Admin

data warehouse project report

**Table of Content**

1. **Objective**
2. **SQL database schema**
3. **OLAP queries for business analysis**
4. **Java code overview**
5. **Why use a MESH**
6. **Output overview of SQL and java code execution**
7. **What did I learned from the project?**
8. **3 shortcomings of MESH join**
9. **Conclusion**

**Objective:**

The task involves working with a relational database system (RDBMS) to design a schema for tracking product sales, and then using Java to load and enrich the data, incorporating data from various dimension tables such as products, customers, and time.

**SQL Database Schema:**

1. **Database Creation**:
   * The database proj is created for tracking sales data.
   * Tables for dimensions (dim\_product, dim\_customer, dim\_time) and a fact table (fact\_sales) are created.
2. **Table Structure**:
   * **Dimension Tables**:
     + dim\_product: Contains product information such as ProductID, ProductName, ProductPrice, and supplier/store details.
     + dim\_customer: Stores customer details like CustomerID, CustomerName, and Gender.
     + dim\_time: Contains time-related information (TimeID, TimeDetails) such as day of the week, month, or season.
   * **Fact Table**:
     + fact\_sales: This table tracks individual transactions (OrderID, OrderDate, ProductID, Quantity, CustomerID, TimeID, TotalSale), with foreign keys referencing the dimension tables. It captures transactional data (such as quantity and total sales amount) for each order.
3. **Data Loading**:
   * The data is loaded into the respective tables using SELECT queries to display the contents of the dimension and fact tables.
   * Before loading the data has to pre-process it as there were issues in data. For instance, had to remove dollar sign from product data. And had to separate date and time into separate columns but it was totally optional.
   * Queries include operations like joining tables to calculate revenue, revenue growth, product sales contributions, and time-based analyses.

**OLAP Queries for Business Analysis:**

The provided SQL queries focus on analyzing sales data across different dimensions and time periods using OLAP (Online Analytical Processing) techniques, specifically grouping, rolling up, and drilling down into the data.

1. **Top Revenue-Generating Products**:
   * **Query**: Identifies top products generating the most revenue during weekdays and weekends, with a monthly drill-down.
   * **Mesh Usage**: Mesh here refers to the interaction of different dimension attributes (ProductName, DayType, Month) in a multidimensional query that categorizes sales across different temporal aspects (e.g., weekdays vs. weekends).
2. **Quarterly Revenue Growth**:
   * **Query**: Analyzes store revenue growth by quarter for 2019, incorporating the LAG function for growth rate calculation.
   * **Mesh Usage**: This query uses multiple dimensions (StoreID, Year, Quarter) and applies a time-based drill-down (quarterly comparison) to measure growth.
3. **Detailed Supplier Sales**:
   * **Query**: Tracks sales contributions from suppliers, categorized by store and product.
   * **Mesh Usage**: The mesh here connects multiple dimension tables (StoreID, SupplierName, ProductName) to provide granular insights into supplier performance at the store level.
4. **Seasonal Analysis**:
   * **Query**: Analyzes product sales by season, dynamically calculating sales per product for different seasons.
   * **Mesh Usage**: This utilizes a time-based mesh to segment sales into seasons and analyze product performance across seasonal periods.
5. **Monthly Revenue Volatility**:
   * **Query**: Analyzes store-wise and supplier-wise revenue volatility across months.
   * **Mesh Usage**: The mesh connects StoreID, SupplierName, and Month, allowing for the detection of sales volatility and highlighting performance differences between suppliers and stores.
6. **Top Products Purchased Together**:
   * **Query**: Identifies pairs of products frequently bought together in the same order.
   * **Mesh Usage**: This query looks at combinations of products within the same OrderID, showcasing the interrelation of items purchased together.
7. **Yearly Revenue Trends with ROLLUP**:
   * **Query**: Uses the ROLLUP function to analyze yearly revenue trends across multiple dimensions (store, supplier, product).
   * **Mesh Usage**: ROLLUP creates a multi-level hierarchical mesh, allowing for summary statistics at different aggregation levels (e.g., yearly totals, store-specific totals).
8. **Sales Analysis for H1 and H2**:
   * **Query**: Analyzes product sales for the first and second halves of the year.
   * **Mesh Usage**: This involves a half-yearly drill-down that categorizes sales into two distinct periods (H1 and H2).
9. **High Revenue Spikes**:
   * **Query**: Identifies significant spikes in daily product sales.
   * **Mesh Usage**: The mesh combines daily sales with historical averages to flag outliers, marking extreme spikes in product sales compared to typical performance.
10. **Store Quarterly Sales View**:
    * **Query**: Creates a view to store aggregated quarterly sales for each store.
    * **Mesh Usage**: This involves grouping sales data by store and quarter, allowing for efficient quarterly reporting.

