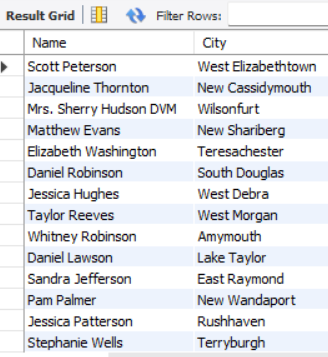
**E-Commerce Store SQL Quries And Its Output:-**

**-- 1. Retrieve all customer names and their cities who registered in the year 2023.**

SELECT Name, City

FROM Customers

WHERE YEAR(RegistrationDate) = 2023;

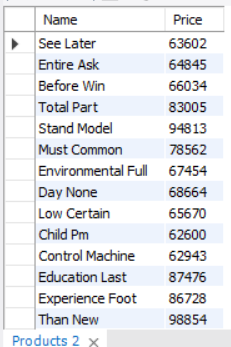


**-- 2. List all products in the "Electronics" category with a price above ₹50,000.**

SELECT Name, Price

FROM Products

WHERE Category = 'Electronics' AND Price > 50000;

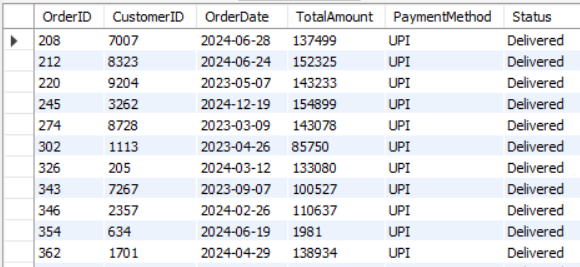


**-- 3. Find all orders that were paid using "UPI" and have a status of "Delivered."**

SELECT \*

FROM Orders

WHERE PaymentMethod = 'UPI' AND Status = 'Delivered';

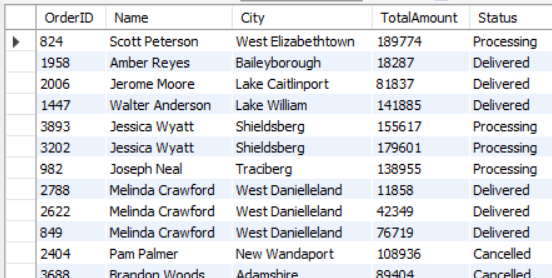


**-- 4. Write a query to fetch all orders along with the customer name and city.**

SELECT o.OrderID, c.Name, c.City, o.TotalAmount, o.Status

FROM Orders o

JOIN Customers c ON o.CustomerID = c.CustomerID;



**-- 5. Retrieve the product name, category, and the corresponding order date for each product ordered.**

SELECT p.Name AS ProductName, p.Category, o.OrderDate

FROM OrderDetails od

JOIN Products p ON od.ProductID = p.ProductID

JOIN Orders o ON od.OrderID = o.OrderID;



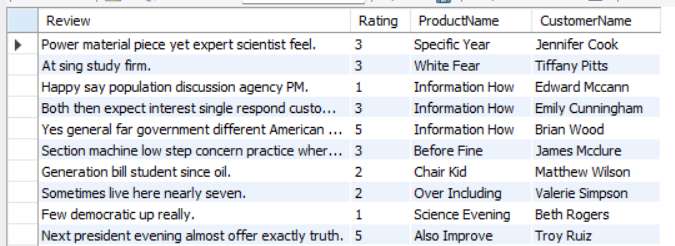
**-- 6. Find all reviews along with the product name and customer name.**

SELECT r.Review, r.Rating, p.Name AS ProductName, c.Name AS CustomerName

FROM Reviews r

JOIN Products p ON r.ProductID = p.ProductID

JOIN Customers c ON r.CustomerID = c.CustomerID;



**-- 7. Find the total number of products sold (use SUM) and the total revenue from the OrderDetails table.**

SELECT SUM(Quantity) AS TotalProductsSold, SUM(Price) AS TotalRevenue

FROM OrderDetails;

