### Name: Thirupugazh S.R.

### Title: Assignment - DBMS

Question 1: Top 3 Departments with Highest Average Salary.

1)SQL QUERY:-

SELECT DepartmentID, DepartmentName, AVG(Salary) AS AvgSalary

FROM Employees

LEFT JOIN Departments ON Employees.DepartmentID = Departments.DepartmentID

GROUP BY DepartmentID, DepartmentName

ORDER BY AvgSalary DESC

LIMIT 3;

### Explanation:-

**Handling Departments with No Employees**:

* 1. The LEFT JOIN ensures that all departments are included, even if they have no employees. For departments with no employees, the average salary will be NULL.

**Calculating Average Salary**:-

* 1. The AVG(Salary) function computes the average salary for employees within each department. The GROUP BY clause groups the result by DepartmentID and DepartmentName.

**Selecting Top 3 Departments**:-

* 1. The ORDER BY AvgSalary DESC sorts departments by average salary in descending order. The LIMIT 3 restricts the output to the top 3 departments with the highest average salaries.

Question 2: Retrieving Hierarchical Category Paths

WITH RECURSIVE CategoryPaths AS (

SELECT CategoryID, CategoryName, CAST(CategoryName AS VARCHAR(MAX)) AS Path

FROM Categories

WHERE ParentCategoryID IS NULL

UNION ALL

SELECT c.CategoryID, c.CategoryName, CONCAT(cp.Path, ' > ', c.CategoryName) AS Path

FROM Categories c

INNER JOIN CategoryPaths cp ON c.ParentCategoryID = cp.CategoryID

)

SELECT CategoryID, CategoryName, Path

FROM CategoryPaths;

### Explanation:-

**Recursive CTE**:-

* 1. **Base Case**: The initial SELECT statement retrieves categories with no parent (root categories). It sets the path to the category name itself.
  2. **Recursive Step**: The second SELECT statement joins the Categories table with the CategoryPaths CTE. It constructs the hierarchical path by concatenating the parent path with the current category name.

**Path Construction**:-

* 1. The CONCAT function creates the hierarchical path string, ensuring each category’s path reflects its position in the hierarchy.

Question 3: Total Distinct Customers by Month

SQL Query:-

WITH Months AS (

SELECT DISTINCT EXTRACT(YEAR FROM OrderDate) AS Year, EXTRACT(MONTH FROM OrderDate) AS Month

FROM Orders

WHERE EXTRACT(YEAR FROM OrderDate) = EXTRACT(YEAR FROM CURRENT\_DATE)

),

CustomerCounts AS (

SELECT EXTRACT(MONTH FROM OrderDate) AS Month, COUNT(DISTINCT CustomerID) AS CustomerCount

FROM Orders

WHERE EXTRACT(YEAR FROM OrderDate) = EXTRACT(YEAR FROM CURRENT\_DATE)

GROUP BY EXTRACT(MONTH FROM OrderDate)

)

SELECT m.Month, COALESCE(cc.CustomerCount, 0) AS CustomerCount

FROM Months m

LEFT JOIN CustomerCounts cc ON m.Month = cc.Month

ORDER BY m.Month;

#### Explanation:-

**Months CTE:-**

* 1. Generates a list of the first days of each month for the current year.

**CustomerCounts CTE:-**

* 1. Aggregates distinct customer counts per month from the Orders table.

**Combining Data:-**

* 1. Uses a LEFT JOIN to ensure that months with no customer activity are included, showing a count of 0 where necessary.

Question 4: Finding Closest Locations

### SQL Query

sql

Copy code

SELECT LocationID, LocationName, Latitude, Longitude,

(6371 \* ACOS(COS(RADIANS(:latitude)) \* COS(RADIANS(Latitude)) \* COS(RADIANS(Longitude) - RADIANS(:longitude)) + SIN(RADIANS(:latitude)) \* SIN(RADIANS(Latitude)))) AS DistanceFROM LocationsORDER BY Distance

LIMIT 5;

### Explanation

**Distance Calculation**:-

* 1. The query uses the Haversine formula to calculate the distance between a given point (:latitude, :longitude) and each location. 6371 is the approximate radius of the Earth in kilometers.

**Ordering and Limiting**:-

* 1. The results are ordered by the calculated distance in ascending order. LIMIT 5 restricts the output to the 5 closest locations.

## Question 5: Optimizing Query for Orders Table

### SQL Query

-- Assuming an index on OrderDate for optimization

SELECT OrderID, CustomerID, OrderDate, TotalAmount

FROM Orders

WHERE OrderDate >= CURRENT\_DATE - INTERVAL '7 DAY'

ORDER BY OrderDate DESC;

### Explanation:-

**Index Utilization**:-

* 1. The query should benefit from an index on the OrderDate column to efficiently filter orders from the last 7 days

**Optimizations**:-

* 1. **Indexing**: Ensure that there is an index on the OrderDate column to speed up the WHERE clause filter.
  2. **Date Calculation**: The use of CURRENT\_DATE - INTERVAL '7 DAY' ensures that only recent orders are considered.
  3. **Sorting**: The ORDER BY OrderDate DESC sorts the results by date in descending order, displaying the most recent orders first.