Fall 2024, BSDS, Course- DW&BI

FAST - National University of Computer and Emerging Sciences

PROJECT

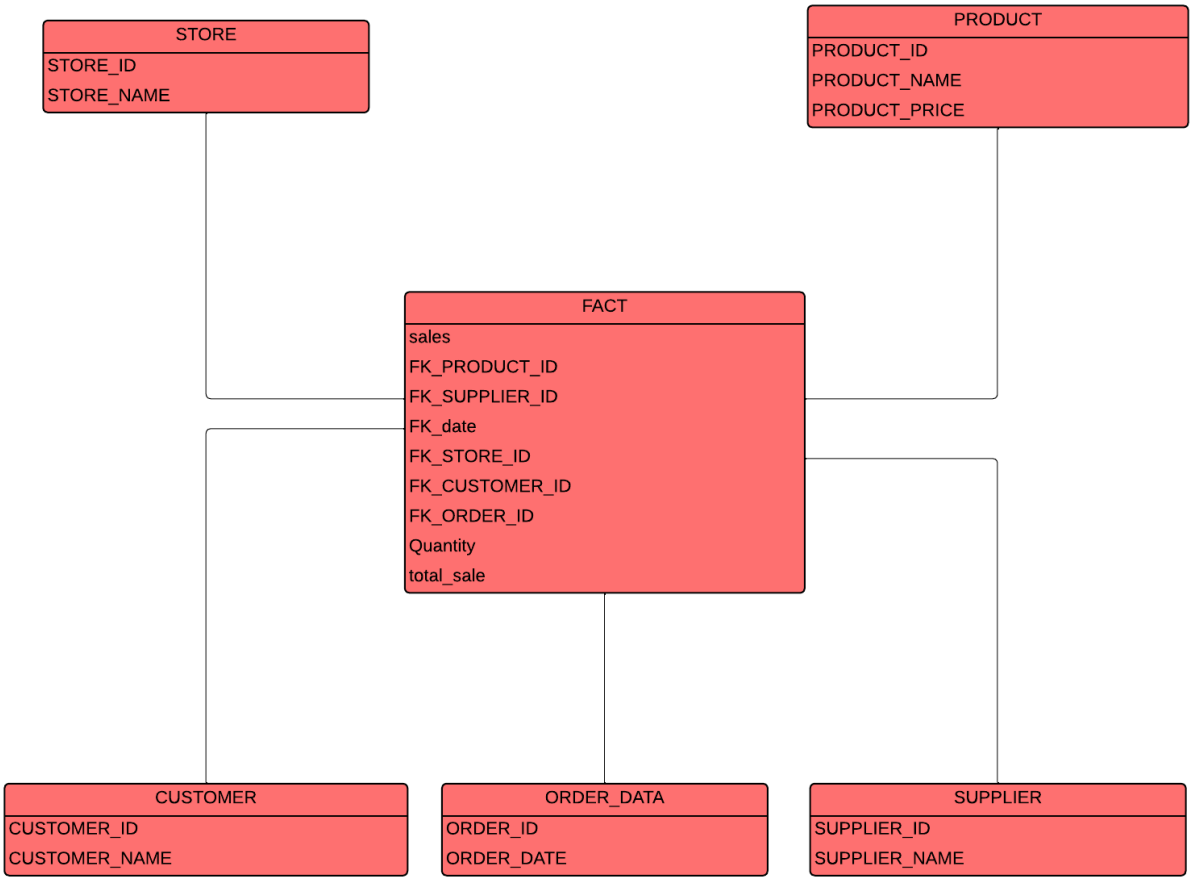
Submitted by:
Syeda Mahum Raza
i21-1662 (B)

The Metro Data Warehouse system is like a well-organized digital filing cabinet for a retail business.

Introduction:

The Metro Data Warehouse project is an advanced retail data management system designed to transform raw transactional data into meaningful business insights. The system collects, organizes, and analyzes data from multiple sources including sales transactions, customer information, and product details.

