● Calculate the total sales revenue from all orders. [Gross Sales Revenue]

❖ Description:

- This query calculates the gross sales revenue by summing all payment amounts, and counts the total number of orders. It includes all completed sales, regardless of whether any returns occurred later.

❖ Objective:

- Track overall business performance and cash inflow.
- Feed financial dashboards and executive summaries.
- Compare against net revenue (after returns/refunds) in financial reports.

❖ Output Columns:

Column Name	Description
`Gross_Sales_Revenue`	Total revenue generated from all payments
`Orders_number`	Total number of distinct orders recorded in `payments`

❖ Explanation of Components:

- SUM Function

```
SUM(amount) AS Gross_Sales_Revenue
```

Calculates the total amount paid across all orders.

- COUNT Function

```
COUNT(order_id) AS Orders_number
```

Counts the total number of payments/orders processed.

- FROM Clause

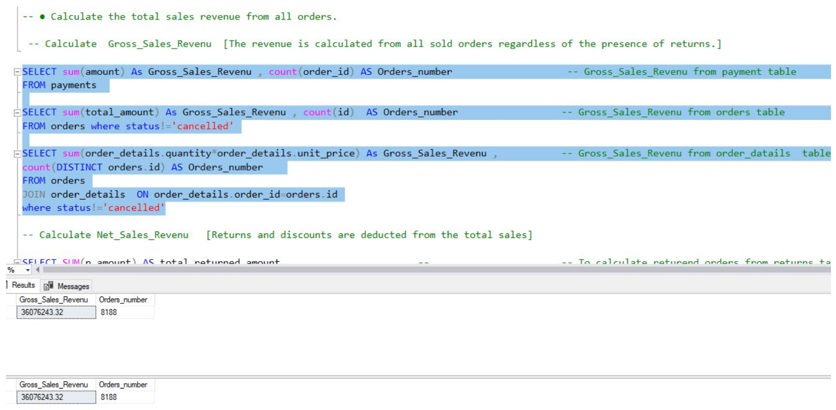
```
FROM payments
```

Uses the `payments` table, which tracks revenue from completed transactions

Assumptions:

- The `payments` table includes only confirmed/processed payments.
- Returns or refunds, if tracked, are **not deducted** in this calculation.

SQL Query:



```
-- • Calculate the total sales revenue from all orders.

 -- Calculate  Gross_Sales_Revenu  [The revenue is calculated from all sold orders regardless of the presence of returns.]

SELECT sum(amount) As Gross_Sales_Revenu , count(order_id) AS Orders_number                    -- Gross_Sales_Revenu from payment table
FROM payments

SELECT sum(total_amount) As Gross_Sales_Revenu , count(id)  AS Orders_number                   -- Gross_Sales_Revenu from orders table
FROM orders where status!='cancelled'

SELECT sum(order_details.quantity*order_details.unit_price) As Gross_Sales_Revenu ,            -- Gross_Sales_Revenu from order_datails  table
count(DISTINCT orders.id) AS Orders_number
FROM orders
JOIN order_details  ON order_details.order_id=orders.id
where status!='cancelled'

-- Calculate Net_Sales_Revenu   [Returns and discounts are deducted from the total sales]

SELECT SUM(r.amount) AS total_returned_amount          --                  -- To calculate returend orders from returns ta
```

```
Results   Messages
Gross_Sales_Revenu   Orders_number
36076243.32          8188
```

```
Gross_Sales_Revenu   Orders_number
36076243.32          8188
```

● Calculate the total sales revenue from all orders. [Net Sales Revenue]

Description:

- This query calculates the **Net Sales Revenue** by subtracting the value of **completed returns** from the **total gross payments** received. It provides a more accurate measure of actual sales revenue after accounting for returned orders.

Objective:

- Financial reporting and revenue analysis.
- Comparing gross vs. net revenue for profitability insights.
- Supporting finance, accounting, and sales performance evaluations.

