RETAIL ANALYSIS USING SQL

Presented By: **Dristi Handique**

Date: **09-08-2025**

INTRODUCTION

The project focuses on a retail company facing challenges with sales transaction, customer profile and inventory management. Through a comprehensive data analysis approach using SQL, this case study aims to address key business problems and insights such as high and low sales products, segment the customer base, and analyze customer behavior.

DATABASE STRUCTURE

Sales_transaction

- TransactionID
- CustomerID
- ProductID
- QuantityPurchased
- TransactionDate
- Price

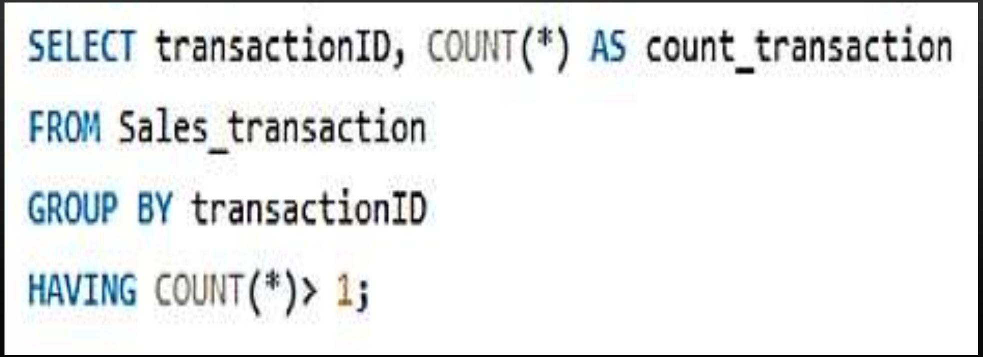
Customer_profiles

- CustomerID
- Age
- Gender
- Location
- JoinDate

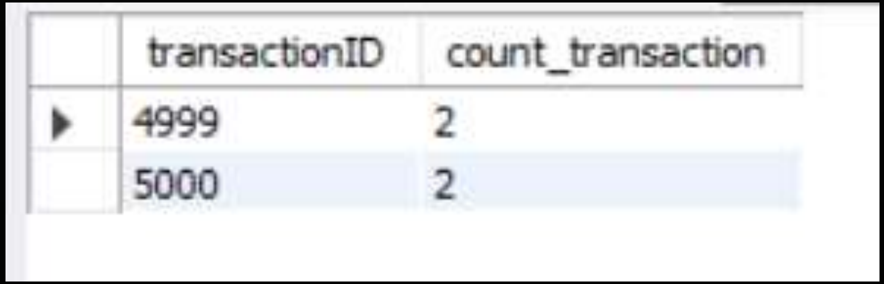
Product_inventory

- ProductID
- ProductName
- Category
- StockLevel
- Price

QUERY 1: Clean the sales_transaction table by identifying and removing duplicate entries.

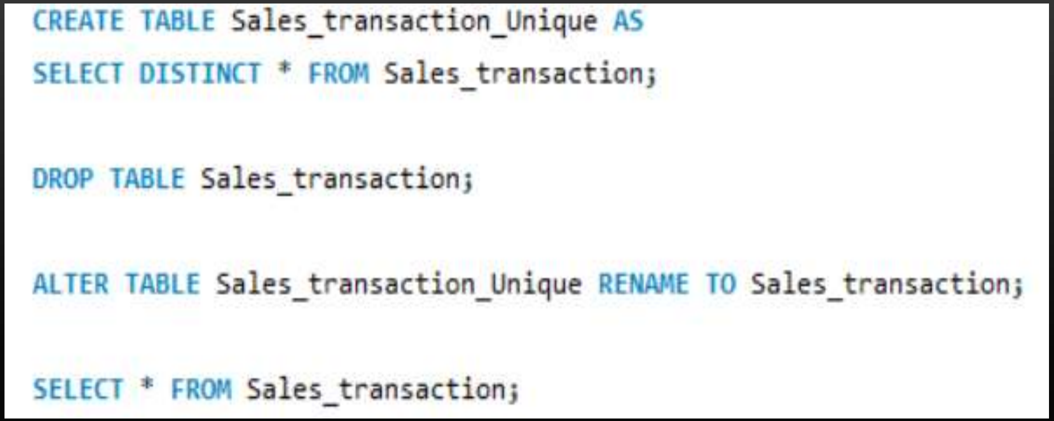


```
SELECT transactionID, COUNT(*) AS count_transaction
FROM Sales_transaction
GROUP BY transactionID
HAVING COUNT(*)> 1;
```

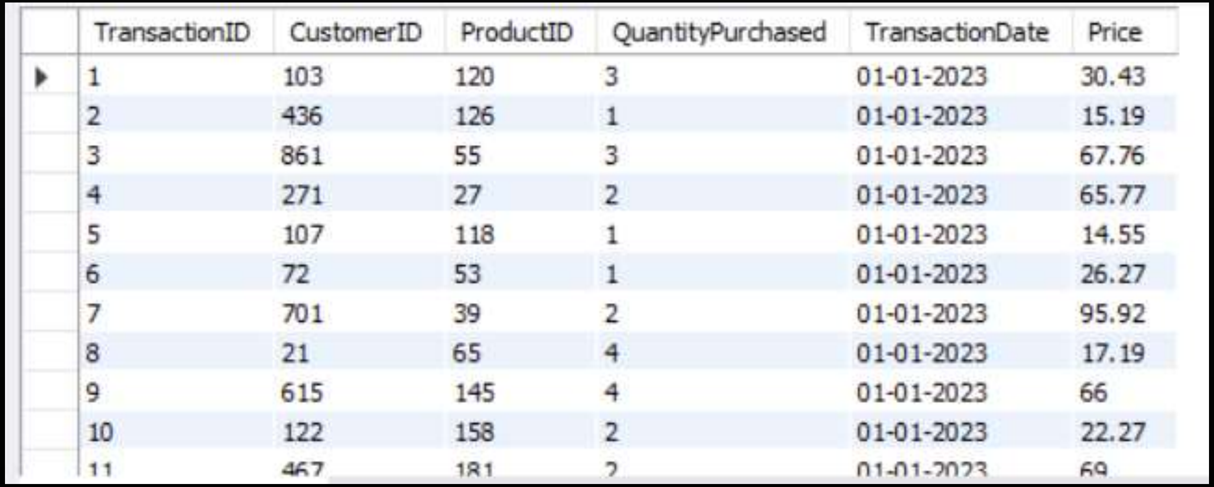


transactionID	count_transaction
4999	2
5000	2

QUERY 2: Clean the sales_transaction table by identifying and removing duplicate entries.

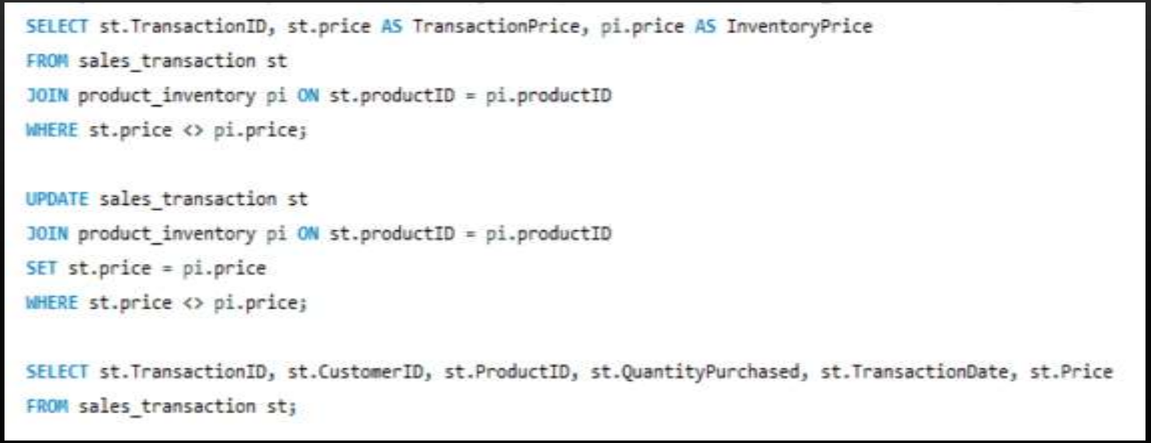


```
CREATE TABLE Sales_transaction_Unique AS
SELECT DISTINCT * FROM Sales_transaction;

DROP TABLE Sales_transaction;

ALTER TABLE Sales_transaction_Unique RENAME TO Sales_transaction;

SELECT * FROM Sales_transaction;
```