It uses a star schema design:



Steps Involved:

1. <u>Data Collection</u>
   - Gather raw data from three main sources:
     - Transactions database (sales records)
     - Customer database (customer information)
     - Products database (product and supplier details)

warehouse | transactions | products_data | queries

Limit to 1000 rows

```
        t.time_id
    FROM
        transactions.transactions t
    JOIN
        customer_data.customer_data c
    ON
        t.customer_id = c.customer_id;


    select *
    from transactions.enriched_transactions;
```

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

ORDER ID	ORDER DATE	ProductID	Quantity Ordered	customer_id	customer_name	gender	time_id
177831	2019-04-01 03:09:00	66	1	17	Ethan Martin	Male	1
185103	2019-04-01 05:18:00	47	1	80	Caroline Morgan	Female	4
191585	2019-04-01 05:58:00	38	1	20	Amelia Gonzalez	Female	5
186650	2019-04-01 06:11:00	2	1	55	Jonathan Bailey	Male	7
192285	2019-04-01 06:23:00	86	1	23	Alexander Walker	Male	10
182960	2019-04-01 06:47:00	45	1	4	Emily Brown	Female	13
186297	2019-04-01 07:01:00	85	1	48	Zoe Morris	Female	16
193621	2019-04-01 07:21:00	4	1	64	Nora Hughes	Female	21

enriched_transactions 4

Read O

Output

warehouse | transactions | products_data | queries

Limit to 1000 rows

```
SELECT * FROM products_data.products_data;

SELECT COLUMN_NAME, DATA_TYPE
FROM INFORMATION_SCHEMA.COLUMNS
WHERE TABLE_SCHEMA = 'products_data'
```

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

productID	productName	productPrice	supplierID	supplierName	storeID	storeName
2	Dell XPS 13 Laptop	1299.99	2	Dell Technologies	2	Tech Haven
21	DJI Mavic Air 2 Drone	799.99	20	DJI	6	Photo World
29	AOC CQ32G1 Curved Gaming Monitor	349.99	27	AOC International	4	Game Zone
237571	LG OLED C1 4K TV	2499.99	11	LG Electronics	1	Electro Mart
240480	2019-08-24 11:10:00	76.00	1	49	84	Sound Zone
238831	2019-08-24 11:13:00	26.00	27	Acer Inc.	4	Game Zone
241372	Lenovo ThinkPad X1 Carbon Gen 9 Laptop	1899.99	32	Lenovo Group	2	Tech Haven
237811	2019-08-24 11:23:00	99.99	1	43	89	Sound Zone
247049	2019-08-24 11:25:00	72.00	1	34	4	Game Zone
247745	2019-08-24 11:28:00	90.00	1	89	6	Photo World
240430	2019-08-24 11:32:00	10.00	12	Logitech	4	Game Zone
241479	2019-08-24 11:38:00	1799.99	8	Alienware (Dell)	5	InnoTech
247254	2019-08-24 11:48:00	8.00	1	41	99	Game Zone
237892	2019-08-24 11:57:00	86.00	1	4	102	Sound Zone
250310	2019-08-24 11:59:00	28.00	19	Canon Inc.	6	Photo World
240974	2019-08-24 12:00:00	31.00	1	Apple Inc.	2	Tech Haven

products_data 6

Read

Output

2. Data Cleaning
    - Fix inconsistent date formats
    - Remove special characters from prices ($)
    - Convert text-based numbers to proper numerical format
    - Combine matching customer and transaction records
    - Remove duplicate entries
    - Handle missing values
3. Data Transformation
    - Create enriched transaction records by joining customer and sales data
    - Convert prices to standard decimal format
    - Standardize date formats
    - Create proper relationships between different data tables
4. Data Loading
    - Load cleaned data into dimension tables:
        - Product table
        - Supplier table
        - Store table
        - Customer table
        - Order table

Concept of Slice and Dice:

- Slice and dice are a powerful feature of the warehouse that allows analyzing data from different perspectives.
    1. Slicing involves the concept of focusing on a specific slice of data.
        - View sales for a specific month
        - Look at transactions from a particular store
        - Analyze purchases by a specific customer group

2. Dicing allows you to ask query across different dimensions. Dicing allows you to ask query across different dimensions.

   - Analyze sales by store, product, and time period
   - Compare supplier performance across different stores and products
   - Study customer buying patterns across different seasons and locations