Output:

Column Name	Description
`Net_Sales_Revenue`	Total sales revenue after subtracting completed returns

Explanation of Components:

- **Total Gross Revenue**

```
(SELECT SUM(amount) FROM payments)
```

Calculates the total revenue from all processed payments.

- **Total Value of Returns**

```
(SELECT SUM(p.amount)
 FROM payments p
 INNER JOIN returns r ON p.order_id = r.order_id
 WHERE r.status = 'completed')
```

Joins payments with returns to get the amount of orders that were returned.
Filters only for **returns with status = 'completed'** to ensure accuracy.

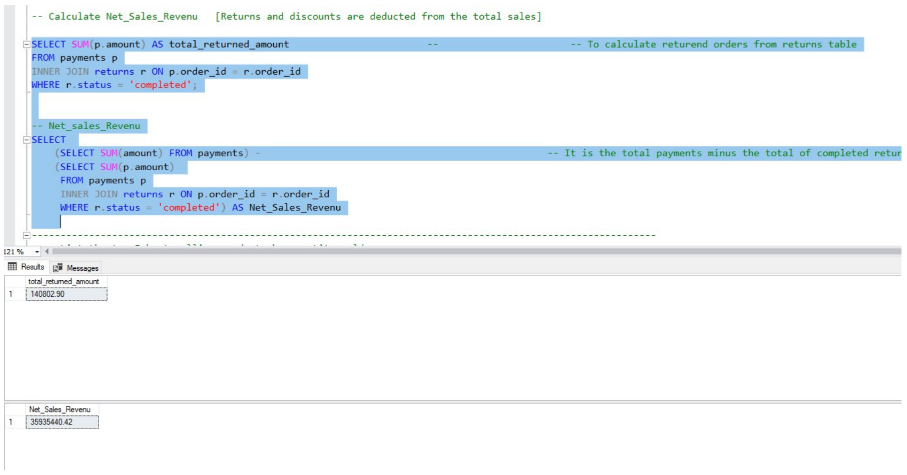
- **Final Calculation**

The outer query subtracts the total value of returns from total payments to get **Net Sales Revenue**.

Assumptions:

- The `payments` table represents completed and confirmed transactions.
- The `returns` table accurately logs all returned orders with their statuses.
- Only returns marked as `'completed'` should reduce net revenue.

SQL Query:



```
-- Calculate Net_Sales_Revenu   [Returns and discounts are deducted from the total sales]

SELECT SUM(p.amount) AS total_returned_amount          --                    -- To calculate returend orders from returns table
FROM payments p
INNER JOIN returns r ON p.order_id = r.order_id
WHERE r.status = 'completed';


-- Net_sales_Revenu
SELECT
    (SELECT SUM(amount) FROM payments) -                          -- It is the total payments minus the total of completed retur
    (SELECT SUM(p.amount)
     FROM payments p
     INNER JOIN returns r ON p.order_id = r.order_id
     WHERE r.status = 'completed') AS Net_Sales_Revenu

-----------------------------------------------------------------------------------------------
```

121 %

Results | Messages

```
   total_returned_amount
1  140802.90
```

```
   Net_Sales_Revenu
1  35935440.42
```

● List the top 5 best-selling products by quantity sold.

❖ Description:

- This query retrieves the top 5 best-selling products based on the total quantity sold. It also calculates the total revenue for each product by multiplying the quantity sold by the unit price. Only completed and non-cancelled orders are included in the results.

❖ Objective:

- Identify the top-performing products by sales volume.
- Analyze product performance in terms of quantity and revenue.
- Provide insight for marketing, inventory, and sales decisions.

