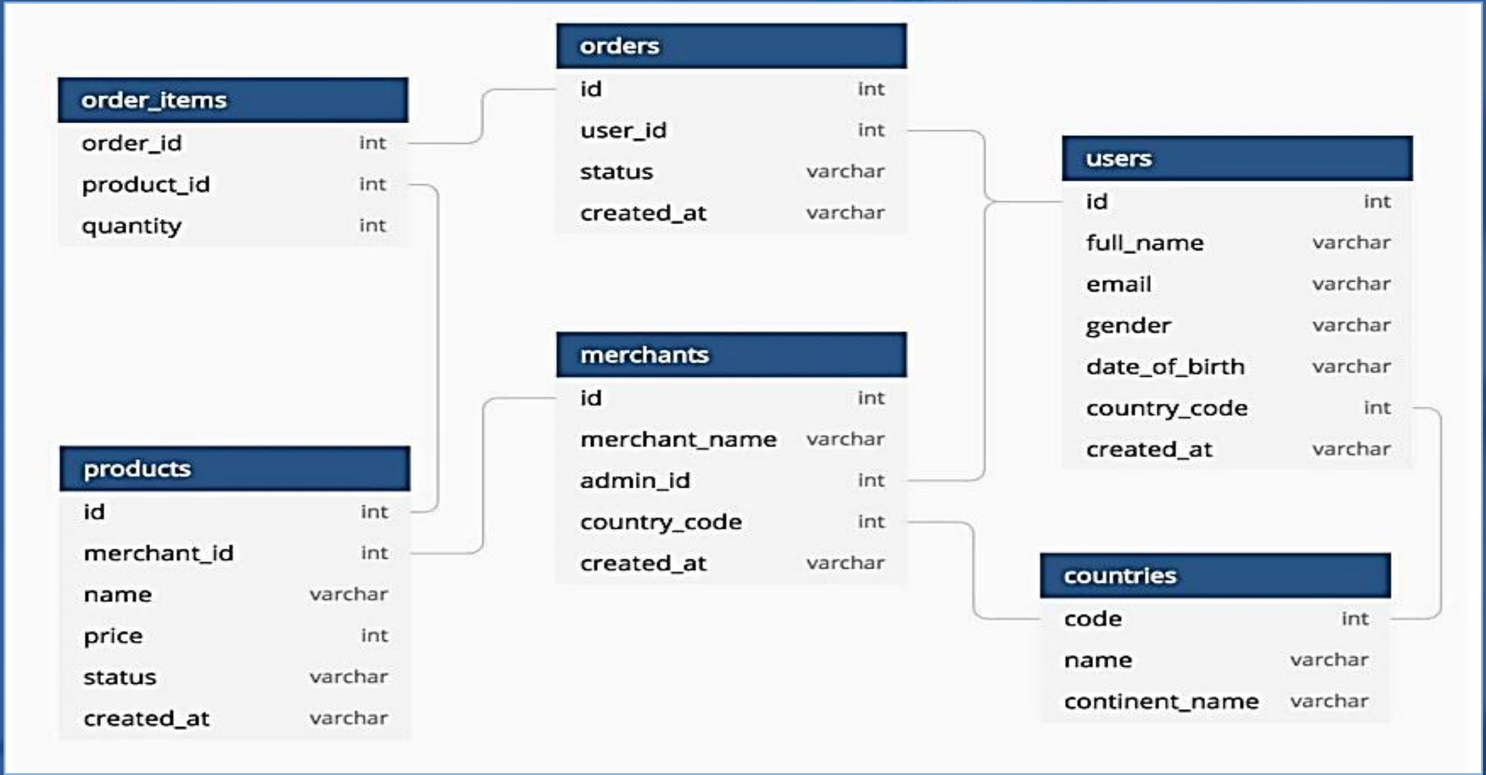
Final Portfolio Project on SQL Part

Topic : Comprehensive E-commerce Data Management and Analysis

Presentor: Sumiya Sadiya

Course Name : Data & Business Analytics Live

Project Schema: eCommerce_DB



orders	
id	int
user_id	int
status	varchar
created_at	varchar

order_items	
order_id	int
product_id	int
quantity	int

users	
id	int
full_name	varchar
email	varchar
gender	varchar
date_of_birth	varchar
country_code	int
created_at	varchar

merchants	
id	int
merchant_name	varchar
admin_id	int
country_code	int
created_at	varchar

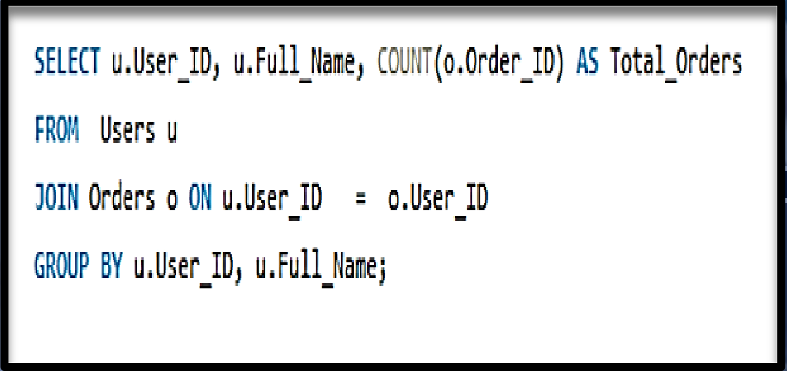
products	
id	int
merchant_id	int
name	varchar
price	int
status	varchar
created_at	varchar

countries	
code	int
name	varchar
continent_name	varchar

Basic Queries

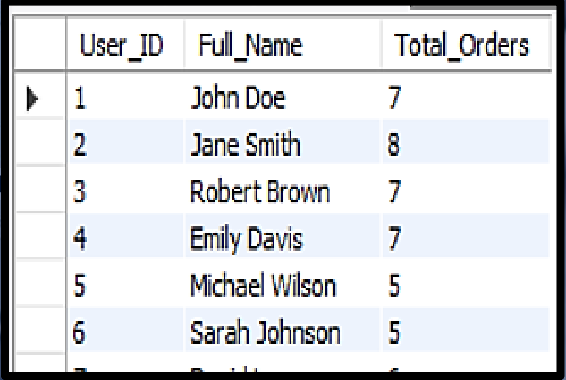
1. Write a query to display the total number of orders made by each user.

MySQL Code



```
SELECT u.User_ID, u.Full_Name, COUNT(o.Order_ID) AS Total_Orders
FROM  Users u
JOIN Orders o ON u.User_ID  =  o.User_ID
GROUP BY u.User_ID, u.Full_Name;
```

Output

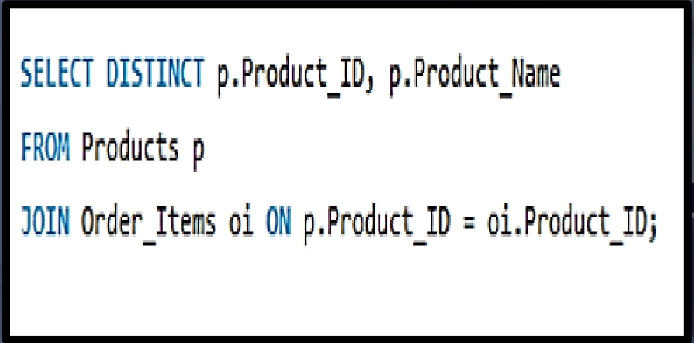


```
User_ID   Full_Name        Total_Orders
1         John Doe         7
2         Jane Smith       8
3         Robert Brown     7
4         Emily Davis      7
5         Michael Wilson   5
6         Sarah Johnson    5
```

- **Users who are highly engaged with the business could be targeted for loyalty programs or retention strategies.**
- **Regular customers could be encouraged through targeted marketing campaigns or discounts to increase their order frequency**

2. Write a query to display the names of products that have been ordered at least once.

MySQL Code

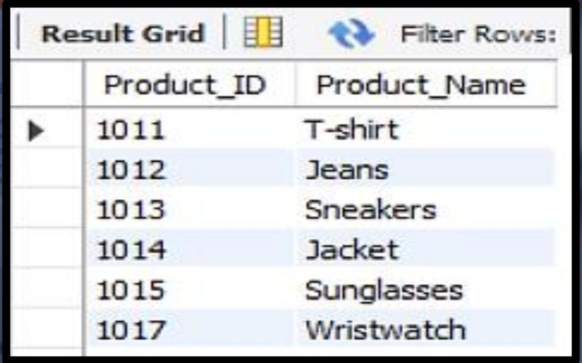


```
SELECT DISTINCT p.Product_ID, p.Product_Name

FROM Products p

JOIN Order_Items oi ON p.Product_ID = oi.Product_ID;
```

Output

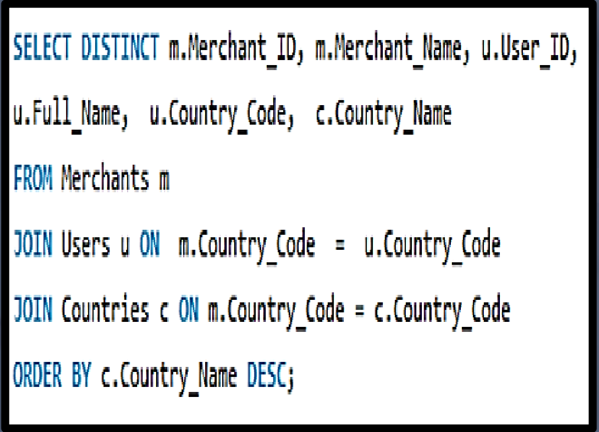


```
Product_ID   Product_Name
1011         T-shirt
1012         Jeans
1013         Sneakers
1014         Jacket
1015         Sunglasses
1017         Wristwatch
```

- **Business should monitor their stock levels closely to avoid upcoming shortages.**
- **In future, implementing bundling strategies can encourage customers to purchase related products together to increase the overall order value.**

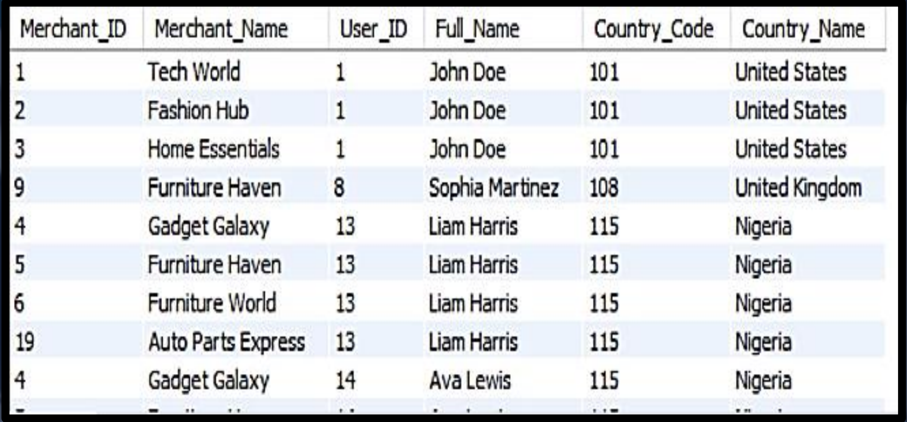
3 . Retrieve the details of all users who are from the same country as the merchant

MySQL Code



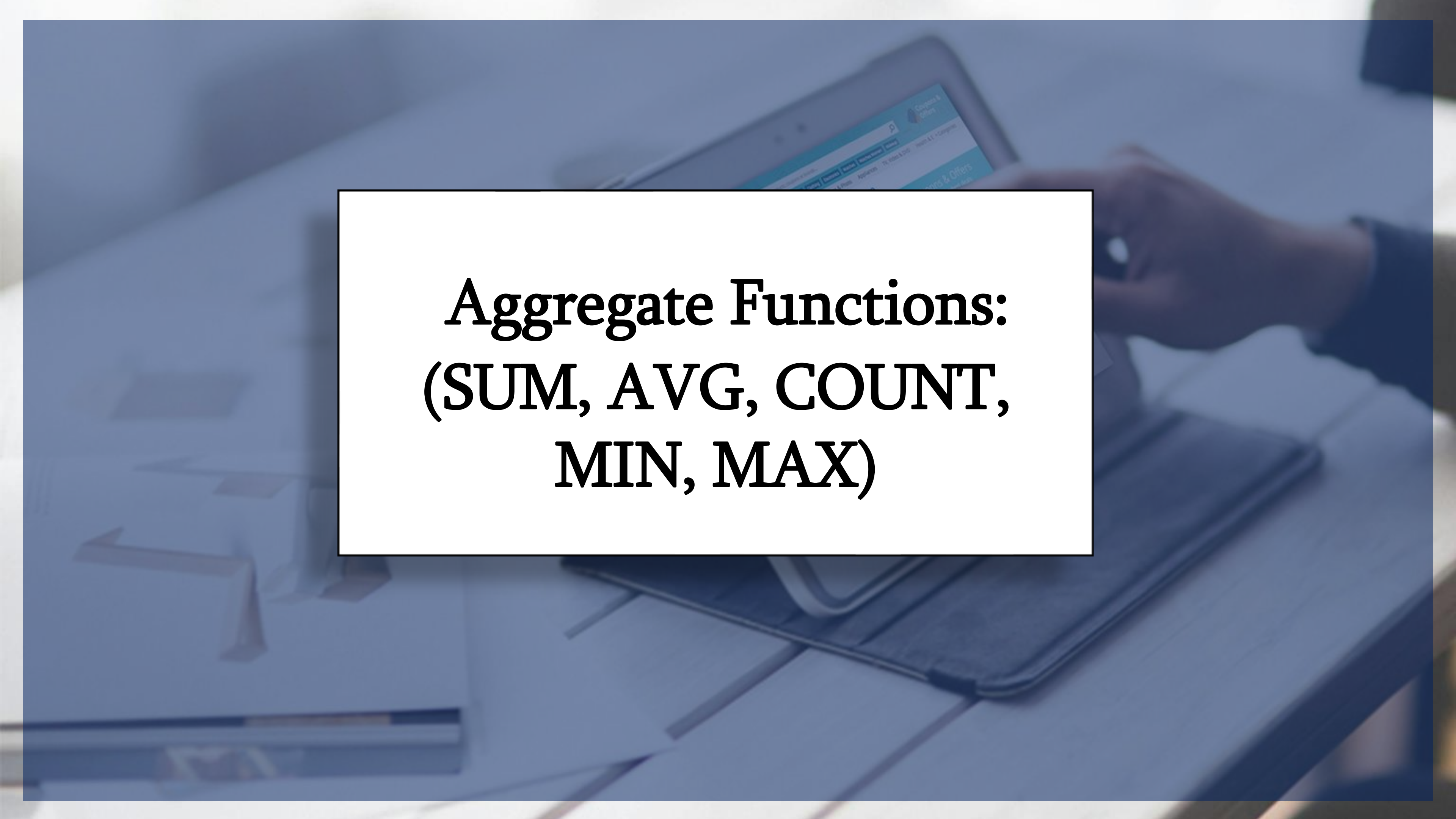
```
SELECT DISTINCT m.Merchant_ID, m.Merchant_Name, u.User_ID,
u.Full_Name,  u.Country_Code,  c.Country_Name
FROM Merchants m
JOIN Users u ON  m.Country_Code  =  u.Country_Code
JOIN Countries c ON m.Country_Code = c.Country_Code
ORDER BY c.Country_Name DESC;
```

Output



Merchant_ID	Merchant_Name	User_ID	Full_Name	Country_Code	Country_Name
1	Tech World	1	John Doe	101	United States
2	Fashion Hub	1	John Doe	101	United States
3	Home Essentials	1	John Doe	101	United States
9	Furniture Haven	8	Sophia Martinez	108	United Kingdom
4	Gadget Galaxy	13	Liam Harris	115	Nigeria
5	Furniture Haven	13	Liam Harris	115	Nigeria
6	Furniture World	13	Liam Harris	115	Nigeria
19	Auto Parts Express	13	Liam Harris	115	Nigeria
4	Gadget Galaxy	14	Ava Lewis	115	Nigeria

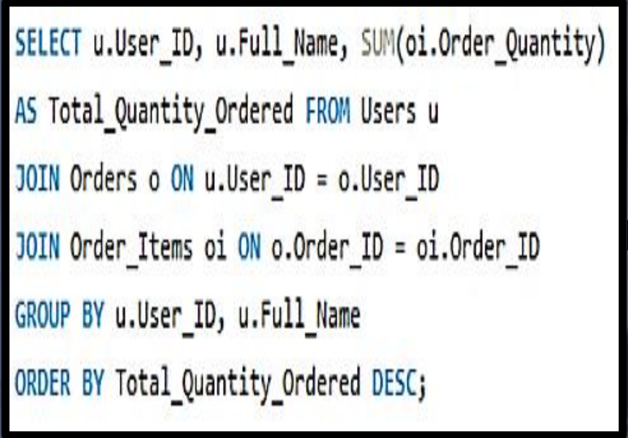
- **Strengthen local marketing campaigns, businesses should prioritize country-specific marketing strategies, fast local delivery, country-specific products, and customer testimonials..**
- **In future, it is recommended introducing customer loyalty programs based on purchase history can help retain local users and encourage repeat purchases..**



Aggregate Functions: (SUM, AVG, COUNT, MIN, MAX)

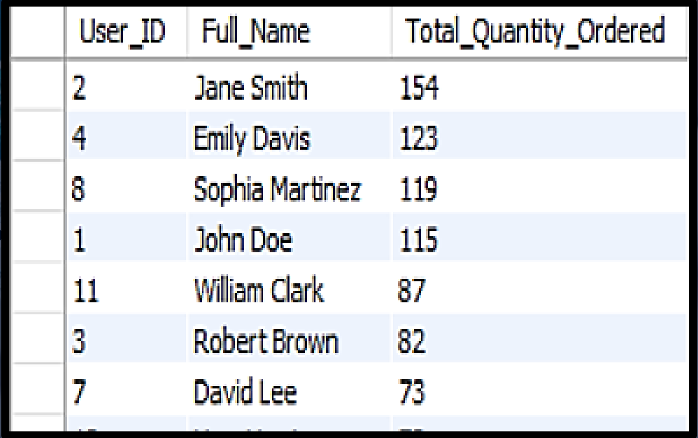
1. Calculate the total quantity of products ordered by each user.

