DATABASE ORGANISATION CS – 425

DELIVERABLE – 3

INVENTORY MANAGEMENT SYSTEM

GROUP – 5

NAME	**CWID**	**CONTRIBUTION**
RITHIKA KAVITA SURESH	A20564346	33.3%
EKTA SHUKLA	A20562374	33.3%
PRAVEENRAJ SEENIVASAN	A20567127	33.3%

Q1.

<u>**Query Description:**</u>
Top 5 Best-Selling Products
This query identifies the top 5 products with the highest total sales quantity.
SELECT Clause:

- p.PID: Selects the Product ID from the Product table
- p.Pname: Selects the Product Name from the Product table
- SUM(sd.SDquantity) AS TotalSold: Calculates the total quantity sold for each product

FROM and JOIN Clauses:

- FROM Product p: Specifies the main table as Product, aliased as 'p'
- JOIN Sale_Details sd ON p.PID = sd.PID: Joins the Sale_Details table with Product table using PID as the joining key

GROUP BY Clause:

- Groups the results by p.PID and p.Pname to aggregate sales for each unique product

ORDER BY Clause:

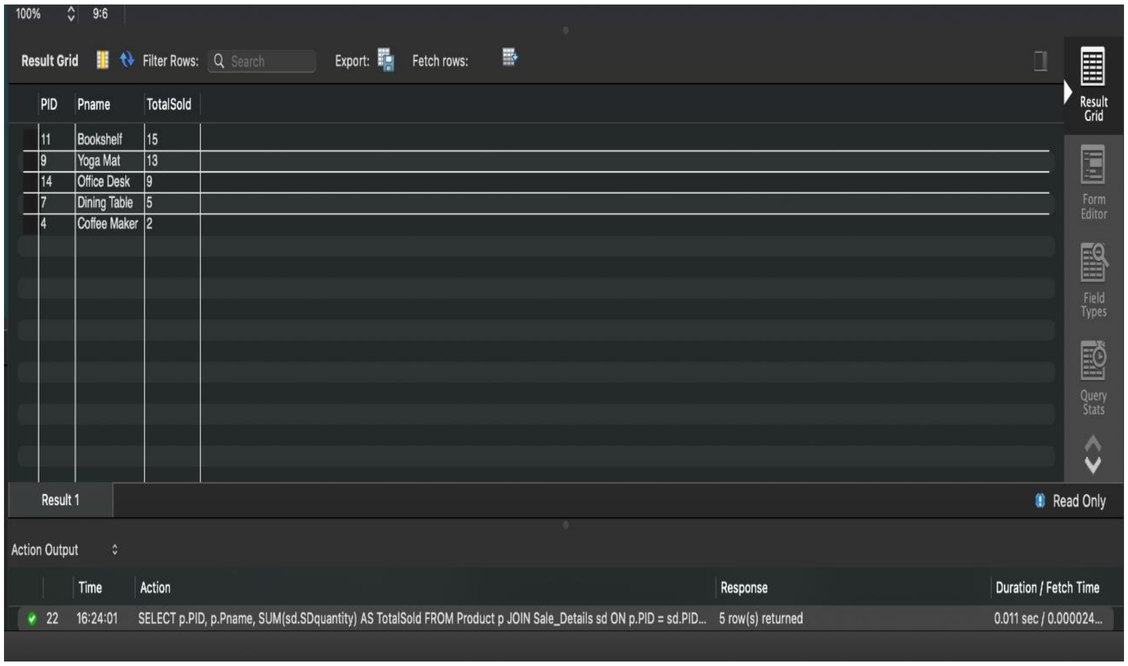
- Sorts the results by TotalSold in descending order (DESC) to rank products from highest to lowest sales

LIMIT Clause:

- Restricts the output to only the top 5 results

<u>**SQL Statement:**</u>

```
SELECT p.PID, p.Pname, SUM(sd.SDquantity) AS TotalSold
FROM Product p
JOIN Sale_Details sd ON p.PID = sd.PID
GROUP BY p.PID, p.Pname
ORDER BY TotalSold DESC
LIMIT 5;
```



PID	Pname	TotalSold
11	Bookshelf	15
9	Yoga Mat	13
14	Office Desk	9
7	Dining Table	5
4	Coffee Maker	2

	Time	Action	Response	Duration / Fetch Time
22	16:24:01	SELECT p.PID, p.Pname, SUM(sd.SDquantity) AS TotalSold FROM Product p JOIN Sale_Details sd ON p.PID = sd.PID...	5 row(s) returned	0.011 sec / 0.000024...

Q2.

<u>Query Description:</u>
Monthly Sales Trend

This query shows the total sales amount for each month, using a window function to calculate the running total.

SELECT Clause:

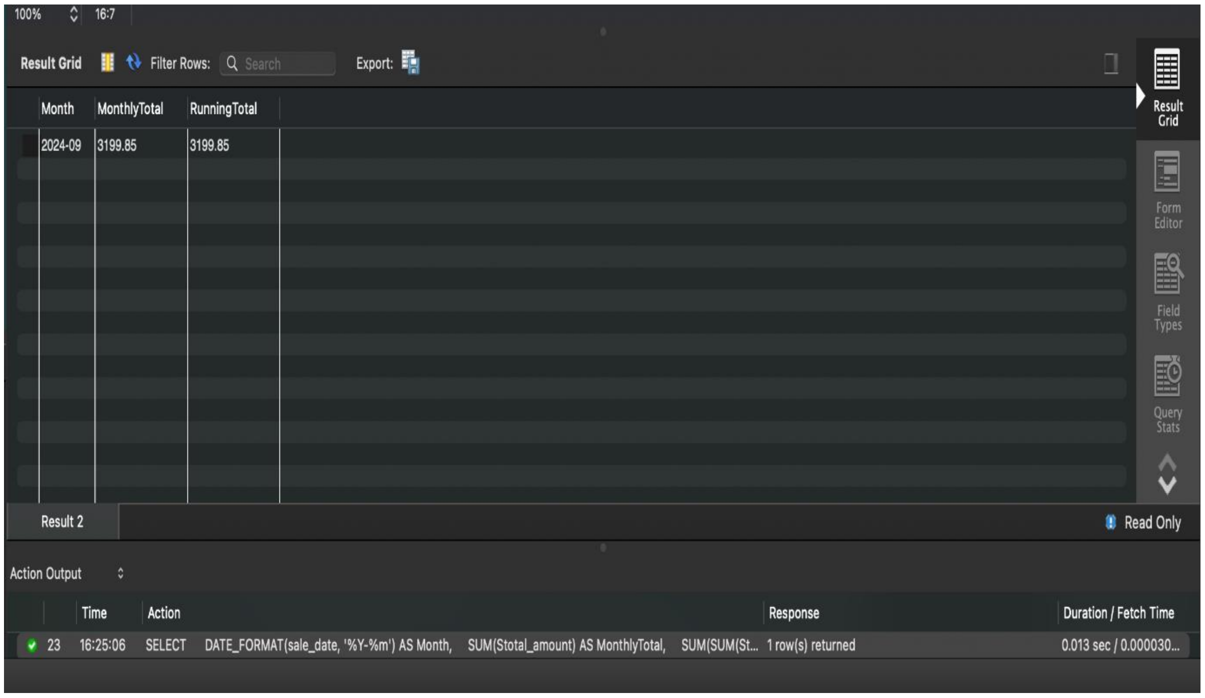
- DATE_FORMAT(sale_date, '%Y-%m') AS Month: Converts the sale date to a year-month format (e.g., '2024-10'), allowing for monthly aggregation.
- SUM(Stotal_amount) AS MonthlyTotal: Calculates the total sales amount for each month.
- SUM(SUM(Stotal_amount)) OVER (ORDER BY DATE_FORMAT(sale_date, '%Y-%m')) AS RunningTotal: Computes a running total of sales across months using a window function.

FROM, GROUP BY, and ORDER BY Clauses:

- The query uses the Sale table, groups results by the formatted month, and orders them chronologically.

<u>SQL Statement:</u>

```
SELECT
   DATE_FORMAT(sale_date, '%Y-%m') AS Month,
   SUM(Stotal_amount) AS MonthlyTotal,
   SUM(SUM(Stotal_amount)) OVER (ORDER BY DATE_FORMAT(sale_date, '%Y-%m'))
AS RunningTotal
FROM Sale
GROUP BY Month
ORDER BY Month;
```



Month	MonthlyTotal	RunningTotal
2024-09	3199.85	3199.85

	Time	Action	Response	Duration / Fetch Time
23	16:25:06	SELECT DATE_FORMAT(sale_date, '%Y-%m') AS Month, SUM(Stotal_amount) AS MonthlyTotal, SUM(SUM(St...	1 row(s) returned	0.013 sec / 0.000030...