❖ Output Columns:

Column Name	Description
`ProductName`	The name of the product
`TotalQuantitySold`	Total units sold for the product
`TotalRevenue`	Total revenue = quantity sold × unit price

❖ Tables Involved:

Table Name	Description
`products`	Contains product information
`order_details`	Contains details of each product within an order (quantity, price)
`orders`	Contains order-level data from customers
`payments`	Tracks payment status for each order

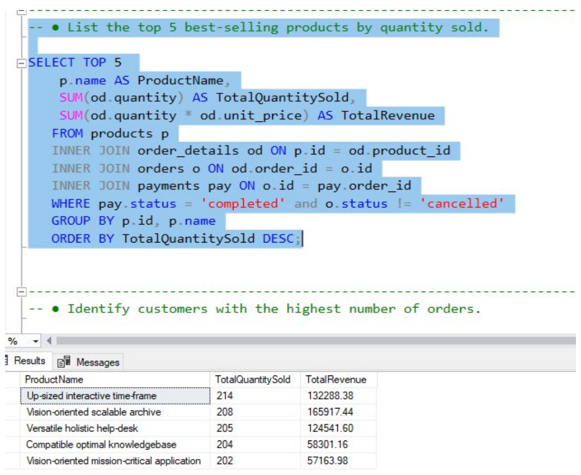
❖ Filtering Conditions:

- pay.status = 'completed': Ensures only completed payments are considered.
- o.status != 'cancelled': Excludes cancelled orders from the result.

◆ Sorting & Limiting:

- ORDER BY TotalQuantitySold DESC: Sorts the products from most to least sold.
- TOP 5: Limits the result to the top 5 products only.

◆ SQL Query:



```
-- • List the top 5 best-selling products by quantity sold.

SELECT TOP 5
    p.name AS ProductName,
    SUM(od.quantity) AS TotalQuantitySold,
    SUM(od.quantity * od.unit_price) AS TotalRevenue
  FROM products p
  INNER JOIN order_details od ON p.id = od.product_id
  INNER JOIN orders o ON od.order_id = o.id
  INNER JOIN payments pay ON o.id = pay.order_id
  WHERE pay.status = 'completed' and o.status != 'cancelled'
  GROUP BY p.id, p.name
  ORDER BY TotalQuantitySold DESC;


-- • Identify customers with the highest number of orders.
```

```
ProductName                               TotalQuantitySold  TotalRevenue
Up-sized interactive time-frame           214                132288.38
Vision-oriented scalable archive          208                165917.44
Versatile holistic help-desk              205                124541.60
Compatible optimal knowledgebase          204                58301.16
Vision-oriented mission-critical application  202            57163.98
```

● Identify customers with the highest number of orders.

◆ Description:

🎬 This query retrieves a list of customers sorted by the number of orders they have placed, in descending order. It helps to identify the most active or loyal customers based on order count.

◆ Objective:

- Identify the top customers by the number of orders.
- Analyze customer engagement and order frequency.

Column Name	**Description**
`customer_id`	Unique identifier of the customer
`customer_name`	Full name of the customer (first + last)

`customer_total_orders`	Total number of orders placed by the customer

Output Columns:

Tables Involved:

Table Name	Description
`orders`	Contains information about customer orders
`customers`	Contains customer personal details

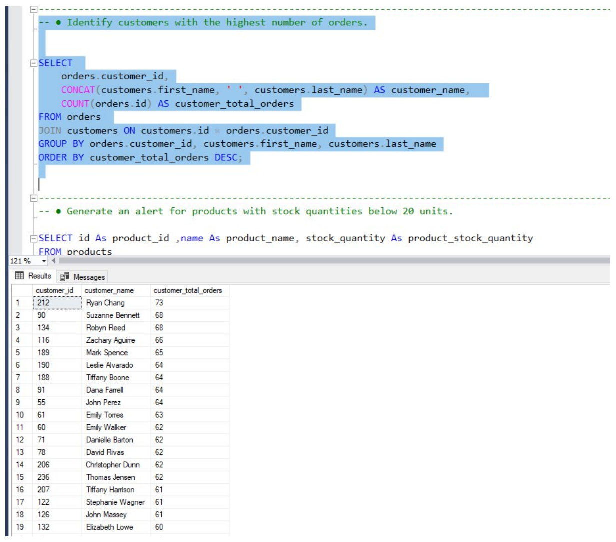
Grouping and Aggregation:

- GROUP BY orders.customer_id, customers.first_name, customers.last_name: Groups orders by customer to calculate totals.
- COUNT(orders.id): Counts the number of orders per customer.