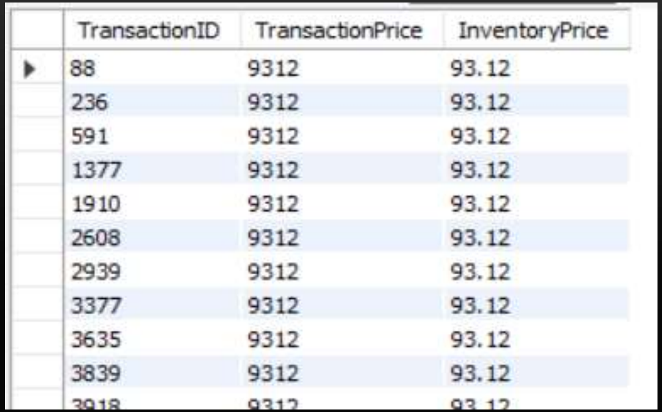


TransactionID	CustomerID	ProductID	QuantityPurchased	TransactionDate	Price
1	103	120	3	01-01-2023	30.43
2	436	126	1	01-01-2023	15.19
3	861	55	3	01-01-2023	67.76
4	271	27	2	01-01-2023	65.77
5	107	118	1	01-01-2023	14.55
6	72	53	1	01-01-2023	26.27
7	701	39	2	01-01-2023	95.92
8	21	65	4	01-01-2023	17.19
9	615	145	4	01-01-2023	66
10	122	158	2	01-01-2023	22.27
11	467	181	2	01-01-2023	69

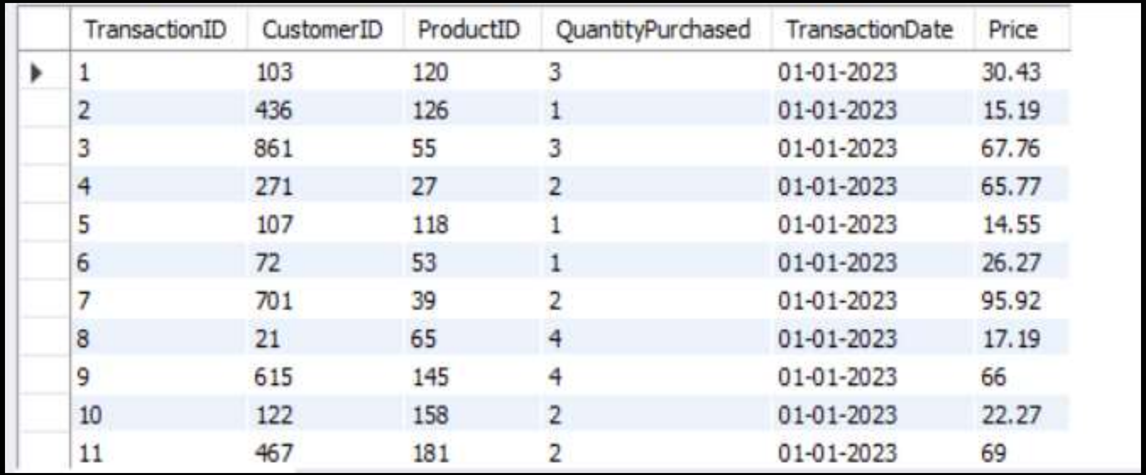
QUERY 3: Identify and resolve pricing discrepancies between the sales_transaction and product_inventory tables by aligning the transaction prices with the correct inventory prices for the same products.



```sql
SELECT st.TransactionID, st.price AS TransactionPrice, pi.price AS InventoryPrice
FROM sales_transaction st
JOIN product_inventory pi ON st.productID = pi.productID
WHERE st.price <> pi.price;

UPDATE sales_transaction st
JOIN product_inventory pi ON st.productID = pi.productID
SET st.price = pi.price
WHERE st.price <> pi.price;

SELECT st.TransactionID, st.CustomerID, st.ProductID, st.QuantityPurchased, st.TransactionDate, st.Price
FROM sales_transaction st;
```

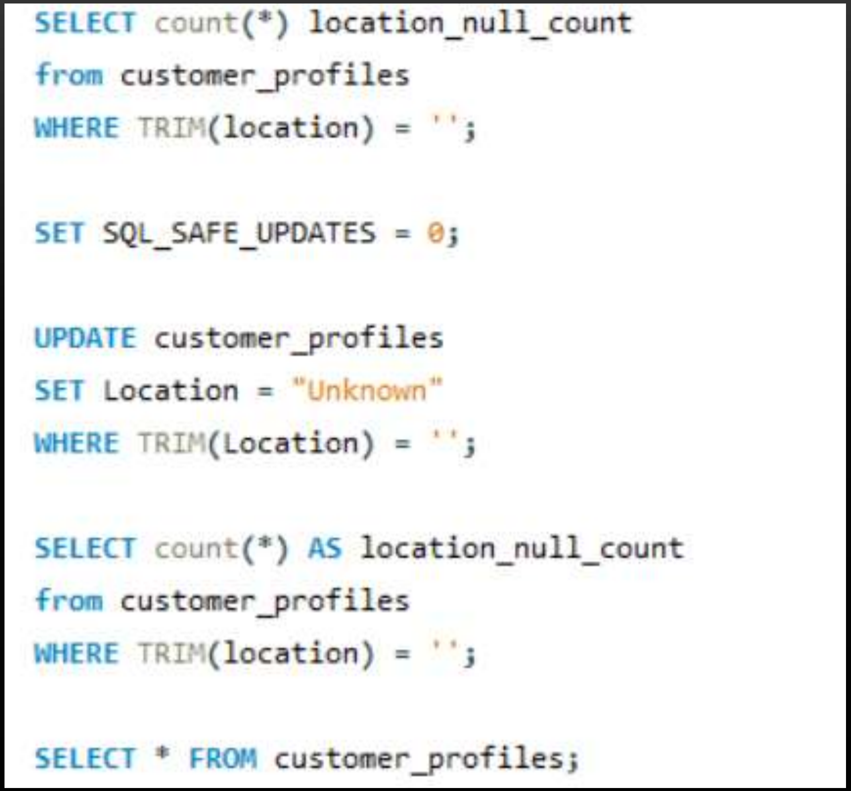


TransactionID	TransactionPrice	InventoryPrice
88	9312	93.12
236	9312	93.12
591	9312	93.12
1377	9312	93.12
1910	9312	93.12
2608	9312	93.12
2939	9312	93.12
3377	9312	93.12
3635	9312	93.12
3839	9312	93.12
3918	9312	93.12

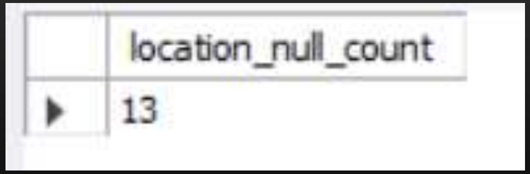


TransactionID	CustomerID	ProductID	QuantityPurchased	TransactionDate	Price
1	103	120	3	01-01-2023	30.43
2	436	126	1	01-01-2023	15.19
3	861	55	3	01-01-2023	67.76
4	271	27	2	01-01-2023	65.77
5	107	118	1	01-01-2023	14.55
6	72	53	1	01-01-2023	26.27
7	701	39	2	01-01-2023	95.92
8	21	65	4	01-01-2023	17.19
9	615	145	4	01-01-2023	66
10	122	158	2	01-01-2023	22.27
11	467	181	2	01-01-2023	69

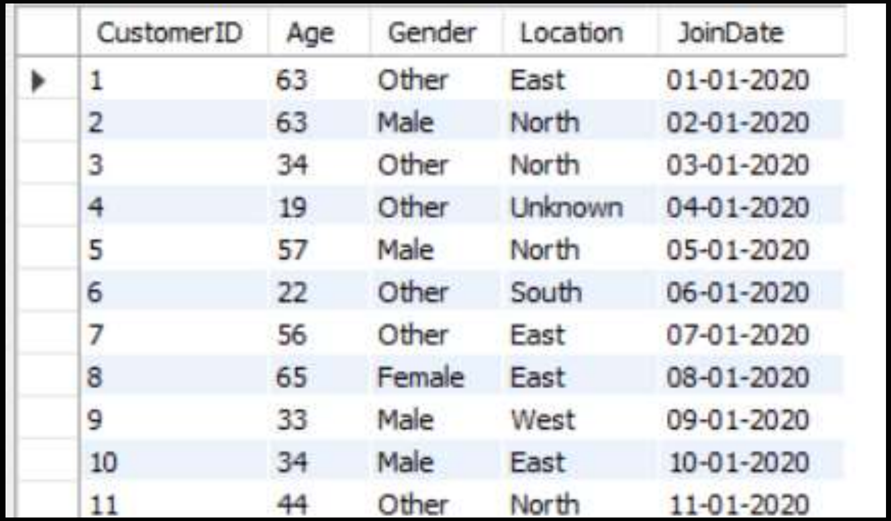
QUERY 4: Clean the sales_transaction table by identifying and removing duplicate entries.



```sql
SELECT count(*) location_null_count
from customer_profiles
WHERE TRIM(location) = '';

SET SQL_SAFE_UPDATES = 0;

UPDATE customer_profiles
SET Location = "Unknown"
WHERE TRIM(Location) = '';

SELECT count(*) AS location_null_count
from customer_profiles
WHERE TRIM(location) = '';

SELECT * FROM customer_profiles;
```