Q3.

<u>**Query Description:**</u>
Product Reorder Alert
This query identifies products that need reordering based on their current stock quantity.
SELECT Clause:

- p.PID: Selects the Product ID
- p.Pname: Selects the Product Name
- p.Pstock_quantity: Selects the current stock quantity of the product
- CASE statement: Creates a new column ReorderStatus based on stock quantity

CASE Statement:

- Assigns 'Urgent Reorder' if stock is less than 10
- Assigns 'Reorder Soon' if stock is between 10 and 49
- Assigns 'Stock Sufficient' if stock is 50 or more

FROM Clause:

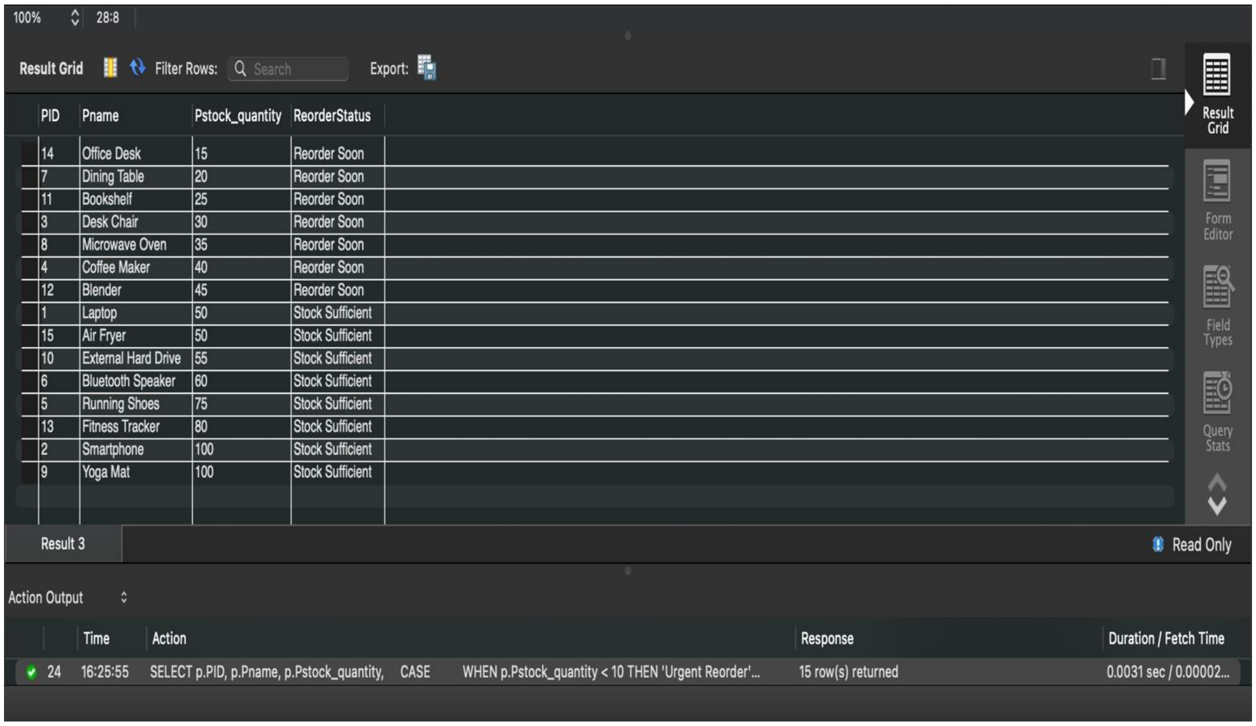
- Uses the Product table, aliased as 'p'

ORDER BY Clause:

- Sorts the results by p.Pstock_quantity in ascending order

<u>**SQL Statement:**</u>

```
SELECT p.PID, p.Pname, p.Pstock_quantity,
   CASE
      WHEN p.Pstock_quantity < 10 THEN 'Urgent Reorder'
      WHEN p.Pstock_quantity < 50 THEN 'Reorder Soon'
      ELSE 'Stock Sufficient'
   END AS ReorderStatus
FROM Product p
ORDER BY p.Pstock_quantity;
```



PID	Pname	Pstock_quantity	ReorderStatus
14	Office Desk	15	Reorder Soon
7	Dining Table	20	Reorder Soon
11	Bookshelf	25	Reorder Soon
3	Desk Chair	30	Reorder Soon
8	Microwave Oven	35	Reorder Soon
4	Coffee Maker	40	Reorder Soon
12	Blender	45	Reorder Soon
1	Laptop	50	Stock Sufficient
15	Air Fryer	50	Stock Sufficient
10	External Hard Drive	55	Stock Sufficient
6	Bluetooth Speaker	60	Stock Sufficient
5	Running Shoes	75	Stock Sufficient
13	Fitness Tracker	80	Stock Sufficient
2	Smartphone	100	Stock Sufficient
9	Yoga Mat	100	Stock Sufficient

	Time	Action	Response	Duration / Fetch Time
24	16:25:55	SELECT p.PID, p.Pname, p.Pstock_quantity, CASE WHEN p.Pstock_quantity < 10 THEN 'Urgent Reorder'...	15 row(s) returned	0.0031 sec / 0.00002...

Q4.

<u>Query Description:</u>
Customer Ranking by Total Purchases

This query ranks customers based on their total purchase amount using a window function.

SELECT Clause:

- c.CID: Selects the Customer ID
- c.Cname: Selects the Customer Name
- SUM(s.Stotal_amount) AS TotalPurchases: Calculates the total purchase amount for each customer
- RANK() OVER (ORDER BY SUM(s.Stotal_amount) DESC) AS CustomerRank: Assigns a rank to each customer based on their total purchases

FROM and JOIN Clauses:

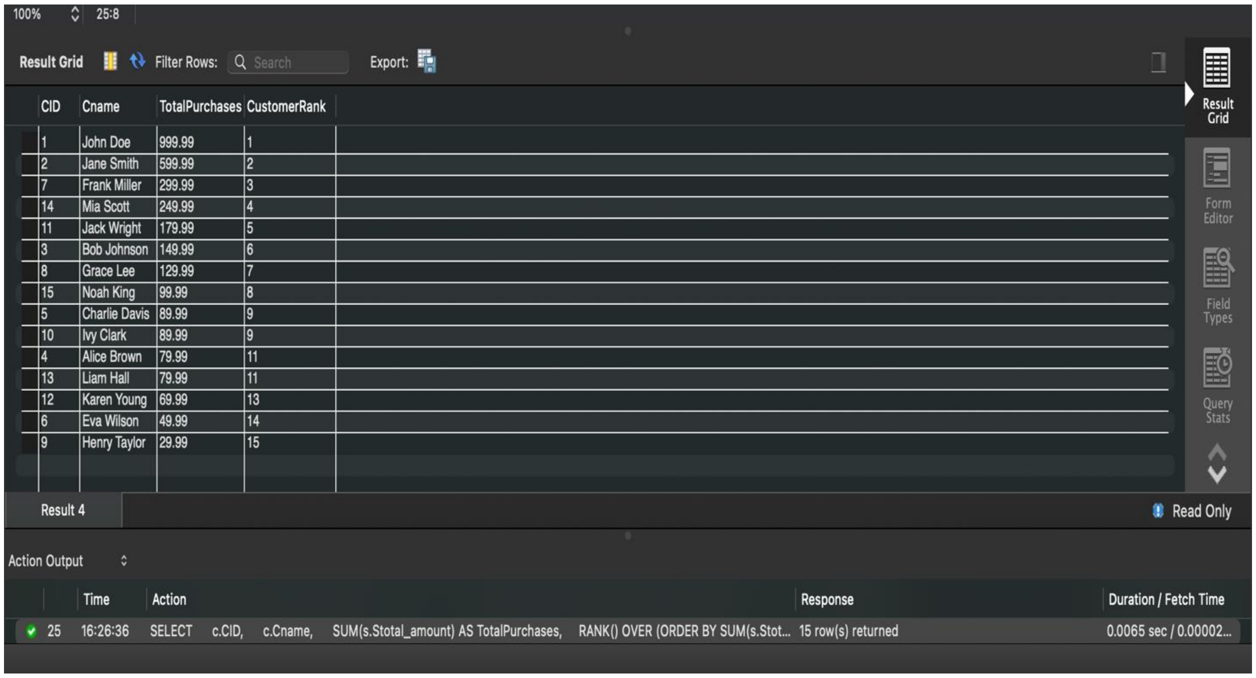
- FROM Customer c: Specifies the main table as Customer, aliased as 'c'
- JOIN Sale s ON c.CID = s.CID: Joins the Sale table with Customer table using CID as the joining key