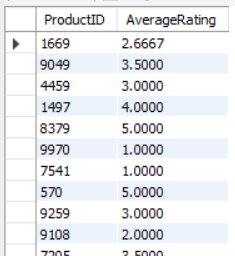


**-- 8. Calculate the average rating for each product.**

SELECT ProductID, AVG(Rating) AS AverageRating

FROM Reviews

GROUP BY ProductID;



**-- 9. Determine which city has the highest number of registered customers.**

SELECT City, COUNT(\*) AS CustomerCount

FROM Customers

GROUP BY City

ORDER BY CustomerCount DESC

LIMIT 1;



**-- 10. List all customers who have placed orders with a total amount greater than ₹1,00,000.**

SELECT Name

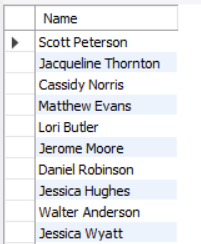
FROM Customers

WHERE CustomerID IN (

SELECT CustomerID

FROM Orders

WHERE TotalAmount > 100000);



**-- 11. Find the product with the highest number of reviews.**

SELECT p.ProductID, p.Name, p.Category,

(SELECT COUNT(\*)

FROM Reviews r

WHERE r.ProductID = p.ProductID) AS ReviewCount

FROM Products p

WHERE p.ProductID = (

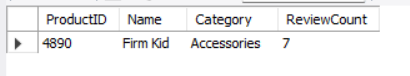
SELECT r.ProductID

FROM Reviews r

GROUP BY r.ProductID

ORDER BY COUNT(\*) DESC

LIMIT 1);



**-- 12. Retrieve all orders that include products from the "Footwear" category.**

SELECT o.\*

FROM Orders o

WHERE o.OrderID IN (

SELECT od.OrderID

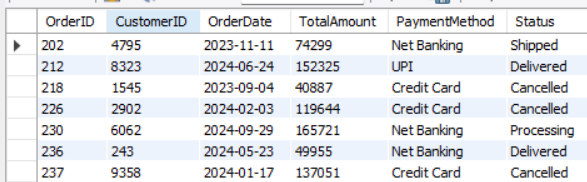
FROM OrderDetails od

WHERE od.ProductID IN (

SELECT p.ProductID

FROM Products p

WHERE p.Category = 'Footwear' ));



**-- 13. find the top 5 products with the highest average ratings.**

WITH AvgRatings AS (

SELECT ProductID, AVG(Rating) AS AverageRating

FROM Reviews

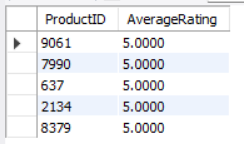
GROUP BY ProductID)

SELECT ProductID, AverageRating

FROM AvgRatings

ORDER BY AverageRating DESC

LIMIT 5;



**-- 14. calculate the total revenue generated by each customer, and then display customers with revenue above ₹2,00,000.**

WITH CustomerRevenue AS (

SELECT CustomerID, SUM(TotalAmount) AS TotalRevenue

FROM Orders

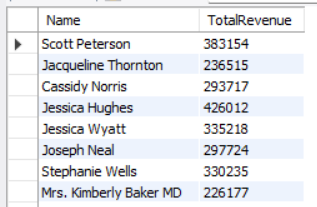
GROUP BY CustomerID)

SELECT c.Name, cr.TotalRevenue

FROM CustomerRevenue cr

JOIN Customers c ON cr.CustomerID = c.CustomerID

WHERE cr.TotalRevenue > 200000;



**-- 15. Write a query using a CTE to list all orders along with the total quantity of products ordered in each.**

WITH OrderQuantities AS (

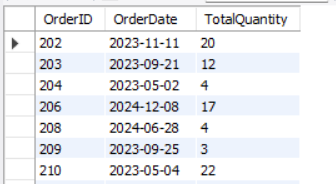
SELECT OrderID, SUM(Quantity) AS TotalQuantity

FROM OrderDetails

GROUP BY OrderID)