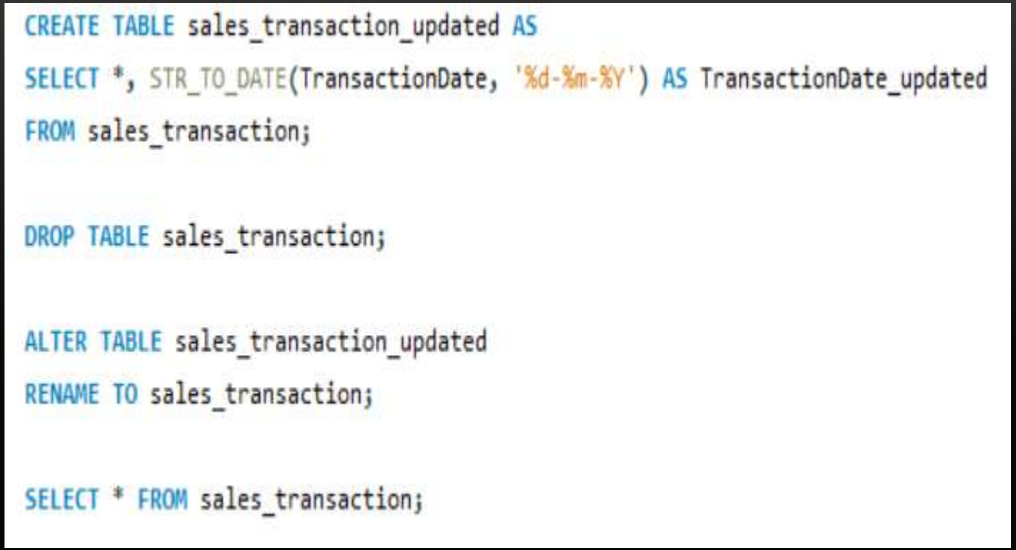


location_null_count
13

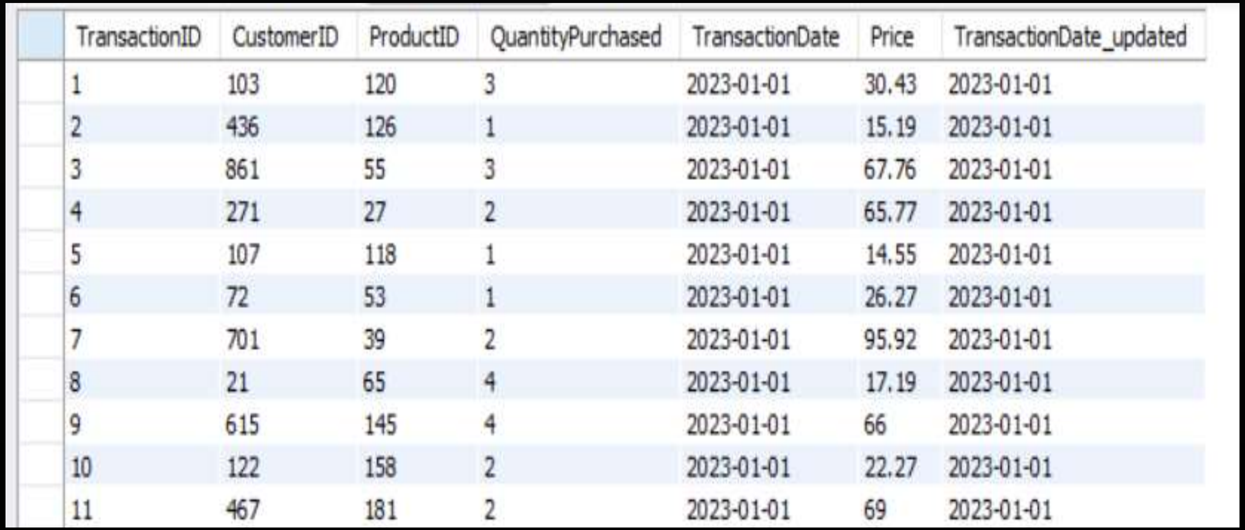


CustomerID	Age	Gender	Location	JoinDate
1	63	Other	East	01-01-2020
2	63	Male	North	02-01-2020
3	34	Other	North	03-01-2020
4	19	Other	Unknown	04-01-2020
5	57	Male	North	05-01-2020
6	22	Other	South	06-01-2020
7	56	Other	East	07-01-2020
8	65	Female	East	08-01-2020
9	33	Male	West	09-01-2020
10	34	Male	East	10-01-2020
11	44	Other	North	11-01-2020

QUERY 5: Clean and correct the data type of the DATE column from TEXT to a standard DATE format.

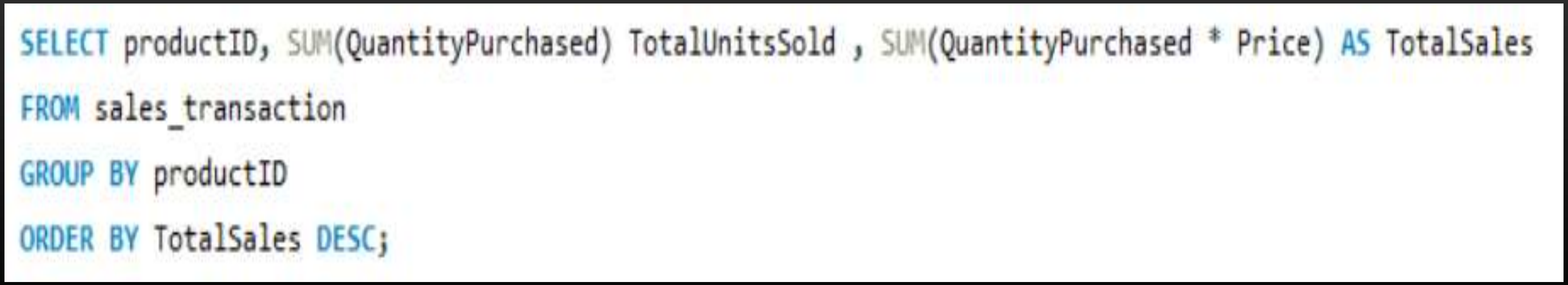


```
CREATE TABLE sales_transaction_updated AS
SELECT *, STR_TO_DATE(TransactionDate, '%d-%m-%Y') AS TransactionDate_updated
FROM sales_transaction;

DROP TABLE sales_transaction;

ALTER TABLE sales_transaction_updated
RENAME TO sales_transaction;

SELECT * FROM sales_transaction;
```

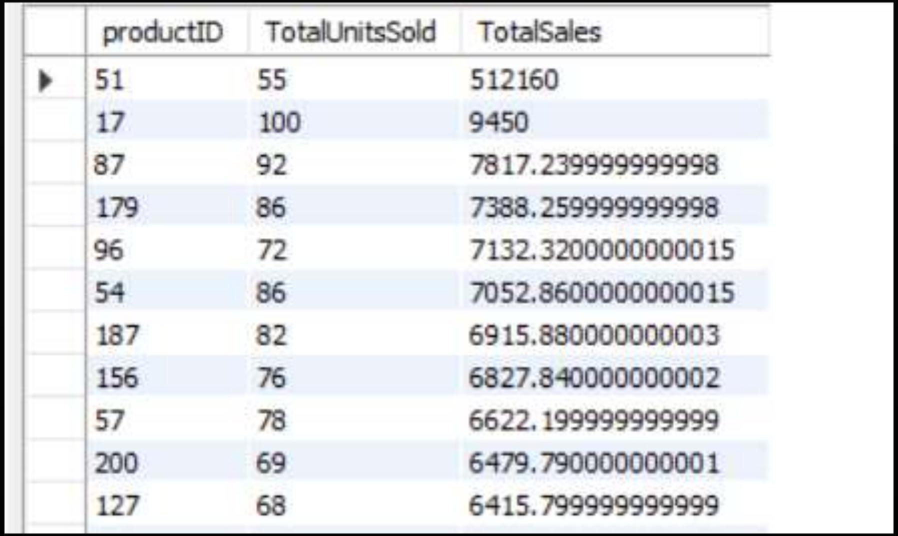


TransactionID	CustomerID	ProductID	QuantityPurchased	TransactionDate	Price	TransactionDate_updated
1	103	120	3	2023-01-01	30.43	2023-01-01
2	436	126	1	2023-01-01	15.19	2023-01-01
3	861	55	3	2023-01-01	67.76	2023-01-01
4	271	27	2	2023-01-01	65.77	2023-01-01
5	107	118	1	2023-01-01	14.55	2023-01-01
6	72	53	1	2023-01-01	26.27	2023-01-01
7	701	39	2	2023-01-01	95.92	2023-01-01
8	21	65	4	2023-01-01	17.19	2023-01-01
9	615	145	4	2023-01-01	66	2023-01-01
10	122	158	2	2023-01-01	22.27	2023-01-01
11	467	181	2	2023-01-01	69	2023-01-01

QUERY 6: Summarize the total sales and quantities sold per product by the company.