Sorting:

- ORDER BY customer_total_orders DESC: Sorts customers from most to fewest orders.

SQL Query:



```sql
-- • Identify customers with the highest number of orders.


SELECT
    orders.customer_id,
    CONCAT(customers.first_name, ' ', customers.last_name) AS customer_name,
    COUNT(orders.id) AS customer_total_orders
FROM orders
JOIN customers ON customers.id = orders.customer_id
GROUP BY orders.customer_id, customers.first_name, customers.last_name
ORDER BY customer_total_orders DESC;


-- • Generate an alert for products with stock quantities below 20 units.

SELECT id As product_id ,name As product_name, stock_quantity As product_stock_quantity
FROM products
```

```
    customer_id  customer_name      customer_total_orders
1   212          Ryan Chang         73
2   90           Suzanne Bennett    68
3   134          Robyn Reed         68
4   116          Zachary Aguirre    66
5   189          Mark Spence        65
6   190          Leslie Alvarado    64
7   188          Tiffany Boone      64
8   91           Dana Farrell       64
9   55           John Perez         64
10  61           Emily Torres       63
11  60           Emily Walker       62
12  71           Danielle Barton    62
13  78           David Rivas        62
14  206          Christopher Dunn   62
15  236          Thomas Jensen      62
16  207          Tiffany Harrison   61
17  122          Stephanie Wagner   61
18  126          John Massey        61
19  132          Elizabeth Lowe     60
```

● Generate an alert for products with stock quantities below 20 units.

❖ Description:

🎬 This query retrieves all products that have less than 20 units remaining in stock. It helps in identifying items that are running low and may require restocking.

❖ Objective:

- Generate alerts for low-stock items.
- Prevent stock outs by enabling timely replenishment and

❖ Output Columns:

Column Name	Description
product_id	Unique ID of the product
product_name	Name of the product
product_stock_quantity	Current available quantity in stock

- **Table Involved:**

Table Name	Description
products	Contains product details and stock levels

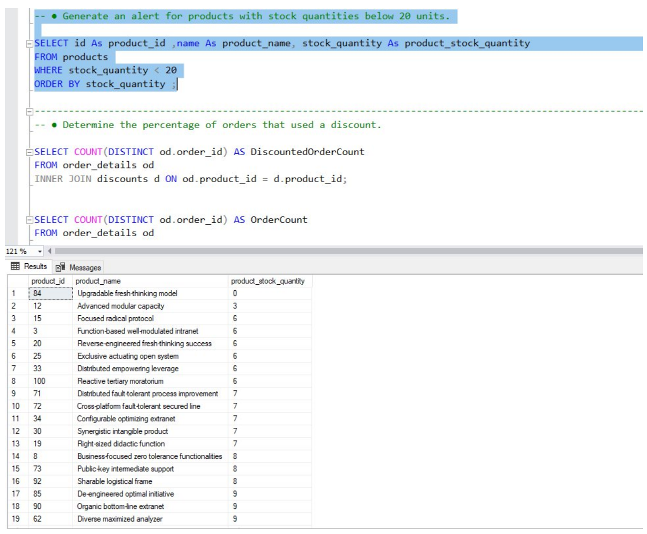
- **Filter Condition:**

  - stock_quantity < 20: Returns only products with less than 20 units in stock.

- **Sorting:**

  - ORDER BY stock_quantity ASC: Displays products starting from the lowest stock quantity.