GROUP BY Clause:

- Groups the results by c.CID and c.Cname to aggregate sales for each unique customer

<u>SQL Statement:</u>

```
SELECT
   c.CID,
   c.Cname,
   SUM(s.Stotal_amount) AS TotalPurchases,
   RANK() OVER (ORDER BY SUM(s.Stotal_amount) DESC) AS CustomerRank
FROM Customer c
JOIN Sale s ON c.CID = s.CID
GROUP BY c.CID, c.Cname;
```



```
CID  Cname         TotalPurchases  CustomerRank
1    John Doe      999.99          1
2    Jane Smith    599.99          2
7    Frank Miller  299.99          3
14   Mia Scott     249.99          4
11   Jack Wright   179.99          5
3    Bob Johnson   149.99          6
8    Grace Lee     129.99          7
15   Noah King     99.99           8
5    Charlie Davis 89.99           9
10   Ivy Clark     89.99           9
4    Alice Brown   79.99           11
13   Liam Hall     79.99           11
12   Karen Young   69.99           13
6    Eva Wilson    49.99           14
9    Henry Taylor  29.99           15
```

```
    Time      Action                                                                                   Response             Duration / Fetch Time
25  16:26:36  SELECT  c.CID,  c.Cname,  SUM(s.Stotal_amount) AS TotalPurchases,  RANK() OVER (ORDER BY SUM(s.Stot...  15 row(s) returned  0.0065 sec / 0.00002...
```

Q5.

<u>Query Description:</u>
Supplier Performance Analysis
This query analyzes supplier performance based on order fulfillment time.
SELECT Clause:

- s.SUPID: Selects the Supplier ID
- s.Sname: Selects the Supplier Name
- AVG(DATEDIFF(o.order_date, o.delivery_date)) AS AvgDeliveryDays: Calculates the average number of days between order and delivery dates
- COUNT(o.OID) AS TotalOrders: Counts the total number of orders for each supplier

FROM and JOIN Clauses:

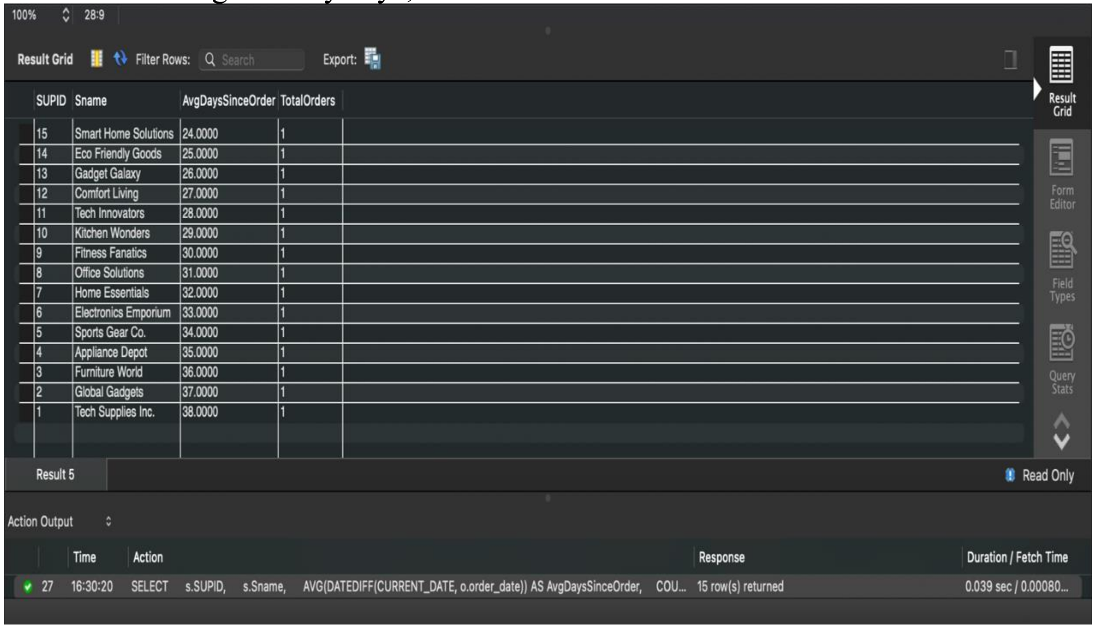
- FROM Supplier s: Specifies the main table as Supplier, aliased as 's'
- JOIN Order o ON s.SUPID = o.SUPID: Joins the Order table with Supplier table using SUPID as the joining key

GROUP BY Clause:

- Groups the results by s.SUPID and s.Sname to aggregate data for each unique supplier

ORDER BY Clause:

- Sorts the results by AvgDeliveryDays in ascending order, prioritizing suppliers with shorter delivery times

<u>SQL Statement:</u>
```
SELECT
   s.SUPID,
   s.Sname,
   AVG(DATEDIFF(o.order_date, o.delivery_date)) AS AvgDeliveryDays,
   COUNT(o.OID) AS TotalOrders
FROM Supplier s
JOIN `Order` o ON s.SUPID = o.SUPID
GROUP BY s.SUPID, s.Sname
ORDER BY AvgDeliveryDays;
```


SUPID	Sname	AvgDaysSinceOrder	TotalOrders
15	Smart Home Solutions	24.0000	1
14	Eco Friendly Goods	25.0000	1
13	Gadget Galaxy	26.0000	1
12	Comfort Living	27.0000	1
11	Tech Innovators	28.0000	1
10	Kitchen Wonders	29.0000	1
9	Fitness Fanatics	30.0000	1
8	Office Solutions	31.0000	1
7	Home Essentials	32.0000	1
6	Electronics Emporium	33.0000	1
5	Sports Gear Co.	34.0000	1
4	Appliance Depot	35.0000	1
3	Furniture World	36.0000	1
2	Global Gadgets	37.0000	1
1	Tech Supplies Inc.	38.0000	1

```
     Time      Action                                                                                          Response             Duration / Fetch Time
✓ 27 16:30:20  SELECT  s.SUPID,  s.Sname,  AVG(DATEDIFF(CURRENT_DATE, o.order_date)) AS AvgDaysSinceOrder,  COU...  15 row(s) returned   0.039 sec / 0.00080...
```

Q6.

<u>**Query Description:**</u>
Top Customers by Total Purchase Amount
This query will show the top 5 customers based on their total purchase amount. It joins the Customer and Sale tables, sums up the total amount for each customer, and orders the results in descending order.

SELECT Clause:
- c.CID: Selects the Customer ID
- c.Cname: Selects the Customer Name
- SUM(s.Stotal_amount) AS TotalPurchaseAmount: Calculates the total purchase amount for each customer

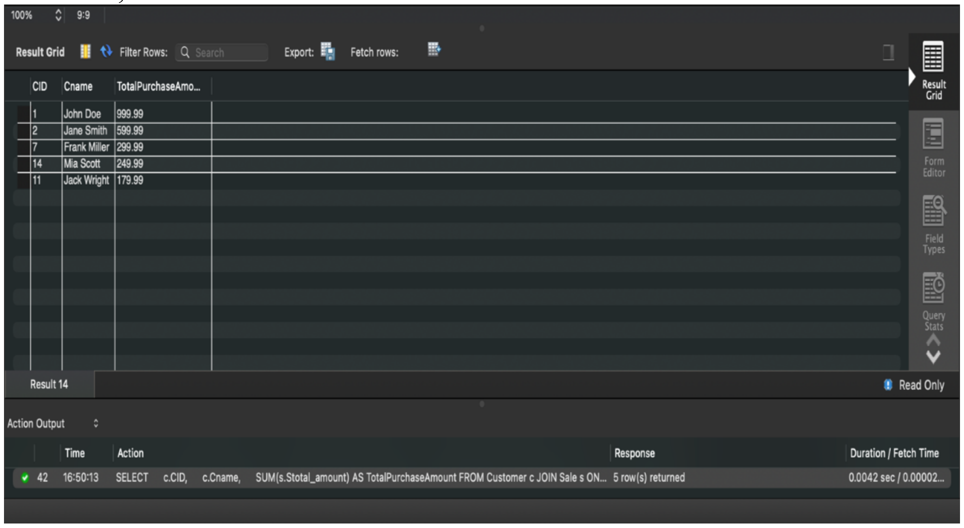
FROM and JOIN Clauses:
- FROM Customer c: Specifies the main table as Customer, aliased as 'c'
- JOIN Sale s ON c.CID = s.CID: Joins the Sale table with Customer table using CID as the joining key