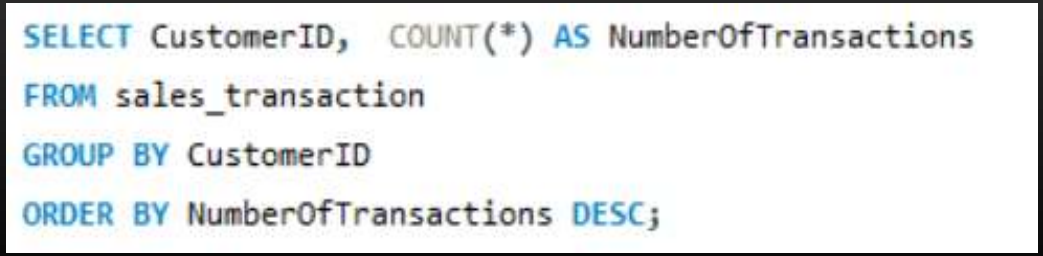


```
SELECT productID, SUM(QuantityPurchased) TotalUnitsSold , SUM(QuantityPurchased * Price) AS TotalSales
FROM sales_transaction
GROUP BY productID
ORDER BY TotalSales DESC;
```

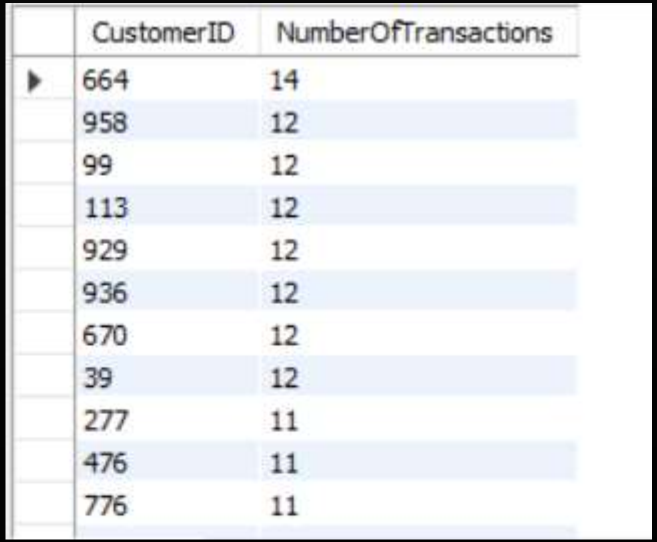


productID	TotalUnitsSold	TotalSales
51	55	512160
17	100	9450
87	92	7817.239999999998
179	86	7388.259999999998
96	72	7132.320000000015
54	86	7052.860000000015
187	82	6915.880000000003
156	76	6827.840000000002
57	78	6622.199999999999
200	69	6479.790000000001
127	68	6415.799999999999

QUERY 7: Count the number of transactions per customer to understand purchase frequency.

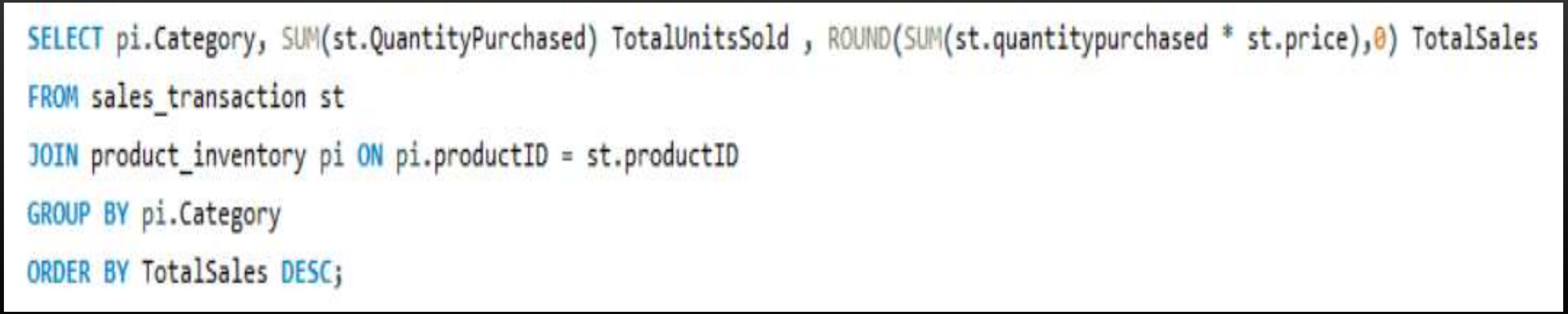


```
SELECT CustomerID,  COUNT(*) AS NumberOfTransactions
FROM sales_transaction
GROUP BY CustomerID
ORDER BY NumberOfTransactions DESC;
```

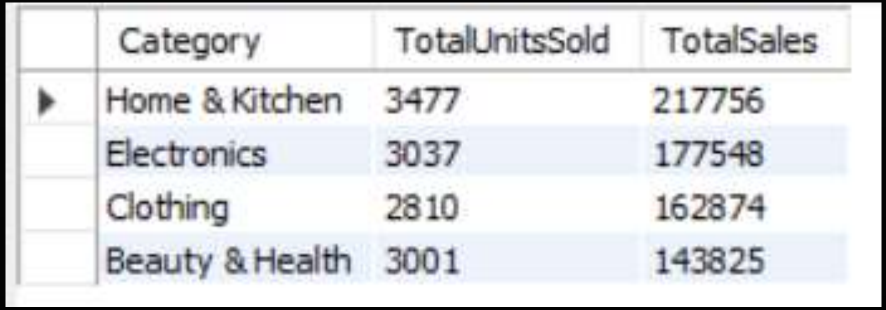


CustomerID	NumberOfTransactions
664	14
958	12
99	12
113	12
929	12
936	12
670	12
39	12
277	11
476	11
776	11

QUERY 8: Evaluate the performance of the product categories based on the total sales.

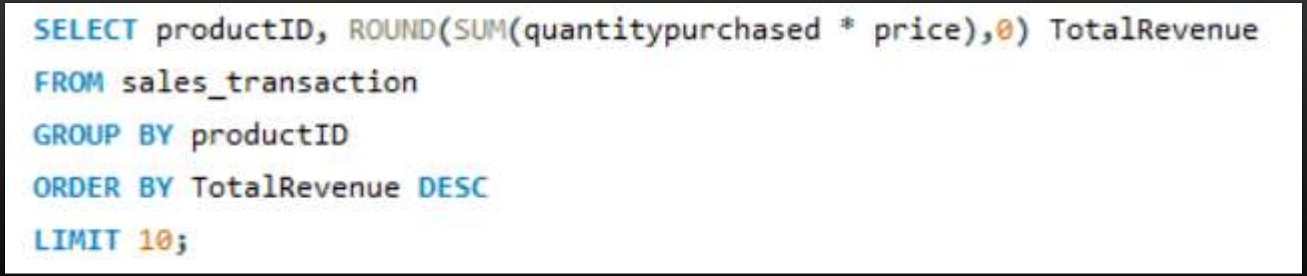


```
SELECT pi.Category, SUM(st.QuantityPurchased) TotalUnitsSold , ROUND(SUM(st.quantitypurchased * st.price),0) TotalSales
FROM sales_transaction st
JOIN product_inventory pi ON pi.productID = st.productID
GROUP BY pi.Category
ORDER BY TotalSales DESC;
```



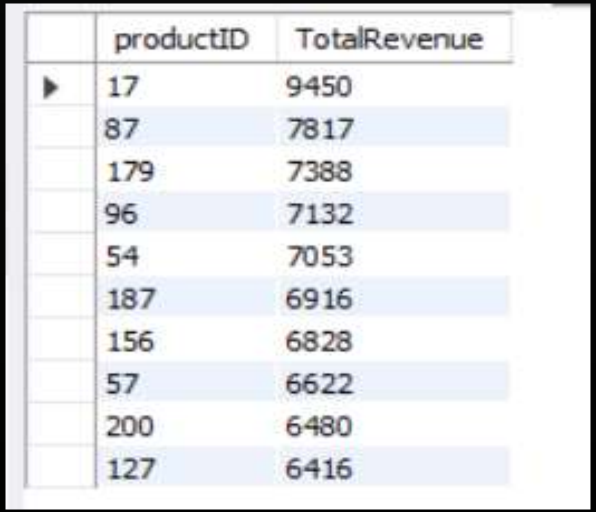
Category	TotalUnitsSold	TotalSales
Home & Kitchen	3477	217756
Electronics	3037	177548
Clothing	2810	162874
Beauty & Health	3001	143825

QUERY 9: Identify the top 10 products by total sales revenue



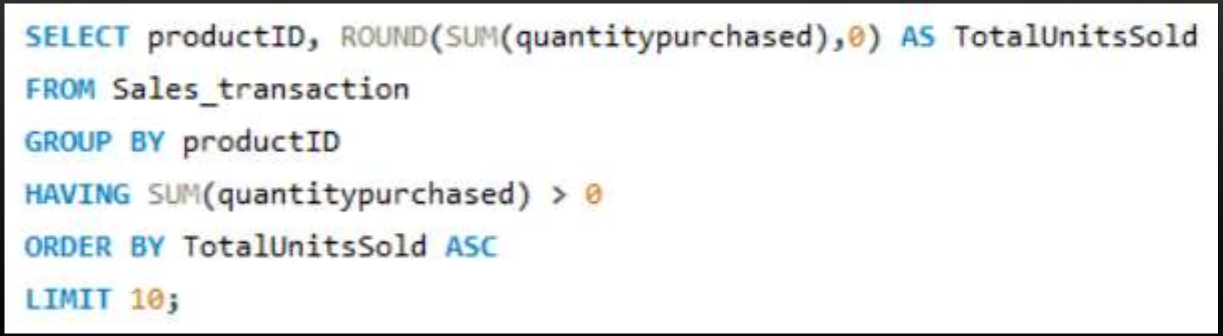
```sql
SELECT productID, ROUND(SUM(quantitypurchased * price),0) TotalRevenue
FROM sales_transaction
GROUP BY productID
ORDER BY TotalRevenue DESC
LIMIT 10;
```

