A black text on a white background

Description automatically generated

**Project Report**

An Assignment Presented to

**Dr. Asif Naeem**

In Partial fulfillment of

the requirement for the course of

**(CS – 3003) Data Warehouse and Business Intelligence**

By

**Huzaifa Rehman(21I - 2694)**

**Extended MESHJOIN Algorithm for Near Realtime METRO Data Warehouse**

1. **Project Overview :**

* **Background :**

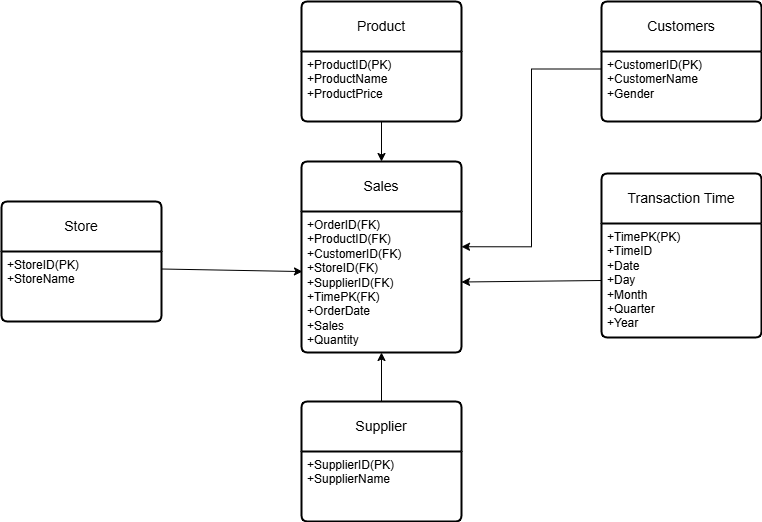
METRO, one of the largest superstore chains in Pakistan, aims to analyze customer shopping behavior for optimizing selling strategies. This analysis involves identifying patterns in transactional data and leveraging promotions effectively. To support these goals, the development of a near-real-time data warehouse (DW) is crucial.

The system must incorporate customer transactions from data sources (DSs) into the DW as they appear. The ETL process must also enrich incomplete transactional data using master data (MD) during the transformation phase. For this, the MESHJOIN algorithm is used to facilitate stream-relation join operations in ETL.

* **Objective :**

This project involves implementing an extended version of the MESHJOIN algorithm using Java and Eclipse IDE. The enriched data will populate a star-schema-based DW, providing multidimensional decision support for METRO's business operations.

1. **Star Schema Diagram :**



1. **Creating Metro Data Warehouse :**  
     
   # Dropping Schema if the schema already exists, then creating metro\_dwh.

Drop schema if exists `metro\_dwh`;

Create schema `metro\_dwh`;

Use `metro\_dwh`;

# Dropping tables if the tables already exists.

Drop table if exists `product`;

Drop table if exists `customers`;

Drop table if exists `store`;

Drop table if exists `supplier`;

Drop table if exists `transaction\_time`;

Drop table if exists `sales`;

# Creating Tables from the Star Schema

Use metro\_dwh;

Create table Product(ProductID int(10) Primary key, ProductName varchar(255) not null, ProductPrice double(10,2) not null);

Create table Customers(CustomerID int(10) Primary key, CustomerName varchar(255) not null, Gender varchar(255) not null);

Create table Store(StoreID int(10) Primary key, StoreName varchar(255) not null);

Create table Supplier(SupplierID int(10) Primary key, SupplierName varchar(255) not null);

Create table Transaction\_Time(TimePK int(10) Primary Key, TimeID int(10), TransactionDate date, TransactionDay varchar(255) not null, TransactionMonth varchar(255) not null, TransactionQuarter int(50) not null, TransactionYear int(50) not null);

