**DATABASE MANAGEMENT SYSTEM**

**CSA0554**

**192311090**

**MANN**

**Question 2:**

**Question 1: Top 3 Departments with Highest Average Salary**

**Task:**

1. Write a SQL query to find the top 3 departments with the highest average salary of employees. Ensure departments with no employees show an average salary of NULL.

**Deliverables:**

1. SQL query that retrieves DepartmentID, DepartmentName, and AvgSalary for the top 3 departments.
2. Explanation of how the query handles departments with no employees and calculates average salary.

CREATE TABLE Departments

( DepartmentID INT PRIMARY KEY,

DepartmentName VARCHAR(100) NOT NULL

);

CREATE TABLE Employees

( EmployeeID INT PRIMARY KEY,

EmployeeName VARCHAR(100) NOT NULL,

Salary DECIMAL(10, 2) NOT NULL,

DepartmentID INT,

FOREIGN KEY (DepartmentID) REFERENCES Departments(DepartmentID)

);

INSERT INTO Departments (DepartmentID, DepartmentName) VALUES (1, 'HR'), (2, 'Finance'), (3, 'Engineering'), (4, 'Marketing');

INSERT INTO Employees (EmployeeID, EmployeeName, Salary, DepartmentID) VALUES (1, 'Alice', 60000, 1), (2, 'Bob', 65000, 1), (3, 'Charlie', 70000, 2), (4, 'David', 72000, 2), (5, 'Eve', 75000, 3), (6, 'Frank', 80000, 3), (7, 'Grace', 55000, 4);

WITH DepartmentSalaries AS (

SELECT

d.DepartmentID,

d.DepartmentName,

AVG(e.Salary) AS AvgSalary

FROM Departments d

LEFT JOIN

Employees e ON d.DepartmentID = e.DepartmentID

GROUP BY d.DepartmentID, d.DepartmentName ),

RankedDepartments AS (

SELECT DepartmentID,

DepartmentName,

AvgSalary,

ROW\_NUMBER() OVER (ORDER BY AvgSalary DESC NULLS LAST) AS rn

FROM DepartmentSalaries )

SELECT DepartmentID, DepartmentName, AvgSalary FROM RankedDepartments WHERE rn <= 3;

**OUTPUT:**



**Question 2: Retrieving Hierarchical Category Paths**

**Task:**

1. Write a SQL query using recursive Common Table Expressions (CTE) to retrieve all categories along with their full hierarchical path (e.g., Category > Subcategory > Sub-subcategory).

CREATE TABLE ProductCategories

( CategoryID INTEGER PRIMARY KEY,

CategoryName TEXT,

ParentCategoryID INTEGER,

FOREIGN KEY (ParentCategoryID) REFERENCES ProductCategories(CategoryID) );

INSERT INTO ProductCategories (CategoryID, CategoryName, ParentCategoryID)

VALUES (1, 'Electronics', NULL), (2, 'Computers', 1), (3, 'Laptops', 2), (4, 'Ultrabooks', 3), (5, 'Gaming Laptops', 3), (6, 'Smartphones', 1), (7, 'Android Phones', 6), (8, 'iPhones', 6), (9, 'Home Appliances', NULL), (10, 'Refrigerators', 9), (11, 'Washing Machines', 9);

WITH RECURSIVE CategoryPath AS (

SELECT CategoryID,

CategoryName,

ParentCategoryID,

CategoryName AS Path

FROM ProductCategories

WHERE

ParentCategoryID IS NULL

UNION ALL

SELECT c.CategoryID,

c.CategoryName,

c.ParentCategoryID,

cp.Path || ' > ' || c.CategoryName

FROM ProductCategories c

INNER JOIN

CategoryPath cp ON c.ParentCategoryID = cp.CategoryID )

SELECT CategoryID, CategoryName, Path

FROM CategoryPath;

**OUTPUT:**

****



**Question 3: Total Distinct Customers by Month**

**Task:**

1. Design a SQL query to find the total number of distinct customers who made a purchase in each month of the current year. Ensure months with no customer activity show a count of 0.

**Deliverables:**

1. SQL query that retrieves MonthName and CustomerCount for each month.

Explanation of how the query ensures all months are included and handles zero customer counts.

CREATE TABLE Purchase

( PurchaseID INTEGER PRIMARY KEY,

CustomerID INTEGER, PurchaseDate DATE );

INSERT INTO Purchase (PurchaseID, CustomerID, PurchaseDate) VALUES (1, 101, '2024-01-15'), (2, 102, '2024-01-25'), (3, 103, '2024-02-10'), (4, 101, '2024-02-20'), (5, 104, '2024-03-05'), (6, 105, '2024-03-15'), (7, 106, '2024-04-01');

WITH RECURSIVE MonthList AS (

SELECT DATE('2024-01-01') AS MonthStart

UNION ALL

SELECT DATE(MonthStart, '+1 month')

FROM MonthList

WHERE strftime('%Y', MonthStart) = '2024' )

SELECT strftime('%Y-%m', MonthStart) AS Month

FROM MonthList

WHERE strftime('%Y', MonthStart) = '2024';

WITH RECURSIVE MonthList AS (

SELECT DATE('2024-01-01') AS MonthStart

UNION ALL SELECT DATE(MonthStart, '+1 month')