SELECT o.OrderID, o.OrderDate, oq.TotalQuantity

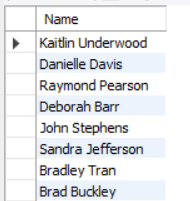
FROM Orders o JOIN OrderQuantities oq ON o.OrderID = oq.OrderID;



**-- 16. Write a query to list all customers who have not placed any orders (use a LEFT JOIN).**

SELECT c.Name FROM Customers c

LEFT JOIN Orders o ON c.CustomerID = o.CustomerID

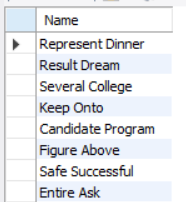


**-- 17. Find all products that have never been reviewed (use a RIGHT JOIN).**

SELECT p.Name FROM Products p

LEFT JOIN Reviews r ON p.ProductID = r.ProductID

WHERE r.ReviewID IS NULL;

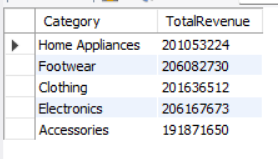


**--18. Retrieve the total revenue for each product category by joining the Products and OrderDetails**

SELECT p.Category, SUM(od.Price) AS TotalRevenue FROM OrderDetails od

JOIN Products p ON od.ProductID = p.ProductID

GROUP BY p.Category;



**-- 19. Find the second-highest total revenue among all customers.**

SELECT MAX(TotalRevenue) AS SecondHighestRevenue

FROM (

SELECT SUM(TotalAmount) AS TotalRevenue

FROM Orders

GROUP BY CustomerID

) AS CustomerRevenues

WHERE TotalRevenue < (

SELECT MAX(TotalRevenue)

FROM (

SELECT SUM(TotalAmount) AS TotalRevenue

FROM Orders

GROUP BY CustomerID

) AS AllRevenues

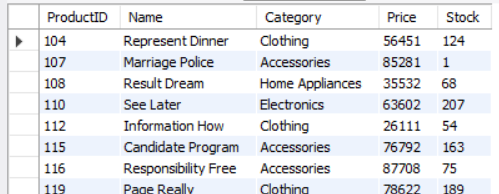
);



**-- 20. Retrieve all products whose stock is less than the average stock across all products.**

SELECT \* FROM Products

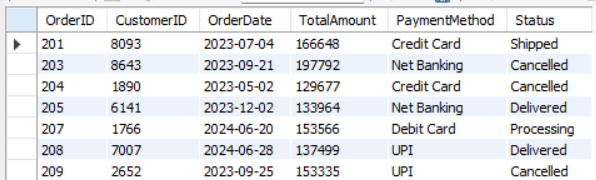
WHERE Stock < (SELECT AVG(Stock) FROM Products);



**-- 21. List all orders where the total amount is greater than the average order amount.**

SELECT \* FROM Orders

WHERE TotalAmount > (SELECT AVG(TotalAmount) FROM Orders);



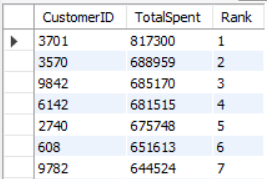
**-- 22. Use a window function to rank customers based on the total amount spent on orders.**

SELECT CustomerID, SUM(TotalAmount) AS TotalSpent,

RANK() OVER (ORDER BY SUM(TotalAmount) DESC) AS `Rank`

FROM Orders

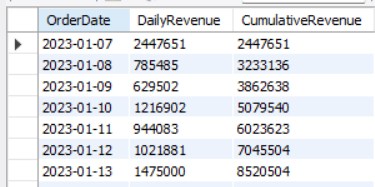
GROUP BY CustomerID;



**-- 23. Calculate the cumulative revenue generated by all orders, ordered by date.**

SELECT OrderDate, SUM(TotalAmount) AS DailyRevenue, SUM(SUM(TotalAmount)) OVER (ORDER BY OrderDate) AS CumulativeRevenue FROM Orders

GROUP BY OrderDate;



