**Instructions:**

* Answer all questions using **MySQL**.
* Use appropriate **subqueries**, **joins**, and **aggregate functions** wherever applicable.
* Make sure to use proper **aliasing**, **GROUP BY**, **HAVING**, **DISTINCT**, etc., as needed.
* Data

-- Customers Table

CREATE TABLE Customers (

CustomerID INT PRIMARY KEY,

Name VARCHAR(100),

City VARCHAR(100)

);

INSERT INTO Customers VALUES

(1, 'John Smith', 'New York'),

(2, 'Jane Doe', 'Delhi'),

(3, 'Robert Johnson', 'London'),

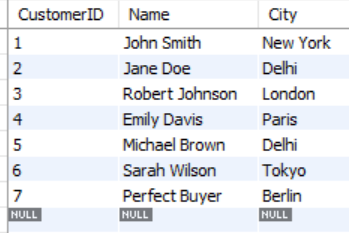
(4, 'Emily Davis', 'Paris'),

(5, 'Michael Brown', 'Delhi'),

(6, 'Sarah Wilson', 'Tokyo');

INSERT INTO Customers VALUES

(7, 'Perfect Buyer', 'Berlin');



-- Orders Table

CREATE TABLE Orders (

OrderID INT PRIMARY KEY,

CustomerID INT,

OrderDate DATE,

Amount DECIMAL(10,2),

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

INSERT INTO Orders VALUES

(1001, 1, '2023-01-15', 1200.00),

(1002, 2, '2023-01-16', 800.00),

(1003, 3, '2023-02-05', 150.00),

(1004, 1, '2023-02-10', 500.00),

(1005, 4, '2023-03-12', 300.00),

(1006, 2, '2023-03-15', 950.00),

(1007, 5, '2023-04-01', 200.00),

(1008, 3, '2023-04-05', 450.00),

(1009, 1, '2023-05-20', 1200.00),

(1010, 6, '2023-05-25', 800.00),

(1011, 2, '2023-06-10', 150.00),

(1012, 4, '2023-06-15', 500.00),

(1013, 5, '2023-07-03', 300.00),

(1014, 3, '2023-07-08', 950.00),

(1015, 1, '2023-08-12', 200.00);

Perfect buyer data:

INSERT INTO Orders VALUES

(1016, 7, '2023-01-10', 100.00),

(1017, 7, '2023-02-10', 100.00),

(1018, 7, '2023-03-10', 100.00),

(1019, 7, '2023-04-10', 100.00),

(1020, 7, '2023-05-10', 100.00),

(1021, 7, '2023-06-10', 100.00),

(1022, 7, '2023-07-10', 100.00),

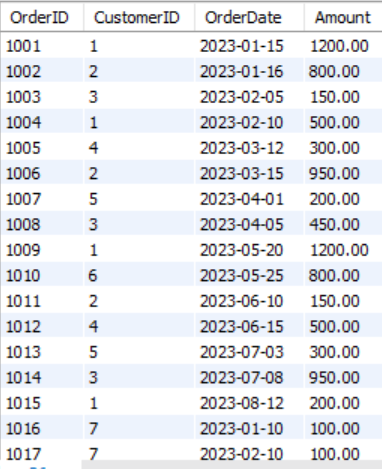
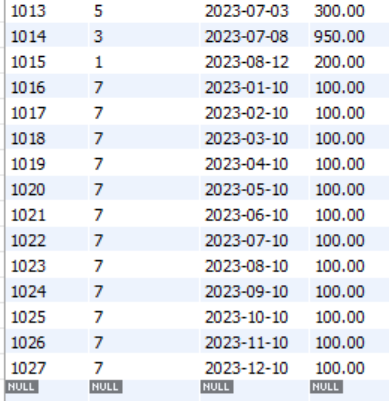
(1023, 7, '2023-08-10', 100.00),

(1024, 7, '2023-09-10', 100.00),

(1025, 7, '2023-10-10', 100.00),

(1026, 7, '2023-11-10', 100.00),

(1027, 7, '2023-12-10', 100.00);

-- Products Table

CREATE TABLE Products (

ProductID INT PRIMARY KEY,

ProductName VARCHAR(100),

Price DECIMAL(10,2)

);