Identify the top 10 products by total sales revenue

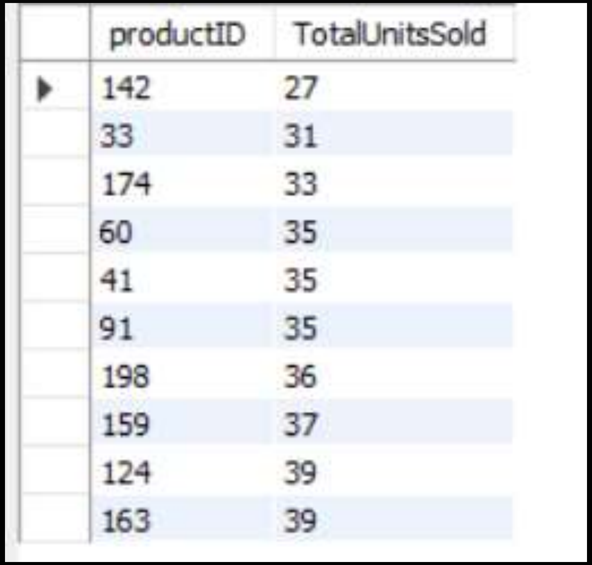


productID	TotalRevenue
17	9450
87	7817
179	7388
96	7132
54	7053
187	6916
156	6828
57	6622
200	6480
127	6416

QUERY 10: Identify the 10 least-selling products.

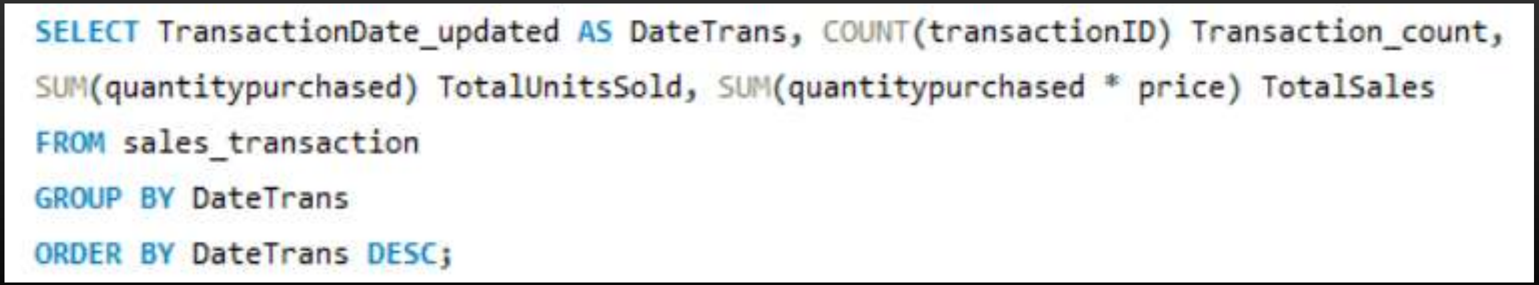


```
SELECT productID, ROUND(SUM(quantitypurchased),0) AS TotalUnitsSold
FROM Sales_transaction
GROUP BY productID
HAVING SUM(quantitypurchased) > 0
ORDER BY TotalUnitsSold ASC
LIMIT 10;
```

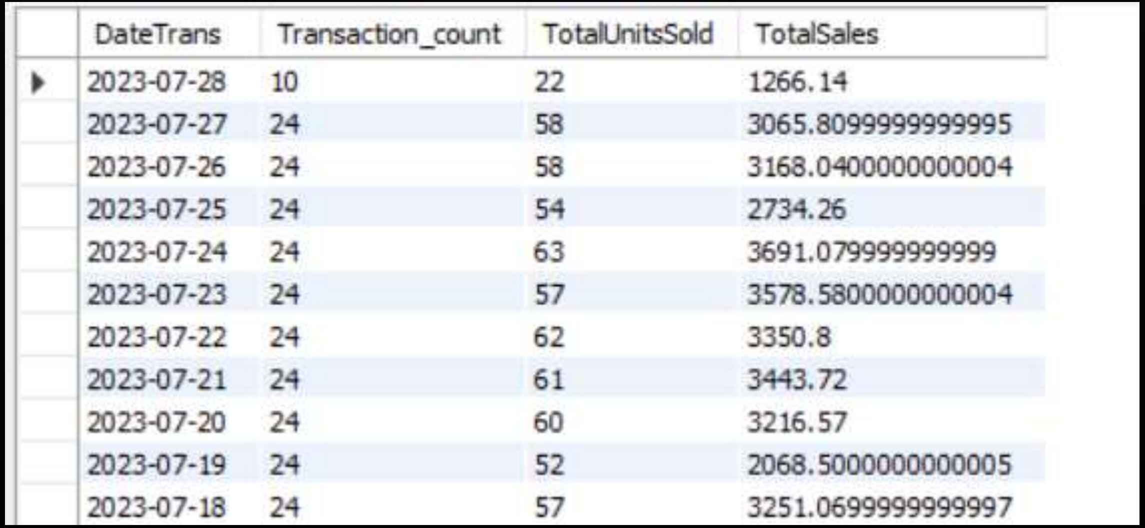


productID	TotalUnitsSold
142	27
33	31
174	33
60	35
41	35
91	35
198	36
159	37
124	39
163	39

QUERY 11: Identify the sales trend from the sales_transaction table.

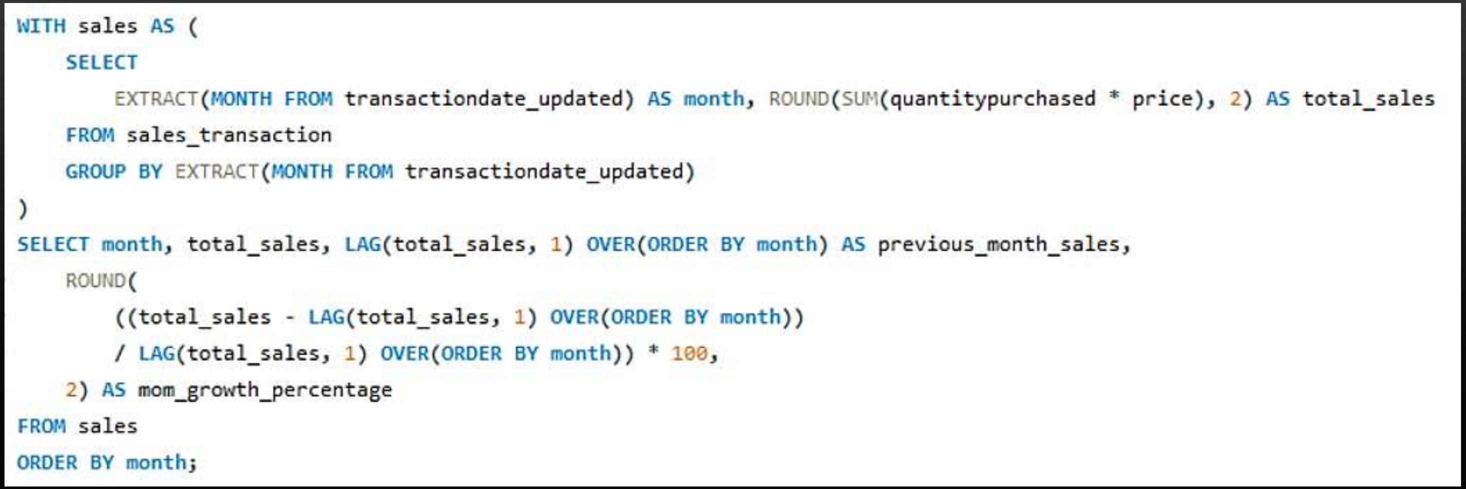


```
SELECT TransactionDate_updated AS DateTrans, COUNT(transactionID) Transaction_count,
SUM(quantitypurchased) TotalUnitsSold, SUM(quantitypurchased * price) TotalSales
FROM sales_transaction
GROUP BY DateTrans
ORDER BY DateTrans DESC;
```

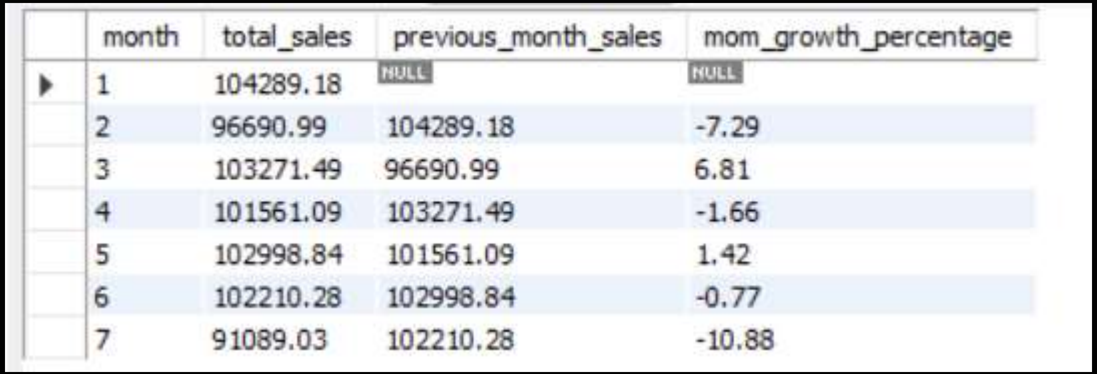


DateTrans	Transaction_count	TotalUnitsSold	TotalSales
2023-07-28	10	22	1266.14
2023-07-27	24	58	3065.8099999999995
2023-07-26	24	58	3168.0400000000004
2023-07-25	24	54	2734.26
2023-07-24	24	63	3691.079999999999
2023-07-23	24	57	3578.5800000000004
2023-07-22	24	62	3350.8
2023-07-21	24	61	3443.72
2023-07-20	24	60	3216.57
2023-07-19	24	52	2068.5000000000005
2023-07-18	24	57	3251.0699999999997

QUERY 12: Analyze the company's month-over-month sales growth rate.