MySQL Code



```sql
SELECT u.User_ID, u.Full_Name, SUM(oi.Order_Quantity)
AS Total_Quantity_Ordered FROM Users u
JOIN Orders o ON u.User_ID = o.User_ID
JOIN Order_Items oi ON o.Order_ID = oi.Order_ID
GROUP BY u.User_ID, u.Full_Name
ORDER BY Total_Quantity_Ordered DESC;
```

Output

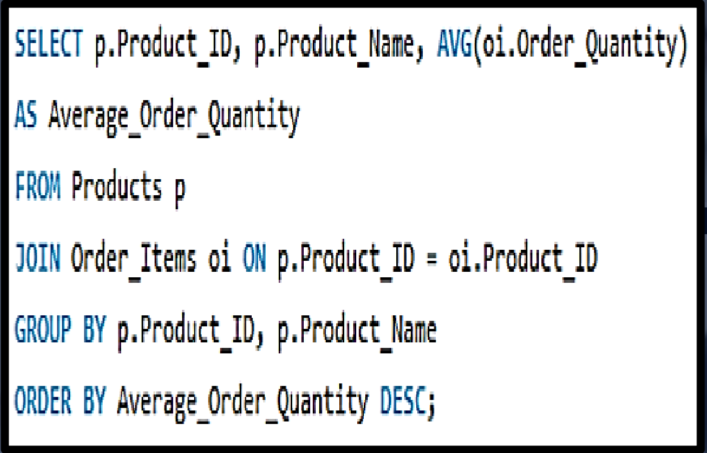


```
User_ID   Full_Name         Total_Quantity_Ordered
2         Jane Smith        154
4         Emily Davis       123
8         Sophia Martinez   119
1         John Doe          115
11        William Clark     87
3         Robert Brown      82
7         David Lee         73
```

- **Highest-volume customers, ordering between 115 and 154 products. In future, engaging with these users through personalized promotions or loyalty programs could increase retention and drive further sales.**
- **lowest-volume customers, re-engagement strategies, such as sending personalized offers or providing incentives for repeat purchases, could help retain these customers.**

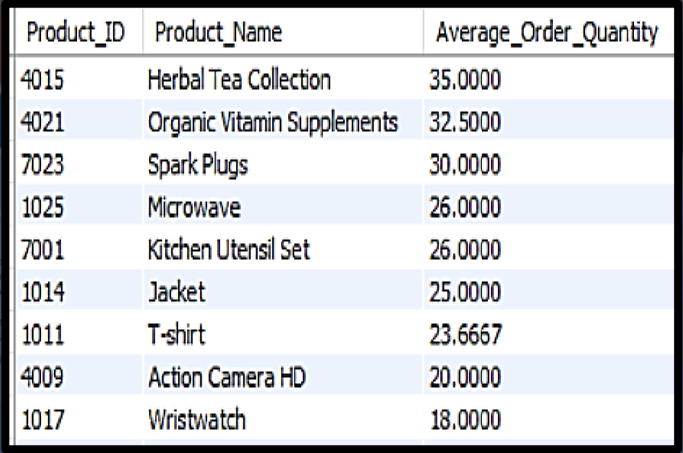
2. Find the average order quantity for each product.

MySQL Code



```
SELECT p.Product_ID, p.Product_Name, AVG(oi.Order_Quantity)
AS Average_Order_Quantity
FROM Products p
JOIN Order_Items oi ON p.Product_ID = oi.Product_ID
GROUP BY p.Product_ID, p.Product_Name
ORDER BY Average_Order_Quantity DESC;
```

Output

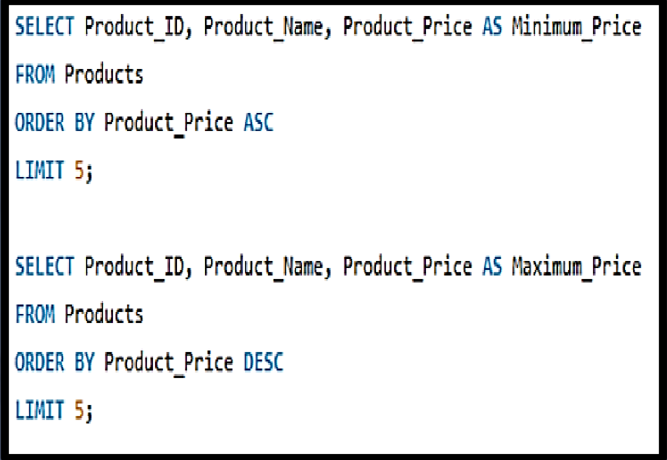


Product_ID	Product_Name	Average_Order_Quantity
4015	Herbal Tea Collection	35.0000
4021	Organic Vitamin Supplements	32.5000
7023	Spark Plugs	30.0000
1025	Microwave	26.0000
7001	Kitchen Utensil Set	26.0000
1014	Jacket	25.0000
1011	T-shirt	23.6667
4009	Action Camera HD	20.0000
1017	Wristwatch	18.0000

- **By analyzing the average order quantities for each product, businesses can better manage their inventory, optimize pricing strategies.**
- **Tailor marketing efforts to maximize sales and enhance the customer experience.**

3. Determine the minimum and maximum prices of the products.

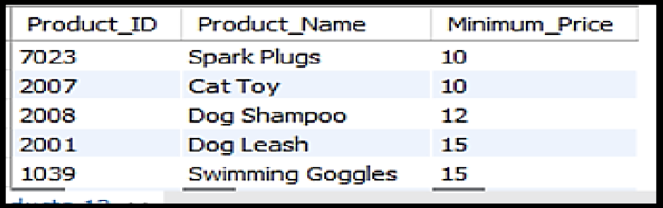
MySQL Code



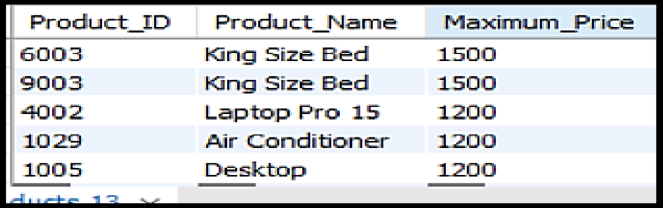
```sql
SELECT Product_ID, Product_Name, Product_Price AS Minimum_Price
FROM Products
ORDER BY Product_Price ASC
LIMIT 5;


SELECT Product_ID, Product_Name, Product_Price AS Maximum_Price
FROM Products
ORDER BY Product_Price DESC
LIMIT 5;
```

Output



```
Product_ID   Product_Name       Minimum_Price
7023         Spark Plugs        10
2007         Cat Toy            10
2008         Dog Shampoo        12
2001         Dog Leash          15
1039         Swimming Goggles   15
```

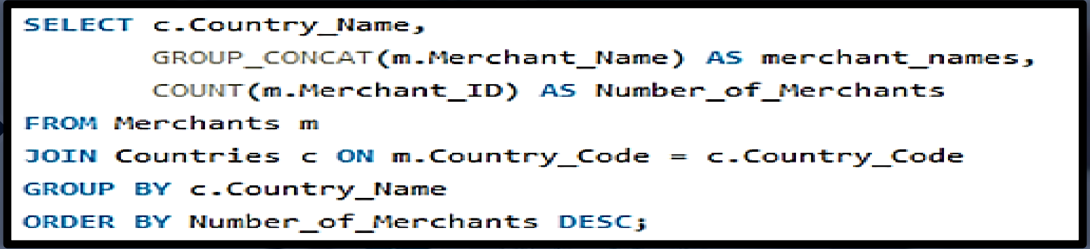


```
Product_ID   Product_Name       Maximum_Price
6003         King Size Bed      1500
9003         King Size Bed      1500
4002         Laptop Pro 15      1200
1029         Air Conditioner    1200
1005         Desktop            1200
```

- **If there's a gap in the product offerings between the lowest and highest prices, consider developing products that could appeal to middle-market consumers, potentially increasing market share.**
- **Use price segmentation to more effectively target marketing and sales efforts.**

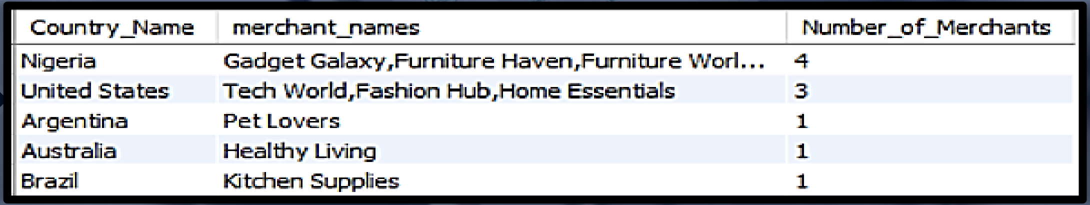
4. Count the total number of merchants operating in each country.

MySQL Code



```
SELECT c.Country_Name,
       GROUP_CONCAT(m.Merchant_Name) AS merchant_names,
       COUNT(m.Merchant_ID) AS Number_of_Merchants
FROM Merchants m
JOIN Countries c ON m.Country_Code = c.Country_Code
GROUP BY c.Country_Name
ORDER BY Number_of_Merchants DESC;
```

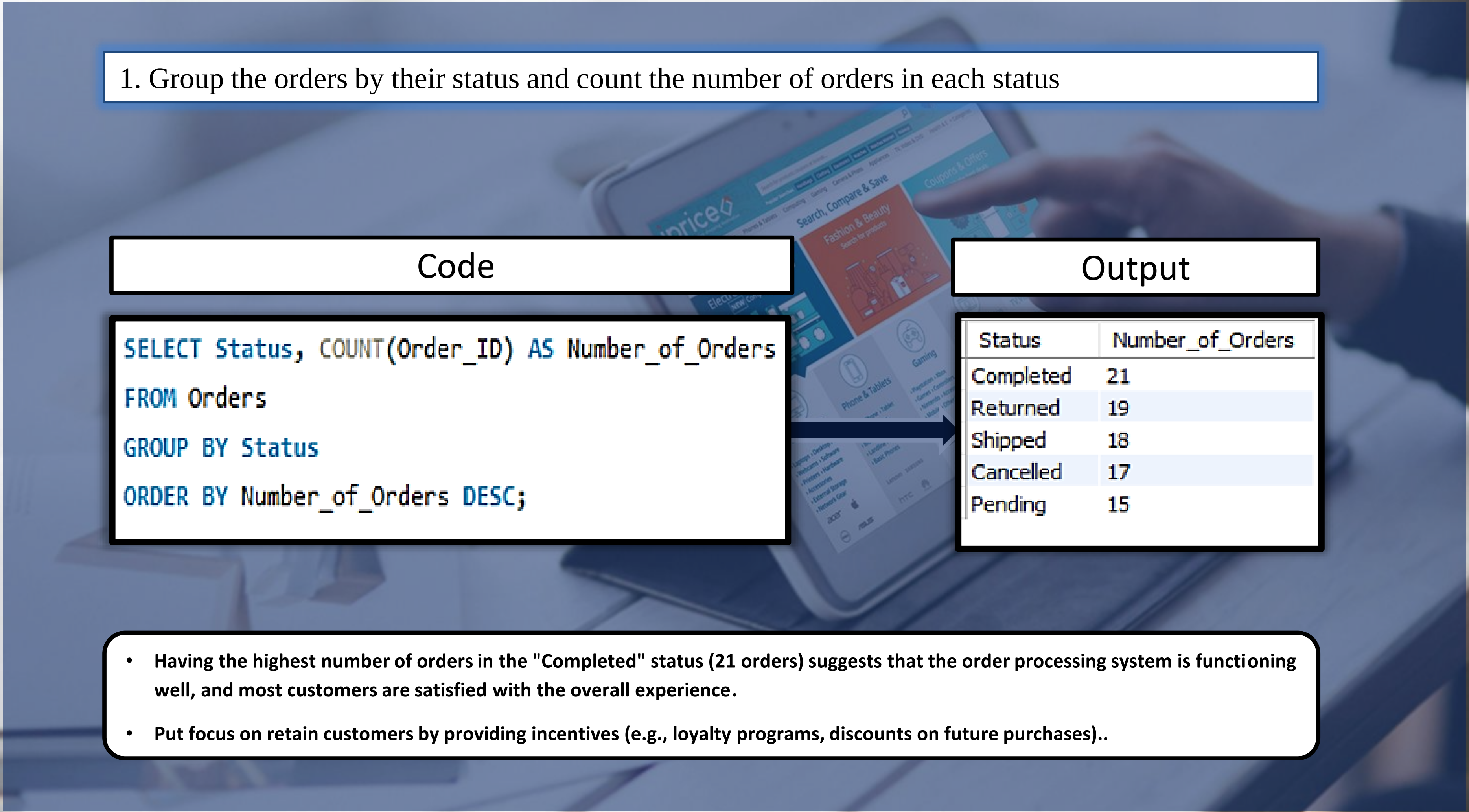
Output



```
Country_Name    merchant_names                                     Number_of_Merchants
Nigeria         Gadget Galaxy,Furniture Haven,Furniture Worl...    4
United States   Tech World,Fashion Hub,Home Essentials             3
Argentina       Pet Lovers                                         1
Australia       Healthy Living                                     1
Brazil          Kitchen Supplies                                   1
```

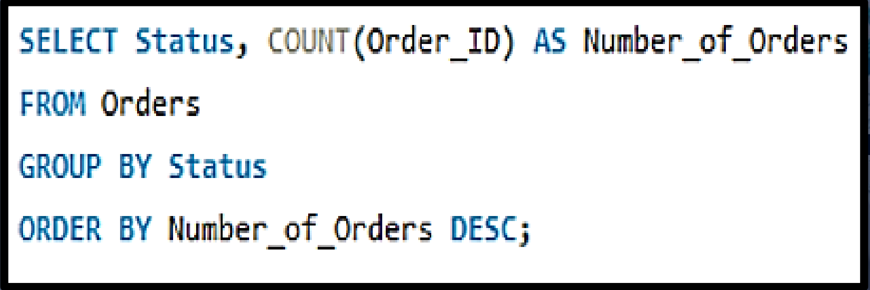
- **High competition in Nigeria with 4 merchants could be more competitive. They need to differentiate their offerings and provide superior customer service to gain market share.**
- **Differentiation through unique product offerings, local partnerships, and customer experiences will be challenging for success .**

Group By & Having Clauses



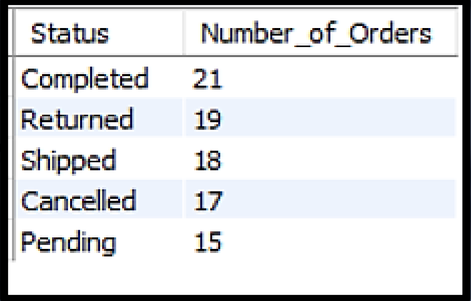
1. Group the orders by their status and count the number of orders in each status

Code



```sql
SELECT Status, COUNT(Order_ID) AS Number_of_Orders
FROM Orders
GROUP BY Status
ORDER BY Number_of_Orders DESC;
```