INSERT INTO Products VALUES

(101, 'Laptop', 1200.00),

(102, 'Smartphone', 800.00),

(103, 'Headphones', 150.00),

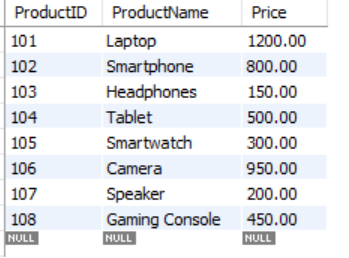
(104, 'Tablet', 500.00),

(105, 'Smartwatch', 300.00),

(106, 'Camera', 950.00),

(107, 'Speaker', 200.00),

(108, 'Gaming Console', 450.00);



-- OrderDetails Table

CREATE TABLE OrderDetails (

OrderDetailID INT PRIMARY KEY,

OrderID INT,

ProductID INT,

Quantity INT,

FOREIGN KEY (OrderID) REFERENCES Orders(OrderID),

FOREIGN KEY (ProductID) REFERENCES Products(ProductID)

);

INSERT INTO OrderDetails VALUES

(1, 1001, 101, 1),

(2, 1002, 102, 1),

(3, 1003, 103, 1),

(4, 1004, 104, 1),

(5, 1005, 105, 1),

(6, 1006, 106, 1),

(7, 1007, 107, 1),

(8, 1008, 108, 1),

(9, 1009, 101, 1),

(10, 1010, 102, 1),

(11, 1011, 103, 1),

(12, 1012, 104, 1),

(13, 1013, 105, 1),

(14, 1014, 106, 1),

(15, 1015, 107, 1),

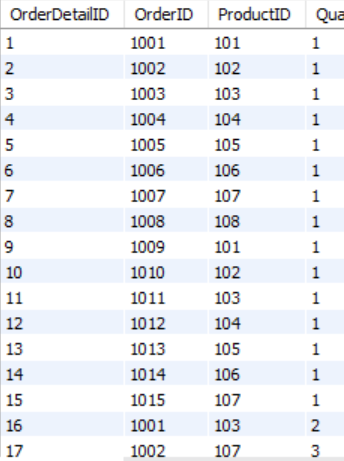
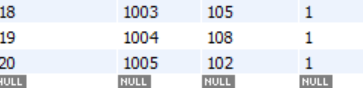
(16, 1001, 103, 2),

(17, 1002, 107, 3),

(18, 1003, 105, 1),

(19, 1004, 108, 1),

(20, 1005, 102, 1);

**Part A – Subqueries (20 marks)**

1. Write a query to find customers who have placed orders in **every month** of the current year.

SELECT c.CustomerID, c.Name

FROM Customers c

WHERE NOT EXISTS (

SELECT 1

FROM (

SELECT 1 AS month\_num UNION SELECT 2 UNION SELECT 3 UNION SELECT 4

UNION SELECT 5 UNION SELECT 6 UNION SELECT 7 UNION SELECT 8

UNION SELECT 9 UNION SELECT 10 UNION SELECT 11 UNION SELECT 12

) months

WHERE NOT EXISTS (

SELECT 1

FROM Orders o

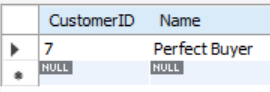
WHERE o.CustomerID = c.CustomerID

AND MONTH(o.OrderDate) = months.month\_num

AND YEAR(o.OrderDate) = 2023

)

);



1. Retrieve the names of products that have been ordered **more than the average quantity** across all products.

SELECT p.ProductName

FROM Products p

WHERE p.ProductID IN (

SELECT od.ProductID

FROM OrderDetails od

GROUP BY od.ProductID

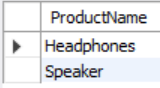
HAVING AVG(od.Quantity) > (

SELECT AVG(Quantity)

FROM OrderDetails

)

);



1. Find customers who have **never ordered a product** priced above ₹1000.

SELECT c.CustomerID, c.Name

FROM Customers c

WHERE NOT EXISTS (

SELECT 1

FROM Orders o

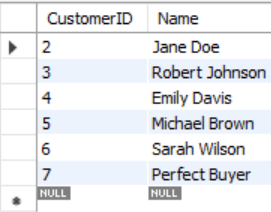
JOIN OrderDetails od ON o.OrderID = od.OrderID

JOIN Products p ON od.ProductID = p.ProductID

WHERE o.CustomerID = c.CustomerID

AND p.Price > 1000

);



1. List the **top 3 products by total revenue** using a subquery.

SELECT p.ProductName,

(SELECT SUM(od.Quantity \* p2.Price)

FROM OrderDetails od

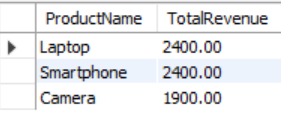
JOIN Products p2 ON od.ProductID = p2.ProductID

