DBMS ASSIGNMENT

**Question 1:**

**ER Diagram Question: Traffic Flow Management System (TFMS)**

**Tasks**

**Task 1: Entity Identification and Attributes**

Identify and list the entities relevant to the TFMS based on the scenario provided (e.g., Roads, Intersections, Traffic Signals, Traffic Data).

Define attributes for each entity, ensuring clarity and completeness.

**1. Roads**

* **Attributes**:
  + RoadID (PK)
  + RoadName
  + Length (in meters)
  + SpeedLimit (in km/h)

**2. Intersections**

* **Attributes**:
  + IntersectionID (PK)
  + IntersectionName
  + Latitude
  + Longitude

**3. Traffic Signals**

* **Attributes**:
  + SignalID (PK)
  + IntersectionID (FK)
  + SignalStatus (Green, Yellow, Red)
  + Timer (countdown to next change)

**4. Traffic Data**

* **Attributes**:
  + TrafficDataID (PK)
  + Timestamp
  + RoadID (FK)
  + Speed (average speed on the road)

**Task 2: Relationship Modeling**

Illustrate the relationships between entities in the ER diagram (e.g., Roads connecting to Intersections, Intersections hosting Traffic Signals).

Specify cardinality (one-to-one, one-to-many, many-to-many) and optionality constraints (mandatory vs. optional relationships).

**1. Roads and Intersections**

* **Relationship**: Roads connect to Intersections
* **Cardinality**: One-to-Many (A road can have multiple intersections, but each intersection is part of one road)
* **Optionality**: Mandatory for Intersections, Optional for Roads

**2. Intersections and Traffic Signals**

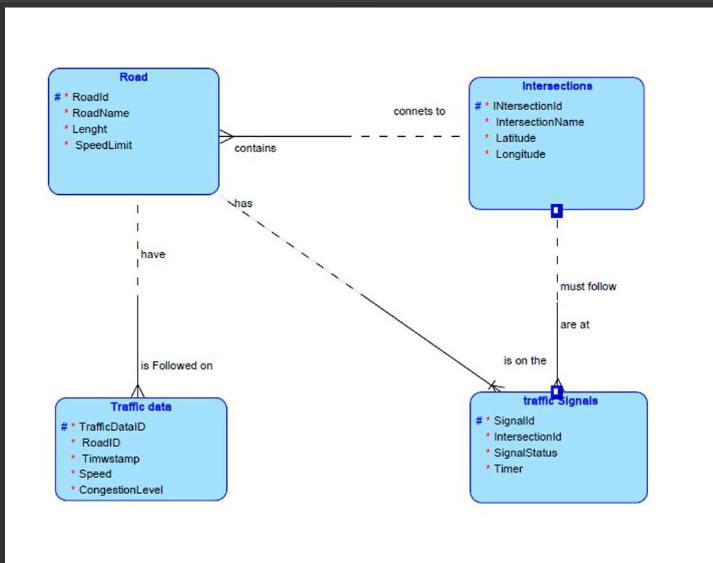
* **Relationship**: Intersections host Traffic Signals
* **Cardinality**: One-to-Many (An intersection can have multiple traffic signals, but each traffic signal is at one intersection)
* **Optionality**: Mandatory for Traffic Signals, Optional for Intersections

**3. Roads and Traffic Data**

* **Relationship**: Roads are associated with Traffic Data
* **Cardinality**: One-to-Many (A road can have multiple traffic data entries, but each traffic data entry is for one road)
* **Optionality**: Mandatory for Traffic Data, Optional for Roads

**Task 3: ER Diagram Design**

Draw the ER diagram for the TFMS, incorporating all identified entities, attributes, and relationships.



Label primary keys (PK) and foreign keys (FK) where applicable to establish relationships between entities.

**Task 4: Justification and Normalization**

Justify your design choices, including considerations for scalability, real-time data processing, and efficient traffic management.

Discuss how you would ensure the ER diagram adheres to normalization principles (1NF, 2NF, 3NF) to minimize redundancy and improve data integrity.

**Design Choices Justification:**

* **Scalability**: The design allows for easy addition of new roads, intersections, and traffic signals without affecting existing data structures. Each entity is clearly defined with its own attributes, ensuring that the system can handle large datasets efficiently.
* **Real-Time Data Processing**: By separating traffic data into its own entity, the system can efficiently handle real-time data updates and queries without affecting the road or intersection data.
* **Efficient Traffic Management**: The relationships between entities ensure that traffic signals can be dynamically controlled based on real-time traffic data, and optimal routes can be suggested by analyzing the traffic data associated with each road.