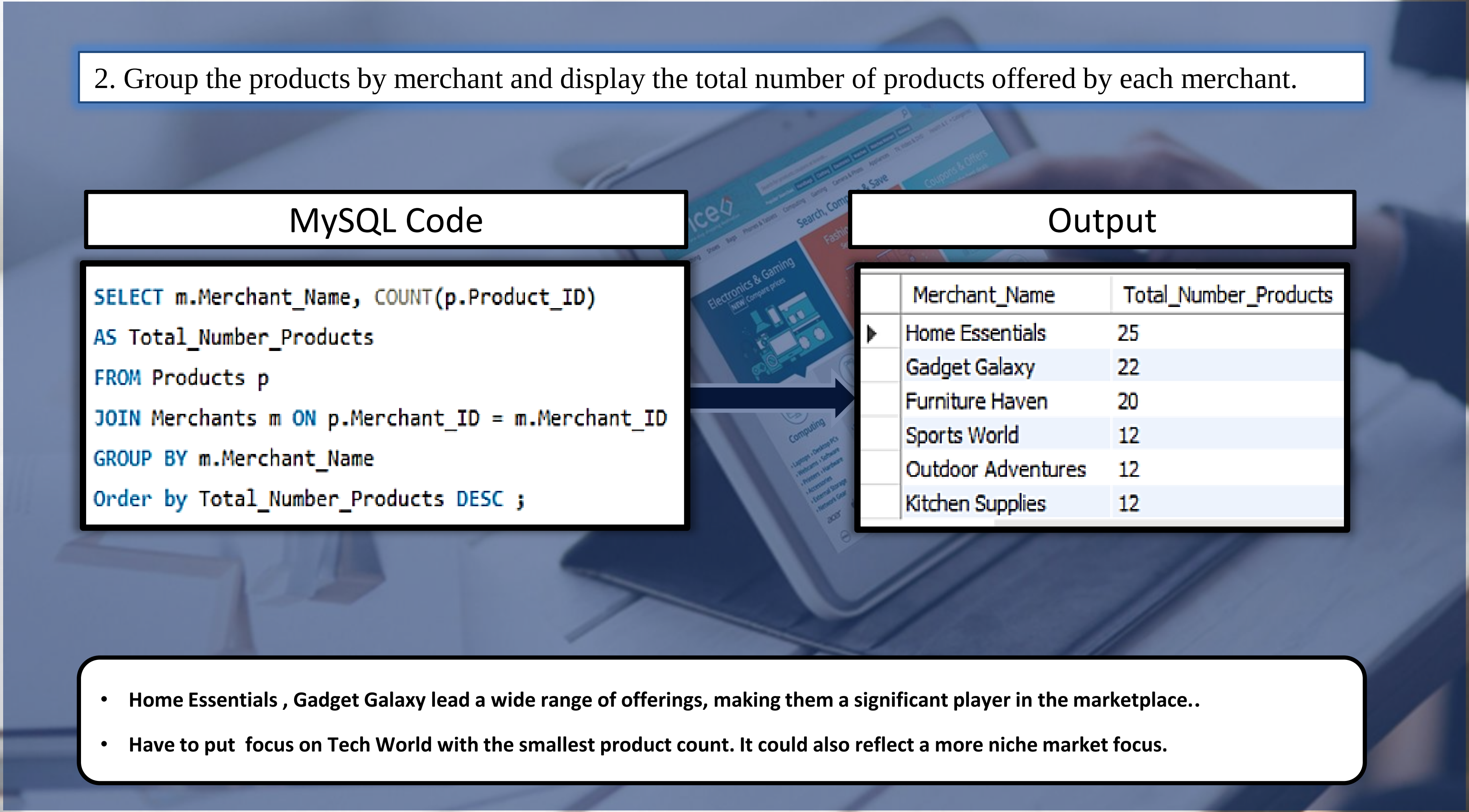
Output



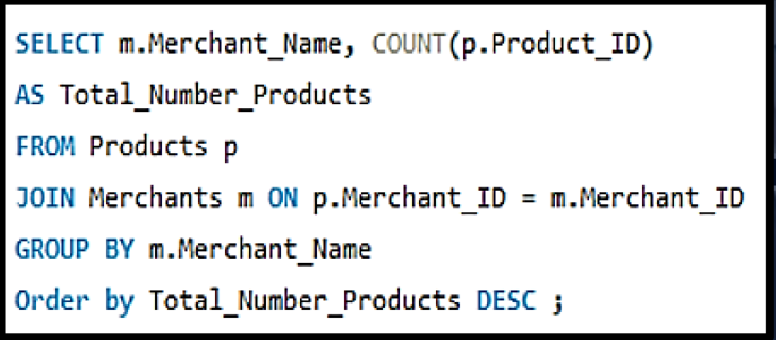
```
Status       Number_of_Orders
Completed    21
Returned     19
Shipped      18
Cancelled    17
Pending      15
```

- **Having the highest number of orders in the "Completed" status (21 orders) suggests that the order processing system is functioning well, and most customers are satisfied with the overall experience.**
- **Put focus on retain customers by providing incentives (e.g., loyalty programs, discounts on future purchases)..**

2. Group the products by merchant and display the total number of products offered by each merchant.

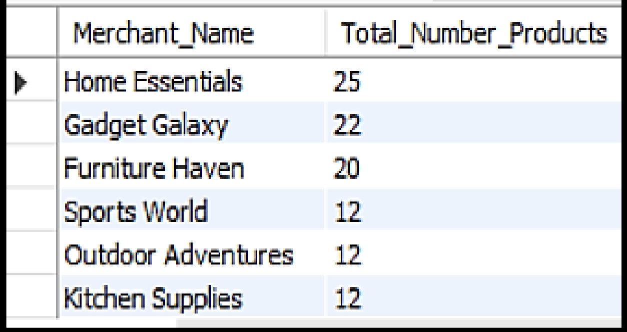


MySQL Code



```sql
SELECT m.Merchant_Name, COUNT(p.Product_ID)
AS Total_Number_Products
FROM Products p
JOIN Merchants m ON p.Merchant_ID = m.Merchant_ID
GROUP BY m.Merchant_Name
Order by Total_Number_Products DESC ;
```

Output

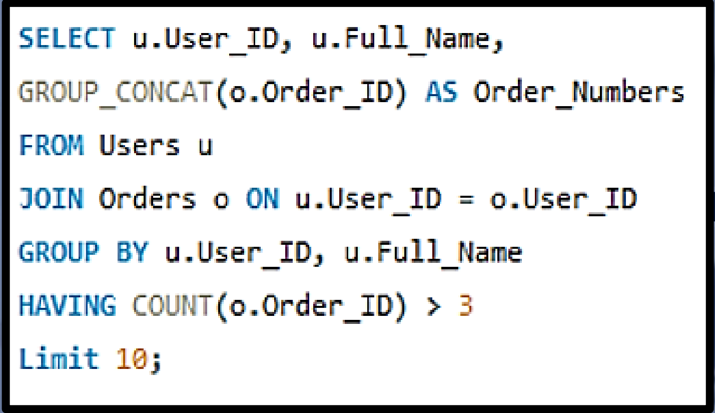


```
   Merchant_Name        Total_Number_Products
▶  Home Essentials      25
   Gadget Galaxy        22
   Furniture Haven      20
   Sports World         12
   Outdoor Adventures   12
   Kitchen Supplies     12
```

- **Home Essentials , Gadget Galaxy lead a wide range of offerings, making them a significant player in the marketplace..**
- **Have to put focus on Tech World with the smallest product count. It could also reflect a more niche market focus.**

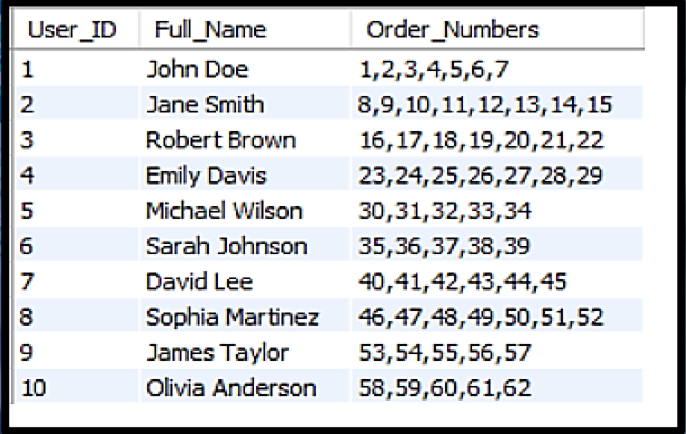
3. Show the users who have placed more than 3 orders.

MySQL Code



```sql
SELECT u.User_ID, u.Full_Name,
GROUP_CONCAT(o.Order_ID) AS Order_Numbers
FROM Users u
JOIN Orders o ON u.User_ID = o.User_ID
GROUP BY u.User_ID, u.Full_Name
HAVING COUNT(o.Order_ID) > 3
Limit 10;
```

Output



User_ID	Full_Name	Order_Numbers
1	John Doe	1,2,3,4,5,6,7
2	Jane Smith	8,9,10,11,12,13,14,15
3	Robert Brown	16,17,18,19,20,21,22
4	Emily Davis	23,24,25,26,27,28,29
5	Michael Wilson	30,31,32,33,34
6	Sarah Johnson	35,36,37,38,39
7	David Lee	40,41,42,43,44,45
8	Sophia Martinez	46,47,48,49,50,51,52
9	James Taylor	53,54,55,56,57
10	Olivia Anderson	58,59,60,61,62

- **Users who have placed more than three orders, providing valuable insights into repeat customers.**
- **Put focus on retain users by providing incentives (e.g., loyalty programs, discounts on future purchases) and drive sales through repeat purchases**.

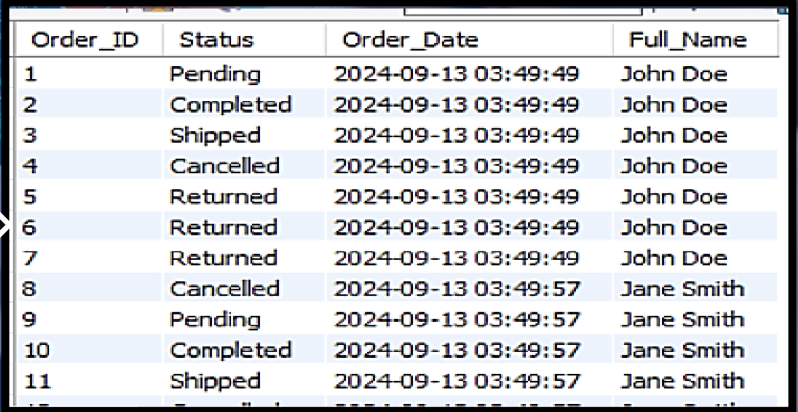
SQL Joins

1 . Perform an inner join between orders and users to retrieve all the orders along with the user's full name

MySQL Code

```
SELECT o.Order_ID, o.Status, o.Created_at
AS Order_Date, u.Full_Name
FROM Orders o
INNER JOIN Users u ON o.User_ID = u.User_ID;
```

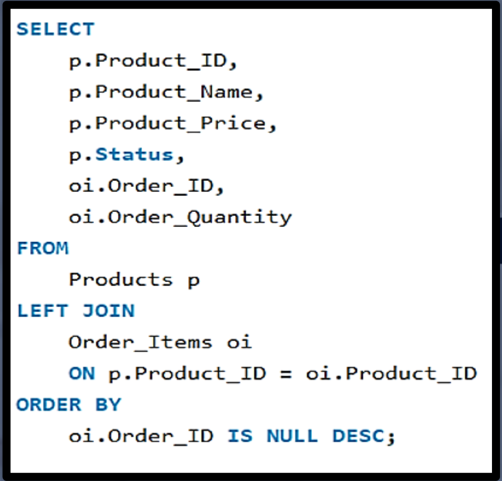
Output



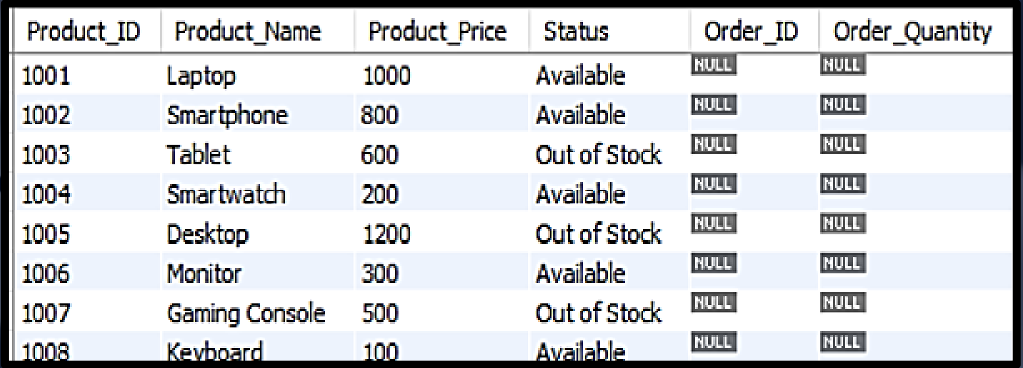
Order_ID	Status	Order_Date	Full_Name
1	Pending	2024-09-13 03:49:49	John Doe
2	Completed	2024-09-13 03:49:49	John Doe
3	Shipped	2024-09-13 03:49:49	John Doe
4	Cancelled	2024-09-13 03:49:49	John Doe
5	Returned	2024-09-13 03:49:49	John Doe
6	Returned	2024-09-13 03:49:49	John Doe
7	Returned	2024-09-13 03:49:49	John Doe
8	Cancelled	2024-09-13 03:49:57	Jane Smith
9	Pending	2024-09-13 03:49:57	Jane Smith
10	Completed	2024-09-13 03:49:57	Jane Smith
11	Shipped	2024-09-13 03:49:57	Jane Smith

- **Analyzing frequent order statuses per customer may provide insights into specific customer behavior.**
- **Specially, return items need for targeted customer service interventions or improved product offerings.**

2. Use a left join to retrieve all products and their respective orders. Include products that haven't been ordered.



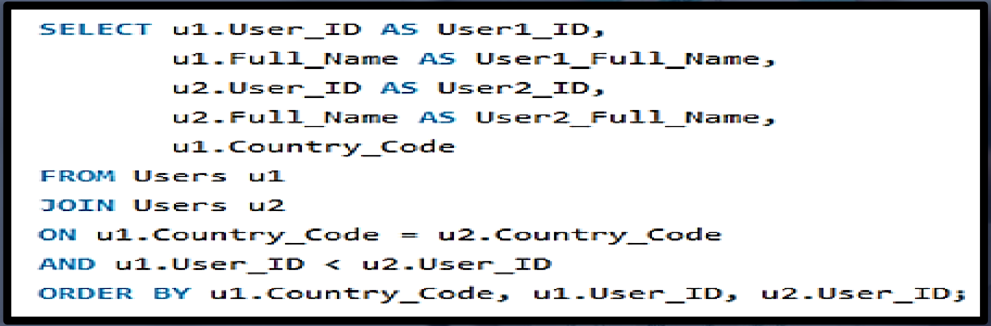
```
SELECT
    p.Product_ID,
    p.Product_Name,
    p.Product_Price,
    p.Status,
    oi.Order_ID,
    oi.Order_Quantity
FROM
    Products p
LEFT JOIN
    Order_Items oi
    ON p.Product_ID = oi.Product_ID
ORDER BY
    oi.Order_ID IS NULL DESC;
```



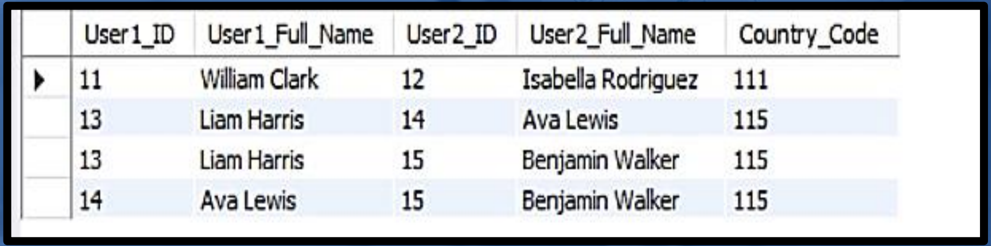
Product_ID	Product_Name	Product_Price	Status	Order_ID	Order_Quantity
1001	Laptop	1000	Available	NULL	NULL
1002	Smartphone	800	Available	NULL	NULL
1003	Tablet	600	Out of Stock	NULL	NULL
1004	Smartwatch	200	Available	NULL	NULL
1005	Desktop	1200	Out of Stock	NULL	NULL
1006	Monitor	300	Available	NULL	NULL
1007	Gaming Console	500	Out of Stock	NULL	NULL
1008	Keyboard	100	Available	NULL	NULL

- **Many products appear without any associated orders (indicated by NULL in Order_ID), signaling that these items may be underperforming.**
- **The business might consider reviewing their marketing strategy for these products or even discontinuing them.**

3. Use a self-join to find users who share the same country code.

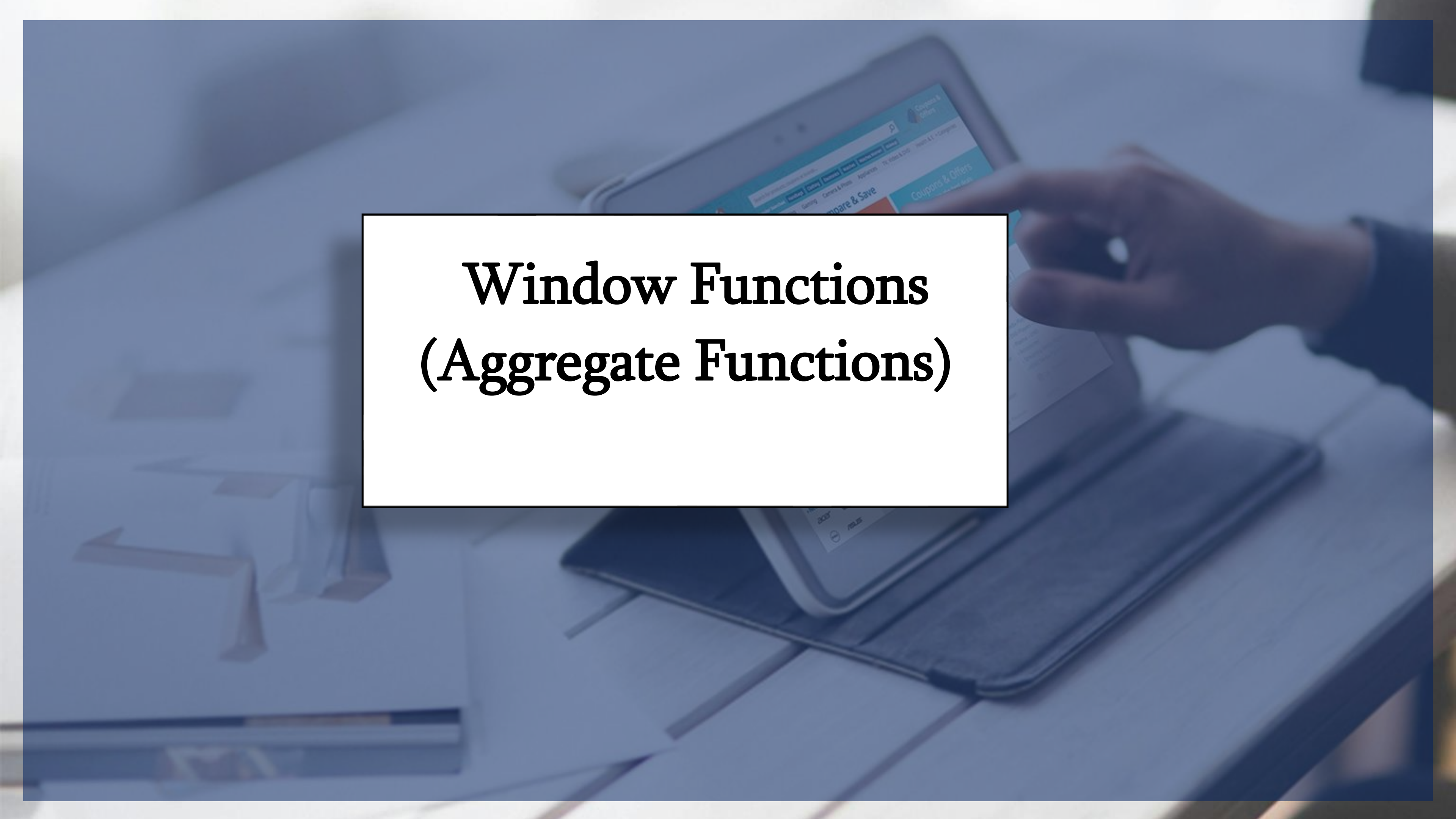


```
SELECT u1.User_ID AS User1_ID,
       u1.Full_Name AS User1_Full_Name,
       u2.User_ID AS User2_ID,
       u2.Full_Name AS User2_Full_Name,
       u1.Country_Code
FROM Users u1
JOIN Users u2
ON u1.Country_Code = u2.Country_Code
AND u1.User_ID < u2.User_ID
ORDER BY u1.Country_Code, u1.User_ID, u2.User_ID;
```



