**MS SQL QUERIES**

**COFFEE SHOP SALES SQL PROJECT**

I enhanced each category with deeper, more complex SQL queries that will be analysed the **Coffee Shop Sales** dataset in-depth. Here’s what I did for each level:

**Basic Level**

I added foundational insights that go beyond simple retrieval, including distinct counts, conditional filtering, and basic arithmetic on columns.

**Intermediate Level**

I added advanced filtering, grouping with multiple conditions, conditional aggregations, and ranking functions for detailed insights.

**Advanced Level**

These involve complex joins, window functions, CTEs (Common Table Expressions), and subqueries for richer, layered analysis.

**Basic Level** **Queries:**

1. **Total DAILY SALES, TOTAL QUANTITY and TOTAL ORDERS.**

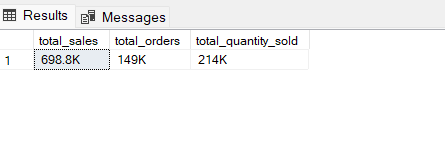
SELECT

CONCAT(ROUND(SUM(unit\_price \* transaction\_qty) / 1000, 1),'K') AS total\_sales,

CONCAT(ROUND(COUNT(transaction\_id) / 1000, 1),'K') AS total\_orders,

CONCAT(ROUND(SUM(transaction\_qty) / 1000, 1),'K') AS total\_quantity\_sold

FROM Transactions



**Insight**: This query helps to find the KPI of Sales, Quantity, Orders .

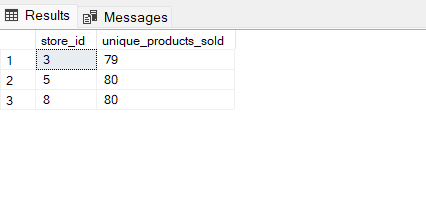
1. **Retrieve the number of unique products sold per store.**

SELECT store\_id,

COUNT(DISTINCT product\_id) AS unique\_products\_sold

FROM Transactions

GROUP BY store\_id;



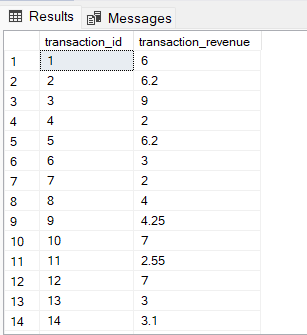
**Insight**: This query helps to understand the variety of products sold at each store.

1. **Calculate total revenue per transaction.**

SELECT transaction\_id,

transaction\_qty \* unit\_price AS transaction\_revenue

FROM Transactions;



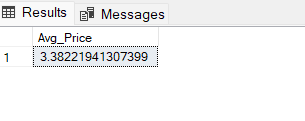
**Insight:** This provides revenue at a per-transaction level, useful for calculating averages or trends.

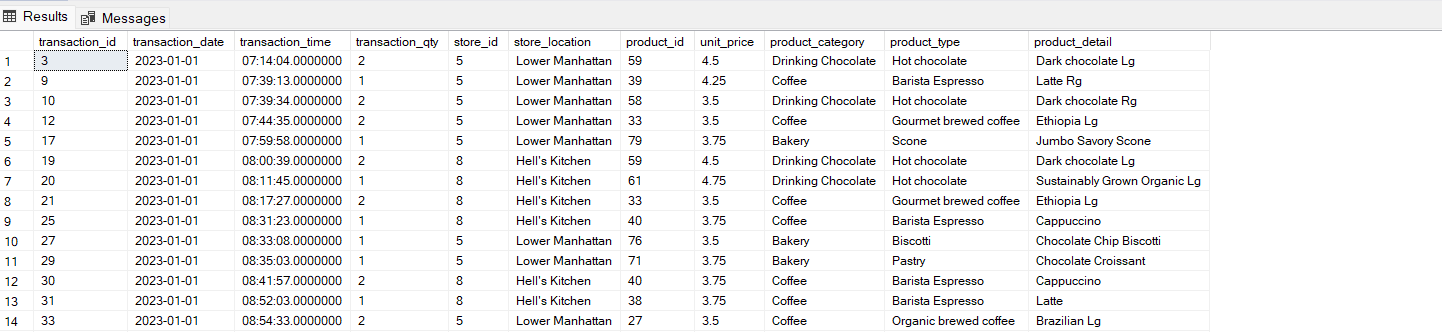
1. **List transactions where the unit price is above the average price of all products.**