OLAP QUERIES:

```sql
-- Q1. Top Revenue-Generating Products on Weekdays and Weekends with Monthly Drill-Down
SELECT
    EXTRACT(YEAR FROM f.FK_date) AS year,
    EXTRACT(MONTH FROM f.FK_date) AS month,
    CASE
        WHEN DAYOFWEEK(f.FK_date) BETWEEN 2 AND 6 THEN 'Weekday'  -- Monday (2) to Friday (6)
        ELSE 'Weekend'  -- Sunday (1) and Saturday (7)
    END AS day_type,
    f.FK_PRODUCT_ID,
    SUM(f.total_sale) AS total_revenue
FROM FACT f
GROUP BY year, month, day_type, f.FK_PRODUCT_ID
ORDER BY total_revenue DESC
LIMIT 5;
```

year	month	day_type	FK_PRODUCT_ID	total_revenue
2019	4	Weekday	89	1221998.12
2019	8	Weekday	89	753998.84
2019	4	Weekend	89	389999.40
2019	4	Weekday	2	223598.28
2019	8	Weekend	89	181999.72

```sql
-- Q2. Trend Analysis of Store Revenue Growth Rate Quarterly for 2017
WITH quarterly_sales AS (
    SELECT f.FK_STORE_ID, EXTRACT(QUARTER FROM f.FK_date) AS quarter,SUM(f.total_sale) AS revenue
    FROM FACT f WHERE EXTRACT(YEAR FROM f.FK_date) = 2017 GROUP BY f.FK_STORE_ID, quarter )
SELECT qs.FK_STORE_ID, qs.quarter, qs.revenue,
LAG(qs.revenue) OVER (PARTITION BY qs.FK_STORE_ID ORDER BY qs.quarter) AS previous_quarter_revenue,
COALESCE(((qs.revenue - LAG(qs.revenue)
 OVER (PARTITION BY qs.FK_STORE_ID ORDER BY qs.quarter)) / LAG(qs.revenue)
 OVER (PARTITION BY qs.FK_STORE_ID ORDER BY qs.quarter)) * 100, 0) AS growth_rate
FROM quarterly_sales qs
ORDER BY qs.FK_STORE_ID, qs.quarter;
-- checked no data of 2017 in csv hence empty --
```

FK_STORE_ID	quarter	revenue	previous_quarter_revenue	growth_rate

warehouse | transactions | products_data | queries*

Limit to 1000 rows

```sql
-- Q3. Detailed Supplier Sales Contribution by Store and Product Name

SELECT
    f.FK_STORE_ID,
    f.FK_SUPPLIER_ID,
    f.FK_PRODUCT_ID,
    SUM(f.total_sale) AS total_sales_contribution
FROM FACT f
GROUP BY f.FK_STORE_ID, f.FK_SUPPLIER_ID, f.FK_PRODUCT_ID
ORDER BY f.FK_STORE_ID, f.FK_SUPPLIER_ID, f.FK_PRODUCT_ID;
```

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

FK_STORE_ID	FK_SUPPLIER_ID	FK_PRODUCT_ID	total_sales_contribution
1	1	1	404796.32
2	1	31	489595.92
2	1	62	29380.00
2	2	2	488796.24
2	2	88	271996.60

Result 28 | Read Only

warehouse | transactions | products_data | queries*

Limit to 1000 rows

```sql
-- Q4. Seasonal Analysis of Product Sales Using Dynamic Drill-Down

SELECT f.FK_PRODUCT_ID,
CASE
        WHEN EXTRACT(MONTH FROM f.FK_date) IN (3, 4, 5) THEN 'Spring'
        WHEN EXTRACT(MONTH FROM f.FK_date) IN (6, 7, 8) THEN 'Summer'
        WHEN EXTRACT(MONTH FROM f.FK_date) IN (9, 10, 11) THEN 'Fall'
        ELSE 'Winter'  -- December, January, February
    END AS season,
    SUM(f.total_sale) AS total_sales
FROM FACT f
GROUP BY f.FK_PRODUCT_ID, season
ORDER BY f.FK_PRODUCT_ID, season;
```

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

FK_PRODUCT_ID	season	total_sales
1	Spring	255197.68
1	Summer	149598.64
13	Spring	11400.00
13	Summer	6840.00
2	Spring	332797.44