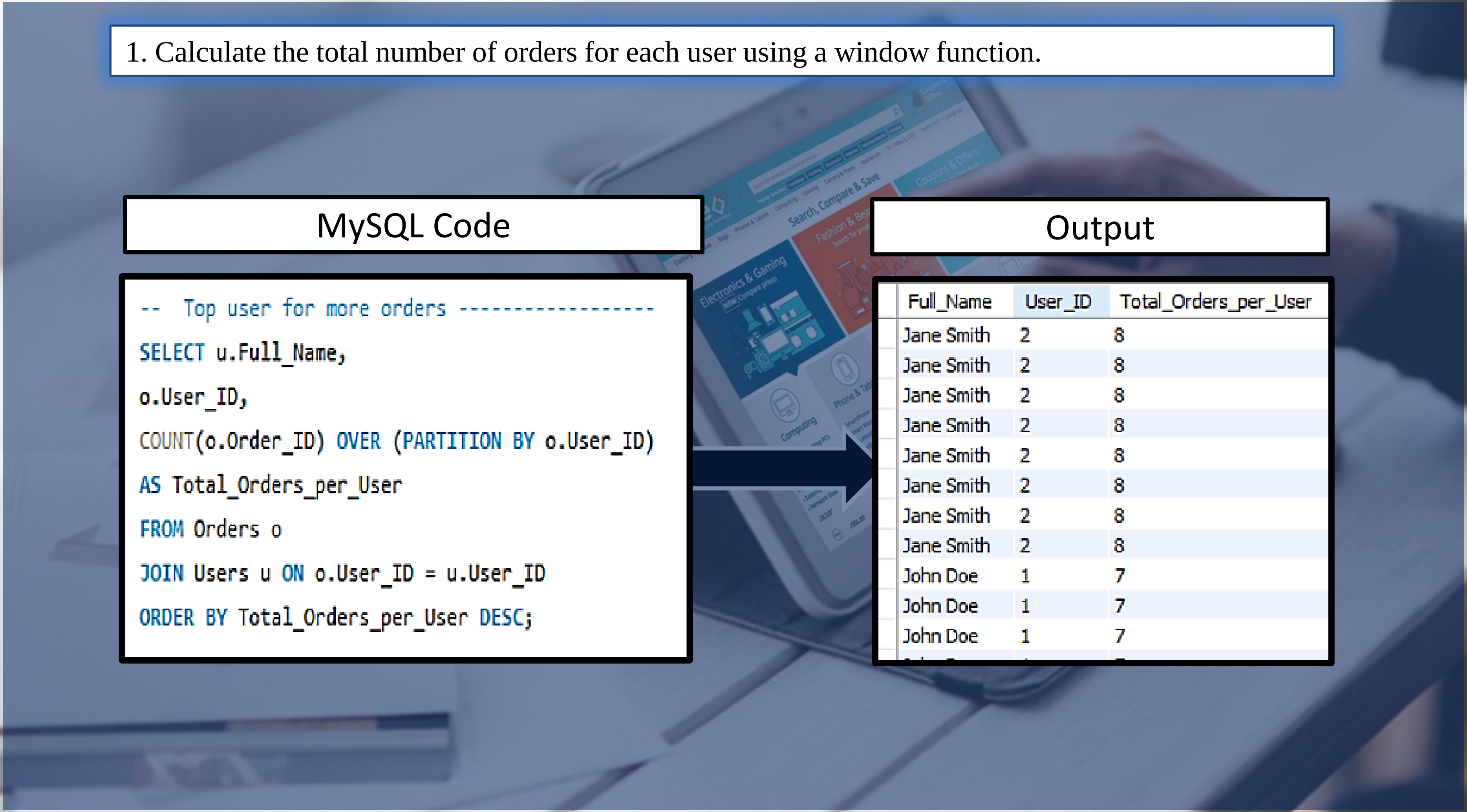
User1_ID	User1_Full_Name	User2_ID	User2_Full_Name	Country_Code
11	William Clark	12	Isabella Rodriguez	111
13	Liam Harris	14	Ava Lewis	115
13	Liam Harris	15	Benjamin Walker	115
14	Ava Lewis	15	Benjamin Walker	115

- This can help tailor regional marketing campaigns, personalized offers, or customer support strategies based on geographic location.Put focus on retain customers by providing incentives (e.g., loyalty programs, discounts on future purchases)..
- Users from the same country may share cultural or regional preferences to foster community building through region-specific engagement, social features, or events.

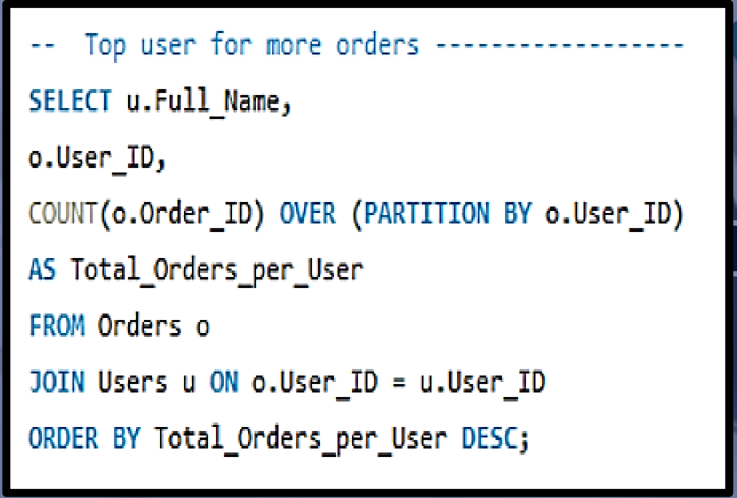


Window Functions (Aggregate Functions)

1. Calculate the total number of orders for each user using a window function.

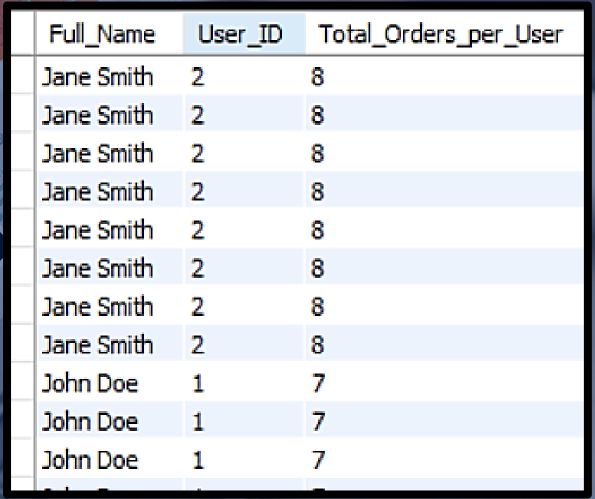


MySQL Code



```
--  Top user for more orders ------------------
SELECT u.Full_Name,
o.User_ID,
COUNT(o.Order_ID) OVER (PARTITION BY o.User_ID)
AS Total_Orders_per_User
FROM Orders o
JOIN Users u ON o.User_ID = u.User_ID
ORDER BY Total_Orders_per_User DESC;
```

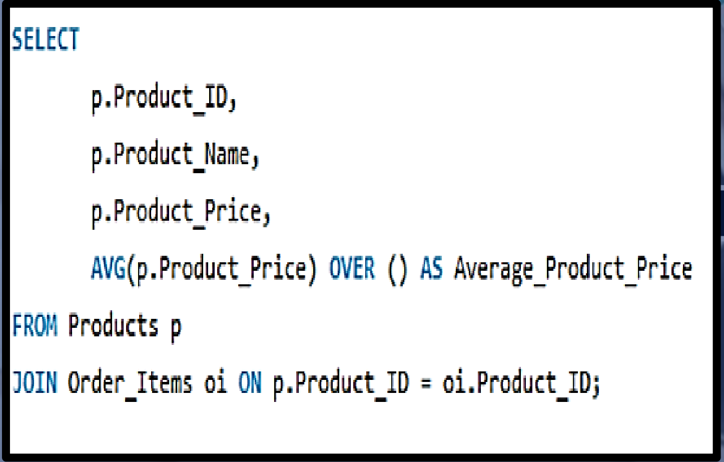
Output



```
Full_Name    User_ID   Total_Orders_per_User
Jane Smith   2         8
Jane Smith   2         8
Jane Smith   2         8
Jane Smith   2         8
Jane Smith   2         8
Jane Smith   2         8
Jane Smith   2         8
Jane Smith   2         8
John Doe     1         7
John Doe     1         7
John Doe     1         7
```

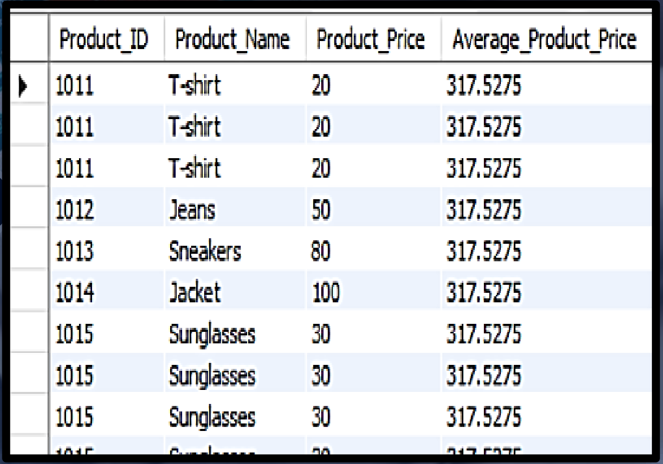
2. Calculate the average price of products over all orders using a window function..

MySQL Code



```
SELECT
	p.Product_ID,
	p.Product_Name,
	p.Product_Price,
	AVG(p.Product_Price) OVER () AS Average_Product_Price
FROM Products p
JOIN Order_Items oi ON p.Product_ID = oi.Product_ID;
```

Output



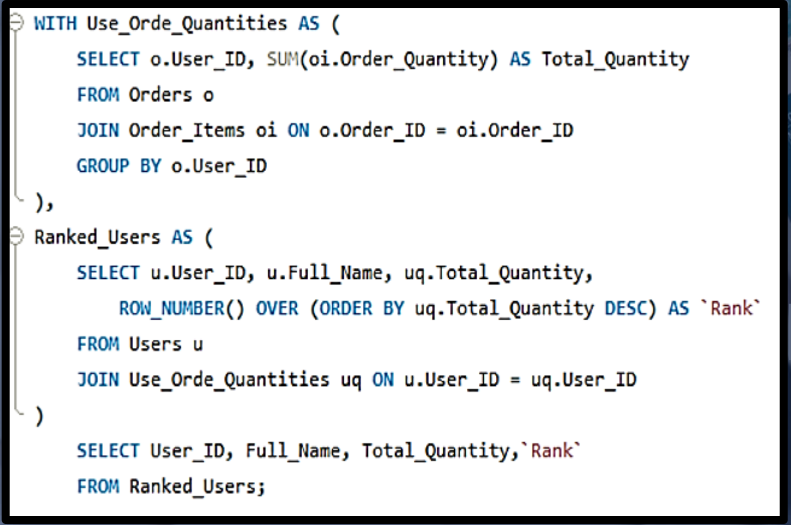
Product_ID	Product_Name	Product_Price	Average_Product_Price
1011	T-shirt	20	317.5275
1011	T-shirt	20	317.5275
1011	T-shirt	20	317.5275
1012	Jeans	50	317.5275
1013	Sneakers	80	317.5275
1014	Jacket	100	317.5275
1015	Sunglasses	30	317.5275
1015	Sunglasses	30	317.5275
1015	Sunglasses	30	317.5275

- **By computing the overall average price help in determining if certain products are overpriced or underpriced relative to the overall product portfolio.**
- **Helping in adjusting prices to optimize revenue, ensuring that products meet market demand and profitability goals.**

Window Functions (Ranking)

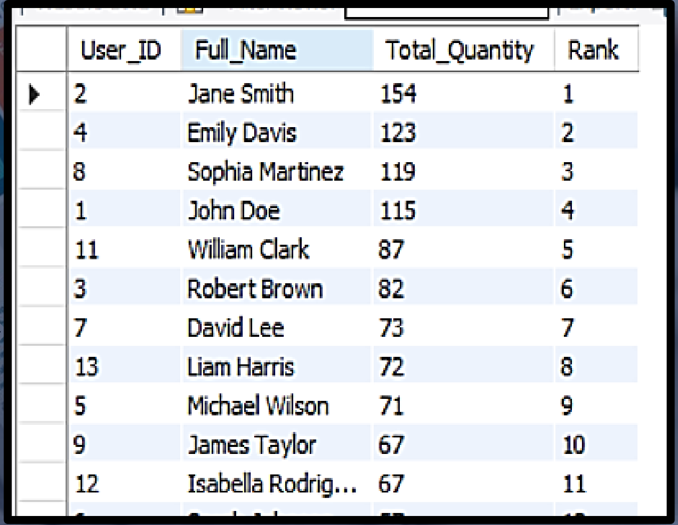
1. Rank the users based on the total quantity of products ordered using ROW_NUMBER().

MySQL Code



```sql
WITH Use_Orde_Quantities AS (
    SELECT o.User_ID, SUM(oi.Order_Quantity) AS Total_Quantity
    FROM Orders o
    JOIN Order_Items oi ON o.Order_ID = oi.Order_ID
    GROUP BY o.User_ID
),
Ranked_Users AS (
    SELECT u.User_ID, u.Full_Name, uq.Total_Quantity,
        ROW_NUMBER() OVER (ORDER BY uq.Total_Quantity DESC) AS `Rank`
    FROM Users u
    JOIN Use_Orde_Quantities uq ON u.User_ID = uq.User_ID
)
    SELECT User_ID, Full_Name, Total_Quantity,`Rank`
    FROM Ranked_Users;
```

Output

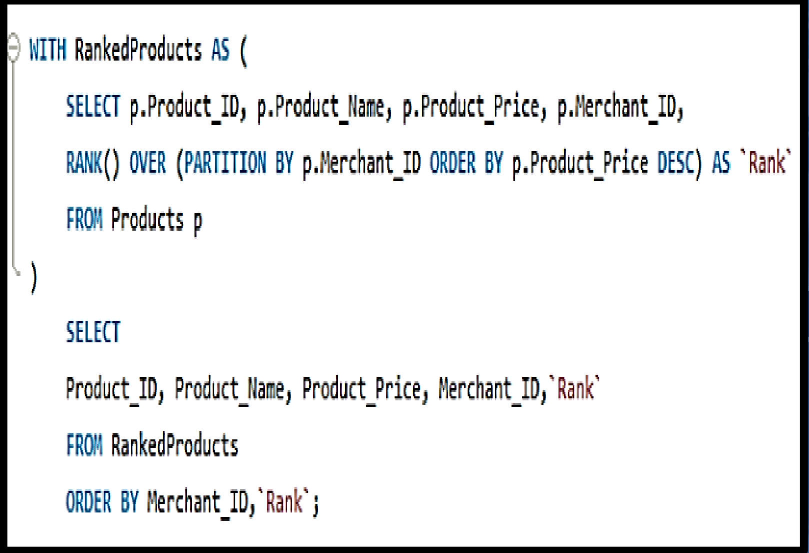


```
User_ID   Full_Name          Total_Quantity   Rank
2         Jane Smith         154              1
4         Emily Davis        123              2
8         Sophia Martinez    119              3
1         John Doe           115              4
11        William Clark      87               5
3         Robert Brown       82               6
7         David Lee          73               7
13        Liam Harris        72               8
5         Michael Wilson     71               9
9         James Taylor       67               10
12        Isabella Rodrig...  67              11
```

- **The most active or high-value customers, who are crucial for driving sales volume. These users may represent a significant portion of revenue.**
- **The ranking provides insight into which customers may benefit from loyalty programs, targeted promotions**

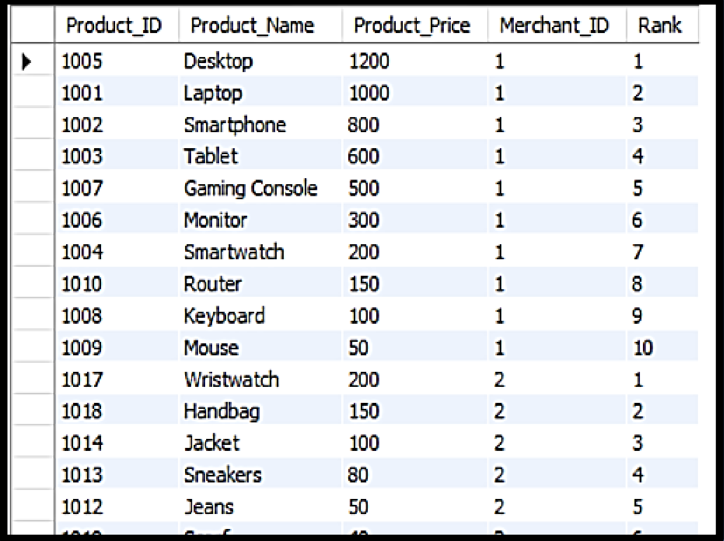
2. Use RANK() to rank products based on their price within each merchant