**Normalization Considerations:**

* **1NF (First Normal Form)**: Ensures that all attributes are atomic (indivisible) and each column contains unique values. The design adheres to 1NF as each attribute contains only a single value, and each entity has a primary key.
* **2NF (Second Normal Form)**: Ensures that all non-key attributes are fully functional dependent on the primary key. The design adheres to 2NF as each non-key attribute depends on the whole primary key, not just part of it.
* **3NF (Third Normal Form)**: Ensures that all non-key attributes are non-transitively dependent on the primary key. The design adheres to 3NF as there are no transitive dependencies; each non-key attribute is directly dependent on the primary key of its entity.

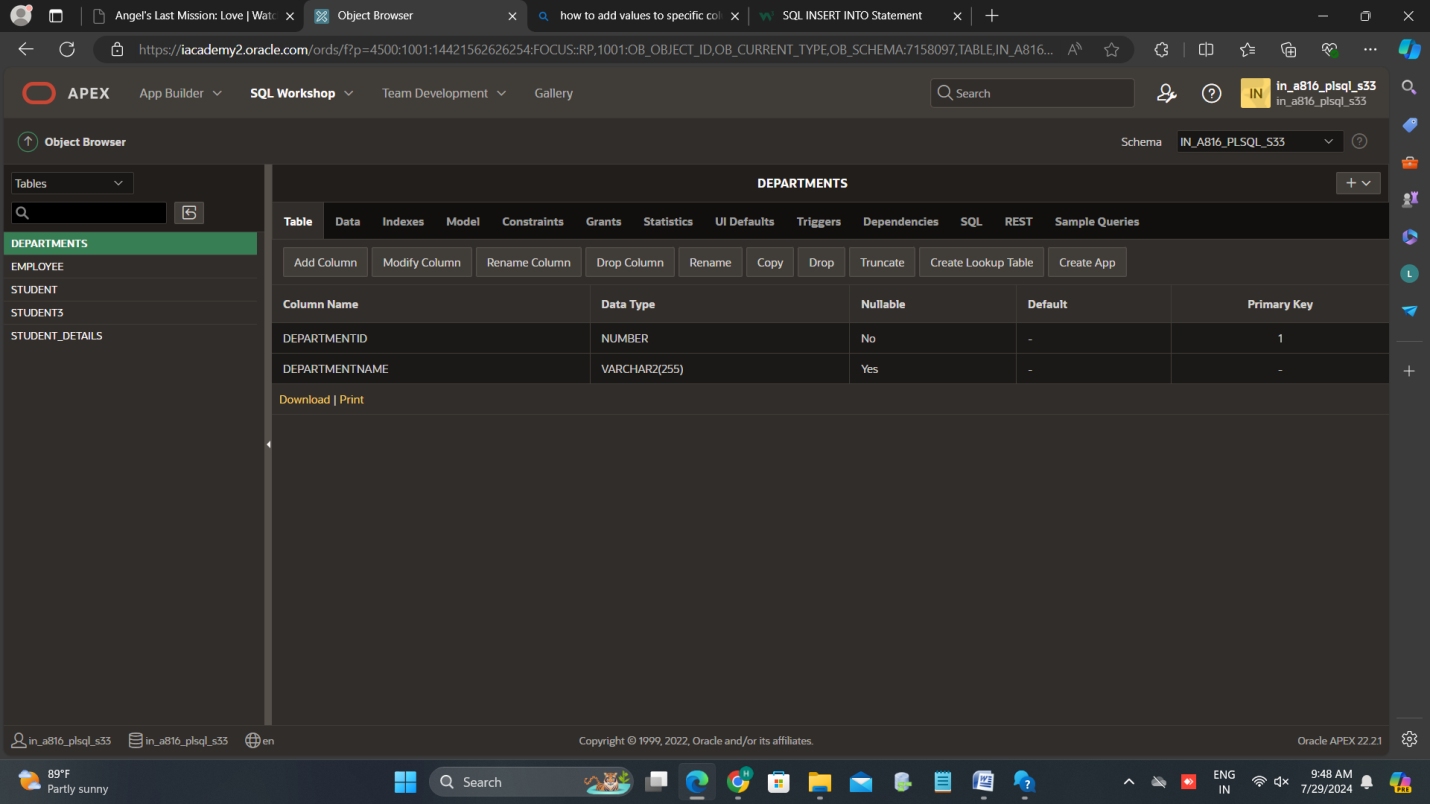
**Question 1: Top 3 Departments with Highest Average Salary**