FROM MonthList WHERE strftime('%Y', MonthStart) = '2024' )

SELECT strftime('%Y-%m', ml.MonthStart) AS Month,

COALESCE(COUNT(DISTINCT p.CustomerID), 0) AS DistinctCustomerCount

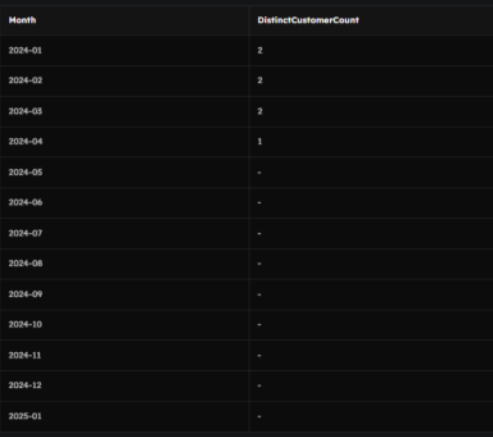
FROM MonthList ml

LEFT JOIN Purchases p

ON strftime('%Y-%m', p.PurchaseDate) = strftime('%Y-%m', ml.MonthStart)

GROUP BY ml.MonthStart

ORDER BY ml.MonthStart;



**Question 4: Finding Closest Locations**

**Task:**

1. Write a SQL query to find the closest 5 locations to a given point specified by latitude and longitude. Use spatial functions or advanced mathematical calculations for proximity.

**Deliverables:**

* + 1. SQL query that calculates the distance and retrieves LocationID, LocationName, Latitude, and Longitude for the closest 5 locations.

Explanation of the spatial or mathematical approach used to determine proximity.

CREATE TABLE locations (

id INT PRIMARY KEY,

name VARCHAR(255),

latitude DECIMAL(10, 8),

longitude DECIMAL(11, 8) );

INSERT INTO locations (id, name, latitude, longitude)

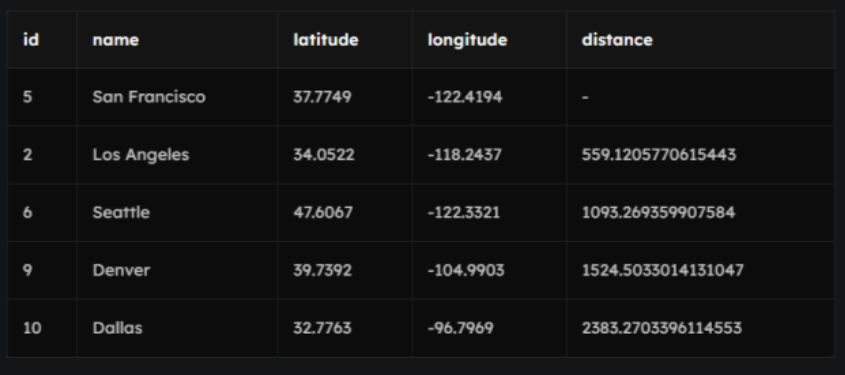
VALUES (1, 'New York City', 40.7128, -74.0060), (2, 'Los Angeles', 34.0522, -118.2437), (3, 'Chicago', 41.8781, -87.6298), (4, 'Houston', 29.7633, -95.3632), (5, 'San Francisco', 37.7749, -122.4194), (6, 'Seattle', 47.6067, -122.3321), (7, 'Miami', 25.7917, -80.1306), (8, 'Boston', 42.3584, -71.0596), (9, 'Denver', 39.7392, -104.9903), (10, 'Dallas', 32.7763, -96.7969);

SELECT id, name, latitude, longitude, (6371 \* ACOS(SIN(RADIANS(37.7749)) \* SIN(RADIANS(latitude)) + COS(RADIANS(37.7749)) \* COS(RADIANS(latitude)) \* COS(RADIANS(longitude) - RADIANS(-122.4194)))) AS distance

FROM locations

ORDER BY distance ASC

LIMIT 5;



**Question 5: Optimizing Query for Orders Table**

**Task:**

1. Write a SQL query to retrieve orders placed in the last 7 days from a large Orders table, sorted by order date in descending order.

**Deliverables:**

1. SQL query optimized for performance, considering indexing, query rewriting, or other techniques.
2. Discussion of strategies used to optimize the query and improve performance.

CREATE TABLE Orders

(order\_id INT PRIMARY KEY,

customer\_id INT,

order\_date DATE,

total DECIMAL(10, 2) );

INSERT INTO Orders (order\_id, customer\_id, order\_date, total)

VALUES (1, 1, '2022-01-01', 100.00), (2, 2, '2022-01-03', 200.00), (3, 3, '2022-01-05', 50.00), (4, 1, '2022-01-08', 150.00), (5, 4, '2022-01-10', 300.00), (6, 2, '2022-01-12', 250.00), (7, 3, '2022-01-14', 100.00), (8, 1, '2022-01-16', 200.00), (9, 4, '2022-01-18', 400.00), (10, 2, '2022-01-20', 350.00);

CREATE TABLE last\_7\_days\_orders AS

SELECT \*

FROM Orders

WHERE order\_date >= DATE('now', '-7 day')

ORDER BY order\_date DESC;