- **SQL Query:**



```
-- • Generate an alert for products with stock quantities below 20 units.

SELECT id As product_id ,name As product_name, stock_quantity As product_stock_quantity
FROM products
WHERE stock_quantity < 20
ORDER BY stock_quantity ;

--------------------------------------------------------------------------------
-- • Determine the percentage of orders that used a discount.

SELECT COUNT(DISTINCT od.order_id) AS DiscountedOrderCount
FROM order_details od
INNER JOIN discounts d ON od.product_id = d.product_id;


SELECT COUNT(DISTINCT od.order_id) AS OrderCount
FROM order_details od
```

	product_id	product_name	product_stock_quantity
1	84	Upgradable fresh-thinking model	0
2	12	Advanced modular capacity	3
3	15	Focused radical protocol	6
4	3	Function-based well-modulated intranet	6
5	20	Reverse-engineered fresh-thinking success	6
6	25	Exclusive actuating open system	6
7	33	Distributed empowering leverage	6
8	100	Reactive tertiary moratorium	6
9	71	Distributed fault-tolerant process improvement	7
10	72	Cross-platform fault-tolerant secured line	7
11	34	Configurable optimizing extranet	7
12	30	Synergistic intangible product	7
13	19	Right-sized didactic function	7
14	8	Business-focused zero tolerance functionalities	8
15	73	Public-key intermediate support	8
16	92	Sharable logistical frame	8
17	85	De-engineered optimal initiative	9
18	90	Organic bottom-line extranet	9
19	62	Diverse maximized analyzer	9

• Determine the percentage of orders that used a discount.

- **Description**:

- This query calculates the percentage of all customer orders that included at least one product with a discount applied. It is used to measure how often discounts are utilized across the total order base.

Objective:

- Track customer adoption of discounts and promotions.
- Evaluate discount strategy effectiveness.
- Support marketing and revenue analytics teams.

Output:

Column Name	Description
`DiscountUsagePercentage`	The percentage of total orders that included at least one discounted item

Explanation of Components:

- Sub query: discounted orders

```
SELECT COUNT(DISTINCT od.order_id) AS DiscountedOrderCount
FROM order_details od
INNER JOIN discounts d ON od.product_id = d.product_id
```

- Joins order_details with the discounts table.
- Counts how many unique orders contained discounted products.
- Sub query: total orders

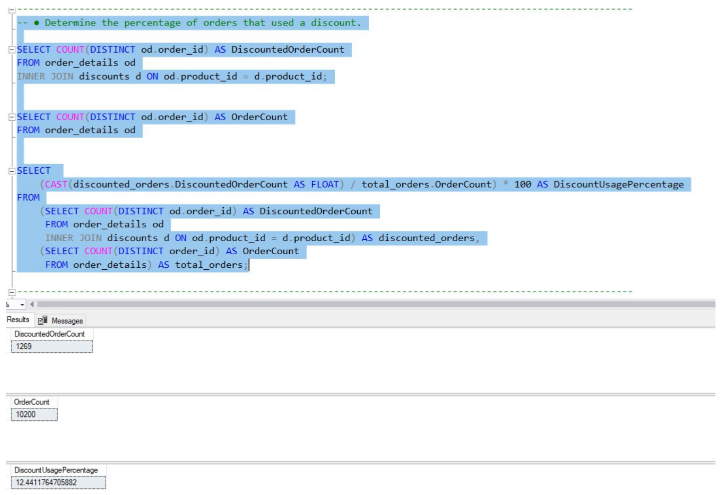
```
SELECT COUNT(DISTINCT order_id) AS OrderCount
FROM order_details
```

- Counts the total number of unique orders in the system.
- Main Calculation

```
(CAST(DiscountedOrderCount AS FLOAT) / OrderCount) * 100
```

- converts the numerator to `FLOAT` to avoid integer division.
- Multiplies by 100 to express the result as a percentage.