MySQL Code



```sql
WITH RankedProducts AS (
    SELECT p.Product_ID, p.Product_Name, p.Product_Price, p.Merchant_ID,
    RANK() OVER (PARTITION BY p.Merchant_ID ORDER BY p.Product_Price DESC) AS `Rank`
    FROM Products p
)
    SELECT
    Product_ID, Product_Name, Product_Price, Merchant_ID,`Rank`
    FROM RankedProducts
    ORDER BY Merchant_ID,`Rank`;
```

Output

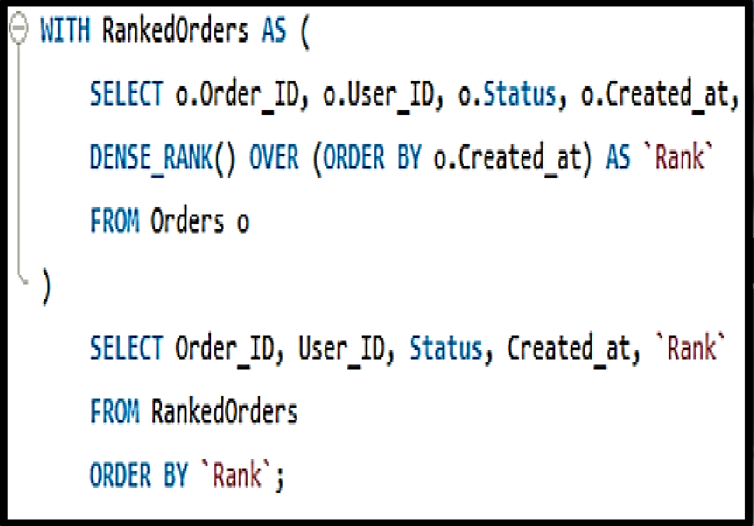


Product_ID	Product_Name	Product_Price	Merchant_ID	Rank
1005	Desktop	1200	1	1
1001	Laptop	1000	1	2
1002	Smartphone	800	1	3
1003	Tablet	600	1	4
1007	Gaming Console	500	1	5
1006	Monitor	300	1	6
1004	Smartwatch	200	1	7
1010	Router	150	1	8
1008	Keyboard	100	1	9
1009	Mouse	50	1	10
1017	Wristwatch	200	2	1
1018	Handbag	150	2	2
1014	Jacket	100	2	3
1013	Sneakers	80	2	4
1012	Jeans	50	2	5

- **Merchants can focus on high-ranking products (the most expensive) for premium customers and ensure these items are well-stocked and promoted.**
- **For lower-ranking, affordable products, merchants can target price-sensitive customers and optimize for volume sales.**

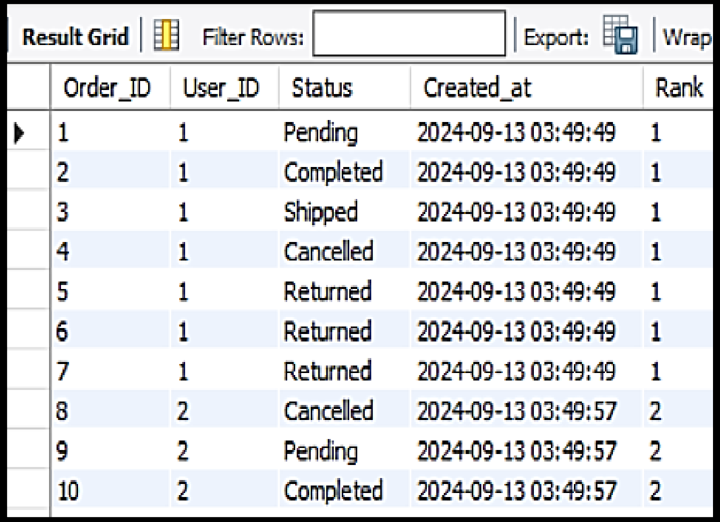
3. Determine the DENSE_RANK() of orders by their created date.

MySQL Code



```
WITH RankedOrders AS (
    SELECT o.Order_ID, o.User_ID, o.Status, o.Created_at,
    DENSE_RANK() OVER (ORDER BY o.Created_at) AS `Rank`
    FROM Orders o
)
    SELECT Order_ID, User_ID, Status, Created_at, `Rank`
    FROM RankedOrders
    ORDER BY `Rank`;
```

Output

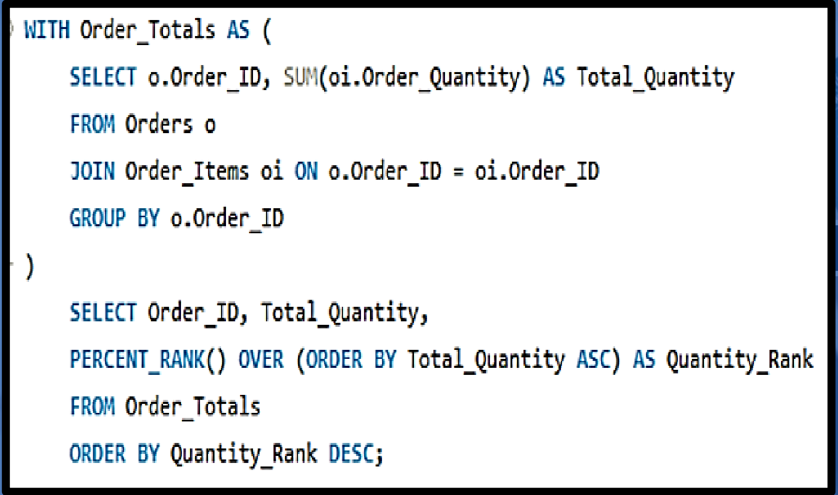


Order_ID	User_ID	Status	Created_at	Rank
1	1	Pending	2024-09-13 03:49:49	1
2	1	Completed	2024-09-13 03:49:49	1
3	1	Shipped	2024-09-13 03:49:49	1
4	1	Cancelled	2024-09-13 03:49:49	1
5	1	Returned	2024-09-13 03:49:49	1
6	1	Returned	2024-09-13 03:49:49	1
7	1	Returned	2024-09-13 03:49:49	1
8	2	Cancelled	2024-09-13 03:49:57	2
9	2	Pending	2024-09-13 03:49:57	2
10	2	Completed	2024-09-13 03:49:57	2

- **Dense ranking helps identify patterns in order creation and lifecycle progression, enabling businesses to improve fulfillment efficiency, customer service, and system performance..**

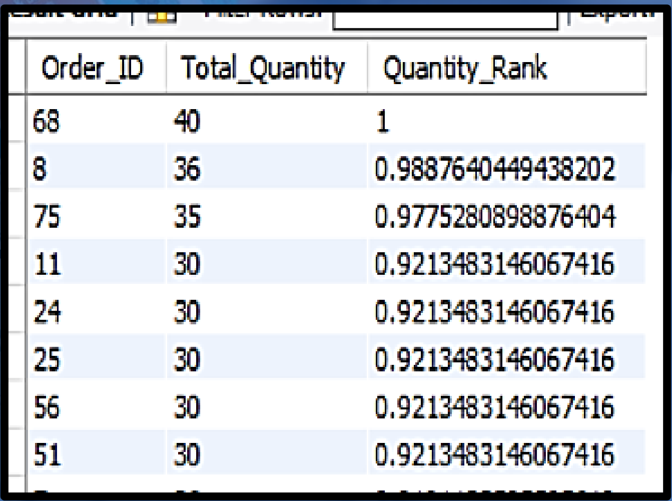
4 . Use PERCENT_RANK() to find the rank of orders by their quantity compared to the overall total.

MySQL Code



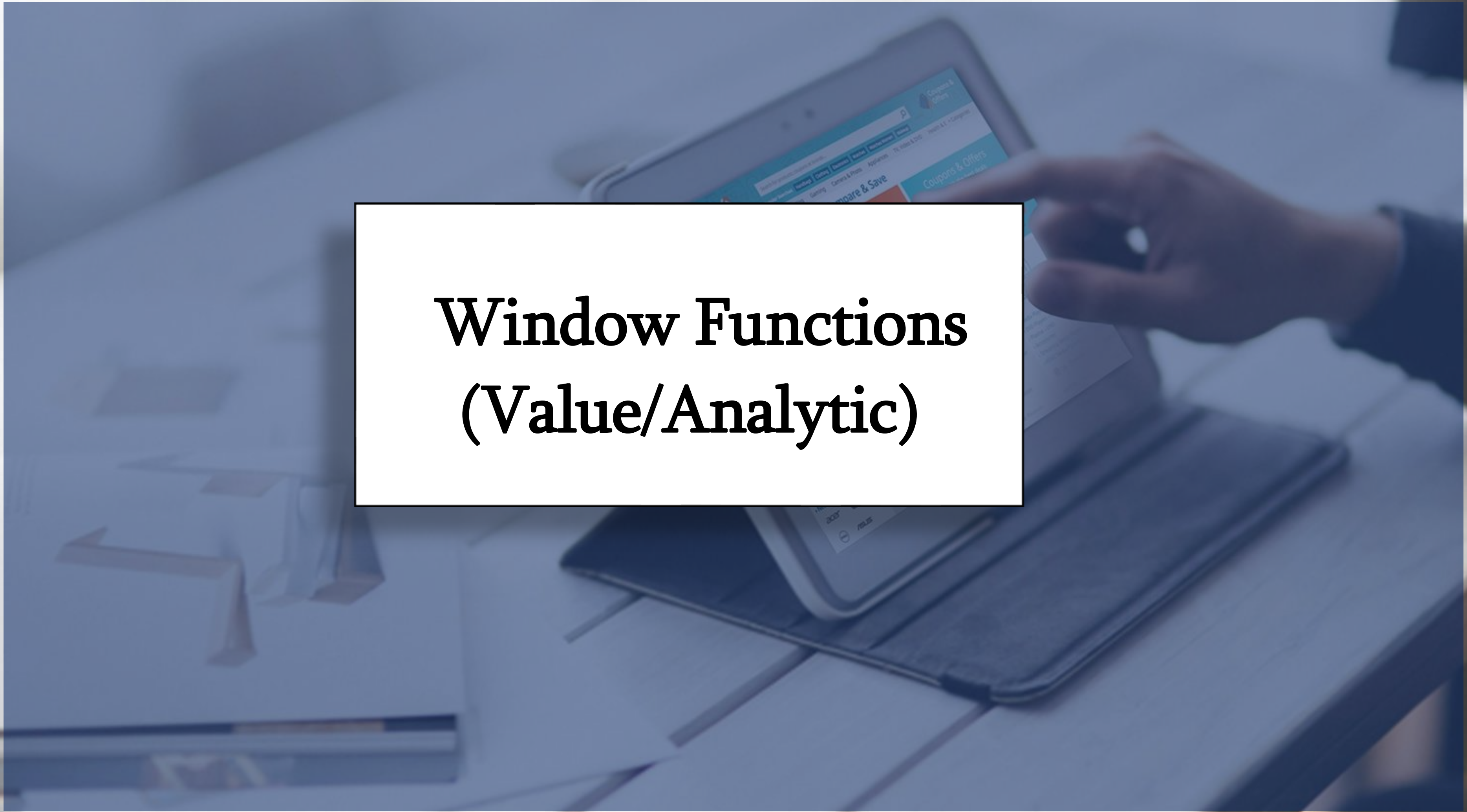
```sql
WITH Order_Totals AS (
    SELECT o.Order_ID, SUM(oi.Order_Quantity) AS Total_Quantity
    FROM Orders o
    JOIN Order_Items oi ON o.Order_ID = oi.Order_ID
    GROUP BY o.Order_ID
)
    SELECT Order_ID, Total_Quantity,
    PERCENT_RANK() OVER (ORDER BY Total_Quantity ASC) AS Quantity_Rank
    FROM Order_Totals
    ORDER BY Quantity_Rank DESC;
```

Output



Order_ID	Total_Quantity	Quantity_Rank
68	40	1
8	36	0.9887640449438202
75	35	0.9775280898876404
11	30	0.9213483146067416
24	30	0.9213483146067416
25	30	0.9213483146067416
56	30	0.9213483146067416
51	30	0.9213483146067416

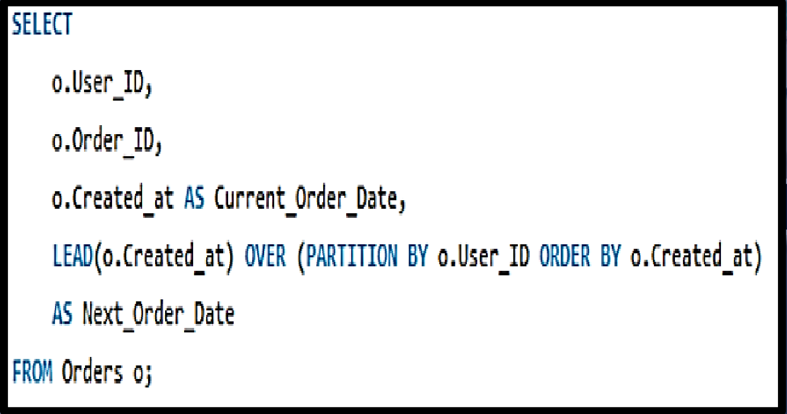
- **This analysis helps to understand how orders compare in size relative to the total quantity distribution across all orders.For lower-ranking, affordable products, merchants can target price-sensitive customers and optimize for volume sales.**
- **By understanding the PERCENT_RANK() , businesses can optimize inventory, pricing strategies, and customer segmentation.**



Window Functions (Value/Analytic)

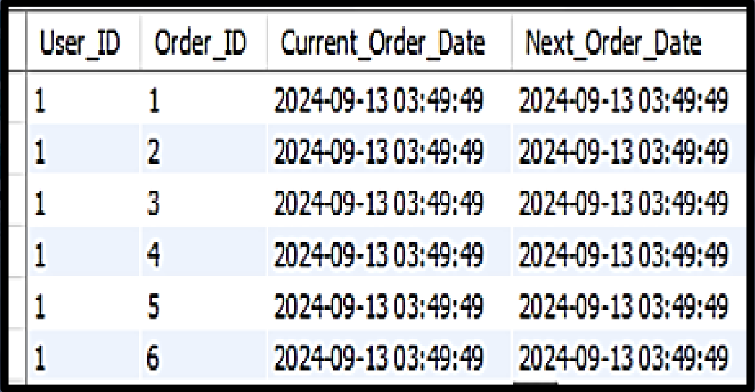
1. Use LEAD() to find the next order date for each user

MySQL Code



```
SELECT
    o.User_ID,
    o.Order_ID,
    o.Created_at AS Current_Order_Date,
    LEAD(o.Created_at) OVER (PARTITION BY o.User_ID ORDER BY o.Created_at)
    AS Next_Order_Date
FROM Orders o;
```

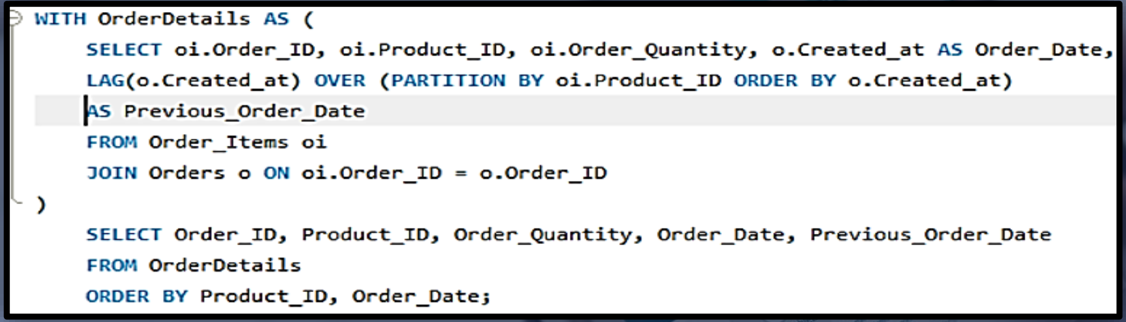
Output



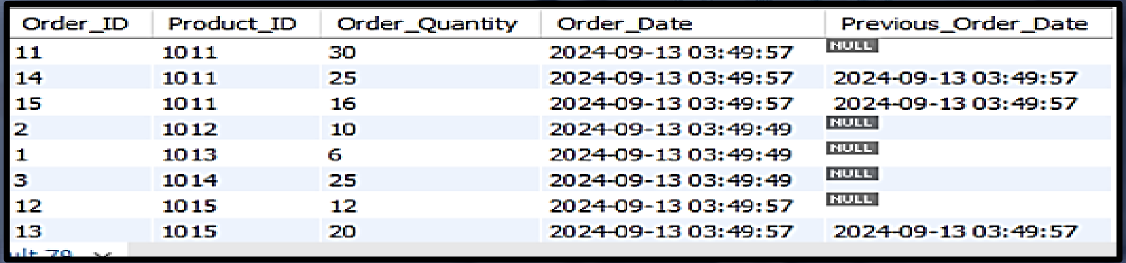
User_ID	Order_ID	Current_Order_Date	Next_Order_Date
1	1	2024-09-13 03:49:49	2024-09-13 03:49:49
1	2	2024-09-13 03:49:49	2024-09-13 03:49:49
1	3	2024-09-13 03:49:49	2024-09-13 03:49:49
1	4	2024-09-13 03:49:49	2024-09-13 03:49:49
1	5	2024-09-13 03:49:49	2024-09-13 03:49:49
1	6	2024-09-13 03:49:49	2024-09-13 03:49:49

