

Data Warehousing and Business Intelligence Project

Ayaan Khan | DS-D | 22I-2066

**Building and Analysing a Near-Real-Time Data
Warehouse Prototype for METRO Pakistan**

Project Report:

1. Project Overview

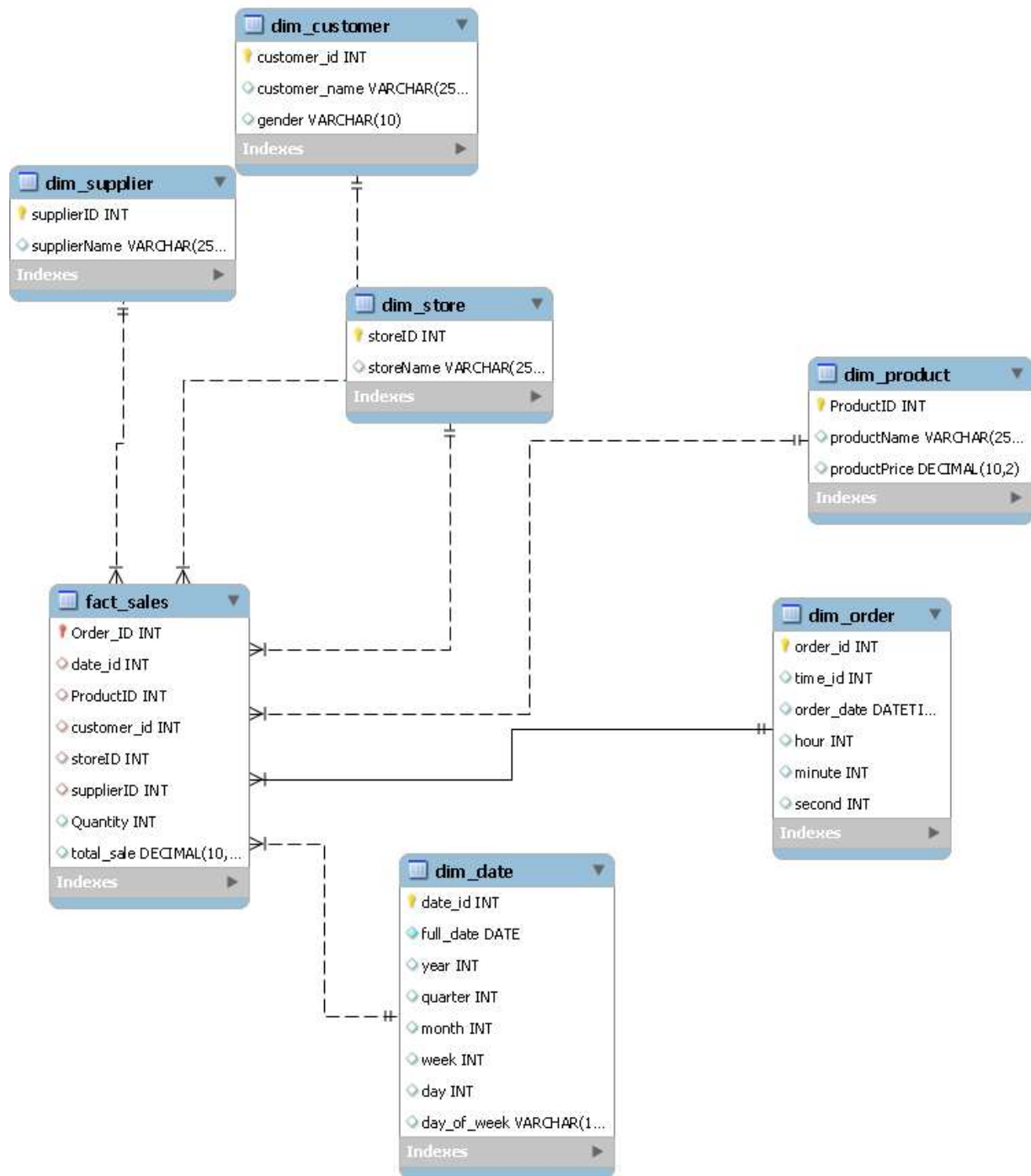
This project focuses on designing and implementing a near-real-time Data Warehouse (DW) prototype for METRO, a leading superstore chain in Pakistan. The objective is to facilitate business intelligence by analyzing customer behavior and sales patterns through real-time data integration. The star schema is used to organize data, and the MESHJOIN algorithm was implemented to enable the ETL(Extract-Transform-Load) process by enriching transactional data with master data.