GROUP BY Clause:
- Groups the results by c.CID and c.Cname to aggregate sales for each unique customer

ORDER BY Clause:
- Sorts the results by TotalPurchaseAmount in descending order (DESC) to rank customers from highest to lowest total purchases

LIMIT Clause:
- Restricts the output to only the top 5 results

<u>**SQL Statement:**</u>
```
SELECT
   c.CID,
   c.Cname,
   SUM(s.Stotal_amount) AS TotalPurchaseAmount
FROM Customer c
JOIN Sale s ON c.CID = s.CID
GROUP BY c.CID, c.Cname
ORDER BY TotalPurchaseAmount DESC
LIMIT 5;
```



CID	Cname	TotalPurchaseAmo...
1	John Doe	999.99
2	Jane Smith	599.99
7	Frank Miller	299.99
14	Mia Scott	249.99
11	Jack Wright	179.99

	Time	Action	Response	Duration / Fetch Time
42	16:50:13	SELECT c.CID, c.Cname, SUM(s.Stotal_amount) AS TotalPurchaseAmount FROM Customer c JOIN Sale s ON...	5 row(s) returned	0.0042 sec / 0.00002...

Q7.

Query Description:

Product Category Sales Analysis

This query uses OLAP features to analyze sales by product category and quarter.

SELECT Clause:

- p.Pcategory: Selects the Product Category
- QUARTER(s.sale_date) AS Quarter: Extracts the quarter from the sale date
- SUM(sd.SDquantity * p.Punit_price) AS TotalSales: Calculates the total sales amount

FROM and JOIN Clauses:

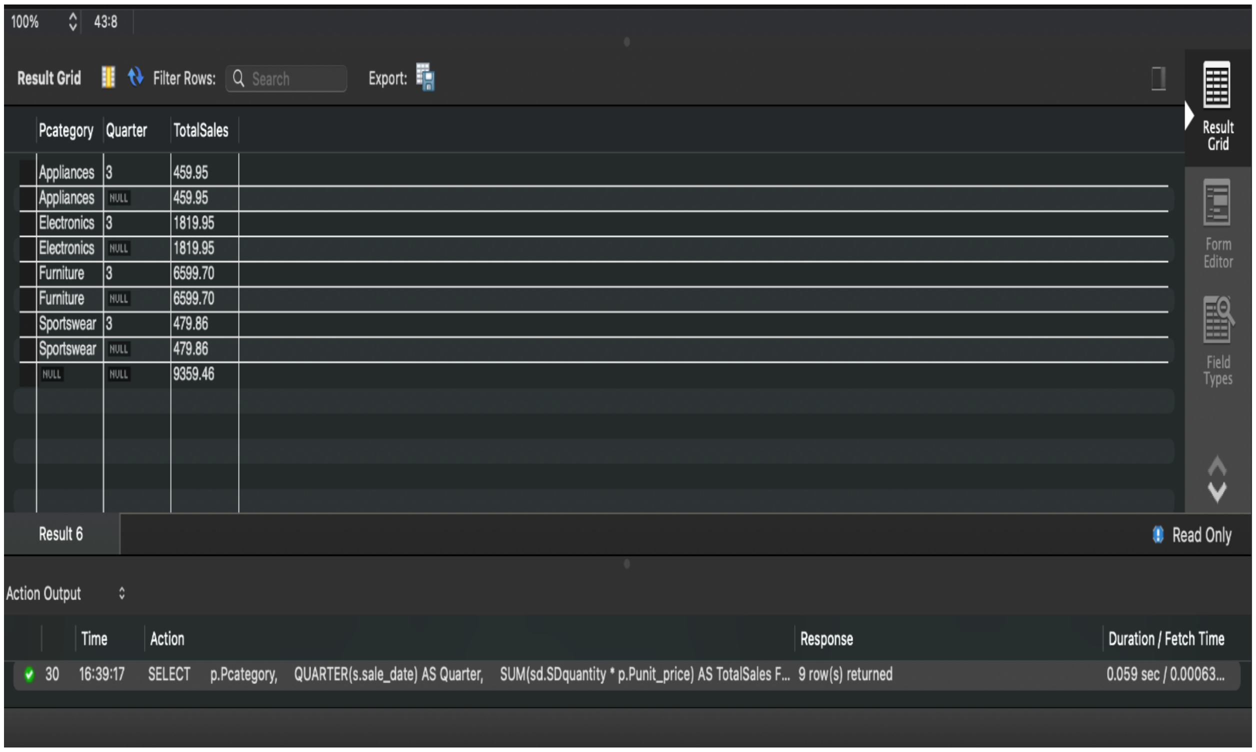
- FROM Product p: Starts with the Product table
- JOIN Sale_Details sd ON p.PID = sd.PID: Links Product to Sale_Details
- JOIN Sale s ON sd.SID = s.SID: Further joins with the Sale table

GROUP BY Clause:

- GROUP BY p.Pcategory, Quarter WITH ROLLUP: Groups results by category and quarter, with additional summary rows

SQL Statement:

```
SELECT
   p.Pcategory,
   QUARTER(s.sale_date) AS Quarter,
   SUM(sd.SDquantity * p.Punit_price) AS TotalSales
FROM Product p
JOIN Sale_Details sd ON p.PID = sd.PID
JOIN Sale s ON sd.SID = s.SID
GROUP BY p.Pcategory, Quarter WITH ROLLUP;
```



Pcategory	Quarter	TotalSales
Appliances	3	459.95
Appliances	NULL	459.95
Electronics	3	1819.95
Electronics	NULL	1819.95
Furniture	3	6599.70
Furniture	NULL	6599.70
Sportswear	3	479.86
Sportswear	NULL	479.86
NULL	NULL	9359.46

	Time	Action	Response	Duration / Fetch Time
✓ 30	16:39:17	SELECT p.Pcategory, QUARTER(s.sale_date) AS Quarter, SUM(sd.SDquantity * p.Punit_price) AS TotalSales F...	9 row(s) returned	0.059 sec / 0.00063...

Q8.

Query Description:
Customer Segmentation
This query segments customers based on their purchase frequency and total spend.
SELECT Clause:

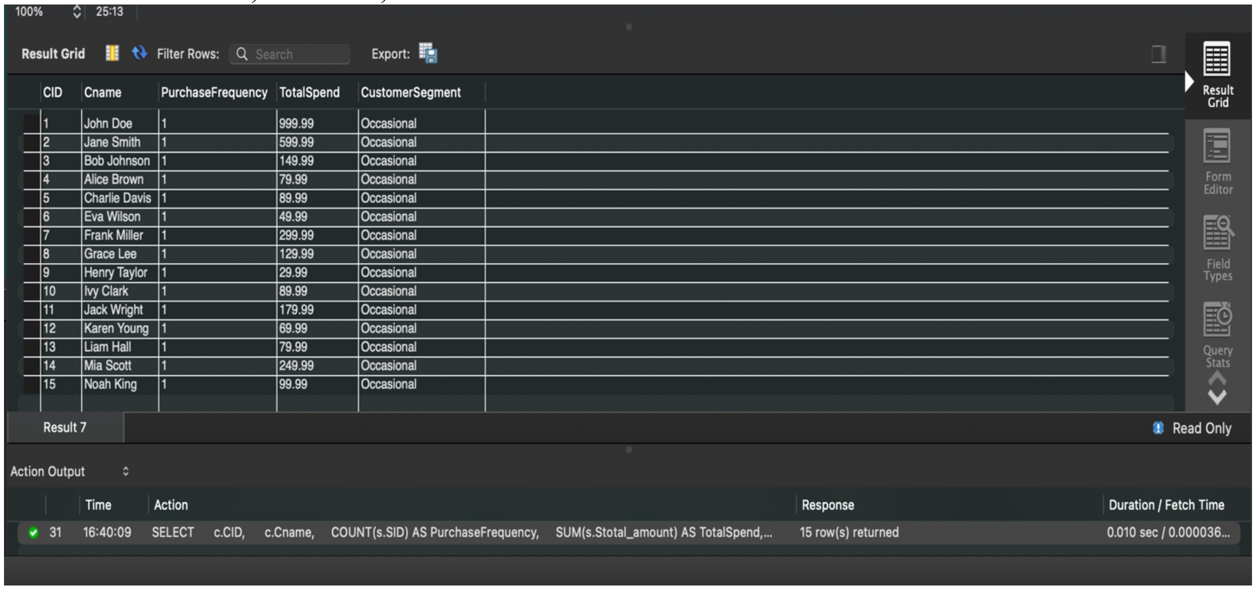
- c.CID and c.Cname: Selects the Customer ID and Name
- COUNT(s.SID) AS PurchaseFrequency: Counts the number of sales for each customer
- SUM(s.Stotal_amount) AS TotalSpend: Calculates the total amount spent by each customer
- CASE statement: Categorizes customers into segments based on their purchase behavior