- **Analysis concept here involves customer purchase behavior analysis by using the LEAD() function to find the next order date for each userFor lower-ranking, affordable products, merchants can target price-sensitive customers and optimize for volume sales.**
- **Helping in identify customer purchase patterns, which is critical for improving retention strategies, inventory management,.**

2. Use LAG() to determine the previous order for each product.



```sql
WITH OrderDetails AS (
    SELECT oi.Order_ID, oi.Product_ID, oi.Order_Quantity, o.Created_at AS Order_Date,
    LAG(o.Created_at) OVER (PARTITION BY oi.Product_ID ORDER BY o.Created_at)
    AS Previous_Order_Date
    FROM Order_Items oi
    JOIN Orders o ON oi.Order_ID = o.Order_ID
)
    SELECT Order_ID, Product_ID, Order_Quantity, Order_Date, Previous_Order_Date
    FROM OrderDetails
    ORDER BY Product_ID, Order_Date;
```

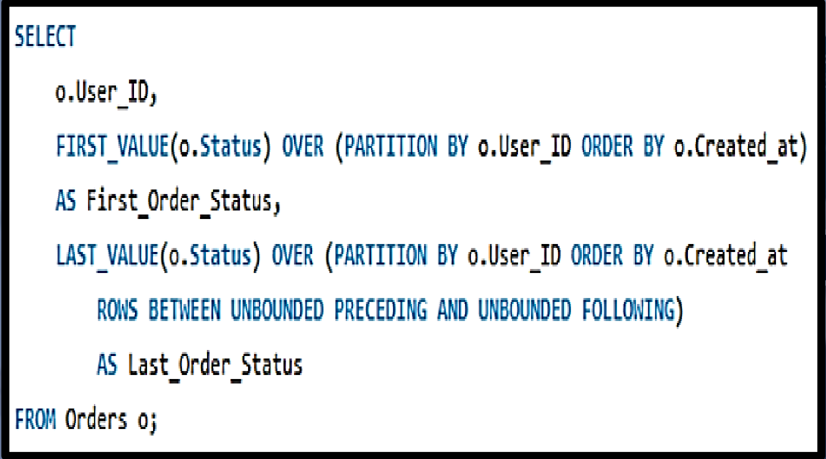


Order_ID	Product_ID	Order_Quantity	Order_Date	Previous_Order_Date
11	1011	30	2024-09-13 03:49:57	NULL
14	1011	25	2024-09-13 03:49:57	2024-09-13 03:49:57
15	1011	16	2024-09-13 03:49:57	2024-09-13 03:49:57
2	1012	10	2024-09-13 03:49:49	NULL
1	1013	6	2024-09-13 03:49:49	NULL
3	1014	25	2024-09-13 03:49:49	NULL
12	1015	12	2024-09-13 03:49:57	NULL
13	1015	20	2024-09-13 03:49:57	2024-09-13 03:49:57

- **The Previous_Order_Date column helps identify how frequently a product is reordered. For instance, products with NULL in this column were being ordered for the first time.**
- **By analyzing the gaps between the Order_Date and Previous_Order_Date, we can gauge demand consistency for each product.**

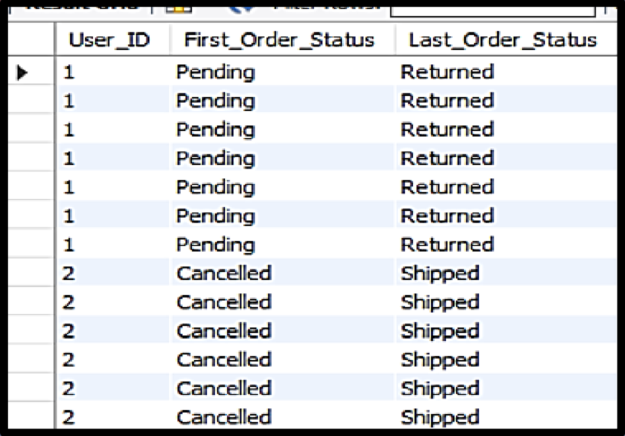
3. Retrieve the FIRST_VALUE() and LAST_VALUE() of order statuses for each user.

MySQL Code



```
SELECT
    o.User_ID,
    FIRST_VALUE(o.Status) OVER (PARTITION BY o.User_ID ORDER BY o.Created_at)
    AS First_Order_Status,
    LAST_VALUE(o.Status) OVER (PARTITION BY o.User_ID ORDER BY o.Created_at
        ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING)
        AS Last_Order_Status
FROM Orders o;
```

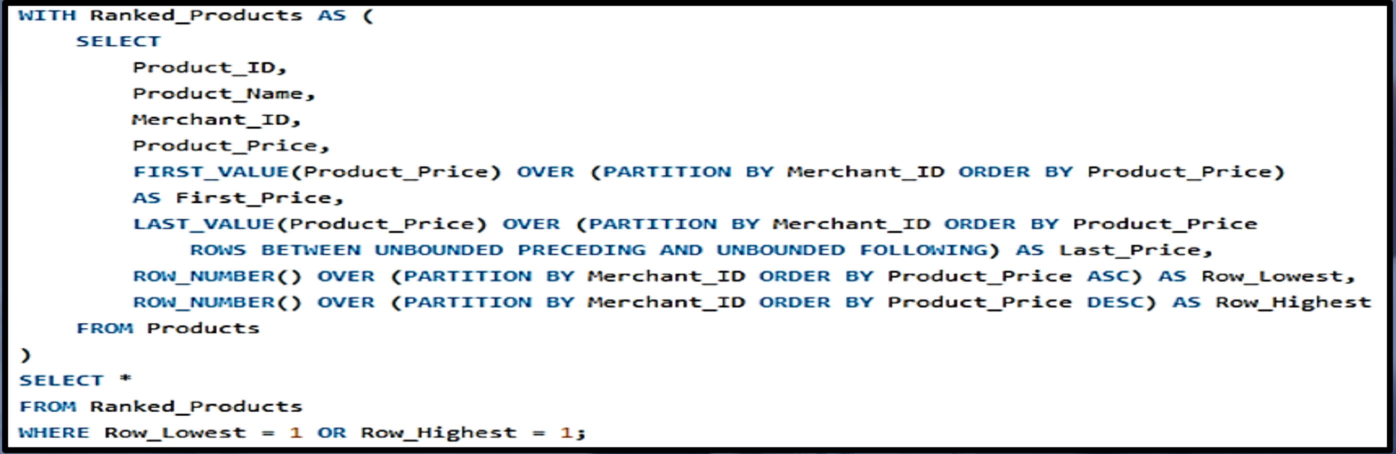
Output



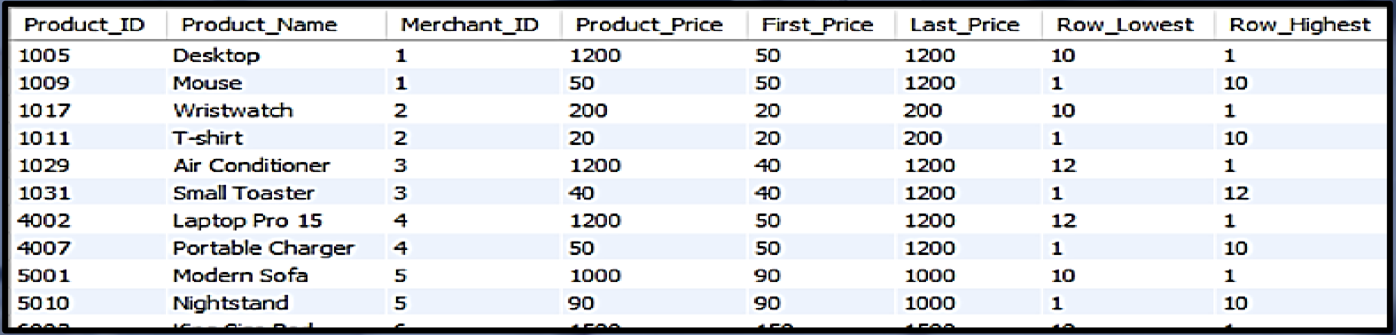
User_ID	First_Order_Status	Last_Order_Status
1	Pending	Returned
1	Pending	Returned
1	Pending	Returned
1	Pending	Returned
1	Pending	Returned
1	Pending	Returned
1	Pending	Returned
2	Cancelled	Shipped
2	Cancelled	Shipped
2	Cancelled	Shipped
2	Cancelled	Shipped
2	Cancelled	Shipped
2	Cancelled	Shipped

- **The goal is to analyze the transition of order statuses over time for individual users.For lower-ranking, affordable products, merchants can target price-sensitive customers and optimize for volume sales.**

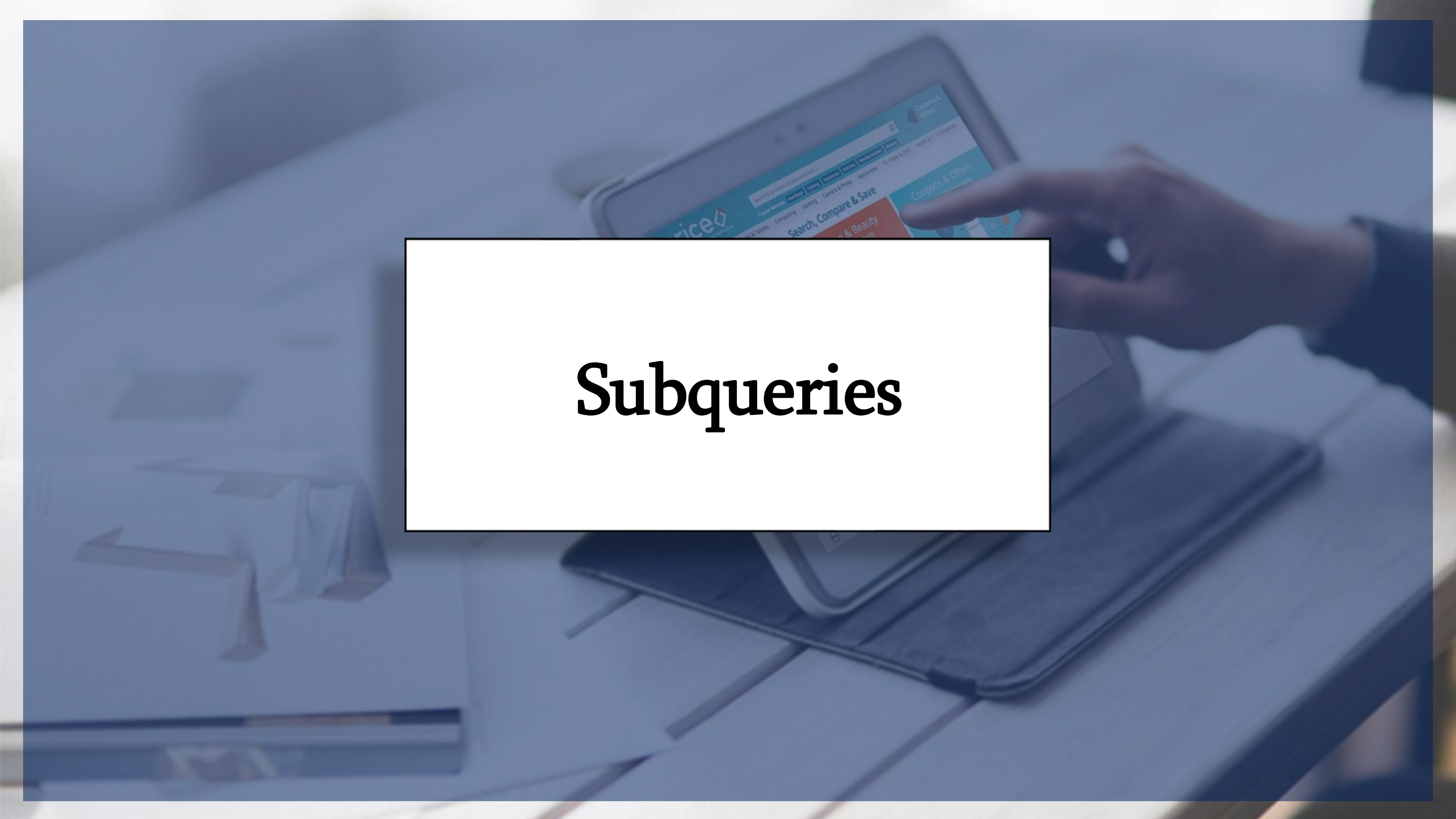
4. Find the FIRST_VALUE() and LAST_VALUE() of prices in each product category.



```
WITH Ranked_Products AS (
    SELECT
        Product_ID,
        Product_Name,
        Merchant_ID,
        Product_Price,
        FIRST_VALUE(Product_Price) OVER (PARTITION BY Merchant_ID ORDER BY Product_Price)
        AS First_Price,
        LAST_VALUE(Product_Price) OVER (PARTITION BY Merchant_ID ORDER BY Product_Price
            ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS Last_Price,
        ROW_NUMBER() OVER (PARTITION BY Merchant_ID ORDER BY Product_Price ASC) AS Row_Lowest,
        ROW_NUMBER() OVER (PARTITION BY Merchant_ID ORDER BY Product_Price DESC) AS Row_Highest
    FROM Products
)
SELECT *
FROM Ranked_Products
WHERE Row_Lowest = 1 OR Row_Highest = 1;
```



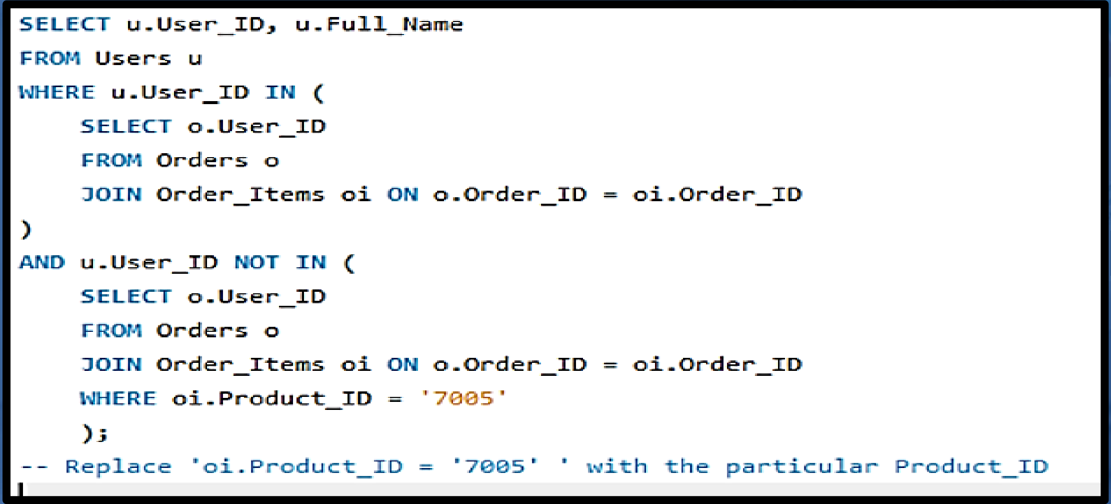
Product_ID	Product_Name	Merchant_ID	Product_Price	First_Price	Last_Price	Row_Lowest	Row_Highest
1005	Desktop	1	1200	50	1200	10	1
1009	Mouse	1	50	50	1200	1	10
1017	Wristwatch	2	200	20	200	10	1
1011	T-shirt	2	20	20	200	1	10
1029	Air Conditioner	3	1200	40	1200	12	1
1031	Small Toaster	3	40	40	1200	1	12
4002	Laptop Pro 15	4	1200	50	1200	12	1
4007	Portable Charger	4	50	50	1200	1	10
5001	Modern Sofa	5	1000	90	1000	10	1
5010	Nightstand	5	90	90	1000	1	10



Subqueries

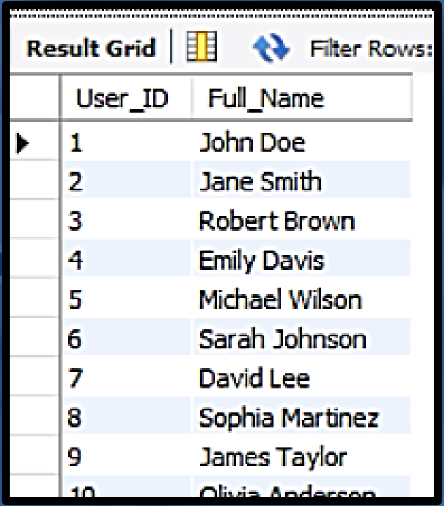
1. Write a subquery to find the users who have placed orders but have not ordered a particular product.