The Data Warehouse allows advanced OLAP queries to be executed, providing insights such as revenue trends, seasonal product performance, and customer purchasing habits. The project will provide METRO with a platform to improve marketing strategies, manage inventory efficiently, and optimize operational processes.

2. Schema for Data Warehouse (DW)

The star schema implemented in this project is designed to support multidimensional analytical queries. The updated schema includes dimensions for product, customer, store, supplier, date, and time to provide granular insights:

Schema Diagram:



Fact Table: fact_sales

Captures transactional data with attributes:

- **Order_ID:** Unique identifier for each transaction.
- **date_id:** Links to dim_date.
- **ProductID:** Links to dim_product.
- **customer_id:** Links to dim_customer.
- **supplier_id:** Links to dim_supplier
- **storeID:** Links to dim_store.
- **Quantity:** Number of units sold.
- **total_sale:** Calculated as $\text{Quantity} \times \text{productPrice}$.

Dimension Tables:

1. **dim_product:** Details about products, including ProductID, productName, productPrice, supplierID, and supplierName.
2. **dim_customer:** Customer attributes like customer_id, customer_name, and gender.
3. **dim_store:** Store-related data, including storeID and storeName.
4. **dim_supplier:** Supplier information (supplierID and supplierName).
5. **dim_date:** Date hierarchy with attributes like full_date, year, quarter, month, week, day, and day_of_week.
6. **dim_order:** Adds time granularity with attributes such as order_id, time_id, order_date, hour, minute, and second.

This schema supports slicing, dicing, and drill-down queries for advanced analytical capabilities.

3. MESHJOIN Algorithm

The MESHJOIN algorithm was implemented as the central component of the ETL pipeline to enrich transactional data in real time. The algorithm integrates stream data (transactions) with master data (products and customers) and processes it for DW storage.

Key Components:

- **Stream Buffer:** Temporarily holds incoming transactional data from the TRANSACTIONS table.
- **Disk Buffer:** Cyclically loads partitions of master data (products and customers) into memory.
- **Hash Table:** Indexes transactional data for quick lookups.
- **Queue:** Tracks the order of transactions to ensure they are fully processed.

Steps in MESHJOIN Implementation:

1. **Load Transactions:** Stream transactions are read into the hash table and queue.
2. **Process Master Data:** Master data is divided into partitions and cyclically loaded into the disk buffer.
3. **Join and Enrich:** Transactions are matched with master data, enriched with attributes like productName and customer_name, and calculated for total_sale.
4. **Load into DW:** Enriched data is inserted into the DW, avoiding duplicates in dimensions.
5. **Repeat:** The process continues until all transactions are fully processed.
6. The staggered processing ensures efficient use of memory while minimizing disk I/O operations.

4. Three Shortcomings of MESHJOIN

1. Memory Limitations:

High memory dependency can lead to performance bottlenecks when processing large datasets or handling high data velocities.

2. Latency for Late Transactions:

Transactions must stay in memory until all master data partitions are processed, introducing delays for transactions arriving later in the cycle.

3. Fixed Partition Sizes:

The need for fixed-size buffers makes the algorithm less adaptable to datasets with varying distributions or dynamic workloads.

5. OLAP Queries

Below are some key OLAP queries used for analysis.

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.

```
4  SELECT
5      p.productName,
6      d.month,
7      CASE
8          WHEN d.day_of_week IN ('Saturday', 'Sunday') THEN 'Weekend'
9          ELSE 'Weekday'
10     END AS day_type,
11     SUM(f.total_sale) AS total_revenue
12 FROM fact_sales f
```

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.

```
23 SELECT
24     s.storeName,
25     d.quarter,
26     SUM(f.total_sale) AS total_revenue,
27     LAG(SUM(f.total_sale)) OVER (PARTITION BY s.storeID ORDER BY d.quarter) AS previous_quarter_revenue,
28     CASE
29         WHEN LAG(SUM(f.total_sale)) OVER (PARTITION BY s.storeID ORDER BY d.quarter) IS NOT NULL
30         THEN (SUM(f.total_sale) - LAG(SUM(f.total_sale)) OVER (PARTITION BY s.storeID ORDER BY d.quarter)) / LAG(SUM(f
31         ELSE 0
32     END AS growth_rate
33 FROM fact_sales f
34 JOIN dim_store s ON f.storeID = s.storeID
35 JOIN dim_date d ON f.date_id = d.date_id
36 WHERE d.year = 2019
37 GROUP BY s.storeID, d.quarter
38 ORDER BY s.storeName, d.quarter;
```