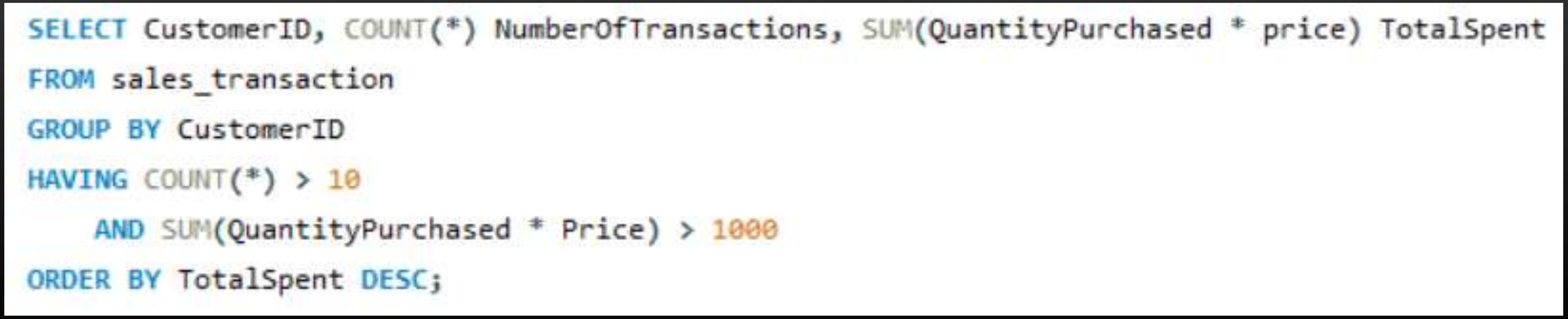


```sql
WITH sales AS (
    SELECT
        EXTRACT(MONTH FROM transactiondate_updated) AS month, ROUND(SUM(quantitypurchased * price), 2) AS total_sales
    FROM sales_transaction
    GROUP BY EXTRACT(MONTH FROM transactiondate_updated)
)
SELECT month, total_sales, LAG(total_sales, 1) OVER(ORDER BY month) AS previous_month_sales,
    ROUND(
        ((total_sales - LAG(total_sales, 1) OVER(ORDER BY month))
        / LAG(total_sales, 1) OVER(ORDER BY month)) * 100,
    2) AS mom_growth_percentage
FROM sales
ORDER BY month;
```

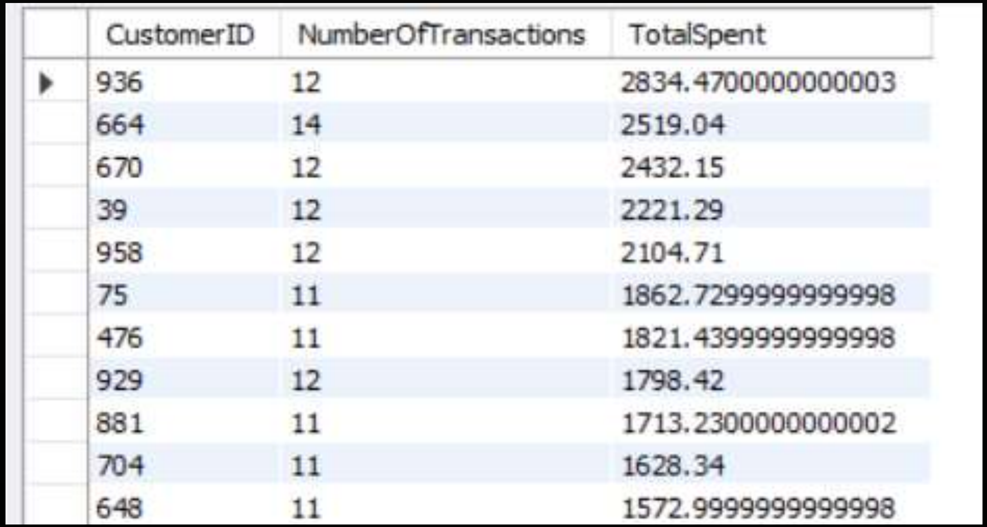


month	total_sales	previous_month_sales	mom_growth_percentage
1	104289.18	NULL	NULL
2	96690.99	104289.18	-7.29
3	103271.49	96690.99	6.81
4	101561.09	103271.49	-1.66
5	102998.84	101561.09	1.42
6	102210.28	102998.84	-0.77
7	91089.03	102210.28	-10.88

QUERY 13: Identify high-frequency customers by retrieving those with more than 10 transactions and a total spend exceeding 1000.

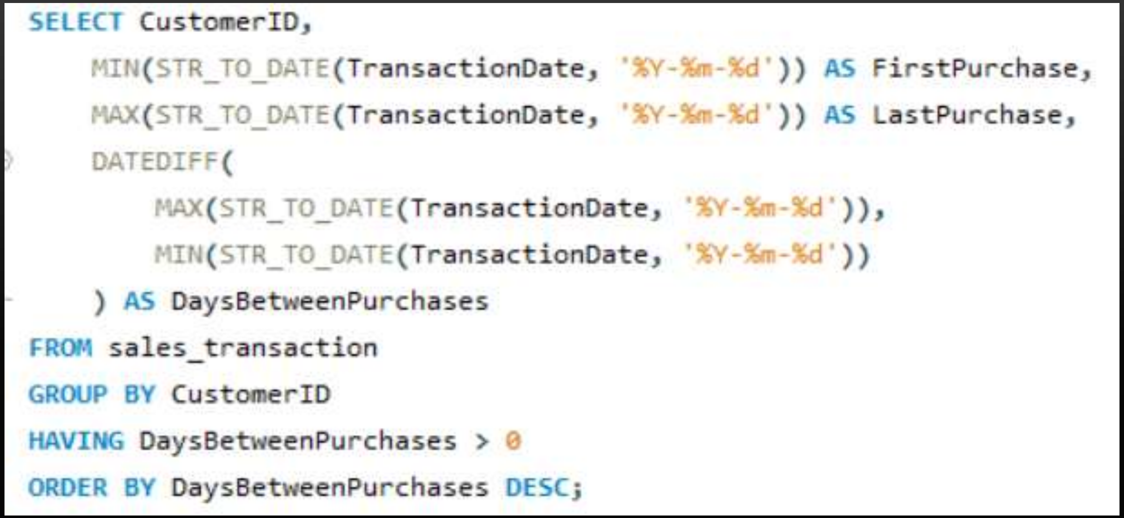


```sql
SELECT CustomerID, COUNT(*) NumberOfTransactions, SUM(QuantityPurchased * price) TotalSpent
FROM sales_transaction
GROUP BY CustomerID
HAVING COUNT(*) > 10
    AND SUM(QuantityPurchased * Price) > 1000
ORDER BY TotalSpent DESC;
```

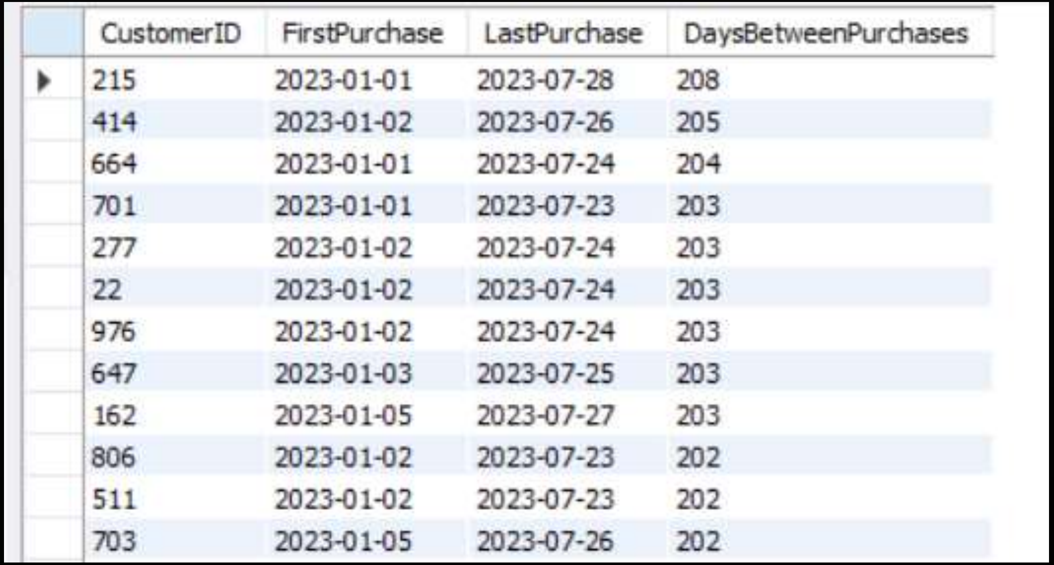


	CustomerID	NumberOfTransactions	TotalSpent
▶	936	12	2834.4700000000003
	664	14	2519.04
	670	12	2432.15
	39	12	2221.29
	958	12	2104.71
	75	11	1862.7299999999998
	476	11	1821.4399999999998
	929	12	1798.42
	881	11	1713.2300000000002
	704	11	1628.34
	648	11	1572.9999999999998

QUERY 14: Calculate the time difference between each customer's first and last purchase to analyze customer loyalty and longevity.