Create table Sales(OrderID int(10), OrderDate date, ProductID int(10), CustomerID int(10), StoreID int(10), SupplierID int(10), TimePK int(10), Sales double(10, 2), Quantity int(10),

constraint Product\_fk foreign key (ProductID) references Product(ProductID),

constraint Customer\_fk foreign key (CustomerID) references Customers(CustomerID),

constraint Store\_fk foreign key (StoreID) references Store(StoreID),

constraint Supplier\_fk foreign key (SupplierID) references Supplier(SupplierID),

constraint Time\_fk foreign key (TimePK) references Transaction\_Time(TimePK));

1. **OLAP Queries:**

-- Query1

SELECT

p.ProductName,

MONTH(t.TransactionDate) AS TransactionMonth,

CASE

WHEN t.TransactionDay IN ('Saturday', 'Sunday') THEN 'Weekend'

ELSE 'Weekday'

END AS DayType,

SUM(s.Sales) AS TotalRevenue

FROM

Sales s

JOIN

Product p ON s.ProductID = p.ProductID

JOIN

Transaction\_Time t ON s.TimePK = t.TimePK

WHERE

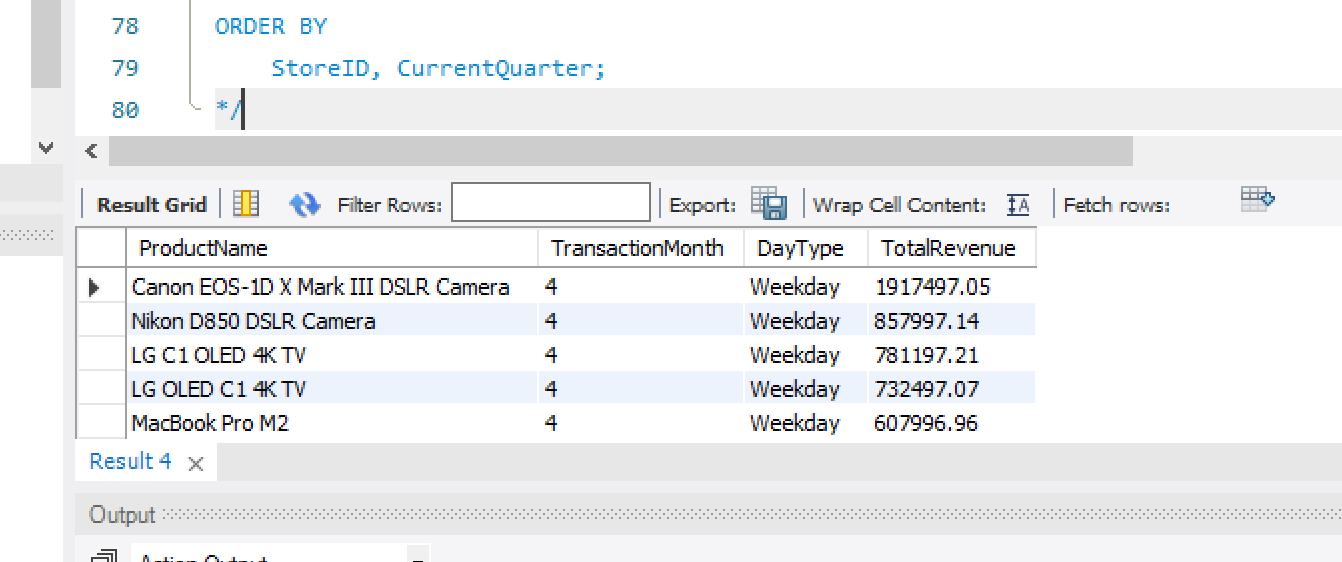
t.TransactionYear = 2019

GROUP BY

p.ProductName, TransactionMonth, DayType

ORDER BY

TransactionMonth, DayType, TotalRevenue DESC

LIMIT 5;

