SQL JOINS ASSIGNMENT DPP

Author: Shivansh Yadav
Database: MySQL 8.0+

Problem Statement

A retail company operating across major Indian cities wants to analyze its customer transactions, order activities, payment records, and employee reporting structure. The data is distributed across multiple tables such as Customers, Orders, Payments, and Employees. Using SQL JOIN operations, the goal is to combine these tables to answer business questions related to customer activity, inactive customers, orphan records, payment behavior, and reporting hierarchy.

Dataset Tables

Table: Customers

CustomerID	CustomerName	City
1	Arjun Mehta	Mumbai
2	Priya Sharma	Delhi
3	Soham Mishra	Bengaluru
4	Sneha Kapoor	Pune
5	Karan Singh	Jaipur

Table: Orders

OrderID	CustomerID	OrderDate	Amount
101	1	2024-09-01	4500
102	2	2024-09-05	5200
103	1	2024-09-07	2100
104	3	2024-09-10	8400
105	7	2024-09-12	7600

Table: Payments

PaymentID	CustomerID	PaymentDate	Amount
P001	1	2024-09-02	4500
P002	2	2024-09-06	5200
P003	3	2024-09-11	8400
P004	4	2024-09-15	3000

Table: Employees

EmployeeID	EmployeeName	ManagerID
1	Amit Khanna	None
2	Neha Joshi	1
3	Vivek Rao	1
4	Rahul Das	2
5	Isha Kulkarni	2

Questions and Solutions

Question 1

Retrieve all customers who have placed at least one order.

```
SELECT DISTINCT c.*
FROM Customers c
INNER JOIN Orders o
ON c.CustomerID = o.CustomerID;
```

Explanation: INNER JOIN ensures only customers with matching orders are returned.

Question 2

Retrieve all customers and their orders, including customers who have not placed any orders.

```
SELECT c.*, o.OrderID, o.OrderDate, o.Amount
FROM Customers c
LEFT JOIN Orders o
```

```
ON c.CustomerID = o.CustomerID;
```

Explanation: LEFT JOIN keeps all customers and shows NULL for missing orders.

Question 3

Retrieve all orders and their corresponding customers, including orders placed by unknown customers.

```
SELECT o.*, c.CustomerName
FROM Orders o
LEFT JOIN Customers c
ON o.CustomerID = c.CustomerID;
```

Explanation: LEFT JOIN ensures orders without valid customers are included.

Question 4

Display all customers and orders, whether matched or not.

```
SELECT c.*, o.*
FROM Customers c
LEFT JOIN Orders o
ON c.CustomerID = o.CustomerID
UNION
SELECT c.*, o.*
FROM Customers c
RIGHT JOIN Orders o
ON c.CustomerID = o.CustomerID;
```

Explanation: UNION simulates FULL OUTER JOIN in MySQL.

Question 5

Find customers who have not placed any orders.

```
SELECT c.*
FROM Customers c
LEFT JOIN Orders o
ON c.CustomerID = o.CustomerID
WHERE o.OrderID IS NULL;
```

Explanation: NULL filtering identifies customers without orders.

Question 6

Retrieve customers who made payments but did not place any orders.

```
SELECT DISTINCT p.CustomerID
FROM Payments p
LEFT JOIN Orders o
ON p.CustomerID = o.CustomerID
WHERE o.OrderID IS NULL;
```

Explanation: Compares payment records against orders to find unmatched customers.

Question 7

Generate a list of all possible combinations between Customers and Orders.

```
SELECT c.CustomerName, o.OrderID
FROM Customers c
CROSS JOIN Orders o;
```

Explanation: CROSS JOIN generates Cartesian combinations.

Question 8

Show all customers along with order and payment amounts in one table.

```
SELECT c.CustomerName, o.Amount AS OrderAmount, p.Amount AS
PaymentAmount
FROM Customers c
LEFT JOIN Orders o ON c.CustomerID = o.CustomerID
LEFT JOIN Payments p ON c.CustomerID = p.CustomerID;
```

Explanation: Multiple LEFT JOINs consolidate customer, order, and payment data.

Question 9

Retrieve all customers who have both placed orders and made payments.

```
SELECT DISTINCT c.CustomerName
FROM Customers c
INNER JOIN Orders o ON c.CustomerID = o.CustomerID
INNER JOIN Payments p ON c.CustomerID = p.CustomerID;
```

Explanation: INNER JOIN across multiple tables ensures complete activity.