Result 29 | Read Only

warehouse | transactions | products_data | queries*

Limit to 1000 rows

```sql
-- Q5. Store-Wise and Supplier-Wise Monthly Revenue Volatility
WITH monthly_sales AS (
SELECT f.FK_STORE_ID, f.FK_SUPPLIER_ID,
EXTRACT(YEAR FROM f.FK_date) AS year, EXTRACT(MONTH FROM f.FK_date) AS month,
SUM(f.total_sale) AS total_revenue
FROM FACT f GROUP BY f.FK_STORE_ID, f.FK_SUPPLIER_ID, year, month)
SELECT ms.FK_STORE_ID, ms.FK_SUPPLIER_ID, ms.month, ms.total_revenue,
LAG(ms.total_revenue)
OVER (PARTITION BY ms.FK_STORE_ID, ms.FK_SUPPLIER_ID ORDER BY ms.month) AS previous_month_revenue,
    ((ms.total_revenue - LAG(ms.total_revenue)
    OVER (PARTITION BY ms.FK_STORE_ID, ms.FK_SUPPLIER_ID ORDER BY ms.month)) / LAG(ms.total_revenue)
    OVER (PARTITION BY ms.FK_STORE_ID, ms.FK_SUPPLIER_ID ORDER BY ms.month)) * 100 AS revenue_volatility
FROM monthly_sales ms
ORDER BY ms.FK_STORE_ID, ms.FK_SUPPLIER_ID, ms.month;
```

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

FK_STORE_ID	FK_SUPPLIER_ID	month	total_revenue	previous_month_revenue	revenue_volatility
1	1	4	255197.68	NULL	NULL
1	1	8	149598.64	255197.68	-41.379310
2	1	4	244578.12	NULL	NULL
2	1	8	274397.80	244578.12	12.192293
2	2	4	484395.52	NULL	NULL

Result 30 | Read Only

warehouse transactions products_data queries

Limit to 1000 rows

```
-- Q6. Top 5 Products Purchased Together Across Multiple Orders (Product Affinity Analysis)

SELECT
    p.PRODUCT_ID,
    p.PRODUCT_NAME,
    SUM(f.Quantity) AS total_quantity_sold
FROM FACT f
JOIN PRODUCT p ON f.FK_PRODUCT_ID = p.PRODUCT_ID
GROUP BY p.PRODUCT_ID, p.PRODUCT_NAME
ORDER BY total_quantity_sold DESC
LIMIT 5;
```

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

PRODUCT_ID	PRODUCT_NAME	total_quantity_sold
21	DJI Mavic Air 2 Drone	496
13	Samsung Galaxy Tab S7	480
62	2019-08-24 16:39:00	452
7	AirPods Pro	436
44	Corsair Virtuoso RGB Wireless Gaming Headset	432

Result 31 Read Only

warehouse transactions products_data queries

Limit to 1000 rows

```
-- Q7. Yearly Revenue Trends by Store, Supplier, and Product with ROLLUP

SELECT
    f.FK_STORE_ID,
    f.FK_SUPPLIER_ID,
    f.FK_PRODUCT_ID,
    SUM(f.total_sale) AS yearly_revenue
FROM FACT f
GROUP BY f.FK_STORE_ID, f.FK_SUPPLIER_ID, f.FK_PRODUCT_ID WITH ROLLUP
ORDER BY f.FK_STORE_ID, f.FK_SUPPLIER_ID, f.FK_PRODUCT_ID;
```

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

FK_STORE_ID	FK_SUPPLIER_ID	FK_PRODUCT_ID	yearly_revenue
NULL	NULL	NULL	5258771.64
1	NULL	NULL	404796.32
1	1	NULL	404796.32
1	1	1	404796.32
2	NULL	NULL	1279768.76

Result 32 Read Only

warehouse transactions products_data queries

Limit to 1000 rows

```
-- Q8. Revenue and Volume-Based Sales Analysis for Each Product for H1 and H2

SELECT
    f.FK_PRODUCT_ID,
    SUM(CASE WHEN EXTRACT(MONTH FROM f.FK_date) BETWEEN 1 AND 6 THEN f.total_sale ELSE 0 END) AS revenue_h:
    SUM(CASE WHEN EXTRACT(MONTH FROM f.FK_date) BETWEEN 1 AND 6 THEN f.Quantity ELSE 0 END) AS quantity_h1,
    SUM(CASE WHEN EXTRACT(MONTH FROM f.FK_date) BETWEEN 7 AND 12 THEN f.total_sale ELSE 0 END) AS revenue_h
    SUM(CASE WHEN EXTRACT(MONTH FROM f.FK_date) BETWEEN 7 AND 12 THEN f.Quantity ELSE 0 END) AS quantity_h:
    SUM(f.total_sale) AS total_revenue,
    SUM(f.Quantity) AS total_quantity
FROM FACT f
GROUP BY f.FK_PRODUCT_ID
ORDER BY f.FK_PRODUCT_ID;
```