-- Query 2

WITH QuarterlyRevenue AS (

SELECT

s.StoreID,

st.StoreName,

t.TransactionYear,

t.TransactionQuarter,

SUM(s.Sales) AS TotalRevenue

FROM

Sales s

JOIN

Store st ON s.StoreID = st.StoreID

JOIN

Transaction\_Time t ON s.TimePK = t.TimePK

WHERE

t.TransactionYear = 2019

GROUP BY

s.StoreID, st.StoreName, t.TransactionYear, t.TransactionQuarter

),

QuarterlyGrowth AS (

SELECT

q1.StoreID,

q1.StoreName,

q1.TransactionQuarter AS CurrentQuarter,

q1.TotalRevenue AS CurrentRevenue,

q2.TotalRevenue AS PreviousRevenue,

CASE

WHEN q2.TotalRevenue IS NOT NULL THEN

((q1.TotalRevenue - q2.TotalRevenue) / q2.TotalRevenue) \* 100

ELSE

NULL

END AS GrowthRate

FROM

QuarterlyRevenue q1

LEFT JOIN

QuarterlyRevenue q2

ON

q1.StoreID = q2.StoreID

AND q1.TransactionYear = q2.TransactionYear

AND q1.TransactionQuarter = q2.TransactionQuarter + 1

)

SELECT

StoreID,

StoreName,

CurrentQuarter,

CurrentRevenue,

PreviousRevenue,

GrowthRate

FROM

QuarterlyGrowth

ORDER BY

StoreID, CurrentQuarter;

A screenshot of a computer

Description automatically generated

-- Query3

SELECT

st.StoreName AS Store,

sp.SupplierName AS Supplier,

p.ProductName AS Product,

SUM(s.Sales) AS TotalSales

FROM

Sales s

JOIN

Store st ON s.StoreID = st.StoreID

JOIN

Supplier sp ON s.SupplierID = sp.SupplierID

JOIN

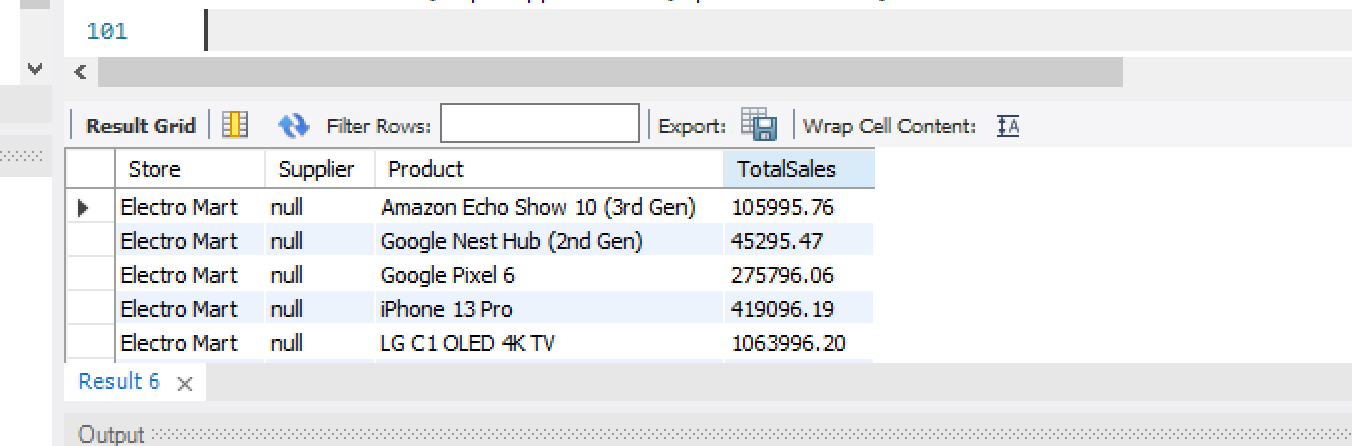
Product p ON s.ProductID = p.ProductID

GROUP BY

st.StoreName, sp.SupplierName, p.ProductName

ORDER BY

st.StoreName, sp.SupplierName, p.ProductName;



-- Query4

SELECT

p.ProductName AS Product,

CASE

WHEN MONTH(t.TransactionDate) IN (3, 4, 5) THEN 'Spring'

WHEN MONTH(t.TransactionDate) IN (6, 7, 8) THEN 'Summer'

WHEN MONTH(t.TransactionDate) IN (9, 10, 11) THEN 'Fall'