SELECT \*

FROM Transactions

WHERE unit\_price > (SELECT AVG(unit\_price) FROM Transactions);





**Insight:** Identifies transactions involving premium-priced products, useful for premium segment analysis.

1. **Count the number of transactions per hour.**

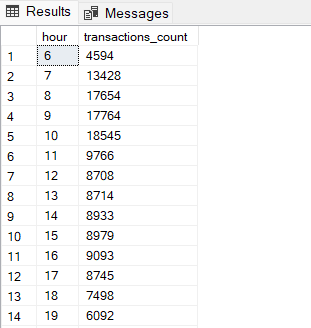
SELECT DATEPART(HOUR, transaction\_time) AS hour,

COUNT(transaction\_id) AS transactions\_count

FROM Transactions

GROUP BY DATEPART(HOUR, transaction\_time)

ORDER BY hour;



**Insight:** Helps identify hourly transaction patterns, indicating peak and off-peak times.

**Intermediate Level Queries:**

1. **Calculate total and average sales for each product category by month.**

SELECT DATEPART(YEAR, transaction\_date) AS year,

DATEPART(MONTH, transaction\_date) AS month,

product\_category,

-- Count(product\_category),

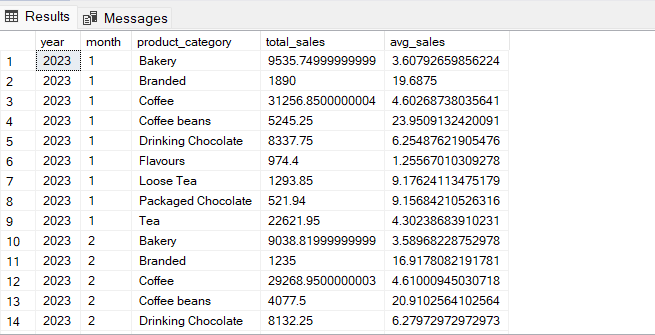
SUM(transaction\_qty \* unit\_price) AS total\_sales,

AVG(transaction\_qty \* unit\_price) AS avg\_sales

FROM Transactions

GROUP BY DATEPART(YEAR, transaction\_date), DATEPART(MONTH, transaction\_date), product\_category

ORDER BY DATEPART(YEAR, transaction\_date), DATEPART(MONTH, transaction\_date), product\_category;



**Insight:** Tracks monthly performance at a category level, helping to spot trends in product sales.

1. **Identify the top 3 best-selling products within each category based on quantity.**

SELECT Top 3 product\_category,

product\_id,

product\_detail,

product\_type,

SUM(transaction\_qty) AS total\_quantity\_sold

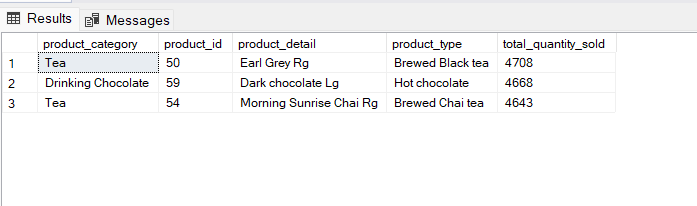
FROM Transactions

GROUP BY product\_category

,product\_id

, product\_detail,product\_type

ORDER BY total\_quantity\_sold DESC



**Insight:** Reveals the most popular products per category, which is useful for inventory and promotional planning.

1. **Calculate revenue generated by each store location for a given quarter (e.g., Q1 of 2023).**