**Java Code Overview:**

The Java code focuses on interacting with the database to load and enrich data from the fact\_sales table and its related dimension tables (dim\_product, dim\_customer, dim\_time). It does the following:

1. **Establish Database Connection**:
   * Prompts the user for database connection details (URL, username, and password) and establishes a connection using JDBC.
2. **Data Enrichment**:
   * For each record in the fact\_sales table, the program fetches additional details from the related dimension tables to enhance the transactional data.
   * For example, the product name, customer name, and time details are fetched for each sale, and the TotalSale is recalculated based on the product price and quantity.
3. **Data Update/Insertion**:
   * Checks if the transaction already exists in the fact\_sales table. If it does, the total sale value is updated; otherwise, a new record is inserted.
4. **Logging**:
   * Logs the details of the enriched records, including the product name, customer name, order date, and calculated total sale for each transaction.

**Why Use a Mesh:**

The "mesh" in the context of this project refers to the interconnections between multiple dimension tables and the fact table, enabling the creation of complex multidimensional queries. By linking various dimensions (e.g., product, customer, time, store), the system can analyze data across multiple perspectives and uncover insights like:

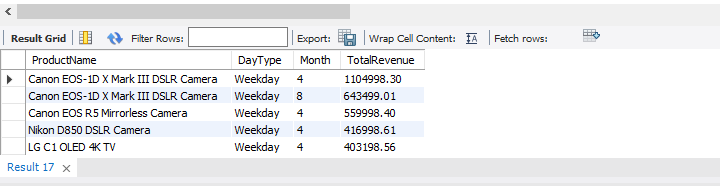
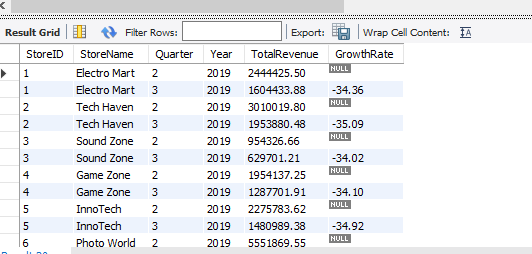
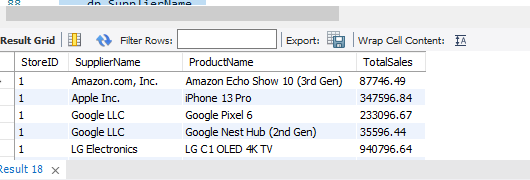
* How product sales differ across seasons or months.
* Identifying high-revenue products or stores.
* Analyzing growth rates or volatility in revenue over time.

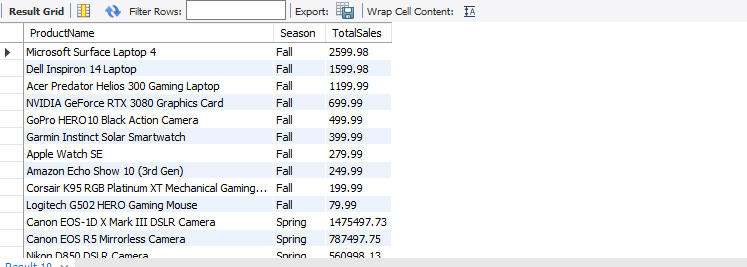
These connections allow for sophisticated analysis and reporting, which is the foundation of OLAP and business intelligence systems. The mesh facilitates drill-downs, roll-ups, and the identification of trends, which are crucial for decision-making in sales, marketing, and inventory management.