MySQL Code



```sql
SELECT u.User_ID, u.Full_Name
FROM Users u
WHERE u.User_ID IN (
    SELECT o.User_ID
    FROM Orders o
    JOIN Order_Items oi ON o.Order_ID = oi.Order_ID
)
AND u.User_ID NOT IN (
    SELECT o.User_ID
    FROM Orders o
    JOIN Order_Items oi ON o.Order_ID = oi.Order_ID
    WHERE oi.Product_ID = '7005'
    );
-- Replace 'oi.Product_ID = '7005' ' with the particular Product_ID
```

Output

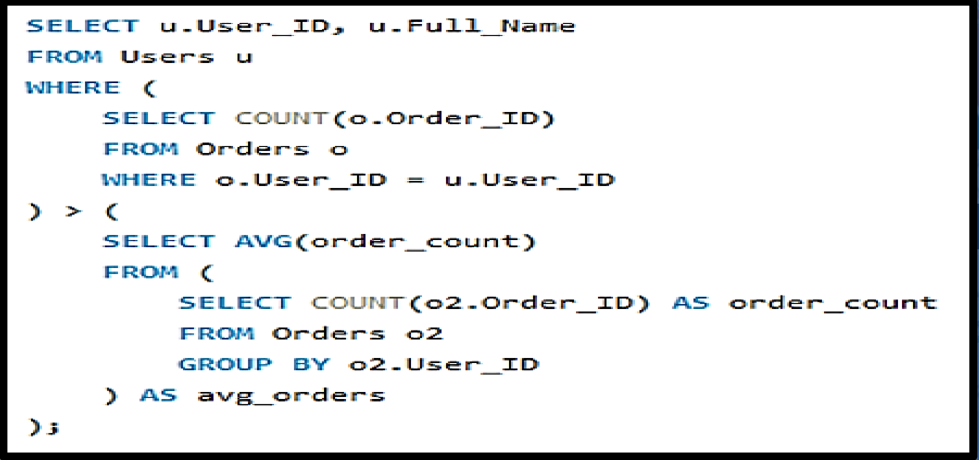


```
User_ID   Full_Name
1         John Doe
2         Jane Smith
3         Robert Brown
4         Emily Davis
5         Michael Wilson
6         Sarah Johnson
7         David Lee
8         Sophia Martinez
9         James Taylor
10        Olivia Anderson
```

- **Businesses can target these users for specific marketing efforts, such as promotions or recommendations.**
- **This insight allows businesses to improve communication or reposition the product to attract this segment.**

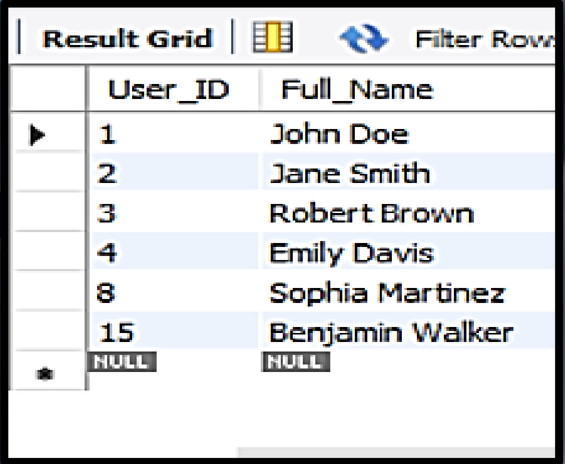
2. Use a correlated subquery to find users who have placed more orders than the average number of orders. .

MySQL Code



```
SELECT u.User_ID, u.Full_Name
FROM Users u
WHERE (
    SELECT COUNT(o.Order_ID)
    FROM Orders o
    WHERE o.User_ID = u.User_ID
) > (
    SELECT AVG(order_count)
    FROM (
        SELECT COUNT(o2.Order_ID) AS order_count
        FROM Orders o2
        GROUP BY o2.User_ID
    ) AS avg_orders
);
```

Output



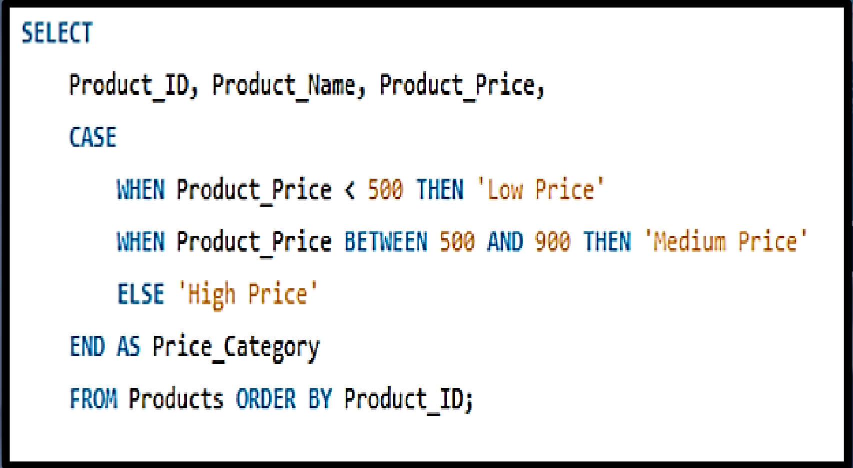
```
   User_ID   Full_Name
▶  1         John Doe
   2         Jane Smith
   3         Robert Brown
   4         Emily Davis
   8         Sophia Martinez
   15        Benjamin Walker
   NULL      NULL
```

- **This query helps businesses identify high-value customers who are placing more orders than the average user.For lower-ranking, affordable products, merchants can target price-sensitive customers and optimize for volume sales.**
- **Businesses can design retention strategies that keep these valuable customers**

Case Statements

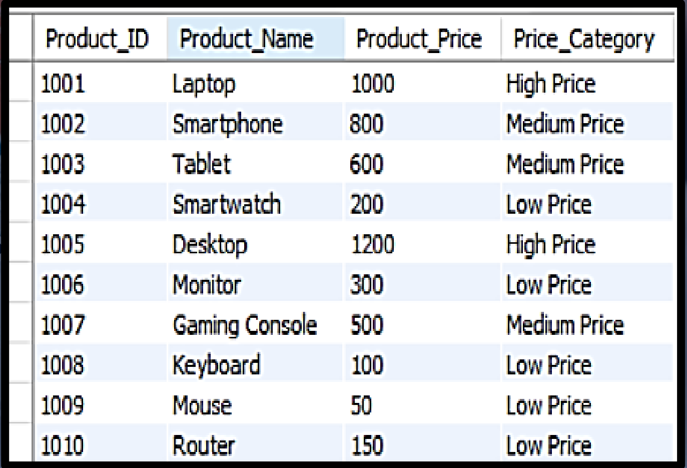
1. Write a query to categorize products into 'Low Price', 'Medium Price', and 'High Price' based on their price.

MySQL Code



```sql
SELECT
    Product_ID, Product_Name, Product_Price,
    CASE
        WHEN Product_Price < 500 THEN 'Low Price'
        WHEN Product_Price BETWEEN 500 AND 900 THEN 'Medium Price'
        ELSE 'High Price'
    END AS Price_Category
    FROM Products ORDER BY Product_ID;
```

Output



```
Product_ID  Product_Name    Product_Price  Price_Category
1001        Laptop          1000           High Price
1002        Smartphone      800            Medium Price
1003        Tablet          600            Medium Price
1004        Smartwatch      200            Low Price
1005        Desktop         1200           High Price
1006        Monitor         300            Low Price
1007        Gaming Console  500            Medium Price
1008        Keyboard        100            Low Price
1009        Mouse           50             Low Price
1010        Router          150            Low Price
```

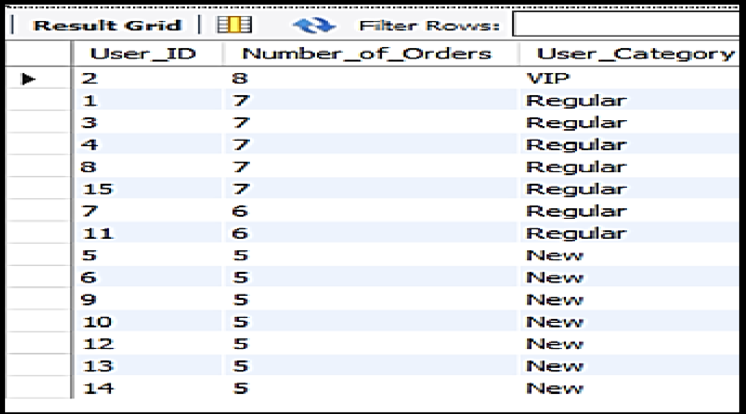
- **By segmenting products into price categories, businesses can better understand their pricing structure and identify how their products align with market expectations..**
- **This segmentation aids in personalized marketing and promotions.**

2. Create a case statement that categorizes users as 'New', 'Regular', or 'VIP' based on the number of orders they have placed.

MySQL Code

```sql
SELECT User_ID, COUNT(Order_ID) AS Number_of_Orders,
    CASE
        WHEN COUNT(Order_ID) = 5 THEN 'New'
        WHEN COUNT(Order_ID) BETWEEN 5 AND 7 THEN 'Regular'
        ELSE 'VIP'
    END AS User_Category
 FROM Orders
GROUP BY User_ID
ORDER BY Number_of_Orders DESC;
```

Output



Result Grid | Filter Rows:

```
  User_ID   Number_of_Orders   User_Category
▶ 2         8                  VIP
  1         7                  Regular
  3         7                  Regular
  4         7                  Regular
  8         7                  Regular
  15        7                  Regular
  7         6                  Regular
  11        6                  Regular
  5         5                  New
  6         5                  New
  9         5                  New
  10        5                  New
  12        5                  New
  13        5                  New
  14        5                  New
```

- **Businesses can use this insight to track the progression of customers through different categories, ensuring effective strategies to move 'New' customers to 'Regular', and 'Regular' to 'VIP'.**

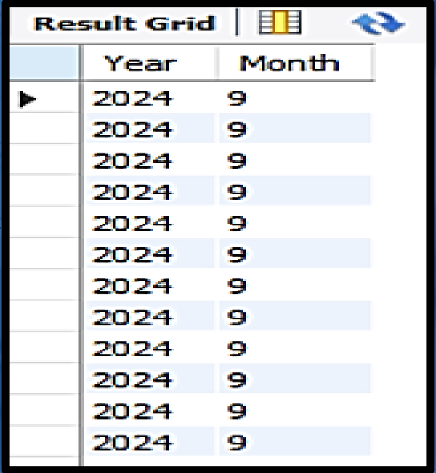
Time-Related Operations

1 . Extract the year and month from the created_at date of orders.

MySQL Code

```
SELECT
    EXTRACT(YEAR FROM created_at) AS Year,
    EXTRACT(MONTH FROM created_at) AS Month
FROM Orders;
```

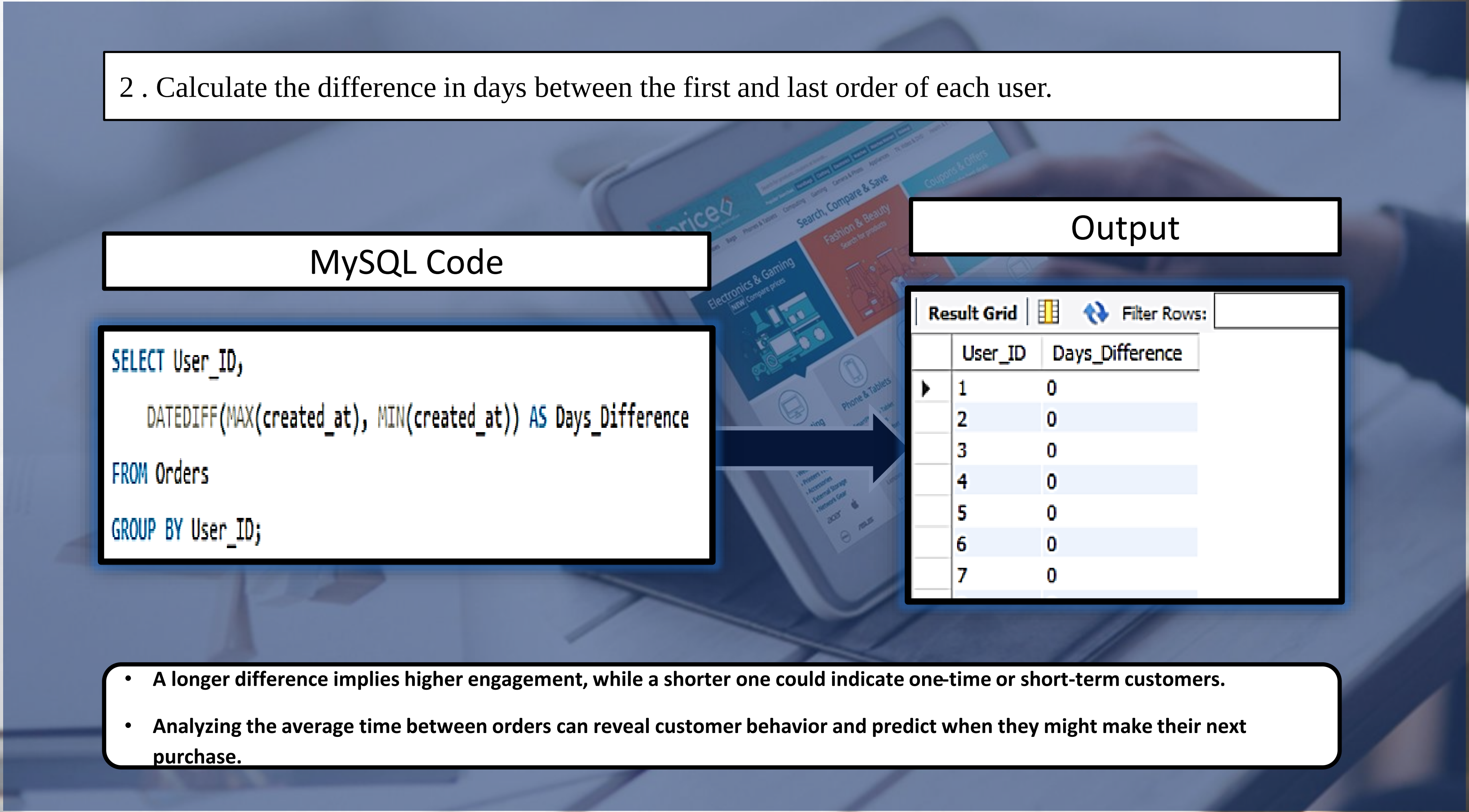
Output



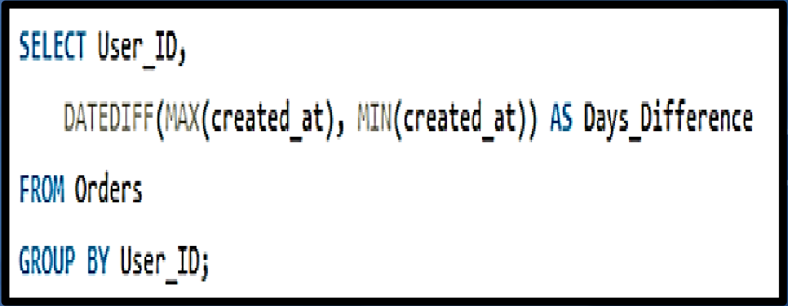
Year	Month
2024	9
2024	9
2024	9
2024	9
2024	9
2024	9
2024	9
2024	9
2024	9
2024	9
2024	9
2024	9

- **Businesses can conduct monthly and yearly trend analysis. This allows for tracking seasonal patterns, spikes, or declines in ordersFor lower-ranking, affordable products, merchants can target price-sensitive customers and optimize for volume sales.**

2 . Calculate the difference in days between the first and last order of each user.

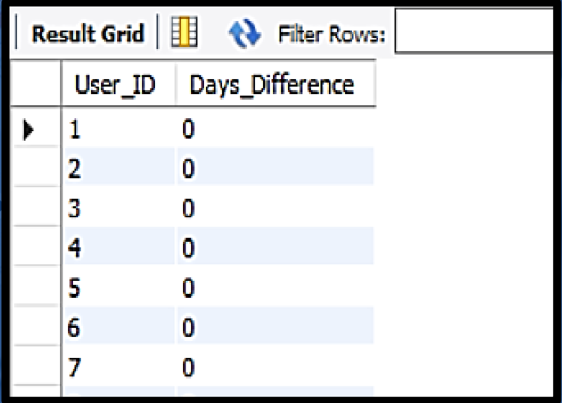


MySQL Code



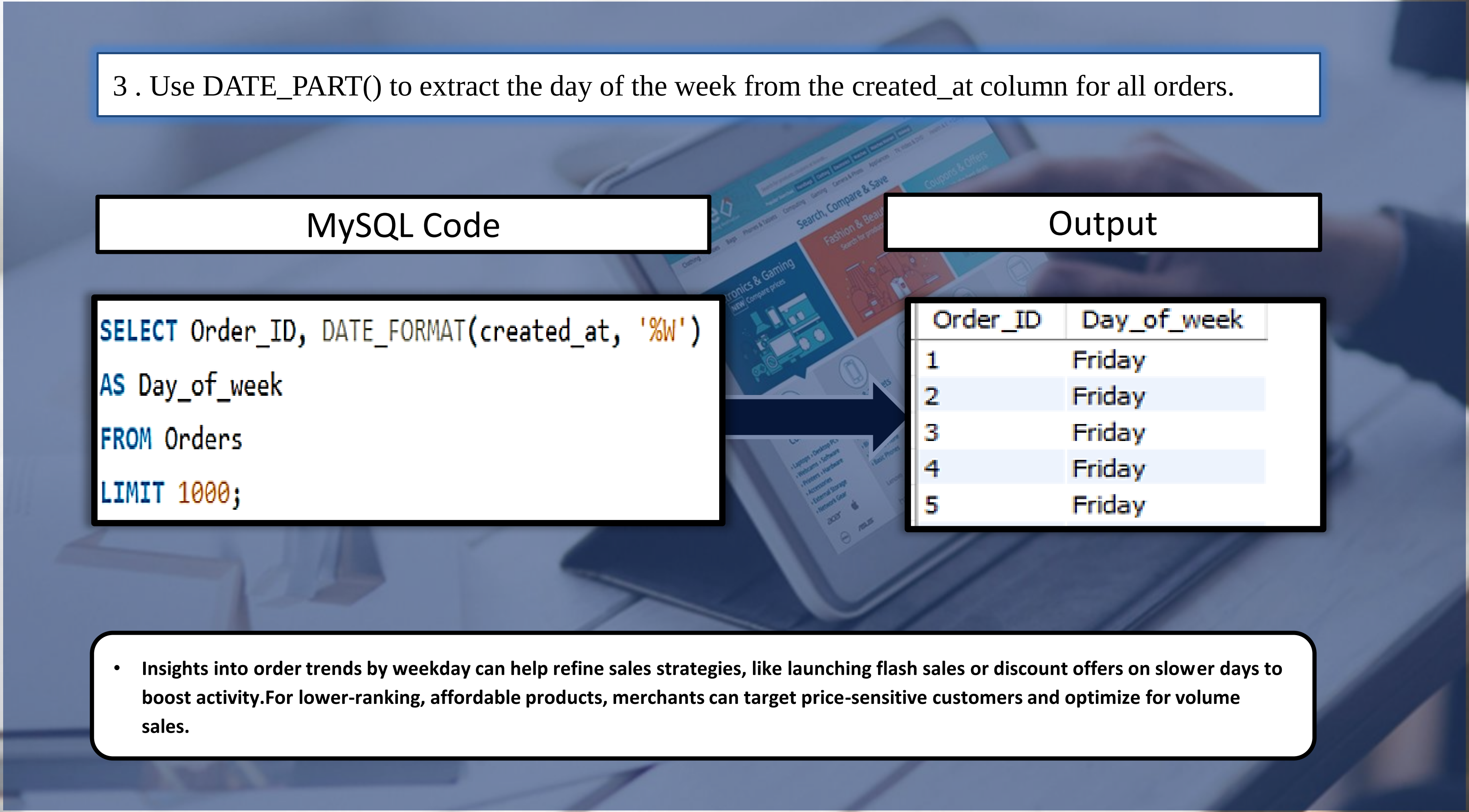
```
SELECT User_ID,
    DATEDIFF(MAX(created_at), MIN(created_at)) AS Days_Difference
FROM Orders
GROUP BY User_ID;
```