SELECT store\_location,

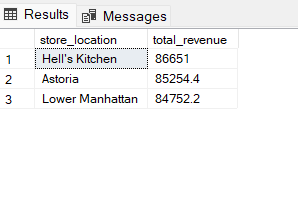
Round(SUM(transaction\_qty \* unit\_price),1) AS total\_revenue

FROM Transactions

WHERE transaction\_date BETWEEN '2023-01-01' AND '2023-03-31'

GROUP BY store\_location

ORDER BY total\_revenue DESC;



**Insight**: Provides a regional breakdown of revenue for specific timeframes, highlighting top-performing stores.

1. **Rank all products by sales within each store.**

SELECT store\_id,

product\_id,

product\_detail,

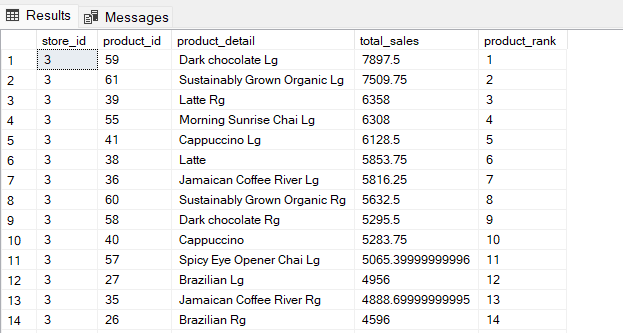
SUM(transaction\_qty \* unit\_price) AS total\_sales,

RANK() OVER (PARTITION BY store\_id ORDER BY SUM(transaction\_qty \* unit\_price) DESC) AS product\_rank

FROM Transactions

where store\_id = 3

GROUP BY store\_id, product\_id, product\_detail;



**Insight:** Ranks products in terms of sales within each store, showing which products perform best at a local level.

1. **Get the percentage share of each product category in total sales.**

SELECT product\_category,

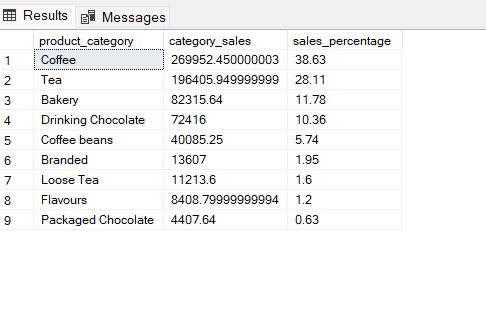
SUM(transaction\_qty \* unit\_price) AS category\_sales,

ROUND((SUM(transaction\_qty \* unit\_price) / (SELECT SUM(transaction\_qty \* unit\_price) FROM Transactions) \* 100), 2) AS sales\_percentage

FROM Transactions

GROUP BY product\_category

ORDER BY sales\_percentage DESC;



**Insight:** Shows how much each product category contributes to total revenue.

**Advanced Level Queries:**

1. **Determine each store’s peak sales hour.**

WITH hourly\_sales AS (

SELECT store\_id,

DATEPART(HOUR,transaction\_time) AS hour,

SUM(transaction\_qty \* unit\_price) AS sales

FROM Transactions

GROUP BY store\_id, DATEPART(HOUR,transaction\_time)

)

SELECT store\_id,

hour,

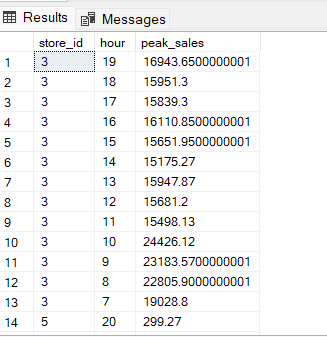
MAX(sales) AS peak\_sales

FROM hourly\_sales

GROUP BY store\_id, hour

ORDER BY store\_id, hour desc;

