ASSIGNMENT QUESTION-2

Question 1: Top 3 Departments with Highest Average Salary

Task:

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.

```
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:

Output

DepartmentID	DepartmentName	AvgSalary
3	Engineering	77500
2	Finance	71000
1	HR	62500

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).

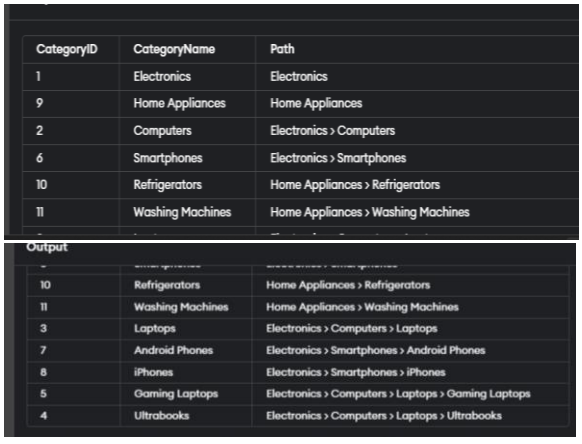
```
CREATE TABLE ProductCategories (
   CategoryID INTEGER PRIMARY KEY,
   CategoryName TEXT,
   ParentCategoryID INTEGER,
   FOREIGN KEY (ParentCategoryID) REFERENCES ProductCategories(CategoryID)
);
INSERT INTO ProductCategories (CategoryID, CategoryName, ParentCategoryID) VALUES
(1, 'Electronics', 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:



CategoryID	CategoryName	Path
1	Electronics	Electronics
9	Home Appliances	Home Appliances
2	Computers	Electronics > Computers
6	Smartphones	Electronics > Smartphones
10	Refrigerators	Home Appliances > Refrigerators
11	Washing Machines	Home Appliances > Washing Machines

Output

10	Refrigerators	Home Appliances > Refrigerators
11	Washing Machines	Home Appliances > Washing Machines
3	Laptops	Electronics > Computers > Laptops
7	Android Phones	Electronics > Smartphones > Android Phones
8	iPhones	Electronics > Smartphones > iPhones
5	Gaming Laptops	Electronics > Computers > Laptops > Gaming Laptops
4	Ultrabooks	Electronics > Computers > Laptops > Ultrabooks

CategoryID	CategoryName	Path
1	Electronics	Electronics
9	Home Appliances	Home Appliances

CategoryID	CategoryName	Path
2	Computers	Electronics > Computers
6	Smartphones	Electronics > Smartphones
10	Refrigerators	Home Appliances > Refrigerators
11	Washing Machines	Home Appliances > Washing Machines
3	Laptops	Electronics > Computers > Laptops
7	Android Phones	Electronics > Smartphones > Android Phones
8	iPhones	Electronics > Smartphones > iPhones
5	Gaming Laptops	Electronics > Computers > Laptops > Gaming Laptops
4	Ultrabooks	Electronics > Computers > Laptops > Ultrabooks

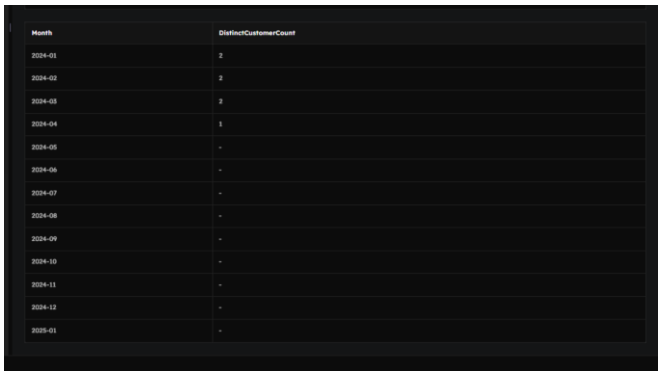
Question 3: Total Distinct Customers by Month

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.

```
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;
```



Month	DistinctCustomerCount
2024-01	2
2024-02	3
2024-03	2
2024-04	1
2024-05	-
2024-06	-
2024-07	-
2024-08	-
2024-09	-
2024-10	-
2024-11	-
2024-12	-
2025-01	-

Question 4: Finding Closest Locations

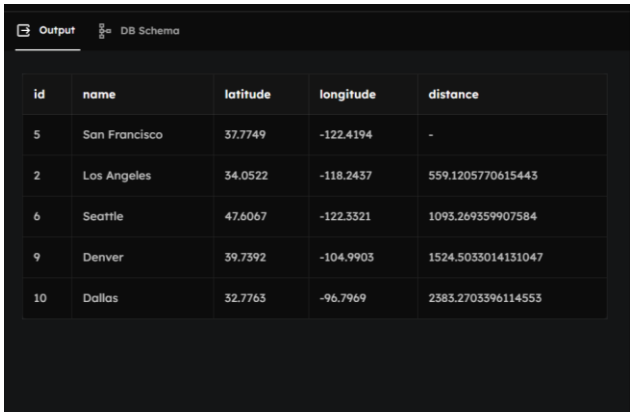
Task:

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

```
-- Create a table to store locations
CREATE TABLE locations (
  id INT PRIMARY KEY,
  name VARCHAR(255),
  latitude DECIMAL(10, 8),
  longitude DECIMAL(11, 8)
);
-- Insert sample data into the table
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);
-- Find the 5 closest locations to San Francisco, CA
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;
```

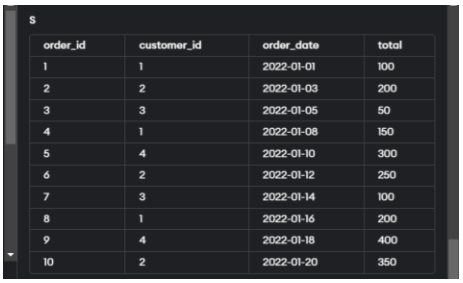


Question 5: Optimizing Query for Orders Table

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.

```
-- Create the Orders table
CREATE TABLE `Orders` (
  `order_id` INT PRIMARY KEY,
  `customer_id` INT,
  `order_date` DATE,
  `total` DECIMAL(10, 2)
);
-- Insert some sample data into the Orders table
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 a new table last_7_days_orders with the query to retrieve orders placed in the last 7 days
CREATE TABLE `last_7_days_orders` AS
SELECT *
FROM `Orders`
WHERE `order_date` >= DATE('now', '-7 day')
ORDER BY `order_date` DESC;
```



s

order_id	customer_id	order_date	total
1	1	2022-01-01	100
2	2	2022-01-03	200
3	3	2022-01-05	50
4	1	2022-01-08	150
5	4	2022-01-10	300
6	2	2022-01-12	250
7	3	2022-01-14	100
8	1	2022-01-16	200
9	4	2022-01-18	400
10	2	2022-01-20	350