Output



```
  User_ID   Days_Difference
▶ 1         0
  2         0
  3         0
  4         0
  5         0
  6         0
  7         0
```

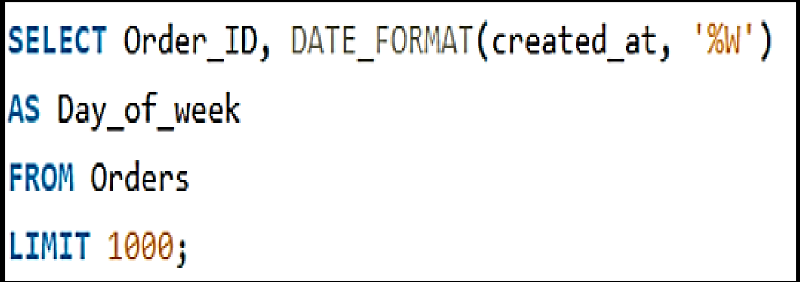
- **A longer difference implies higher engagement, while a shorter one could indicate one-time or short-term customers.**
- **Analyzing the average time between orders can reveal customer behavior and predict when they might make their next purchase.**



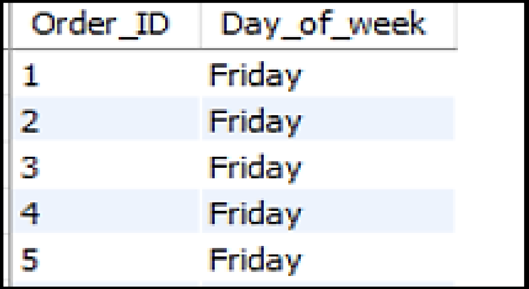
3 . Use DATE_PART() to extract the day of the week from the created_at column for all orders.

MySQL Code

Output

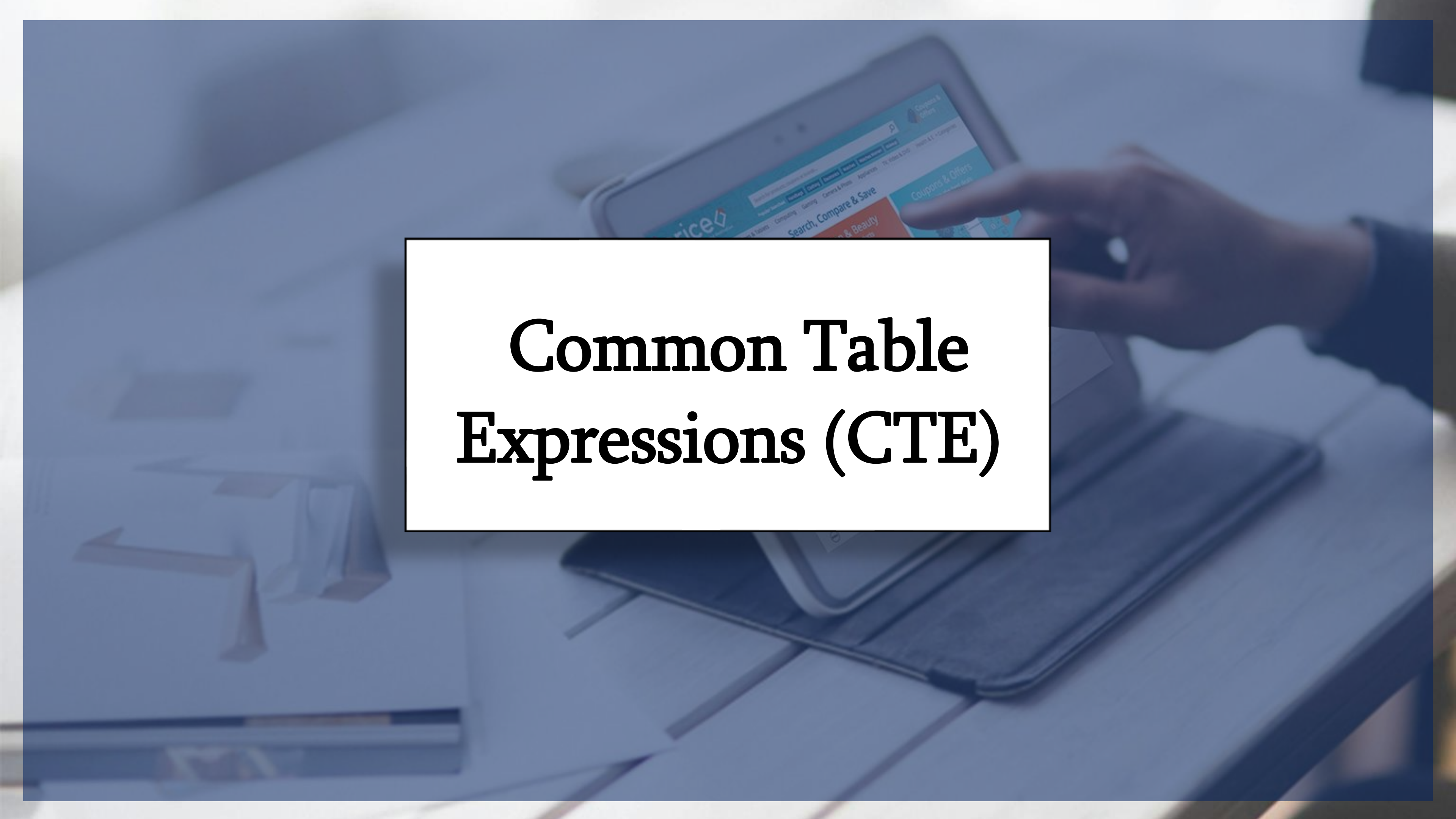


```
SELECT Order_ID, DATE_FORMAT(created_at, '%W')
AS Day_of_week
FROM Orders
LIMIT 1000;
```



```
Order_ID   Day_of_week
1          Friday
2          Friday
3          Friday
4          Friday
5          Friday
```

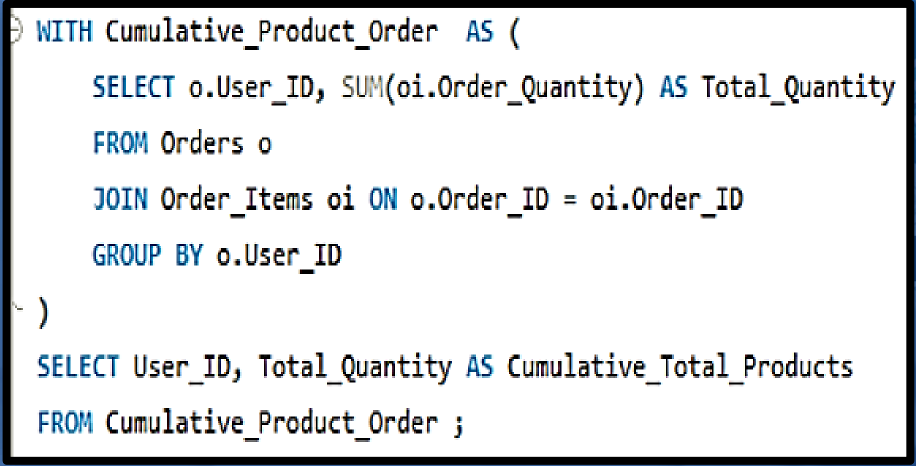
- **Insights into order trends by weekday can help refine sales strategies, like launching flash sales or discount offers on slower days to boost activity.For lower-ranking, affordable products, merchants can target price-sensitive customers and optimize for volume sales.**



Common Table Expressions (CTE)

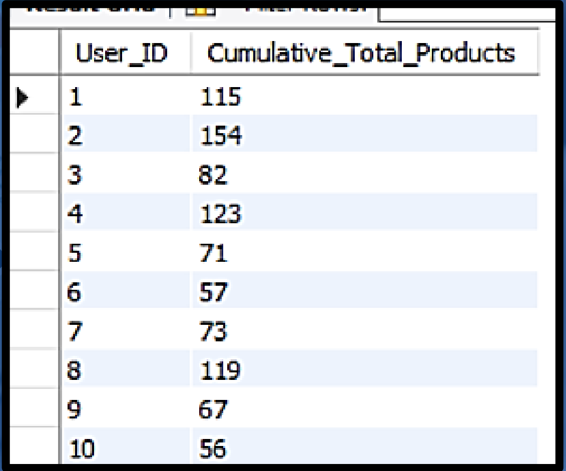
1 . Write a CTE to calculate the cumulative total of products ordered by each user

MySQL Code



```
WITH Cumulative_Product_Order  AS (
    SELECT o.User_ID, SUM(oi.Order_Quantity) AS Total_Quantity
    FROM Orders o
    JOIN Order_Items oi ON o.Order_ID = oi.Order_ID
    GROUP BY o.User_ID
)
SELECT User_ID, Total_Quantity AS Cumulative_Total_Products
FROM Cumulative_Product_Order ;
```

Output

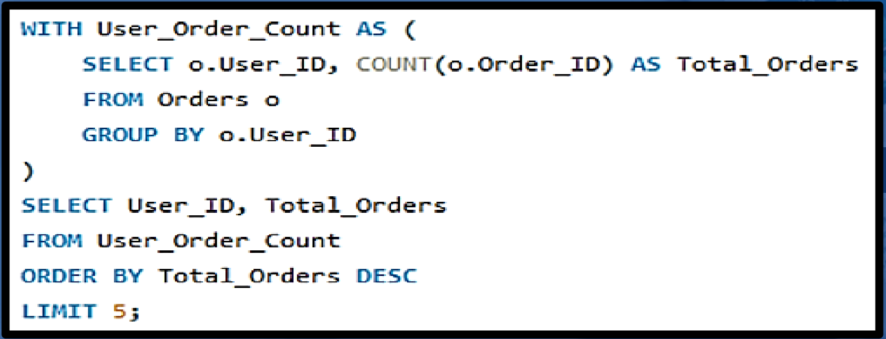


User_ID	Cumulative_Total_Products
1	115
2	154
3	82
4	123
5	71
6	57
7	73
8	119
9	67
10	56

- **Analyzing cumulative orders can help design effective loyalty programs. Users with high totals may be targeted for exclusive rewards or recognition, fostering customer retention.**

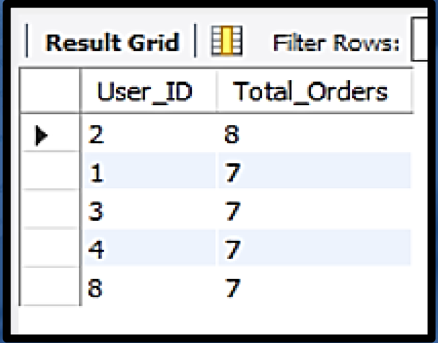
2 . Use a CTE to find the top 5 users with the highest number of orders.

MySQL Code



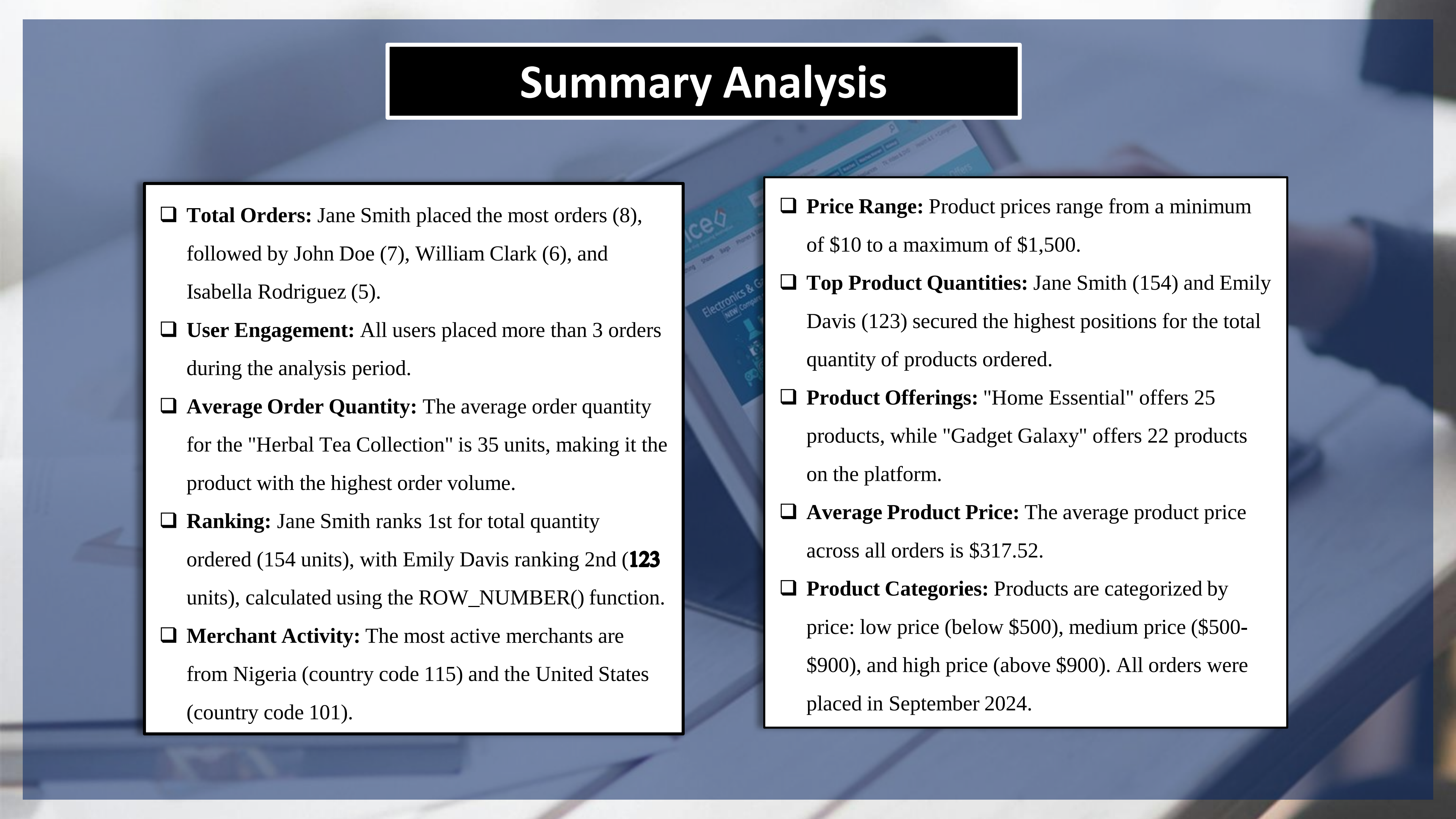
```
WITH User_Order_Count AS (
    SELECT o.User_ID, COUNT(o.Order_ID) AS Total_Orders
    FROM Orders o
    GROUP BY o.User_ID
)
SELECT User_ID, Total_Orders
FROM User_Order_Count
ORDER BY Total_Orders DESC
LIMIT 5;
```

Output



```
User_ID  Total_Orders
2        8
1        7
3        7
4        7
8        7
```

- Top users helps recognize high-value customers who significantly contribute to revenue.For lower-ranking, affordable products, merchants can target price-sensitive customers and optimize for volume sales.
- Analyzing order counts can serve as a metric for evaluating sales team performance



Summary Analysis

- **Total Orders:** Jane Smith placed the most orders (8), followed by John Doe (7), William Clark (6), and Isabella Rodriguez (5).
- **User Engagement:** All users placed more than 3 orders during the analysis period.
- **Average Order Quantity:** The average order quantity for the "Herbal Tea Collection" is 35 units, making it the product with the highest order volume.
- **Ranking:** Jane Smith ranks 1st for total quantity ordered (154 units), with Emily Davis ranking 2nd (**123** units), calculated using the ROW_NUMBER() function.
- **Merchant Activity:** The most active merchants are from Nigeria (country code 115) and the United States (country code 101).

- **Price Range:** Product prices range from a minimum of $10 to a maximum of $1,500.
- **Top Product Quantities:** Jane Smith (154) and Emily Davis (123) secured the highest positions for the total quantity of products ordered.
- **Product Offerings:** "Home Essential" offers 25 products, while "Gadget Galaxy" offers 22 products on the platform.
- **Average Product Price:** The average product price across all orders is $317.52.
- **Product Categories:** Products are categorized by price: low price (below $500), medium price ($500-$900), and high price (above $900). All orders were placed in September 2024.