Result Grid

	storeName	quarter	total_revenue	previous_quarter_revenue	growth_rate
▶	Electro Mart	2	2444425.50	NULL	0
	Electro Mart	3	1604433.88	2444425.50	-34.363560
	Game Zone	2	1954137.25	NULL	0
	Game Zone	3	1287701.91	1954137.25	-34.103814
	Health Zone	2	566624.03	NULL	0

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.

```
44 SELECT
45     st.storeName,
46     sp.supplierName,
47     p.productName,
48     SUM(f.total_sale) AS total_sales
49 FROM fact_sales f
50 JOIN dim_store st ON f.storeID = st.storeID
51 JOIN dim_product p ON f.ProductID = p.ProductID
52 JOIN dim_supplier sp ON p.supplierID = sp.supplierID
53 GROUP BY st.storeName, sp.supplierName, p.productName
54 ORDER BY st.storeName, sp.supplierName, p.productName;
```

Result Grid

	storeName	supplierName	productName	total_sales
▶	Electro Mart	Amazon.com Inc.	Amazon Echo Show 10 (3rd Gen)	87746.49
	Electro Mart	Apple Inc.	iPhone 13 Pro	347596.84
	Electro Mart	Google LLC	Google Nest Hub (2nd Gen)	35596.44
	Electro Mart	Google LLC	Google Pixel 6	233096.67
	Electro Mart	LG Electronics	LG C1 OLED 4K TV	940796.64

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.

```
59 SELECT
60     p.productName,
61     CASE
62         WHEN d.month IN (3, 4, 5) THEN 'Spring'
63         WHEN d.month IN (6, 7, 8) THEN 'Summer'
64         WHEN d.month IN (9, 10, 11) THEN 'Fall'
65         WHEN d.month IN (12, 1, 2) THEN 'Winter'
66     END AS season,
67     SUM(f.total_sale) AS total_sales
68 FROM fact_sales f
69 JOIN dim_product p ON f.ProductID = p.ProductID
70 JOIN dim_date d ON f.date_id = d.date_id
71 GROUP BY p.productName, season
72 ORDER BY season, total_sales DESC;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Cor

	productName	season	total_sales
▶	Microsoft Surface Laptop 4	Fall	2599.98
	Dell Inspiron 14 Laptop	Fall	1599.98
	Acer Predator Helios 300 Gaming Laptop	Fall	1199.99
	NVIDIA GeForce RTX 3080 Graphics Card	Fall	699.99
	GoPro HERO10 Black Action Camera	Fall	499.99

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 stores or suppliers with highly fluctuating sales.

```
77 SELECT
78     st.storeName,
79     sp.supplierName,
80     d.month,
81     SUM(f.total_sale) AS total_sales,
82     LAG(SUM(f.total_sale)) OVER (PARTITION BY st.storeID, sp.supplierID ORDER BY d.month) AS previous_month_sales,
83     CASE
84         WHEN LAG(SUM(f.total_sale)) OVER (PARTITION BY st.storeID, sp.supplierID ORDER BY d.month) IS NOT NULL
85         THEN (SUM(f.total_sale) - LAG(SUM(f.total_sale)) OVER (PARTITION BY st.storeID, sp.supplierID ORDER BY d.month)
86              / LAG(SUM(f.total_sale)) OVER (PARTITION BY st.storeID, sp.supplierID ORDER BY d.month) * 100)
87         ELSE 0
88     END AS revenue_volatility
89 FROM fact_sales f
90 JOIN dim_store st ON f.storeID = st.storeID
91 JOIN dim_product p ON f.ProductID = p.ProductID
92 JOIN dim_supplier sp ON p.supplierID = sp.supplierID
93 JOIN dim_date d ON f.date_id = d.date_id
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: |

	storeName	supplierName	month	total_sales	previous_month_sales	revenue_volatility
▶	Electro Mart	Amazon.com Inc.	4	50747.97	NULL	0
	Electro Mart	Amazon.com Inc.	8	36748.53	50747.97	-27.586207
	Electro Mart	Amazon.com Inc.	9	249.99	36748.53	-99.319728
	Electro Mart	Apple Inc.	4	201298.17	NULL	0
	Electro Mart	Apple Inc.	8	146298.67	201298.17	-27.322404