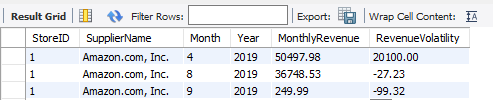
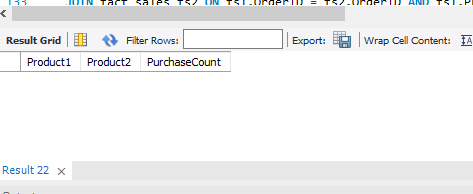
**Output Overview for SQL and Java Code Execution:**

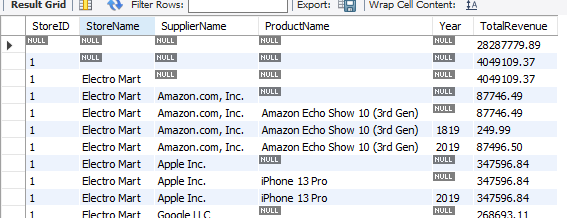
**1. SQL Output:**

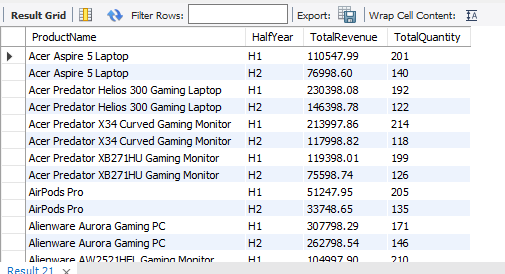
The SQL queries provided in the code produce various outputs related to sales analysis, such as revenue generation, growth rates, seasonal trends, and high-revenue spikes. Here’s a breakdown of the expected output for each query:

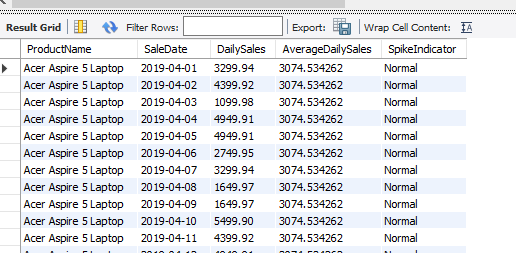
1. **Top Revenue-Generating Products on Weekdays and Weekends**:
   * **Output**: This query returns the top 5 products generating the most revenue during weekdays and weekends, broken down by month. The output will list product names, day types (Weekday/Weekend), months, and total revenue.
   *  **Output**:
2. **Quarterly Revenue Growth for 2019**:
   * **Output**: This query calculates the revenue growth for each store, quarter-wise for 2019. It also includes a percentage growth rate.
   *  **Output**:
3. **Supplier Sales Contribution by Store and Product**:
   * **Output**: Displays total sales per store and product for each supplier.
   * **Output**:
4. **Seasonal Analysis of Product Sales**:
   * **Output**: Analyzes sales of products by season (Spring, Summer, Fall, Winter).
   * **Output**:



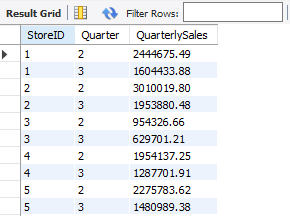
1. **Store-Wise and Supplier-Wise Monthly Revenue Volatility**:
   * **Output**: Shows how monthly revenue changes, calculating revenue volatility for each store and supplier.
   * **Output**:
2. **Top 5 Products Purchased Together Across Multiple Orders**:
   * **Output**: Lists the top 5 product pairs that are often purchased together in the same transaction.
   * **Output**:
3. **Yearly Revenue Trends by Store, Supplier, and Product with ROLLUP**:
   * **Output**: Shows yearly revenue trends for stores, suppliers, and products, with summary rows using ROLLUP.
   * **Output**:

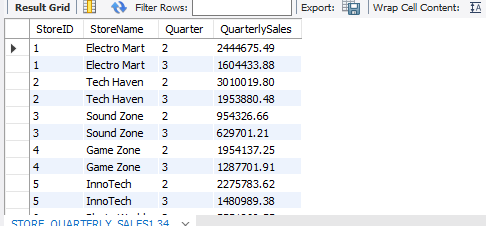


1. **Revenue and Volume-Based Sales Analysis for Each Product for H1 and H2**:
   * **Output**: Displays the sales revenue and quantities for each product split between the first and second halves of the year.
   * **Output**:
2. **High Revenue Spikes in Product Sales**:
   * **Output**: Identifies dates with exceptionally high sales, flagging them as "spikes" if sales exceed twice the average daily sales.
   * **Output**:



1. **Store Quarterly Sales View**:
   * **Output**: Displays aggregated quarterly sales per store.
   * **Output**:

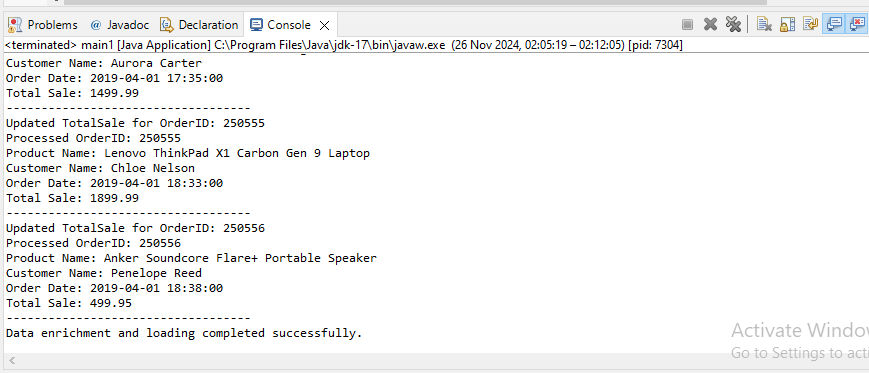




**2. Java Mesh join Output:**

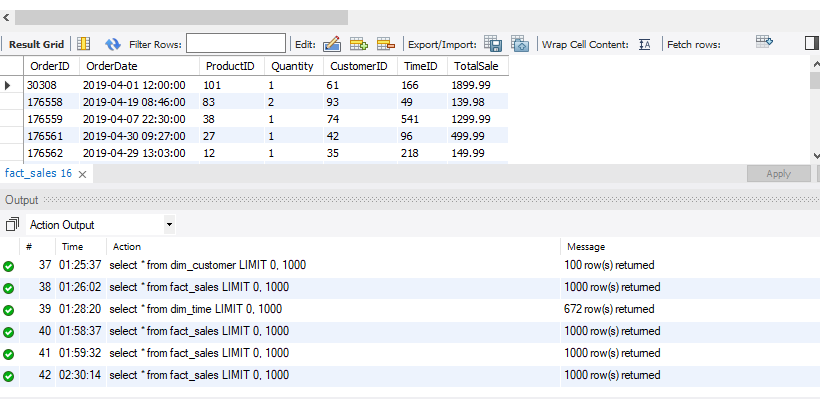
The Java code enriches and loads the data into the fact\_sales table from the related dimension tables. The expected output from running the Java code is as follows:

1. **Successful Database Connection**:
   * The message "Connected to the database." confirms that the connection to the database was established successfully.
2. **Data Processing for Each Order**:
   * For each order in the fact\_sales table, the program fetches related data (product name, customer details, time details) and computes the TotalSale value based on product price and quantity.
   * For each order, the following information is logged:



1. **Data Insertion/Update**:
   * If the order exists, the system updates the TotalSale value.
   * If the order is new, the system inserts a new record into the fact\_sales table.
   * Log messages indicate whether a record was updated or inserted
2. **Completion**:
   * After processing all records, the system prints: "Data enrichment and loading completed successfully."