**Insight:** Identifies peak sales hour for each store, which is valuable for staffing and promotional strategies.



1. **Calculate the average time between transactions for each store.**

WITH transaction\_diffs AS (

SELECT store\_id,

DATEDIFF(SECOND, LAG(transaction\_time) OVER (PARTITION BY store\_id ORDER BY transaction\_time), transaction\_time) AS time\_diff

FROM Transactions

)

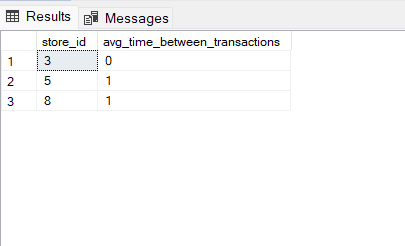
SELECT store\_id,

AVG(time\_diff) AS avg\_time\_between\_transactions

FROM transaction\_diffs

WHERE time\_diff IS NOT NULL

GROUP BY store\_id;



**Insight:** Measures transaction frequency at each store, giving insight into customer traffic.

1. **List stores with above-average sales performance for a specified period.**

WITH avg\_sales AS (

SELECT AVG(transaction\_qty \* unit\_price) AS average\_sales

FROM Transactions

WHERE transaction\_date BETWEEN '2023-04-01' AND '2023-06-30'

)

SELECT t.store\_id,

t.store\_location,

SUM(t.transaction\_qty \* t.unit\_price) AS total\_sales

FROM Transactions t

WHERE t.transaction\_date BETWEEN '2023-04-01' AND '2023-06-30'

GROUP BY t.store\_id, t.store\_location

HAVING SUM(t.transaction\_qty \* t.unit\_price) > (

SELECT average\_sales

FROM avg\_sales

);

-- Another way using CROSS JOIN:

WITH avg\_sales AS (

SELECT AVG(transaction\_qty \* unit\_price) AS average\_sales

FROM Transactions

WHERE transaction\_date BETWEEN '2023-04-01' AND '2023-06-30'

)

SELECT t.store\_id,

t.store\_location,

SUM(t.transaction\_qty \* t.unit\_price) AS total\_sales

FROM Transactions t

CROSS JOIN avg\_sales

WHERE t.transaction\_date BETWEEN '2023-04-01' AND '2023-06-30'

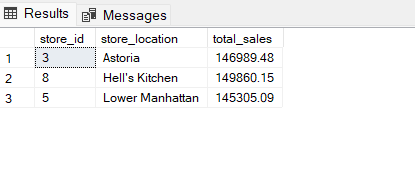
GROUP BY t.store\_id, t.store\_location

HAVING SUM(t.transaction\_qty \* t.unit\_price) > (

SELECT average\_sales

FROM avg\_sales

);



**Insight:** Highlights stores that are overperforming relative to the average sales.

1. **Identify products that make up the top 20% of total sales.**

WITH sales\_data AS (

SELECT product\_id,

product\_detail,

SUM(transaction\_qty \* unit\_price) AS product\_sales,

NTILE(5) OVER (ORDER BY SUM(transaction\_qty \* unit\_price) DESC) AS sales\_quintile

FROM Transactions

GROUP BY product\_id, product\_detail

)

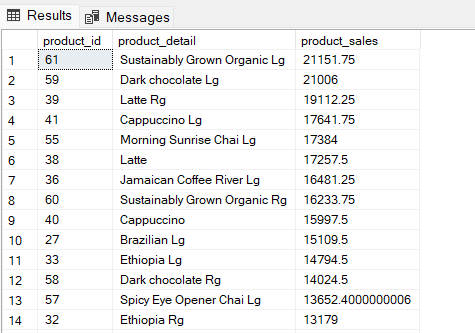
SELECT product\_id,

product\_detail,

product\_sales

FROM sales\_data

WHERE sales\_quintile = 1;



**Insight:** Lists high-performing products by sales, focusing on those that contribute significantly to revenue.

1. **Analyze month-over-month growth % for each product category.**

SELECT product\_category,

DATEPART(YEAR, transaction\_date) AS year,

DATEPART(MONTH, transaction\_date) AS month,

SUM(transaction\_qty \* unit\_price) AS monthly\_sales,

LAG(SUM(transaction\_qty \* unit\_price)) OVER (PARTITION BY product\_category ORDER BY DATEPART(YEAR, transaction\_date), DATEPART(MONTH, transaction\_date)) AS previous\_month\_sales,