SQL Query:



```sql
-- • Determine the percentage of orders that used a discount.

SELECT COUNT(DISTINCT od.order_id) AS DiscountedOrderCount
FROM order_details od
INNER JOIN discounts d ON od.product_id = d.product_id;


SELECT COUNT(DISTINCT od.order_id) AS OrderCount
FROM order_details od


SELECT
    (CAST(discounted_orders.DiscountedOrderCount AS FLOAT) / total_orders.OrderCount) * 100 AS DiscountUsagePercentage
FROM
    (SELECT COUNT(DISTINCT od.order_id) AS DiscountedOrderCount
     FROM order_details od
     INNER JOIN discounts d ON od.product_id = d.product_id) AS discounted_orders,
    (SELECT COUNT(DISTINCT order_id) AS OrderCount
     FROM order_details) AS total_orders;
```

```
DiscountedOrderCount
1269

OrderCount
10200

DiscountUsagePercentage
12.4411764705882
```

• Calculate the average rating for each product.

❖ **Description**:

- This query calculates the average customer rating for each product based on review data. It helps identify product performance and customer satisfaction levels.

❖ **Objective:**

- Analyze customer satisfaction across the product catalog.
- Identify high- and low-performing products.
- Power product listing pages, dashboards, or recommendation engines.

Column Name	**Description**
`product_id`	Unique identifier of the product
`product_name`	Name of the product

`average_rating`	The average of all ratings given to the product

◆ Output:

◆ Explanation of Components:

- JOIN Clause

  ```
  JOIN products p ON r.product_id = p.id
  ```

  Connects reviews to the corresponding products using product_id.

- AVG Function

  ```
  AVG(CAST(r.rating AS FLOAT)) AS average_rating
  ```

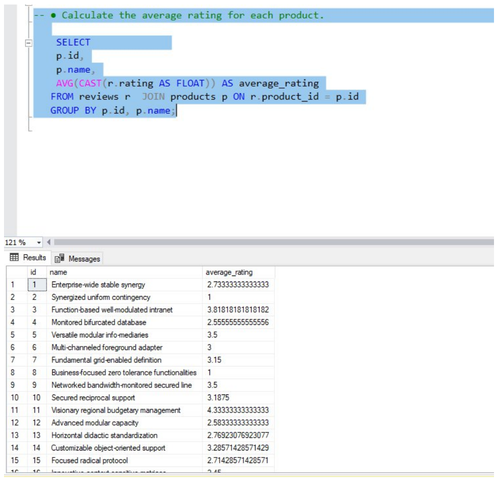
  Calculates the average rating.
  Casts the rating to FLOAT for accurate decimal results (in case it's stored as an integer).

- GROUP BY Clause

  ```
  GROUP BY p.id, p.name
  ```

  Groups reviews by product to compute the average per product.

◆ SQL Query:



```
-- • Calculate the average rating for each product.

    SELECT
    p.id,
    p.name,
    AVG(CAST(r.rating AS FLOAT)) AS average_rating
   FROM reviews r  JOIN products p ON r.product_id = p.id
   GROUP BY p.id, p.name;
```

	id	name	average_rating
1	1	Enterprise-wide stable synergy	2.73333333333333
2	2	Synergized uniform contingency	1
3	3	Function-based well-modulated intranet	3.81818181818182
4	4	Monitored bifurcated database	2.55555555555556
5	5	Versatile modular info-mediaries	3.5
6	6	Multi-channeled foreground adapter	3
7	7	Fundamental grid-enabled definition	3.15
8	8	Business-focused zero tolerance functionalities	1
9	9	Networked bandwidth-monitored secured line	3.5
10	10	Secured reciprocal support	3.1875
11	11	Visionary regional budgetary management	4.33333333333333
12	12	Advanced modular capacity	2.58333333333333
13	13	Horizontal didactic standardization	2.76923076923077
14	14	Customizable object-oriented support	3.28571428571429
15	15	Focused radical protocol	2.71428571428571