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.

```
101 SELECT
102     p1.productName AS product1,
103     p2.productName AS product2,
104     COUNT(*) AS frequency
105 FROM fact_sales f1
106 JOIN fact_sales f2 ON f1.Order_ID = f2.Order_ID AND f1.ProductID < f2.ProductID
107 JOIN dim_product p1 ON f1.ProductID = p1.ProductID
108 JOIN dim_product p2 ON f2.ProductID = p2.ProductID
109 GROUP BY p1.productName, p2.productName
110 ORDER BY frequency DESC
111 LIMIT 5;
```

Result Grid

Filter Rows: Export: Wrap Cell Content:

product1	product2	frequency
----------	----------	-----------

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.

```
117 SELECT
118     st.storeName,
119     sp.supplierName,
120     p.productName,
121     SUM(f.total_sale) AS total_sales
122 FROM fact_sales f
123 JOIN dim_store st ON f.storeID = st.storeID
124 JOIN dim_product p ON f.ProductID = p.ProductID
125 JOIN dim_supplier sp ON p.supplierID = sp.supplierID
126 GROUP BY st.storeName, sp.supplierName, p.productName WITH ROLLUP
127 ORDER BY st.storeName, sp.supplierName, p.productName;
```

Result Grid

Filter Rows: Export: Wrap Cell Content:

	storeName	supplierName	productName	total_sales
	NULL	NULL	NULL	28287779.89
	Electro Mart	NULL	NULL	4049109.37
	Electro Mart	Amazon.com Inc.	NULL	87746.49
	Electro Mart	Amazon.com Inc.	Amazon Echo Show 10 (3rd Gen)	87746.49
	Electro Mart	Apple Inc.	NULL	347596.84