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

FK_PRODUCT_ID	revenue_h1	quantity_h1	revenue_h2	quantity_h2	total_revenue	total_quantity
1	255197.68	232	149598.64	136	404796.32	368
13	11400.00	300	6840.00	180	18240.00	480
2	332797.44	256	155998.80	120	488796.24	376
21	236797.04	296	159998.00	200	396795.04	496
25	24797.52	248	16798.32	168	41595.84	416

Result 33 Read Only

warehouse transactions products_data queries*

Limit to 1000 rows

```
-- Q9. Identify High Revenue Spikes in Product Sales and Highlight Outliers

WITH daily_avg_sales AS ( SELECT f.FK_PRODUCT_ID, EXTRACT(DAY FROM f.FK_date) AS day,
    AVG(f.total_sale) AS avg_daily_sales
    FROM FACT f GROUP BY f.FK_PRODUCT_ID, day
)
SELECT f.FK_PRODUCT_ID, f.FK_date AS order_date, f.total_sale, das.avg_daily_sales,
CASE  WHEN f.total_sale > 2 * das.avg_daily_sales THEN 'Outlier' ELSE 'Normal' END AS sales_type
FROM FACT f JOIN daily_avg_sales das ON f.FK_PRODUCT_ID = das.FK_PRODUCT_ID AND
EXTRACT(DAY FROM f.FK_date) = das.day
WHERE f.total_sale > 2 * das.avg_daily_sales;
```

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

FK_PRODUCT_ID	order_date	total_sale	avg_daily_sales	sales_type
62	2019-04-06	195.00	91.000000	Outlier
13	2019-04-16	152.00	66.500000	Outlier
62	2019-04-16	260.00	111.428571	Outlier
13	2019-04-20	114.00	53.200000	Outlier
39	2019-08-06	2199.96	785.700000	Outlier

Result 34 Read Only

warehouse transactions products_data queries*

Limit to 1000 rows

```
-- Q10. Create a View STORE_QUARTERLY_SALES for Optimized Sales Analysis

CREATE VIEW STORE_QUARTERLY_SALES AS
SELECT
    f.FK_STORE_ID,
    EXTRACT(QUARTER FROM f.FK_date) AS quarter,
    EXTRACT(YEAR FROM f.FK_date) AS year,
    SUM(f.total_sale) AS total_sales
FROM FACT f
GROUP BY f.FK_STORE_ID, quarter, year
ORDER BY f.FK_STORE_ID, year, quarter;

SELECT * FROM STORE_QUARTERLY_SALES;
```

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

FK_STORE_ID	quarter	year	total_sales
1	2	2019	255197.68
1	3	2019	149598.64
2	2	2019	734173.60
2	3	2019	545595.16
244	2	2019	11400.00

STORE_QUARTERLY_SALES 35 Read Only

Warehouse:

warehouse* | transactions | products_data | queries

Limit to 1000 rows

```
CREATE DATABASE METRO_DW;
USE METRO_DW;
```

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

sales	FK_PRODUCT_ID	FK_SUPPLIER_ID	FK_date	FK_STORE_ID	FK_CUSTOMER_ID	FK_ORDER_ID	Quantity	total_sale
1099.99	1	1	2019-04-01	1	81	177937	1	1099.99
1299.99	2	2	2019-04-01	2	59	186974	1	1299.99
1299.99	2	2	2019-04-01	2	64	180088	1	1299.99
1299.99	2	2	2019-04-01	2	80	177773	1	1299.99
349.99	29	27	2019-04-01	4	96	187325	1	349.99
1199.99	31	1	2019-04-01	2	98	187572	1	1199.99
999.96	7	7	2019-04-01	3	20	181642	4	999.96
749.97	7	7	2019-04-01	3	96	182505	3	749.97
499.98	7	7	2019-04-01	3	86	179308	2	499.98
99.99	25	24	2019-04-01	3	66	193269	1	99.99
65.00	62	1	2019-04-01	2	80	190361	1	65.00
130.00	62	1	2019-04-01	2	48	190798	2	130.00
65.00	62	1	2019-04-01	2	98	248188	1	65.00
1049.97	29	27	2019-04-01	4	23	178037	3	1049.97
549.99	39	1	2019-04-01	3	86	187346	1	549.99
179.99	44	21	2019-04-02	4	56	180571	1	179.99

FACT 37 | Read Only

This structure allows business users to:

1) Quickly find specific information
2) Analyze trends and patterns
3) Make data-driven decisions
4) Monitor business performance
5) Identify opportunities and challenges
6) Track progress over time