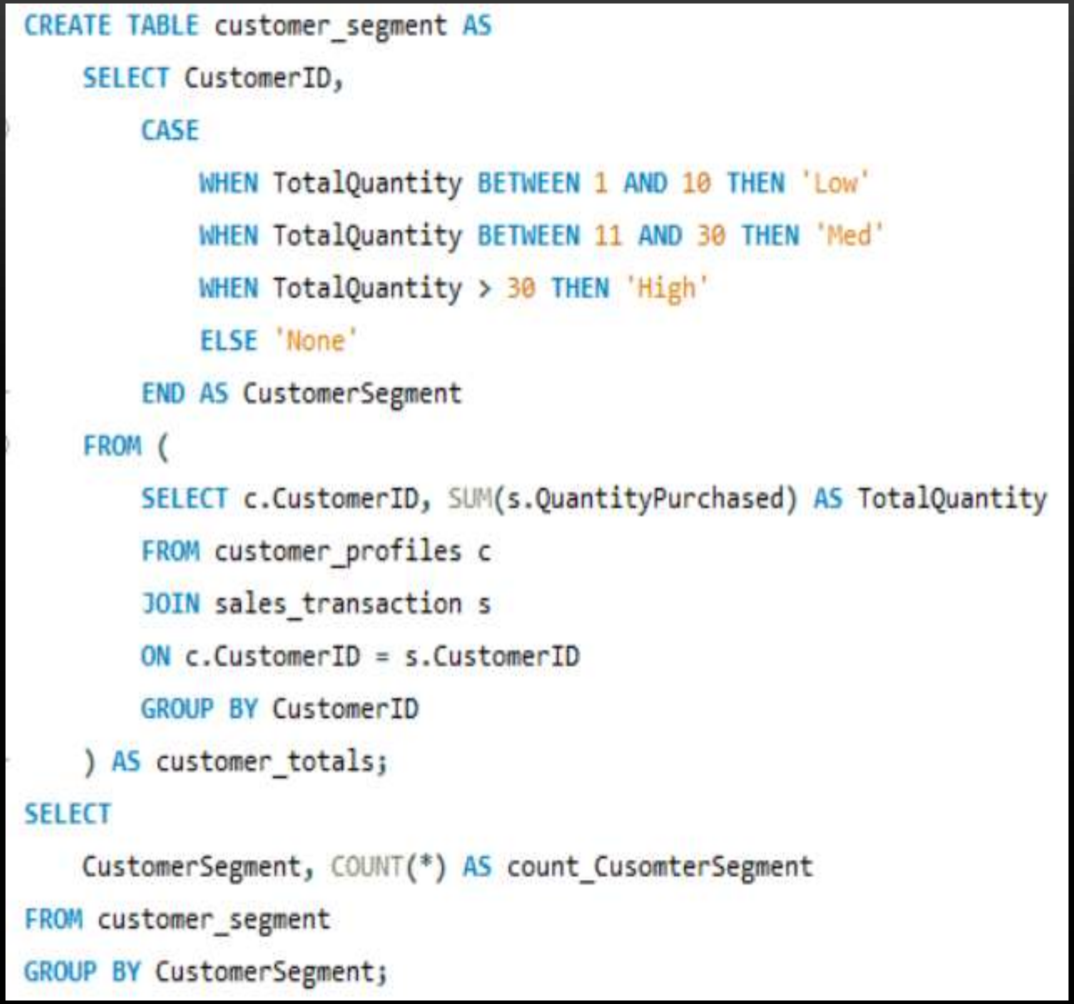


```sql
SELECT CustomerID,
    MIN(STR_TO_DATE(TransactionDate, '%Y-%m-%d')) AS FirstPurchase,
    MAX(STR_TO_DATE(TransactionDate, '%Y-%m-%d')) AS LastPurchase,
    DATEDIFF(
        MAX(STR_TO_DATE(TransactionDate, '%Y-%m-%d')),
        MIN(STR_TO_DATE(TransactionDate, '%Y-%m-%d'))
    ) AS DaysBetweenPurchases
FROM sales_transaction
GROUP BY CustomerID
HAVING DaysBetweenPurchases > 0
ORDER BY DaysBetweenPurchases DESC;
```

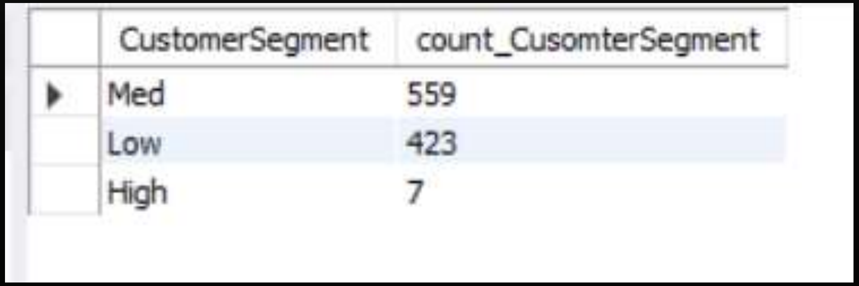


CustomerID	FirstPurchase	LastPurchase	DaysBetweenPurchases
215	2023-01-01	2023-07-28	208
414	2023-01-02	2023-07-26	205
664	2023-01-01	2023-07-24	204
701	2023-01-01	2023-07-23	203
277	2023-01-02	2023-07-24	203
22	2023-01-02	2023-07-24	203
976	2023-01-02	2023-07-24	203
647	2023-01-03	2023-07-25	203
162	2023-01-05	2023-07-27	203
806	2023-01-02	2023-07-23	202
511	2023-01-02	2023-07-23	202
703	2023-01-05	2023-07-26	202

QUERY 15: Segment customers into purchasing tiers based on the total quantity of products bought and count the number of customers in each segment.

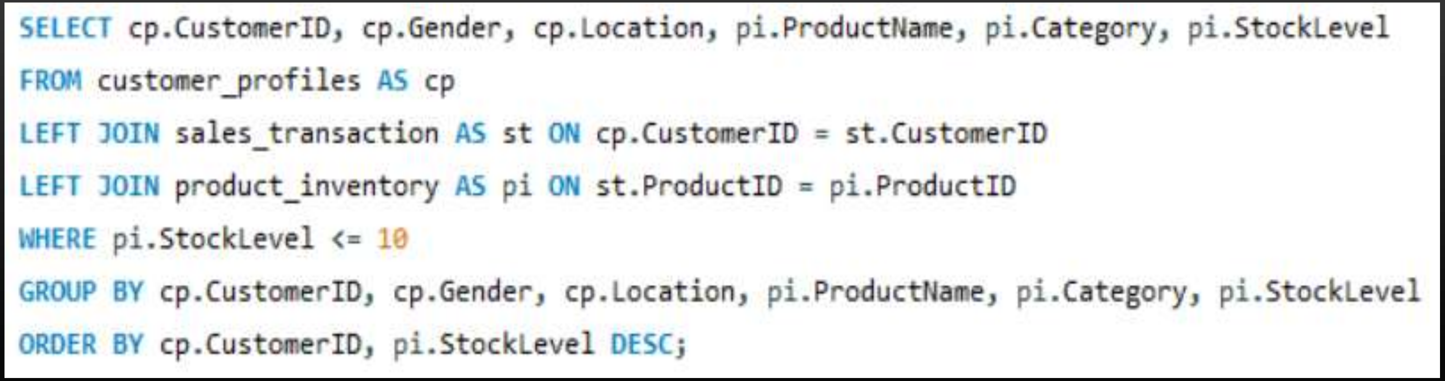


```sql
CREATE TABLE customer_segment AS
    SELECT CustomerID,
        CASE
            WHEN TotalQuantity BETWEEN 1 AND 10 THEN 'Low'
            WHEN TotalQuantity BETWEEN 11 AND 30 THEN 'Med'
            WHEN TotalQuantity > 30 THEN 'High'
            ELSE 'None'
        END AS CustomerSegment
    FROM (
        SELECT c.CustomerID, SUM(s.QuantityPurchased) AS TotalQuantity
        FROM customer_profiles c
        JOIN sales_transaction s
        ON c.CustomerID = s.CustomerID
        GROUP BY CustomerID
    ) AS customer_totals;
SELECT
    CustomerSegment, COUNT(*) AS count_CusomterSegment
FROM customer_segment
GROUP BY CustomerSegment;
```

