CREATE TABLE Customers (

CustomerID INT PRIMARY KEY,

FirstName NVARCHAR(50),

LastName NVARCHAR(50),

City NVARCHAR(50)

);

CREATE TABLE Orders (

OrderID INT PRIMARY KEY,

CustomerID INT,

OrderDate DATE,

OrderAmount DECIMAL(10, 2),

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

INSERT INTO Customers (CustomerID, FirstName, LastName, City) VALUES

(1, 'John', 'Doe', 'Mumbai'),

(2, 'Jane', 'Smith', 'Delhi'),

(3, 'Emily', 'Jones', 'Bangalore'),

(4, 'Michael', 'Brown', 'Mumbai'),

(5, 'Sarah', 'Davis', 'Chennai');

INSERT INTO Orders (OrderID, CustomerID, OrderDate, OrderAmount) VALUES

(101, 1, '2023-01-15', 500.00),

(102, 1, '2023-03-10', 700.00),

(103, 2, '2023-02-20', 1200.00),

(104, 3, '2023-04-05', 300.00),

(105, 4, '2023-06-12', 1500.00),

(106, 5, '2023-07-19', 2000.00),

(107, 1, '2023-09-25', 800.00),

(108, 3, '2023-10-01', 900.00);

-- Query Exercises

1. Filter and Aggregate on Join Results using SQL

-- Task: Join the Orders and Customers tables to find the total order amount per customer and filter out customers who have spent less than $1,000.

SELECT c.CustomerID, c.FirstName, c.LastName,

SUM(o.OrderAmount) AS TotalSpent

FROM Orders o

JOIN Customers c ON o.CustomerID = c.CustomerID

GROUP BY c.CustomerID, c.FirstName, c.LastName

HAVING SUM(o.OrderAmount) >= 1000;

2. Cumulative Aggregations and Ranking in SQL Queries

Create a cumulative sum of the OrderAmount for each customer to track the running total of how much each customer has spent.

WITH CumulativeSum AS (

SELECT c.CustomerID, c.FirstName, c.LastName, o.OrderDate, o.OrderAmount,

SUM(o.OrderAmount) OVER (PARTITION BY c.CustomerID ORDER BY o.OrderDate) AS RunningTotal

FROM Orders o

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

SELECT \* FROM CumulativeSum;

3. OVER and PARTITION BY Clause in SQL Queries

-- Task: Rank the customers based on the total amount they have spent, partitioned by city.

WITH CustomerTotal AS (

SELECT c.CustomerID, c.FirstName, c.LastName, c.City, SUM(o.OrderAmount) AS TotalSpent

FROM Orders o

JOIN Customers c ON o.CustomerID = c.CustomerID

GROUP BY c.CustomerID, c.FirstName, c.LastName, c.City )

SELECT CustomerID, FirstName, LastName, City, TotalSpent,

RANK() OVER (PARTITION BY City ORDER BY TotalSpent DESC) AS CityRank

FROM CustomerTotal;

4. Total Aggregation using OVER and PARTITION BY in SQL Queries

-- Task: Calculate the total amount of all orders (overall total) and the percentage each customer's total spending contributes to the overall total.

WITH TotalOrders AS (

SELECT SUM(OrderAmount) AS OverallTotal FROM Orders),

CustomerTotal AS (

SELECT c.CustomerID, c.FirstName, c.LastName,

SUM(o.OrderAmount) AS TotalSpent

FROM Orders o

JOIN Customers c ON o.CustomerID = c.CustomerID

GROUP BY c.CustomerID, c.FirstName, c.LastName )

SELECT CustomerID, FirstName, LastName, TotalSpent,

(TotalSpent \* 100.0 / (SELECT OverallTotal FROM TotalOrders)) AS PercentageOfTotal

FROM CustomerTotal;

5. Ranking in SQL

-- Task: Rank all customers based on the total amount they have spent, without partitioning.

WITH CustomerTotal AS (

SELECT c.CustomerID, c.FirstName, c.LastName,

SUM(o.OrderAmount) AS TotalSpent

FROM Orders o

JOIN Customers c ON o.CustomerID = c.CustomerID

GROUP BY c.CustomerID, c.FirstName, c.LastName )

SELECT CustomerID, FirstName, LastName, TotalSpent, RANK() OVER (ORDER BY TotalSpent DESC) AS OverallRank

FROM CustomerTotal;

6. Calculate the Average Order Amount per City\*\*

-- Task: Write a query that joins the Orders and Customers tables, calculates the average order amount for each city, and orders the results by the average amount in descending order.

SELECT c.City, AVG(o.OrderAmount) AS AverageOrderAmount

FROM Orders o

JOIN Customers c ON o.CustomerID = c.CustomerID

GROUP BY c.City

ORDER BY AverageOrderAmount DESC;

7. Find Top N Customers by Total Spending

-- Task: Write a query to find the top 3 customers who have spent the most, using ORDER BY and LIMIT.

WITH CustomerTotal AS (

SELECT c.CustomerID, c.FirstName, c.LastName, SUM(o.OrderAmount) AS TotalSpent

FROM Orders o

JOIN Customers c ON o.CustomerID = c.CustomerID

GROUP BY c.CustomerID, c.FirstName, c.LastName

)

SELECT TOP 3 CustomerID, FirstName, LastName, TotalSpent

FROM CustomerTotal

ORDER BY TotalSpent DESC;

8. Calculate Yearly Order Totals

-- Task: Write a query that groups orders by year (using OrderDate), calculates the total amount of orders for each year, and orders the results by year.

SELECT

YEAR(OrderDate) AS OrderYear,

SUM(OrderAmount) AS TotalAmount

FROM Orders

GROUP BY YEAR(OrderDate)

ORDER BY OrderYear;

9. Rank Customers by Total Order Amount in Mumbai

-- Task: Write a query that ranks customers by their total spending, but only for customers located in "Mumbai". The rank should reset for each customer in Mumbai.

WITH CustomerTotal AS (

SELECT c.CustomerID, c.FirstName, c.LastName, c.City,

SUM(o.OrderAmount) AS TotalSpent

FROM Orders o

JOIN Customers c ON o.CustomerID = c.CustomerID

WHERE c.City = 'Mumbai'

GROUP BY c.CustomerID, c.FirstName, c.LastName, c.City )

SELECT CustomerID, FirstName, LastName, TotalSpent,

RANK() OVER (ORDER BY TotalSpent DESC) AS MumbaiRank

FROM CustomerTotal;

10. Compare Each Customer's Total Order to the Average Order Amount

-- Task: Write a query that calculates each customer's total order amount and compares it to the average order amount for all customers.

WITH CustomerTotal AS (

SELECT c.CustomerID, c.FirstName, c.LastName,

SUM(o.OrderAmount) AS TotalSpent

FROM Orders o

JOIN Customers c ON o.CustomerID = c.CustomerID

GROUP BY c.CustomerID, c.FirstName, c.LastName ),

AverageOrder AS (

SELECT AVG(OrderAmount) AS AvgOrderAmount

FROM Orders)

SELECT c.CustomerID, c.FirstName, c.LastName, c.TotalSpent,

CASE

WHEN c.TotalSpent > (SELECT AvgOrderAmount FROM AverageOrder) THEN 'Above Average'

ELSE 'Below Average'

END AS SpendingComparison

FROM CustomerTotal c;