WHERE od.ProductID = p.ProductID) AS TotalRevenue

FROM Products p

ORDER BY TotalRevenue DESC

LIMIT 3;



1. Find orders that contain **only one product** using a **correlated subquery**.

SELECT o.OrderID

FROM Orders o

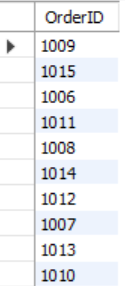
WHERE 1 = (

SELECT COUNT(DISTINCT od.ProductID)

FROM OrderDetails od

WHERE od.OrderID = o.OrderID

);



**Part B – Correlated & Nested Subqueries (25 marks)**

1. Retrieve the names of customers who placed an order on the **same date as 'John'**.

SELECT DISTINCT c.Name

FROM Customers c

JOIN Orders o ON c.CustomerID = o.CustomerID

WHERE o.OrderDate IN (

SELECT o2.OrderDate

FROM Orders o2

JOIN Customers c2 ON o2.CustomerID = c2.CustomerID

WHERE c2.Name = 'John Smith'

)

AND c.Name != 'John Smith';



1. Find the name of the customer who placed the **most recent order**.

SELECT c.Name

FROM Customers c

WHERE c.CustomerID = (

SELECT o.CustomerID

FROM Orders o

ORDER BY o.OrderDate DESC

LIMIT 1

);



1. Write a query to find the product that has the **second lowest price** using a subquery.

SELECT p.ProductName

FROM Products p

WHERE p.Price = (

SELECT MIN(Price)

FROM Products

WHERE Price > (

SELECT MIN(Price)

FROM Products

)

);



1. Display customer names who have spent **more than double the average spending**.

SELECT c.Name, SUM(o.Amount) AS TotalSpent

FROM Customers c

JOIN Orders o ON c.CustomerID = o.CustomerID

GROUP BY c.CustomerID, c.Name

HAVING TotalSpent > 2 \* (

SELECT AVG(OrderTotal)

FROM (

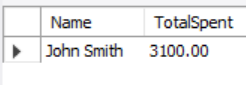
SELECT SUM(Amount) AS OrderTotal

FROM Orders

GROUP BY CustomerID

) AS AvgSpending

);



1. List customers whose **total order amount is more than the total order amount of any customer from 'Delhi'**.

SELECT c.Name, SUM(o.Amount) AS TotalAmount

FROM Customers c

JOIN Orders o ON c.CustomerID = o.CustomerID

GROUP BY c.CustomerID, c.Name

HAVING TotalAmount > ALL (

SELECT SUM(o2.Amount)

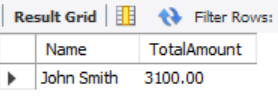
FROM Orders o2

JOIN Customers c2 ON o2.CustomerID = c2.CustomerID

WHERE c2.City = 'Delhi'

GROUP BY o2.CustomerID

);



**Part C – Join + Subquery Mix (30 marks)**

1. Use a correlated subquery to find customers who have placed **more orders than the average** number of orders placed by all customers.

SELECT c.CustomerID, c.Name, COUNT(o.OrderID) AS OrderCount

FROM Customers c

JOIN Orders o ON c.CustomerID = o.CustomerID

GROUP BY c.CustomerID, c.Name

HAVING OrderCount > (

SELECT AVG(OrderCount)

FROM (

SELECT COUNT(OrderID) AS OrderCount

FROM Orders

GROUP BY CustomerID

) AS AvgOrders

);



1. Find all products whose **total sales quantity** is higher than the average total quantity sold per product.

SELECT p.ProductName, SUM(od.Quantity) AS TotalQuantity

FROM Products p

JOIN OrderDetails od ON p.ProductID = od.ProductID

GROUP BY p.ProductID, p.ProductName

HAVING TotalQuantity > (

SELECT AVG(ProductQuantity)

FROM (

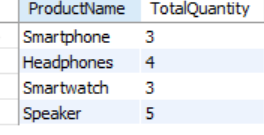
SELECT SUM(Quantity) AS ProductQuantity

FROM OrderDetails

GROUP BY ProductID

) AS AvgQuantities

);



1. Get customers who have ordered at least **one product that no one else has ordered**.

SELECT DISTINCT c.Name

FROM Customers c

JOIN Orders o ON c.CustomerID = o.CustomerID

JOIN OrderDetails od ON o.OrderID = od.OrderID

WHERE od.ProductID IN (

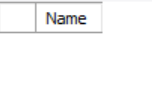
SELECT od2.ProductID

FROM OrderDetails od2

GROUP BY od2.ProductID

HAVING COUNT(DISTINCT od2.OrderID) = 1

);