This output indicates that the Java program has successfully enriched the data by fetching additional details from the dimension tables and updated the fact table accordingly.



**What Did I Learn from the Project?**

The project demonstrated the use of **relational database design**, **OLAP analysis**, and **data enrichment** using both SQL and Java. Key takeaways include:

1. **Importance of Dimensional Modeling**:
   * The use of **dimension tables** (dim\_product, dim\_customer, dim\_time) and a **fact table** (fact\_sales) allows for a **structured** way of analyzing and reporting sales data. It is a typical design in **Data Warehousing** (DW) systems and enables **complex reporting** and **multidimensional analysis** (OLAP).
2. **OLAP Queries and Business Intelligence**:
   * By utilizing **complex SQL queries** (such as GROUP BY, ROLLUP, and LAG), we can perform detailed analyses like revenue growth, seasonal trends, and identifying high-performing products and stores.
   * The project showcases how **sales data** can be segmented by time (months, quarters, seasons) and customer or store dimensions for targeted business insights.
3. **Data Enrichment and Transformation Using Java**:
   * The Java application shows how to **enrich transactional data** from a **fact table** by **joining** it with dimension tables (e.g., pulling product names, customer details, and time information). It also demonstrates the use of **JDBC** to interact with a MySQL database.
   * The program also ensures **data integrity** by updating or inserting records into the fact\_sales table based on whether the data already exists, which is important for **data consistency**.
4. **SQL-Driven Data Analysis and Reporting**:
   * Through various queries, we learned how to generate business reports like **top revenue-generating products**, **seasonal sales trends**, and **supplier contributions**, all essential for strategic decision-making.
5. **Optimizing Sales Analysis with Views**:
   * The use of **views** (e.g., STORE\_QUARTERLY\_SALES1) helps in simplifying complex queries, making sales data accessible in an optimized and structured way for further analysis.

**3 Shortcomings of Mesh Join:**

While the **mesh join** approach offers great flexibility and power in multi-dimensional queries, it does come with its limitations. Some key shortcomings of mesh joins include:

1. **Complexity in Query Writing**:
   * **Mesh joins** often require multiple **self-joins** or **cross-table joins** which can make the SQL queries more complex and harder to read. For example, if you need to perform analysis across multiple dimensions (products, customers, time), the queries can become increasingly intricate and difficult to maintain. This complexity can lead to longer development cycles, higher chances of errors, and challenges in debugging.
2. **Performance Issues with Large Datasets**:
   * **Mesh joins** typically involve **multiple joins** across large tables (especially dimension tables and fact tables). As the dataset grows, the query performance can degrade significantly due to the computational overhead required to process these joins. **Joins on large fact tables** can be slow, and as the number of dimensions increases, the complexity of the join grows, resulting in long execution times, especially in OLAP systems with large-scale data.
3. **Risk of Cartesian Products**:
   * When working with complex mesh joins, there is a **higher risk of generating Cartesian products** if the join conditions are not carefully defined. This can result in an **explosion of data** that doesn't provide meaningful results, leading to incorrect analysis or unnecessary computational resources being consumed. Ensuring correct filtering and join conditions is crucial to prevent this from happening, but it can be tricky in multi-dimensional joins.

**Conclusion:**

The combination of SQL and Java facilitates an effective OLAP system for analyzing and managing sales data. By using a mesh of interconnected dimension tables, the system enables flexible, multidimensional analysis that is essential for making informed business decisions. **What did we learn?** From this project, we learned how to design and query relational databases for **multi-dimensional analysis**, how to enrich transactional data using **Java**, and how to apply **OLAP techniques** to generate insightful reports. While mesh joins offer powerful analysis capabilities, they also come with challenges such as complexity, performance degradation, and the potential for Cartesian products, which need to be managed carefully for efficient query performance.

Thank you!

☺