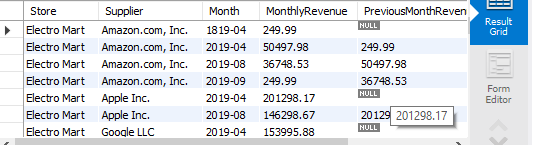


Q4. Seasonal Analysis of Product Sales Using Dynamic Drill-Down. Present total sales for each product, drilled down by seasonal periods (Spring, Summer, Fall, Winter). This can help understand product performance across seasonal periods.

A screenshot of a computer

Description automatically generated

Q5. Store-Wise and Supplier-Wise Monthly Revenue Volatility. Calculate the month-to-month revenue volatility for each store and supplier pair. Volatility can be defined as the percentage change in revenue from one month to the next, helping identify store or suppliers with highly fluctuating sales.



Q6. Top 5 Products Purchased Together Across Multiple Orders (Product Affinity Analysis). Identify the top 5 products frequently bought together within a set of orders (i.e., multiple products purchased in the same transaction). This product affinity analysis could inform potential product bundling strategies.

**Problem**: All OrderIDs have only one product:

-- This means product affinity analysis (identifying frequently bought-together products) is not possible with the current data, as it requires orders with multiple products.

-- hence shifting to Product Popularity Analysis Script

-- This query identifies the most purchased products and their total revenue.

A screenshot of a computer

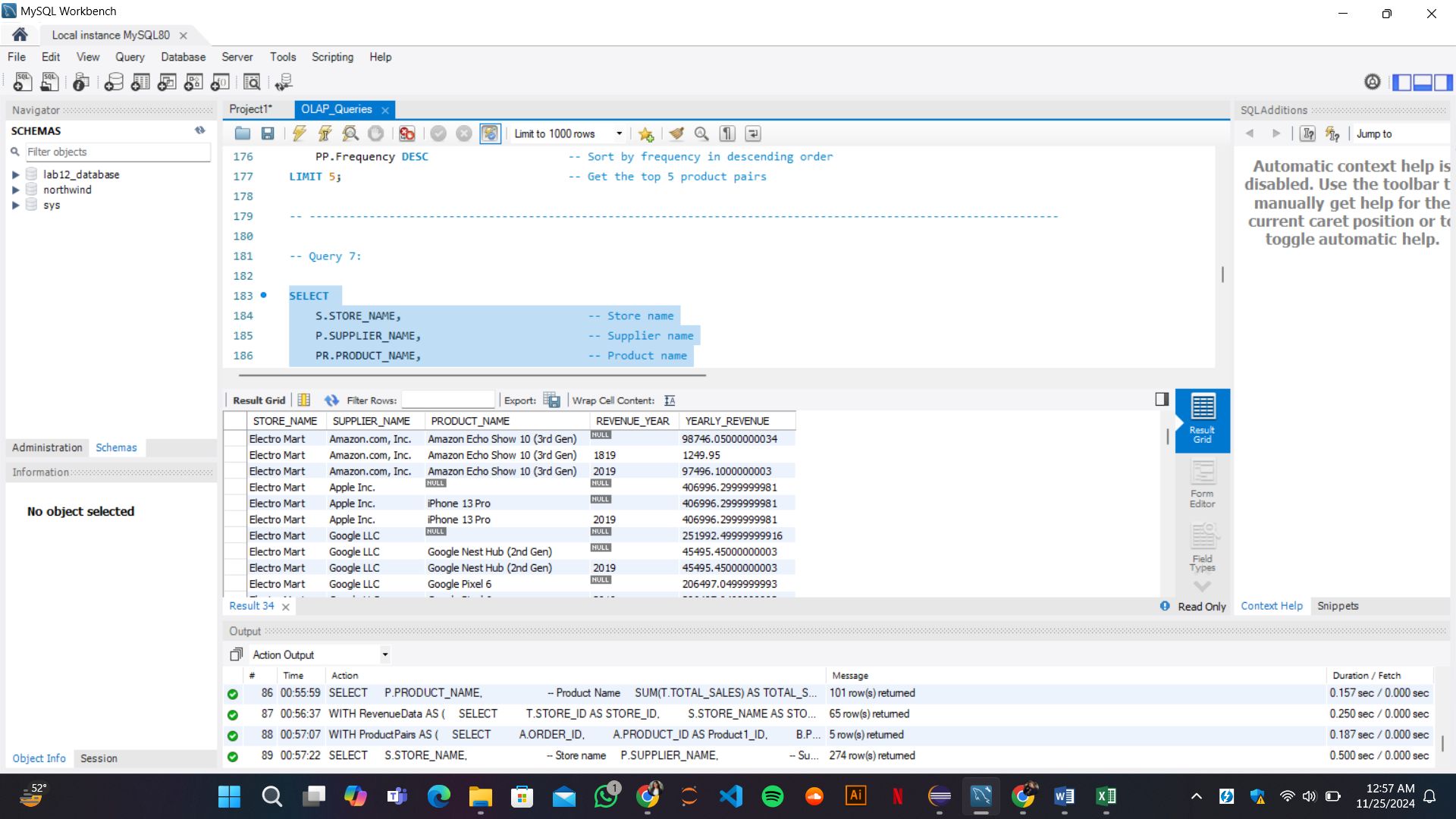
Description automatically generated

Hence I showed top 5 most selling items. Here is the screen shot below

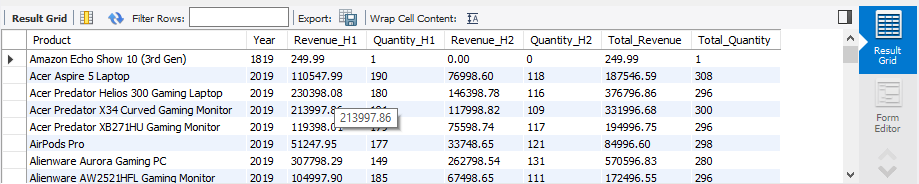
A screenshot of a computer

Description automatically generated

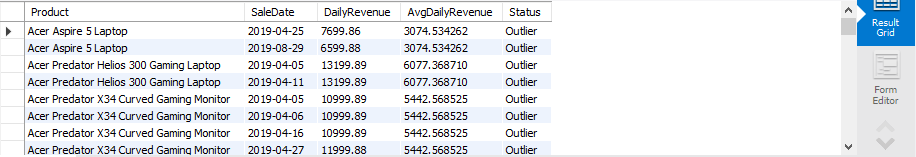
Q7. Yearly Revenue Trends by Store, Supplier, and Product with ROLLUP. Use the ROLLUP operation to aggregate yearly revenue data by store, supplier, and product, enabling a comprehensive overview from individual product-level details up to total revenue per store. This query should provide an overview of cumulative and hierarchical sales figures.



Q8. Revenue and Volume-Based Sales Analysis for Each Product for H1 and H2. For each product, calculate the total revenue and quantity sold in the first and second halves of the year, along with yearly totals. This split-by-time-period analysis can reveal changes in product popularity or demand over the year.



Q9. Identify High Revenue Spikes in Product Sales and Highlight Outliers. Calculate daily average sales for each product and flag days where the sales exceed twice the daily average by product as potential outliers or spikes. Explain any identified anomalies in the report, as these may indicate unusual demand events.



Q10. Create a View STORE\_QUARTERLY\_SALES for Optimized Sales Analysis. Create a view named STORE\_QUARTERLY\_SALES that aggregates total quarterly sales by store, ordered by store name. This view allows quick retrieval of store-specific trends across quarters, significantly improving query performance for regular sales analysis.