1. Retrieve all orders where the total order amount is equal to the **maximum order amount for that customer**.

SELECT o.OrderID, o.CustomerID, o.Amount

FROM Orders o

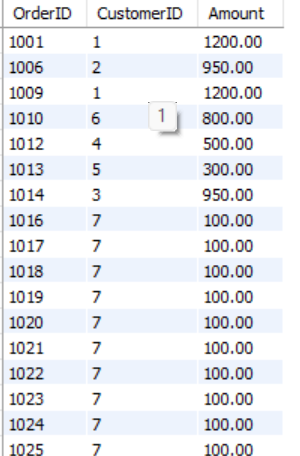
WHERE o.Amount = (

SELECT MAX(Amount)

FROM Orders o2

WHERE o2.CustomerID = o.CustomerID

);



1. Write a query to list customers who have **never placed an order with a quantity greater than 5**.

SELECT c.CustomerID, c.Name

FROM Customers c

WHERE NOT EXISTS (

SELECT 1

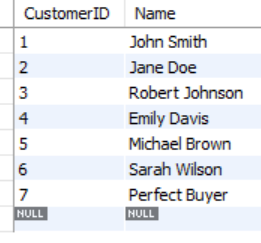
FROM Orders o

JOIN OrderDetails od ON o.OrderID = od.OrderID

WHERE o.CustomerID = c.CustomerID

AND od.Quantity > 5

);



**Part D – Joins & Set Operations (25 marks)**

1. Use a subquery to list the **top 5 customers by total spending**.

SELECT c.CustomerID, c.Name, SUM(o.Amount) AS TotalSpending

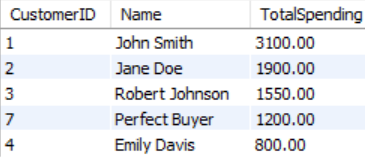
FROM Customers c

JOIN Orders o ON c.CustomerID = o.CustomerID

GROUP BY c.CustomerID, c.Name

ORDER BY TotalSpending DESC

LIMIT 5;



1. Find all customers who have only ordered **one unique product** using subqueries.

SELECT c.CustomerID, c.Name

FROM Customers c

WHERE 1 = (

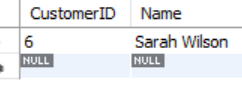
SELECT COUNT(DISTINCT od.ProductID)

FROM Orders o

JOIN OrderDetails od ON o.OrderID = od.OrderID

WHERE o.CustomerID = c.CustomerID

);



1. List all orders where the amount is **not in the top 10 highest order amounts**.

SELECT o.OrderID, o.Amount

FROM Orders o

WHERE o.Amount NOT IN (

SELECT Amount FROM (

SELECT Amount

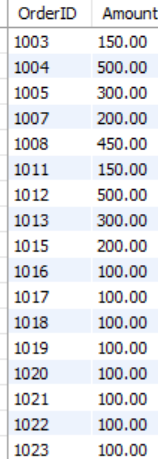
FROM Orders

ORDER BY Amount DESC

LIMIT 5

) AS Top5Amounts

);



1. Retrieve customer names who placed an order in the **last 7 days** but **not** in the **previous 30 days** before that.

SELECT DISTINCT c.Name

FROM Customers c

JOIN Orders o ON c.CustomerID = o.CustomerID

WHERE o.OrderDate BETWEEN DATE\_SUB(CURRENT\_DATE, INTERVAL 7 DAY) AND CURRENT\_DATE

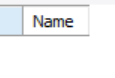
AND c.CustomerID NOT IN (

SELECT DISTINCT o2.CustomerID

FROM Orders o2

WHERE o2.OrderDate BETWEEN DATE\_SUB(CURRENT\_DATE, INTERVAL 37 DAY) AND DATE\_SUB(CURRENT\_DATE, INTERVAL 7 DAY)

);



1. Write a query to list all products ordered in the **highest number of distinct orders**.

SELECT p.ProductName, COUNT(DISTINCT od.OrderID) AS OrderCount

FROM Products p

JOIN OrderDetails od ON p.ProductID = od.ProductID

GROUP BY p.ProductID, p.ProductName

HAVING OrderCount = (

SELECT COUNT(DISTINCT od2.OrderID)

FROM OrderDetails od2

GROUP BY od2.ProductID

ORDER BY COUNT(DISTINCT od2.OrderID) DESC

LIMIT 1

);