WHEN MONTH(t.TransactionDate) IN (12, 1, 2) THEN 'Winter'

END AS Season,

SUM(s.Sales) AS TotalSales

FROM

Sales s

JOIN

Product p ON s.ProductID = p.ProductID

JOIN

Transaction\_Time t ON s.TimePK = t.TimePK

GROUP BY

p.ProductName, Season

ORDER BY

p.ProductName,

FIELD(Season, 'Spring', 'Summer', 'Fall', 'Winter');

A screenshot of a computer

Description automatically generated

-- Query5

WITH MonthlyRevenue AS (

SELECT

s.StoreID,

st.StoreName,

s.SupplierID,

sp.SupplierName,

YEAR(t.TransactionDate) AS Year,

MONTH(t.TransactionDate) AS Month,

SUM(s.Sales) AS TotalRevenue

FROM

Sales s

JOIN

Store st ON s.StoreID = st.StoreID

JOIN

Supplier sp ON s.SupplierID = sp.SupplierID

JOIN

Transaction\_Time t ON s.TimePK = t.TimePK

GROUP BY

s.StoreID, st.StoreName, s.SupplierID, sp.SupplierName, Year, Month

),

MonthlyVolatility AS (

SELECT

mr1.StoreID,

mr1.StoreName,

mr1.SupplierID,

mr1.SupplierName,

mr1.Year,

mr1.Month AS CurrentMonth,

mr1.TotalRevenue AS CurrentRevenue,

mr2.TotalRevenue AS PreviousRevenue,

CASE

WHEN mr2.TotalRevenue IS NOT NULL THEN

((mr1.TotalRevenue - mr2.TotalRevenue) / mr2.TotalRevenue) \* 100

ELSE

NULL

END AS RevenueVolatility

FROM

MonthlyRevenue mr1

LEFT JOIN

MonthlyRevenue mr2

ON

mr1.StoreID = mr2.StoreID

AND mr1.SupplierID = mr2.SupplierID

AND mr1.Year = mr2.Year

AND mr1.Month = mr2.Month + 1

)

SELECT

StoreID,

StoreName,

SupplierID,

SupplierName,

Year,

CurrentMonth,

CurrentRevenue,

PreviousRevenue,

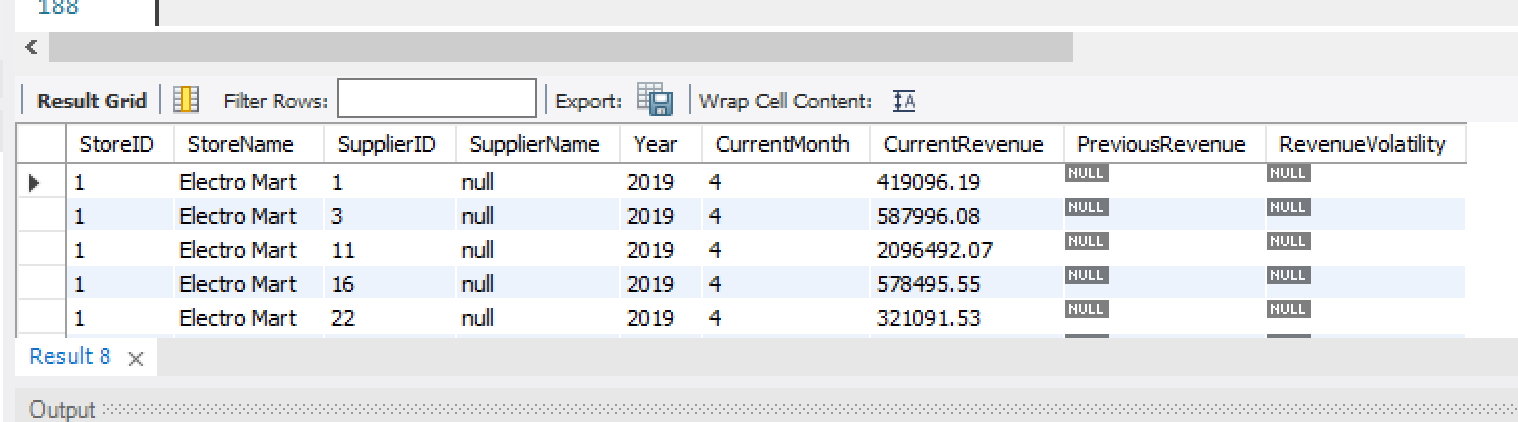
RevenueVolatility

FROM

MonthlyVolatility

ORDER BY

StoreID, SupplierID, Year, CurrentMonth;