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.

```
132 SELECT
133     p.productName,
134     SUM(CASE WHEN d.month <= 6 THEN f.total_sale ELSE 0 END) AS H1_total_sales,
135     SUM(CASE WHEN d.month <= 6 THEN f.Quantity ELSE 0 END) AS H1_quantity,
136     SUM(CASE WHEN d.month > 6 THEN f.total_sale ELSE 0 END) AS H2_total_sales,
137     SUM(CASE WHEN d.month > 6 THEN f.Quantity ELSE 0 END) AS H2_quantity,
138     SUM(f.total_sale) AS total_sales,
139     SUM(f.Quantity) AS total_quantity
140 FROM fact_sales f
141 JOIN dim_product p ON f.ProductID = p.ProductID
142 JOIN dim_date d ON f.date_id = d.date_id
143 GROUP BY p.productName
144 ORDER BY p.productName;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: |

productName	H1_total_sales	H1_quantity	H2_total_sales	H2_quantity	total_sales	total_quantity
Acer Aspire 5 Laptop	110547.99	201	76998.60	140	187546.59	341
Acer Predator Helios 300 Gaming Laptop	230398.08	192	146398.78	122	376796.86	314
Acer Predator X34 Curved Gaming Monitor	213997.86	214	117998.82	118	331996.68	332
Acer Predator XB271HU Gaming Monitor	119398.01	199	75598.74	126	194996.75	325
AirPods Pro	51247.95	205	33748.65	135	84996.60	340

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.

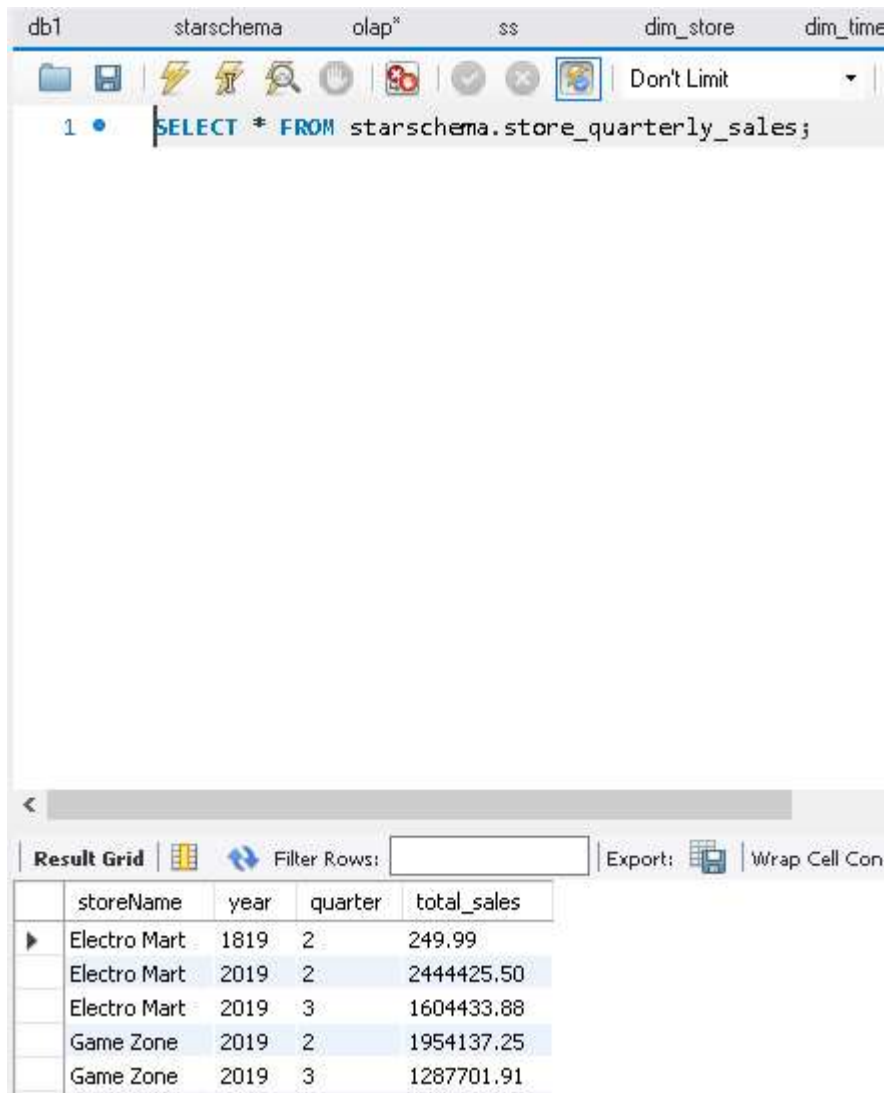
```
149 WITH avg_daily_sales AS (
150     SELECT
151         f.ProductID,
152         d.date_id,
153         d.full_date,
154         AVG(f.total_sale) AS avg_sales
155     FROM fact_sales f
156     JOIN dim_date d ON f.date_id = d.date_id
157     GROUP BY f.ProductID, d.date_id, d.full_date
158 )
159 SELECT
160     f.ProductID,
161     f.Order_ID,
162     f.total_sale,
163     a.avg_sales,
164     CASE
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: |

	ProductID	Order_ID	total_sale	avg_sales	sale_status
▶	1	176659	1099.99	1191.655833	Normal
	1	176718	1099.99	1539.986000	Normal
	1	176778	1099.99	1099.990000	Normal
	1	176914	1099.99	1099.990000	Normal

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.



The screenshot shows a SQL IDE interface. At the top, there are tabs for 'db1', 'starschema', 'olap*', 'ss', 'dim_store', and 'dim_time'. Below the tabs is a toolbar with various icons. The main query editor displays the following SQL statement:

```
1 • SELECT * FROM starschema.store_quarterly_sales;
```

Below the query editor, there is a 'Result Grid' section. It includes a 'Filter Rows:' input field and an 'Export:' button. The result grid displays the following data:

	storeName	year	quarter	total_sales
▶	Electro Mart	1819	2	249.99
	Electro Mart	2019	2	2444425.50
	Electro Mart	2019	3	1604433.88
	Game Zone	2019	2	1954137.25
	Game Zone	2019	3	1287701.91

6. What I Learned from the Project

This project provided hands-on experience in designing, implementing, and analyzing data warehouses. Key learnings include:

- **Star Schema Design:** Creating an effective star schema for multidimensional analysis.
- **Real-Time ETL Implementation:** Applying the MESHJOIN algorithm for enriching and loading transactional data into the DW.
- **OLAP Query Development:** Writing and optimizing complex queries for business intelligence.
- **Data Cleaning and Preprocessing:** Ensuring data consistency using Python scripts for transformations.
- **Granular Time Analysis:** Including a time dimension to support detailed temporal insights.

The project bridged theoretical knowledge and practical applications which is essential for real-world challenges in data warehousing and analytics.