FROM and JOIN Clauses:

- FROM Customer c: Specifies the main table as Customer
- LEFT JOIN Sale s ON c.CID = s.CID: Ensures all customers are included, even those without sales

GROUP BY Clause:

- Groups the results by c.CID and c.Cname to aggregate data for each unique customer

SQL Statement:
```
SELECT
   c.CID,
   c.Cname,
   COUNT(s.SID) AS PurchaseFrequency,
   SUM(s.Stotal_amount) AS TotalSpend,
   CASE
      WHEN COUNT(s.SID) > 10 AND SUM(s.Stotal_amount) > 10000 THEN 'VIP'
      WHEN COUNT(s.SID) > 5 OR SUM(s.Stotal_amount) > 5000 THEN 'Regular'
      ELSE 'Occasional'
   END AS CustomerSegment
FROM Customer c
LEFT JOIN Sale s ON c.CID = s.CID
GROUP BY c.CID, c.Cname;
```


CID	Cname	PurchaseFrequency	TotalSpend	CustomerSegment
1	John Doe	1	999.99	Occasional
2	Jane Smith	1	599.99	Occasional
3	Bob Johnson	1	149.99	Occasional
4	Alice Brown	1	79.99	Occasional
5	Charlie Davis	1	89.99	Occasional
6	Eva Wilson	1	49.99	Occasional
7	Frank Miller	1	299.99	Occasional
8	Grace Lee	1	129.99	Occasional
9	Henry Taylor	1	29.99	Occasional
10	Ivy Clark	1	89.99	Occasional
11	Jack Wright	1	179.99	Occasional
12	Karen Young	1	69.99	Occasional
13	Liam Hall	1	79.99	Occasional
14	Mia Scott	1	249.99	Occasional
15	Noah King	1	99.99	Occasional

	Time	Action	Response	Duration / Fetch Time
31	16:40:09	SELECT c.CID, c.Cname, COUNT(s.SID) AS PurchaseFrequency, SUM(s.Stotal_amount) AS TotalSpend,...	15 row(s) returned	0.010 sec / 0.000036...

Q9.

Query Description:

Product Reorder Alert - This query identifies products that need reordering based on their current stock quantity. It categorizes products into 'Urgent Reorder', 'Reorder Soon', and 'Stock Sufficient' based on their stock levels.

SELECT Clause:

- p.PID: Selects the Product ID
- p.Pname: Selects the Product Name
- p.Pstock_quantity: Selects the current stock quantity of the product
- CASE statement: Creates a new column ReorderStatus based on stock quantity

CASE Statement:

- Assigns 'Urgent Reorder' if stock is less than 10
- Assigns 'Reorder Soon' if stock is between 10 and 49
- Assigns 'Stock Sufficient' for all other cases (though this won't appear due to the WHERE clause)

FROM Clause:

- Uses the Product table, aliased as 'p'

WHERE Clause:

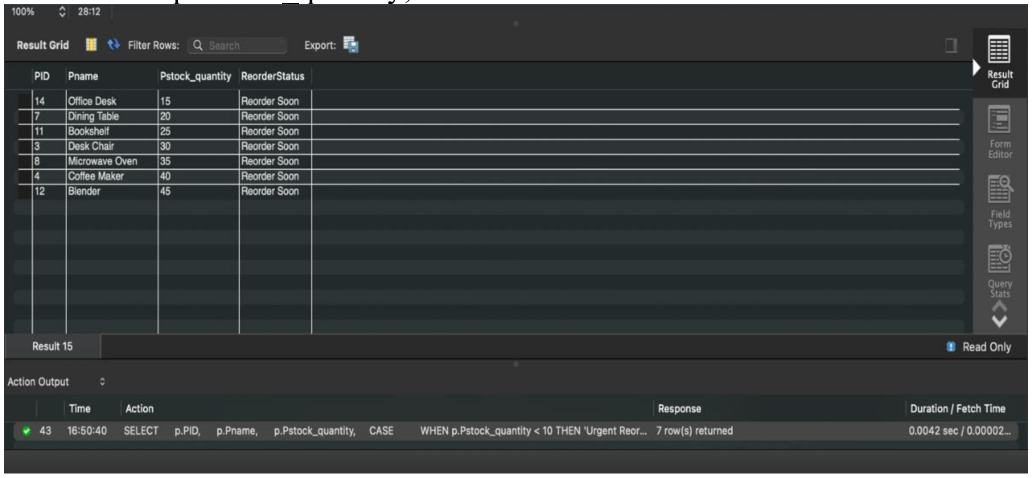
- Filters products to include only those with stock quantity less than 50

ORDER BY Clause:

- Sorts the results by p.Pstock_quantity in ascending order

SQL Statement:

```
SELECT
   p.PID,
   p.Pname,
   p.Pstock_quantity,
   CASE
      WHEN p.Pstock_quantity < 10 THEN 'Urgent Reorder'
      WHEN p.Pstock_quantity < 50 THEN 'Reorder Soon'
      ELSE 'Stock Sufficient'
   END AS ReorderStatus
FROM Product p
WHERE p.Pstock_quantity < 50
ORDER BY p.Pstock_quantity;
```



PID	Pname	Pstock_quantity	ReorderStatus
14	Office Desk	15	Reorder Soon
7	Dining Table	20	Reorder Soon
11	Bookshelf	25	Reorder Soon
3	Desk Chair	30	Reorder Soon
8	Microwave Oven	35	Reorder Soon
4	Coffee Maker	40	Reorder Soon
12	Blender	45	Reorder Soon

```
    Time      Action                                                                                     Response            Duration / Fetch Time
 43 16:50:40  SELECT  p.PID,  p.Pname,  p.Pstock_quantity,  CASE  WHEN p.Pstock_quantity < 10 THEN 'Urgent Reor...  7 row(s) returned  0.0042 sec / 0.00002...
```

Q10.

<u>Query Description:</u>

Order Fulfillment Time Trend

This query analyzes the trend in order fulfillment time over months.

SELECT Clause:

- DATE_FORMAT(order_date, '%Y-%m') AS Month: Converts the order date to a year-month format (e.g., '2024-10').
- AVG(DATEDIFF(delivery_date, order_date)) AS AvgFulfillmentDays: Calculates the average number of days between order and delivery dates for each month.
- LAG(AVG(DATEDIFF(delivery_date, order_date))) OVER (ORDER BY DATE_FORMAT(order_date, '%Y-%m')) AS PrevMonthAvg: Uses the LAG window function to retrieve the previous month's average fulfillment time.

FROM Clause:

- Specifies the Order table as the data source.