**-- 24. Find the product with the highest price in each category (use PARTITION BY).**

SELECT ProductID, Category, Price

FROM (

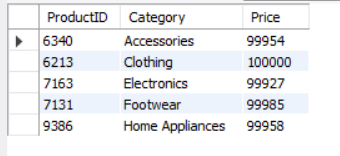
SELECT ProductID, Category, Price,

RANK() OVER (PARTITION BY Category ORDER BY Price DESC) AS `Rank`

FROM Products

) AS RankedProducts

WHERE `Rank` = 1;



**-- 25. Write a query to categorize products as "High Stock" (above 100 units) or "Low Stock" (100 units or less).**

SELECT ProductID, Name, Stock,

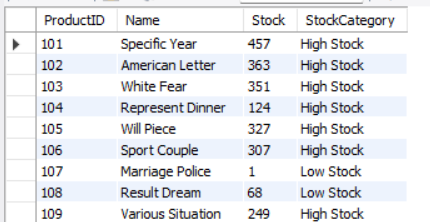
CASE

WHEN Stock > 100 THEN 'High Stock'

ELSE 'Low Stock'

END AS StockCategory

FROM Products;



**-- 26. Create a query to display customer names and a label indicating if they are "New" (registered in the last 6 months) or "Old" (registered earlier).**

SELECT Name,

RegistrationDate,

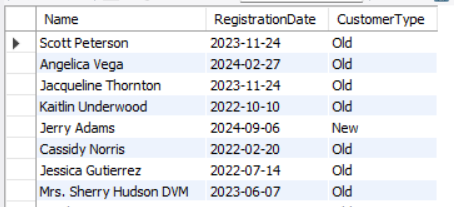
CASE

WHEN RegistrationDate >= DATE\_SUB(CURDATE(), INTERVAL 6 MONTH) THEN 'New'

ELSE 'Old'

END AS CustomerType

FROM Customers;



**-- 27. Find the customer who placed the most orders and calculate the total amount of all their orders.**

SELECT o.CustomerID, c.Name, COUNT(o.OrderID) AS TotalOrders, SUM(o.TotalAmount) AS TotalSpent

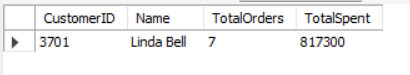
FROM Orders o

JOIN Customers c ON o.CustomerID = c.CustomerID

GROUP BY o.CustomerID, c.Name

ORDER BY TotalOrders DESC

LIMIT 1;



**-- 28. Write a query to find the top 3 most reviewed products in each category.**

WITH ProductReviewCounts AS (

SELECT p.ProductID, p.Name, p.Category, COUNT(r.ReviewID) AS ReviewCount

FROM Products p

JOIN Reviews r ON p.ProductID = r.ProductID

GROUP BY p.ProductID, p.Name, p.Category),

RankedProducts AS (

SELECT ProductID, Name, Category, ReviewCount,

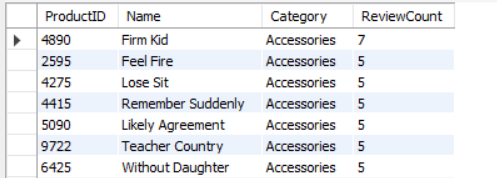
RANK() OVER (PARTITION BY Category ORDER BY ReviewCount DESC) AS `Rank`

FROM ProductReviewCounts)

SELECT ProductID, Name, Category, ReviewCount

FROM RankedProducts

WHERE `Rank` <= 3;



**-- 29. Identify customers who have purchased products from at least 3 different categories.**

SELECT o.CustomerID, c.Name, COUNT(DISTINCT p.Category) AS CategoriesPurchased

FROM Orders o

JOIN OrderDetails od ON o.OrderID = od.OrderID

JOIN Products p ON od.ProductID = p.ProductID

JOIN Customers c ON o.CustomerID = c.CustomerID

GROUP BY o.CustomerID, c.Name

HAVING COUNT(DISTINCT p.Category) >= 3; 