(FORMAT(

(SUM(transaction\_qty \* unit\_price) - LAG(SUM(transaction\_qty \* unit\_price)) OVER (PARTITION BY product\_category ORDER BY DATEPART(YEAR, transaction\_date), DATEPART(MONTH, transaction\_date)))

/ NULLIF(LAG(SUM(transaction\_qty \* unit\_price)) OVER (PARTITION BY product\_category ORDER BY DATEPART(YEAR, transaction\_date), DATEPART(MONTH, transaction\_date)), 0) \* 100,

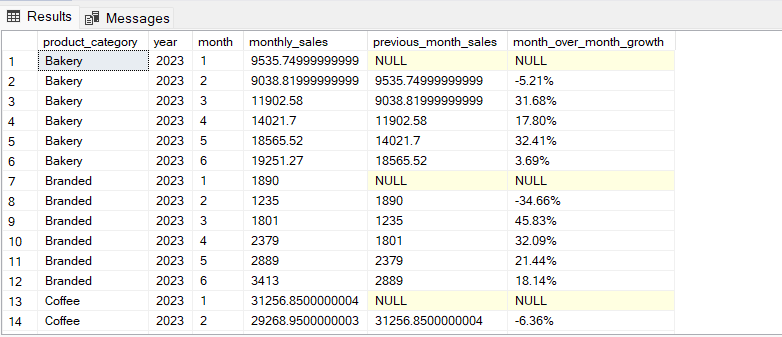
'N2'

) + '%' ) AS month\_over\_month\_growth

FROM Transactions

GROUP BY product\_category, DATEPART(YEAR, transaction\_date), DATEPART(MONTH, transaction\_date)

ORDER BY product\_category, year, month;



**Insight:** Measures growth trends for each category month-over-month, revealing patterns in product demand.

1. **Find Products Performing Above Average in Specific Stores.**

WITH ProductAvgRevenue AS (

SELECT product\_id,

product\_detail,

Round(AVG(transaction\_qty \* unit\_price),2) AS avg\_product\_revenue

FROM Transactions

GROUP BY product\_id, product\_detail

)

SELECT t.store\_id,

t.store\_location,

t.product\_id,

t.product\_detail,

Round(SUM(t.transaction\_qty \* t.unit\_price),2) AS store\_product\_revenue,

par.avg\_product\_revenue

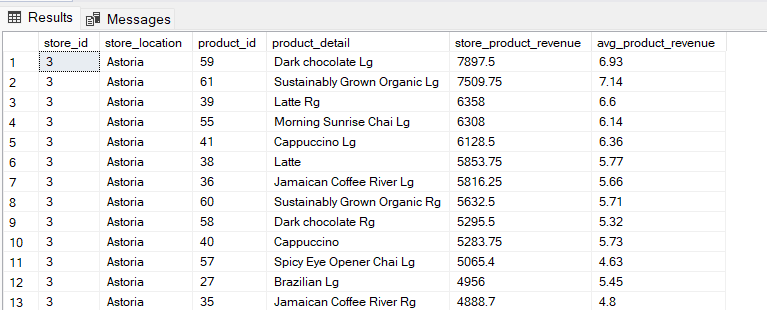
FROM Transactions t

JOIN ProductAvgRevenue par ON t.product\_id = par.product\_id

GROUP BY t.store\_id, t.store\_location, t.product\_id, t.product\_detail, par.avg\_product\_revenue

HAVING Round(SUM(t.transaction\_qty \* t.unit\_price),2) > par.avg\_product\_revenue

ORDER BY t.store\_id, store\_product\_revenue DESC;

****

**Insights:** This query helps identify product strengths at the store level and can guide targeted inventory decisions or promotions.