GROUP BY Clause:

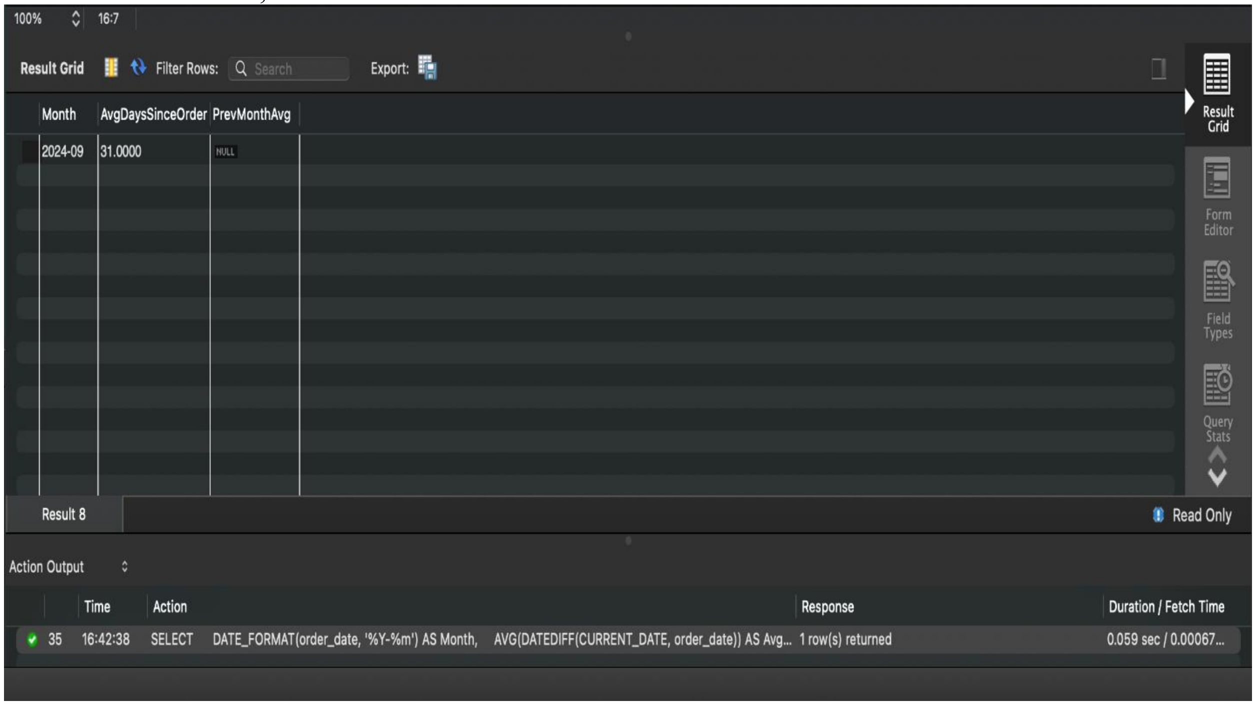
- Groups the results by the formatted month.

ORDER BY Clause:

- Sorts the results chronologically by month.

<u>SQL Statement:</u>

```
SELECT
   DATE_FORMAT(order_date, '%Y-%m') AS Month,
   AVG(DATEDIFF(delivery_date, order_date)) AS AvgFulfillmentDays,
   LAG(AVG(DATEDIFF(delivery_date, order_date))) OVER (ORDER BY
DATE_FORMAT(order_date, '%Y-%m')) AS PrevMonthAvg
FROM `Order`
GROUP BY Month
ORDER BY Month;
```



Month	AvgDaysSinceOrder	PrevMonthAvg
2024-09	31.0000	NULL

```
    Time      Action                                                                                      Response            Duration / Fetch Time
35  16:42:38  SELECT  DATE_FORMAT(order_date, '%Y-%m') AS Month,  AVG(DATEDIFF(CURRENT_DATE, order_date)) AS Avg...  1 row(s) returned   0.059 sec / 0.00067...
```

Q11.

<u>Query Description:</u>

Product Profit Margin Analysis

This query calculates the profit margin for each product.

SELECT Clause:

- p.PID: Selects the Product ID
- p.Pname: Selects the Product Name
- p.Punit_price AS SellingPrice: Retrieves the unit price from the Product table as the selling price
- AVG(od.ODunit_price) AS AvgCostPrice: Calculates the average cost price from Order_Details
- (p.Punit_price - AVG(od.ODunit_price)) / p.Punit_price * 100 AS ProfitMarginPercentage: Computes the profit margin percentage

FROM and JOIN Clauses:

- FROM Product p: Specifies the main table as Product
- JOIN Order_Details od ON p.PID = od.PID: Joins with Order_Details table using Product ID

GROUP BY Clause:

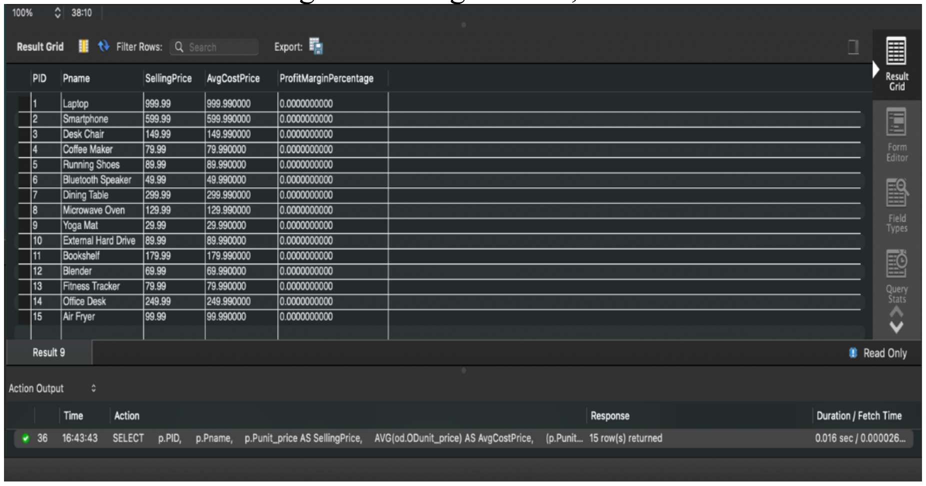
- Groups results by p.PID, p.Pname, and p.Punit_price to aggregate data for each unique product

ORDER BY Clause:

- Sorts the results by ProfitMarginPercentage in descending order, showing highest profit margins first

<u>SQL Statement:</u>

```
SELECT
   p.PID,
   p.Pname,
   p.Punit_price AS SellingPrice,
   AVG(od.ODunit_price) AS AvgCostPrice,
   (p.Punit_price - AVG(od.ODunit_price)) / p.Punit_price * 100 AS ProfitMarginPercentage
FROM Product p
JOIN Order_Details od ON p.PID = od.PID
GROUP BY p.PID, p.Pname, p.Punit_price
ORDER BY ProfitMarginPercentage DESC;
```



PID	Pname	SellingPrice	AvgCostPrice	ProfitMarginPercentage
1	Laptop	999.99	999.990000	0.0000000000
2	Smartphone	599.99	599.990000	0.0000000000
3	Desk Chair	149.99	149.990000	0.0000000000
4	Coffee Maker	79.99	79.990000	0.0000000000
5	Running Shoes	89.99	89.990000	0.0000000000
6	Bluetooth Speaker	49.99	49.990000	0.0000000000
7	Dining Table	299.99	299.990000	0.0000000000
8	Microwave Oven	129.99	129.990000	0.0000000000
9	Yoga Mat	29.99	29.990000	0.0000000000
10	External Hard Drive	89.99	89.990000	0.0000000000
11	Bookshelf	179.99	179.990000	0.0000000000
12	Blender	69.99	69.990000	0.0000000000
13	Fitness Tracker	79.99	79.990000	0.0000000000
14	Office Desk	249.99	249.990000	0.0000000000
15	Air Fryer	99.99	99.990000	0.0000000000

```
Result 9                                                        Read Only

Action Output
     Time      Action                                                                                   Response              Duration / Fetch Time
✓ 36 16:43:43  SELECT  p.PID,  p.Pname,  p.Punit_price AS SellingPrice,  AVG(od.ODunit_price) AS AvgCostPrice,  (p.Punit...  15 row(s) returned  0.016 sec / 0.000026...
```

Q12.
Query Description:
Customer Retention Analysis
This query analyzes customer retention by comparing purchases in consecutive years.
Common Table Expression (CTE)

```
WITH CustomerYearlyPurchases AS (
   SELECT
      CID,
      YEAR(sale_date) AS Year,
      SUM(Stotal_amount) AS YearlyPurchase
   FROM Sale
   GROUP BY CID, YEAR(sale_date)
)
```

This CTE calculates the total purchase amount for each customer per year. It:

- Extracts the year from the sale date
- Sums up the total amount spent by each customer in each year
- Groups the results by customer ID and year