-- Query6

SELECT

p1.ProductName AS ProductA,

p2.ProductName AS ProductB,

COUNT(\*) AS Frequency

FROM

Sales s1

JOIN

Sales s2 ON s1.OrderID = s2.OrderID AND s1.ProductID < s2.ProductID

JOIN

Product p1 ON s1.ProductID = p1.ProductID

JOIN

Product p2 ON s2.ProductID = p2.ProductID

GROUP BY

p1.ProductName, p2.ProductName

ORDER BY

Frequency DESC

LIMIT 5;

A screenshot of a computer

Description automatically generated

-- Query7

SELECT

st.StoreName AS Store,

sp.SupplierName AS Supplier,

p.ProductName AS Product,

t.TransactionYear AS Year,

SUM(s.Sales) AS TotalRevenue

FROM

Sales s

JOIN

Store st ON s.StoreID = st.StoreID

JOIN

Supplier sp ON s.SupplierID = sp.SupplierID

JOIN

Product p ON s.ProductID = p.ProductID

JOIN

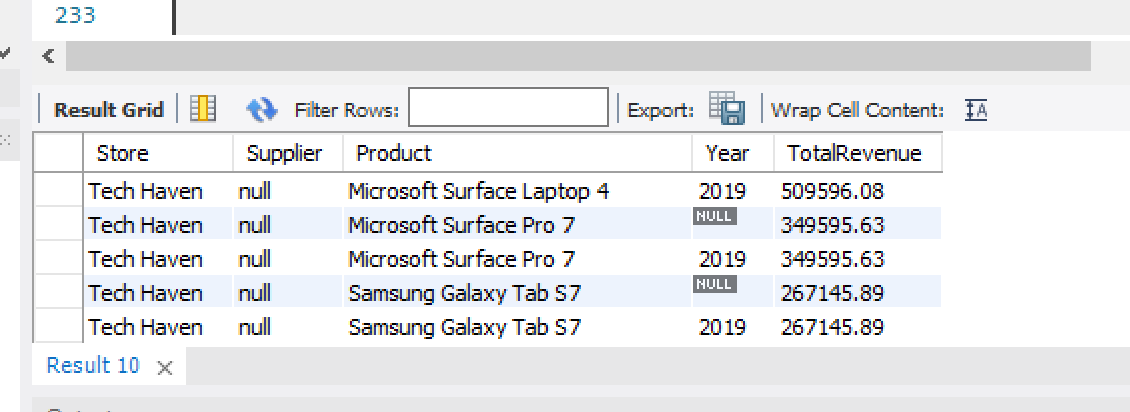
Transaction\_Time t ON s.TimePK = t.TimePK

GROUP BY

ROLLUP (st.StoreName, sp.SupplierName, p.ProductName, t.TransactionYear)

ORDER BY

st.StoreName, sp.SupplierName, p.ProductName, t.TransactionYear;



-- Query8

SELECT

p.ProductName AS Product,

CASE

WHEN MONTH(t.TransactionDate) BETWEEN 1 AND 6 THEN 'H1'

WHEN MONTH(t.TransactionDate) BETWEEN 7 AND 12 THEN 'H2'

ELSE 'Yearly Total'

END AS TimePeriod,

YEAR(t.TransactionDate) AS Year,

SUM(s.Sales) AS TotalRevenue,

SUM(s.Quantity) AS TotalQuantity

FROM

Sales s

JOIN

Product p ON s.ProductID = p.ProductID

JOIN

Transaction\_Time t ON s.TimePK = t.TimePK

GROUP BY

p.ProductName, Year, TimePeriod

ORDER BY

p.ProductName, Year, FIELD(TimePeriod, 'H1', 'H2', 'Yearly Total');