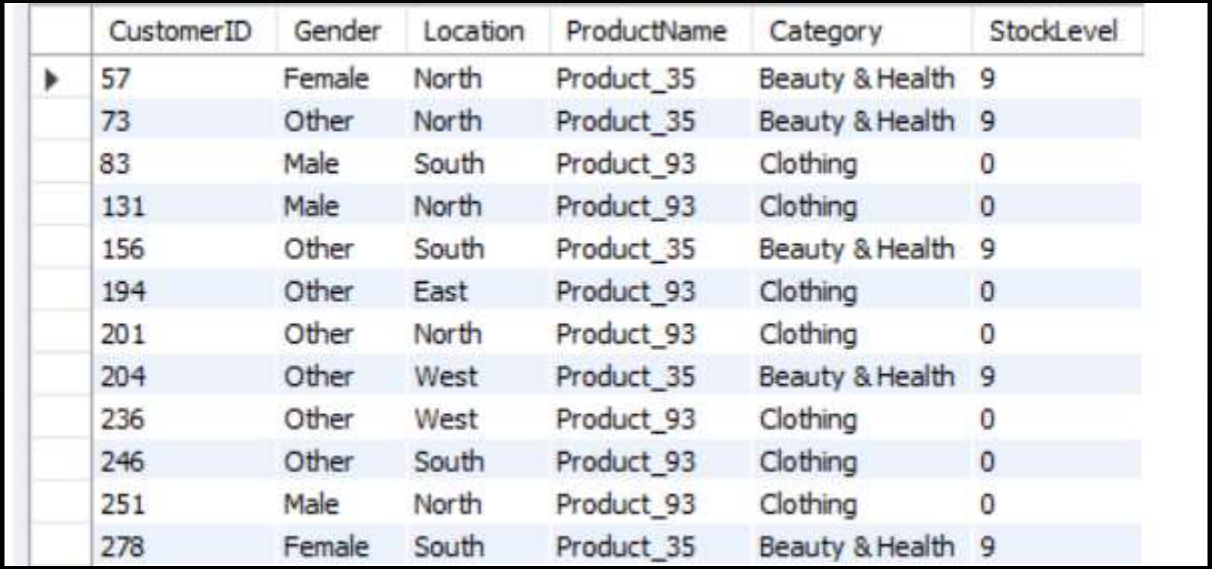


CustomerSegment	count_CusomterSegment
Med	559
Low	423
High	7

QUERY 16: To identify customers who have previously purchased products that are currently low in stock.

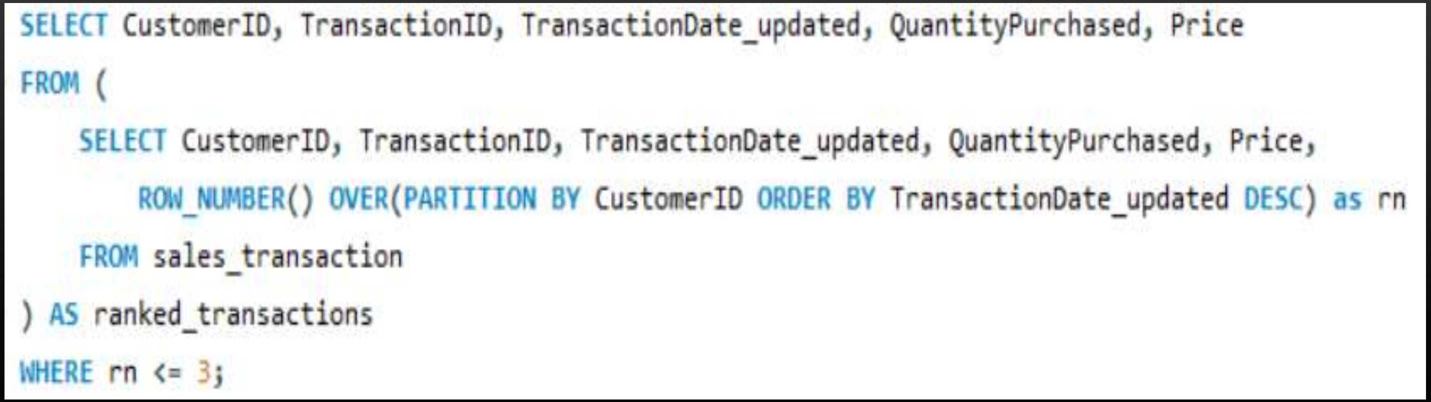


```sql
SELECT cp.CustomerID, cp.Gender, cp.Location, pi.ProductName, pi.Category, pi.StockLevel
FROM customer_profiles AS cp
LEFT JOIN sales_transaction AS st ON cp.CustomerID = st.CustomerID
LEFT JOIN product_inventory AS pi ON st.ProductID = pi.ProductID
WHERE pi.StockLevel <= 10
GROUP BY cp.CustomerID, cp.Gender, cp.Location, pi.ProductName, pi.Category, pi.StockLevel
ORDER BY cp.CustomerID, pi.StockLevel DESC;
```

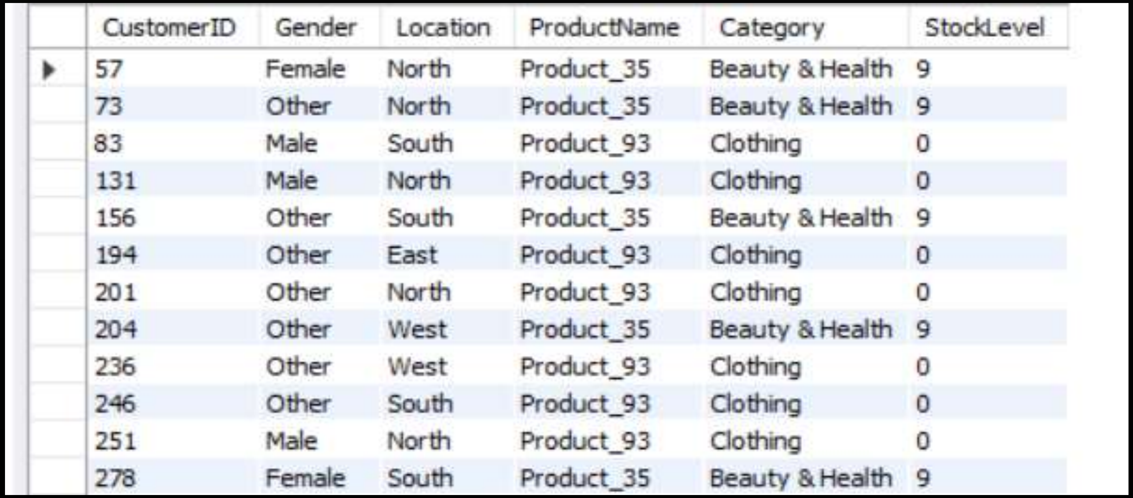


CustomerID	Gender	Location	ProductName	Category	StockLevel
57	Female	North	Product_35	Beauty & Health	9
73	Other	North	Product_35	Beauty & Health	9
83	Male	South	Product_93	Clothing	0
131	Male	North	Product_93	Clothing	0
156	Other	South	Product_35	Beauty & Health	9
194	Other	East	Product_93	Clothing	0
201	Other	North	Product_93	Clothing	0
204	Other	West	Product_35	Beauty & Health	9
236	Other	West	Product_93	Clothing	0
246	Other	South	Product_93	Clothing	0
251	Male	North	Product_93	Clothing	0
278	Female	South	Product_35	Beauty & Health	9

QUERY 17: Identify the top 3 most recent transactions for each customer.



```
SELECT CustomerID, TransactionID, TransactionDate_updated, QuantityPurchased, Price
FROM (
    SELECT CustomerID, TransactionID, TransactionDate_updated, QuantityPurchased, Price,
        ROW_NUMBER() OVER(PARTITION BY CustomerID ORDER BY TransactionDate_updated DESC) as rn
    FROM sales_transaction
) AS ranked_transactions
WHERE rn <= 3;
```



CustomerID	Gender	Location	ProductName	Category	StockLevel
57	Female	North	Product_35	Beauty & Health	9
73	Other	North	Product_35	Beauty & Health	9
83	Male	South	Product_93	Clothing	0
131	Male	North	Product_93	Clothing	0
156	Other	South	Product_35	Beauty & Health	9
194	Other	East	Product_93	Clothing	0
201	Other	North	Product_93	Clothing	0
204	Other	West	Product_35	Beauty & Health	9
236	Other	West	Product_93	Clothing	0
246	Other	South	Product_93	Clothing	0
251	Male	North	Product_93	Clothing	0
278	Female	South	Product_35	Beauty & Health	9

THANK YOU

Contact:

Email: dristi1249@gmail.com

LinkedIn: https://www.linkedin.com/in/dristi-handique/