Main SELECT Statement

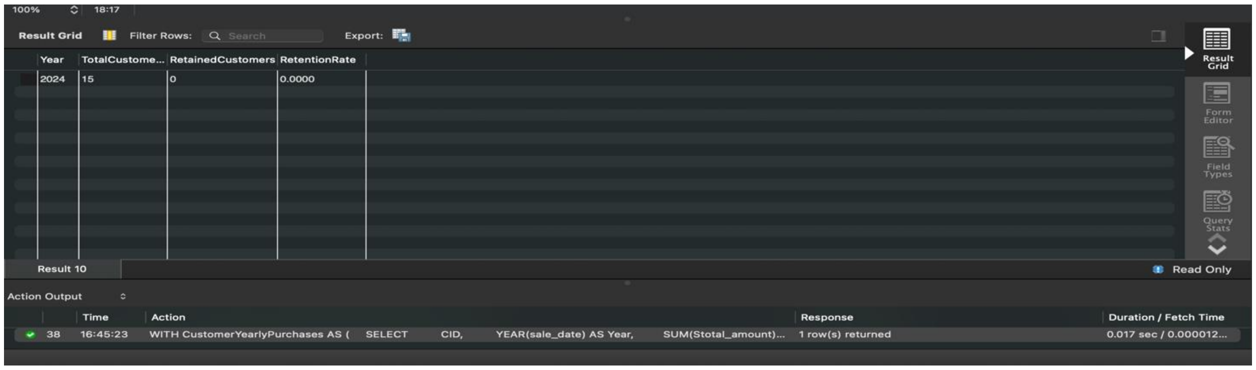
```
SELECT
   c1.Year,
   COUNT(DISTINCT c1.CID) AS TotalCustomers,
   COUNT(DISTINCT c2.CID) AS RetainedCustomers,
   COUNT(DISTINCT c2.CID) / COUNT(DISTINCT c1.CID) * 100 AS RetentionRate
FROM CustomerYearlyPurchases c1
LEFT JOIN CustomerYearlyPurchases c2 ON c1.CID = c2.CID AND c1.Year = c2.Year - 1
GROUP BY c1.Year
ORDER BY c1.Year;
```

This part of the query calculates the retention rate:

- It joins the CTE with itself (c1 and c2) to compare consecutive years
- Counts total customers for each year
- Counts retained customers (those who made purchases in consecutive years)
- Calculates the retention rate as a percentage

SQL Statement:

```
WITH CustomerYearlyPurchases AS (
   SELECT
      CID,
      YEAR(sale_date) AS Year,
      SUM(Stotal_amount) AS YearlyPurchase
   FROM Sale
   GROUP BY CID, YEAR(sale_date)
)
SELECT
   c1.Year,
   COUNT(DISTINCT c1.CID) AS TotalCustomers,
   COUNT(DISTINCT c2.CID) AS RetainedCustomers,
    COUNT(DISTINCT c2.CID) / COUNT(DISTINCT c1.CID) * 100 AS RetentionRate
FROM CustomerYearlyPurchases c1
LEFT JOIN CustomerYearlyPurchases c2 ON c1.CID = c2.CID AND c1.Year = c2.Year - 1
GROUP BY c1.Year
ORDER BY c1.Year;
```



```
Year  TotalCustome...  RetainedCustomers  RetentionRate
2024  15               0                  0.0000

Result 10

Action Output
    Time      Action                                                                                   Response            Duration / Fetch Time
38  16:45:23  WITH CustomerYearlyPurchases AS (  SELECT  CID,  YEAR(sale_date) AS Year,  SUM(Stotal_amount)...  1 row(s) returned  0.017 sec / 0.000012...
```

Q13.

Query Description:

Supplier Order Value Distribution

This query uses window functions to analyze the distribution of order values for each supplier.

SELECT Clause:

- s.SUPID: Selects the Supplier ID
- s.Sname: Selects the Supplier Name
- o.Ototal_amount: Selects the total amount of each order
- PERCENT_RANK(): Calculates the percent rank of each order amount within a supplier's orders
- NTILE(4): Divides the orders into quartiles for each supplier

FROM and JOIN Clauses:

- FROM Supplier s: Specifies the main table as Supplier
- JOIN Order o ON s.SUPID = o.SUPID: Joins with the Order table using Supplier ID

Window Functions

1. **PERCENT_RANK():**

PERCENT_RANK() **OVER** (**PARTITION BY** s.SUPID **ORDER BY** o.Ototal_amount) **AS** PercentRank

- Calculates the relative rank of each order amount within a supplier's set of orders
- Values range from 0 to 1, indicating the percentage of values below the current value

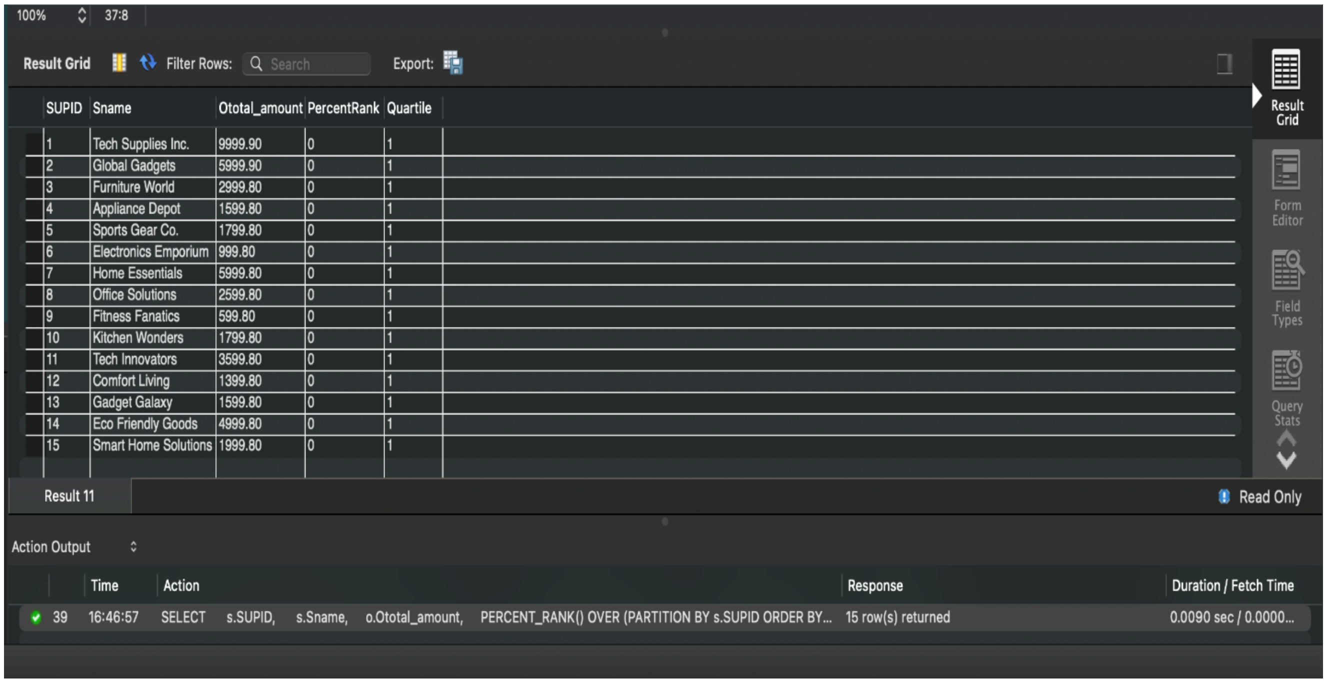
2. **NTILE(4):**

NTILE(4) **OVER** (**PARTITION BY** s.SUPID **ORDER BY** o.Ototal_amount) **AS** Quartile

- Divides the orders for each supplier into 4 equal groups (quartiles)
- Assigns a value from 1 to 4 to each order, representing which quartile it falls into

SQL Statement:

```
SELECT
   s.SUPID,
   s.Sname,
   o.Ototal_amount,
   PERCENT_RANK() OVER (PARTITION BY s.SUPID ORDER BY o.Ototal_amount) AS
PercentRank,
   NTILE(4) OVER (PARTITION BY s.SUPID ORDER BY o.Ototal_amount) AS Quartile
FROM Supplier s
JOIN `Order` o ON s.SUPID = o.SUPID;
```



SUPID	Sname	Ototal_amount	PercentRank	Quartile
1	Tech Supplies Inc.	9999.90	0	1
2	Global Gadgets	5999.90	0	1
3	Furniture World	2999.80	0	1
4	Appliance Depot	1599.80	0	1
5	Sports Gear Co.	1799.80	0	1
6	Electronics Emporium	999.80	0	1
7	Home Essentials	5999.80	0	1
8	Office Solutions	2599.80	0	1
9	Fitness Fanatics	599.80	0	1
10	Kitchen Wonders	1799.80	0	1
11	Tech Innovators	3599.80	0	1
12	Comfort Living	1399.80	0	1
13	Gadget Galaxy	1599.80	0	1
14	Eco Friendly Goods	4999.80	0	1
15	Smart Home Solutions	1999.80	0	1

```
39  16:46:57  SELECT   s.SUPID,   s.Sname,   o.Ototal_amount,   PERCENT_RANK() OVER (PARTITION BY s.SUPID ORDER BY...  15 row(s) returned  0.0090 sec / 0.0000...
```

Q14.