\*/

A screenshot of a computer

Description automatically generated

/\*

-- Query9

WITH DailyAverageSales AS (

SELECT

p.ProductName AS Product,

t.TransactionDate AS SaleDate,

SUM(s.Sales) / COUNT(DISTINCT t.TransactionDate) AS DailyAverageSales

FROM

Sales s

JOIN

Product p ON s.ProductID = p.ProductID

JOIN

Transaction\_Time t ON s.TimePK = t.TimePK

GROUP BY

p.ProductName, t.TransactionDate

),

SalesWithOutliers AS (

SELECT

p.ProductName AS Product,

t.TransactionDate AS SaleDate,

SUM(s.Sales) AS TotalSales,

da.DailyAverageSales,

CASE

WHEN SUM(s.Sales) > (2 \* da.DailyAverageSales) THEN 'Outlier'

ELSE 'Normal'

END AS SalesStatus

FROM

Sales s

JOIN

Product p ON s.ProductID = p.ProductID

JOIN

Transaction\_Time t ON s.TimePK = t.TimePK

JOIN

DailyAverageSales da ON p.ProductName = da.Product AND t.TransactionDate = da.SaleDate

GROUP BY

p.ProductName, t.TransactionDate, da.DailyAverageSales

)

SELECT

Product,

SaleDate,

TotalSales,

DailyAverageSales,

SalesStatus

FROM

SalesWithOutliers

ORDER BY

Product, SaleDate;

\*/

A screenshot of a computer

Description automatically generated

-- Query10

CREATE VIEW STORE\_QUARTERLY\_SALES AS

SELECT

st.StoreName AS Store,

QUARTER(t.TransactionDate) AS Quarter,

YEAR(t.TransactionDate) AS Year,

SUM(s.Sales) AS TotalSales

FROM

Sales s

JOIN

Store st ON s.StoreID = st.StoreID

JOIN

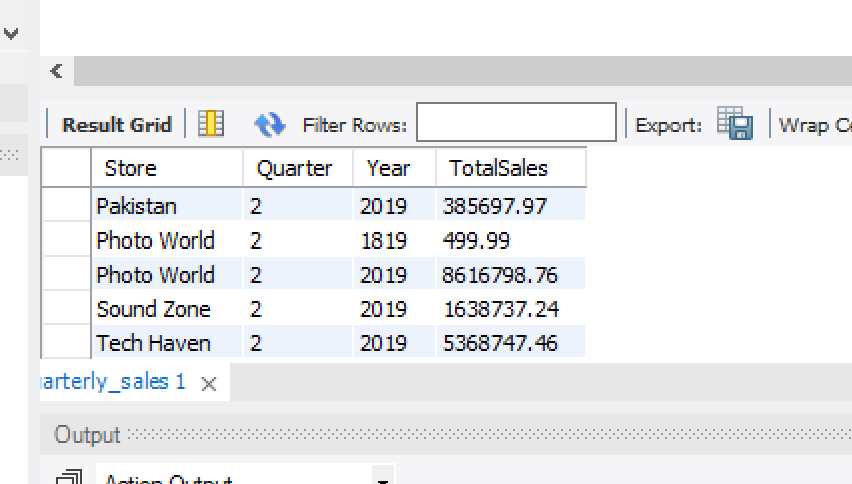
Transaction\_Time t ON s.TimePK = t.TimePK

GROUP BY

st.StoreName, QUARTER(t.TransactionDate), YEAR(t.TransactionDate)

ORDER BY

st.StoreName, Year, Quarter;



1. **Conclusion:**

The extended MESHJOIN algorithm provides a robust framework for implementing near-real-time stream-relation joins in ETL. Its efficient handling of large datasets and ability to enrich incomplete transactional data ensures timely and accurate updates to the DW. By leveraging this system, METRO can gain valuable insights into customer behavior, enabling optimized selling strategies and improved decision-making capabilities.