Query Description:

Monthly Sales Trend

This query shows the sales trend over the last 12 months. It provides the number of sales, total sales amount, and average sale amount for each month.

SELECT Clause:

- DATE_FORMAT(s.sale_date, '%Y-%m') AS Month: Formats the sale date into a year-month string
- COUNT(DISTINCT s.SID) AS NumberOfSales: Counts unique sale IDs per month
- SUM(s.Stotal_amount) AS TotalSalesAmount: Calculates total sales amount per month
- AVG(s.Stotal_amount) AS AverageSaleAmount: Computes average sale amount per month

FROM Clause:

- Uses the Sale table, aliased as 's'

GROUP BY Clause:

- Groups results by the formatted month

ORDER BY Clause:

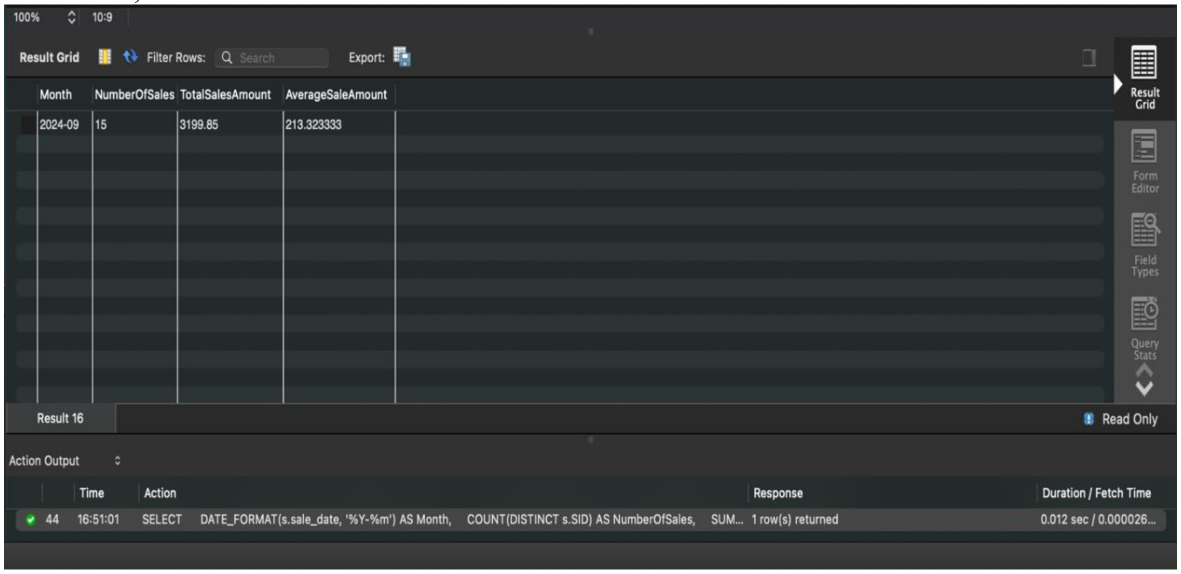
- Sorts results by month in descending order (most recent first)

LIMIT Clause:

- Restricts the output to the last 12 months

SQL Statement:

```
SELECT
   DATE_FORMAT(s.sale_date, '%Y-%m') AS Month,
   COUNT(DISTINCT s.SID) AS NumberOfSales,
   SUM(s.Stotal_amount) AS TotalSalesAmount,
   AVG(s.Stotal_amount) AS AverageSaleAmount
FROM Sale s
GROUP BY Month
ORDER BY Month DESC
LIMIT 12;
```



Month	NumberOfSales	TotalSalesAmount	AverageSaleAmount
2024-09	15	3199.85	213.323333

	Time	Action	Response	Duration / Fetch Time
44	16:51:01	SELECT DATE_FORMAT(s.sale_date, '%Y-%m') AS Month, COUNT(DISTINCT s.SID) AS NumberOfSales, SUM...	1 row(s) returned	0.012 sec / 0.000026...

Q15.
Query Description:
Seasonal Sales Pattern
This query analyzes seasonal sales patterns using OLAP features.
SELECT Clause:

- YEAR(sale_date) AS Year: Extracts the year from the sale date
- QUARTER(sale_date) AS Quarter: Extracts the quarter from the sale date
- SUM(Stotal_amount) AS TotalSales: Calculates total sales for each group
- AVG(SUM(Stotal_amount)) OVER (PARTITION BY QUARTER(sale_date)) AS AvgQuarterlySales: Computes the average quarterly sales across years

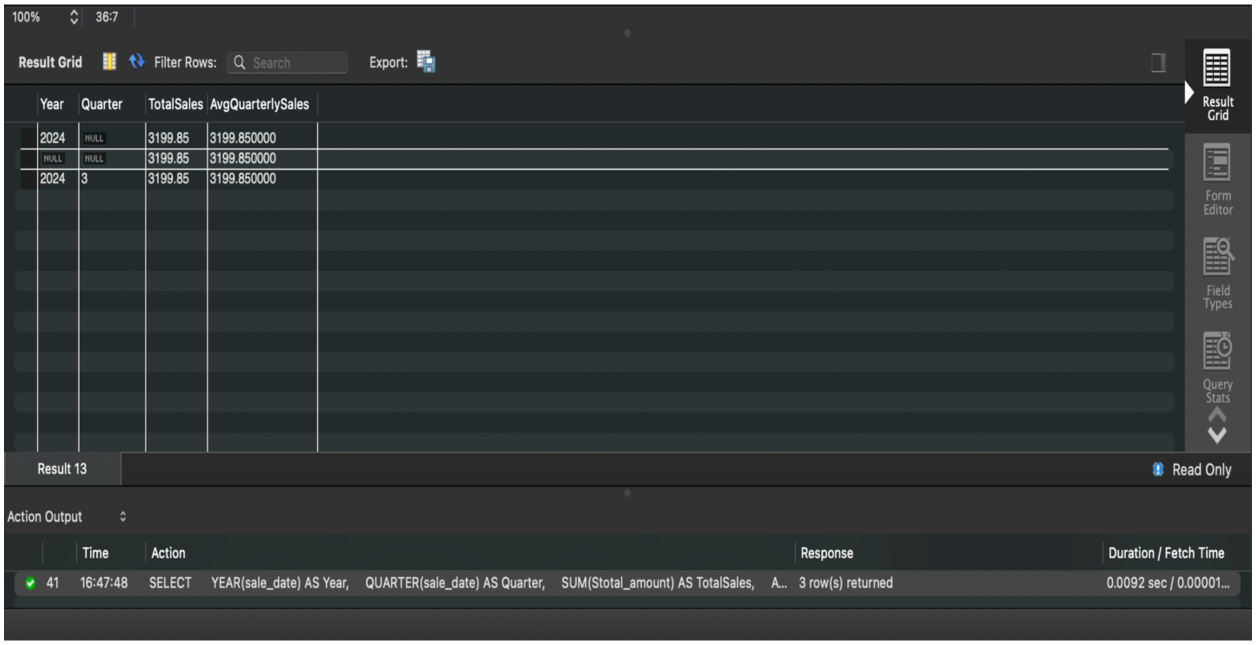
FROM Clause:

- Uses the Sale table

GROUP BY Clause:

- Groups results by Year and Quarter
- WITH ROLLUP: Generates subtotals and grand totals

SQL Statement:
```
SELECT
   YEAR(sale_date) AS Year,
   QUARTER(sale_date) AS Quarter,
   SUM(Stotal_amount) AS TotalSales,
   AVG(SUM(Stotal_amount)) OVER (PARTITION BY QUARTER(sale_date)) AS
AvgQuarterlySales
FROM Sale
GROUP BY Year, Quarter WITH ROLLUP;
```



Year	Quarter	TotalSales	AvgQuarterlySales
2024	NULL	3199.85	3199.850000
NULL	NULL	3199.85	3199.850000
2024	3	3199.85	3199.850000

	Time	Action	Response	Duration / Fetch Time
41	16:47:48	SELECT YEAR(sale_date) AS Year, QUARTER(sale_date) AS Quarter, SUM(Stotal_amount) AS TotalSales, A...	3 row(s) returned	0.0092 sec / 0.00001...