**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.

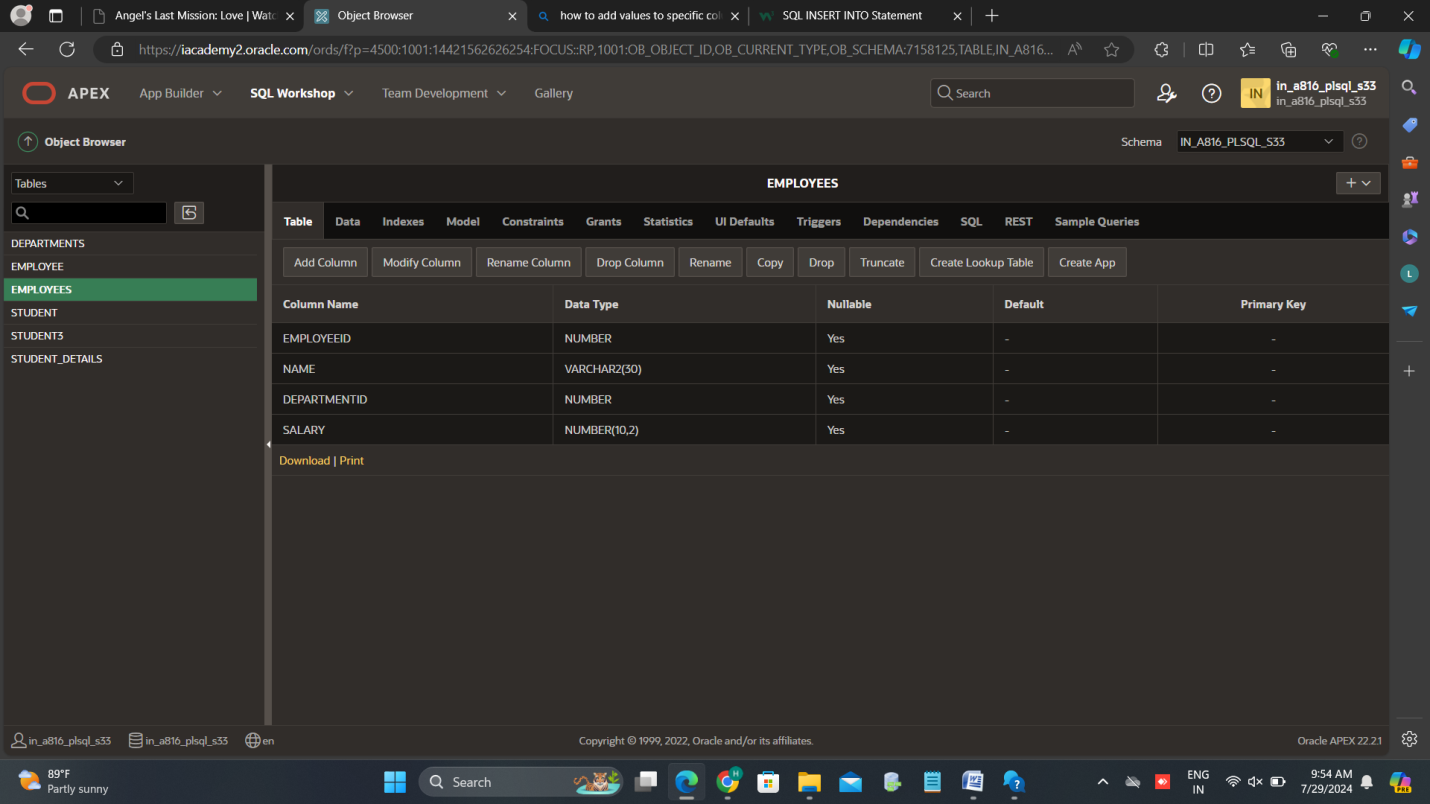
QUERY FOR CREATING TABLE “DEPARTMENTS”

CREATE TABLE Departments (DepartmentID INT ,DepartmentName VARCHAR(30);



## QUERY FOR CREATING TABLE “EMPLOYEES’’

CREATE TABLE Employees (EmployeeID INT,Name VARCHAR(30),DepartmentID INT,Salary DECIMAL(10, 2));



QUERY FOR INSERTING VALUES INTO “DEPARTMENTS’’

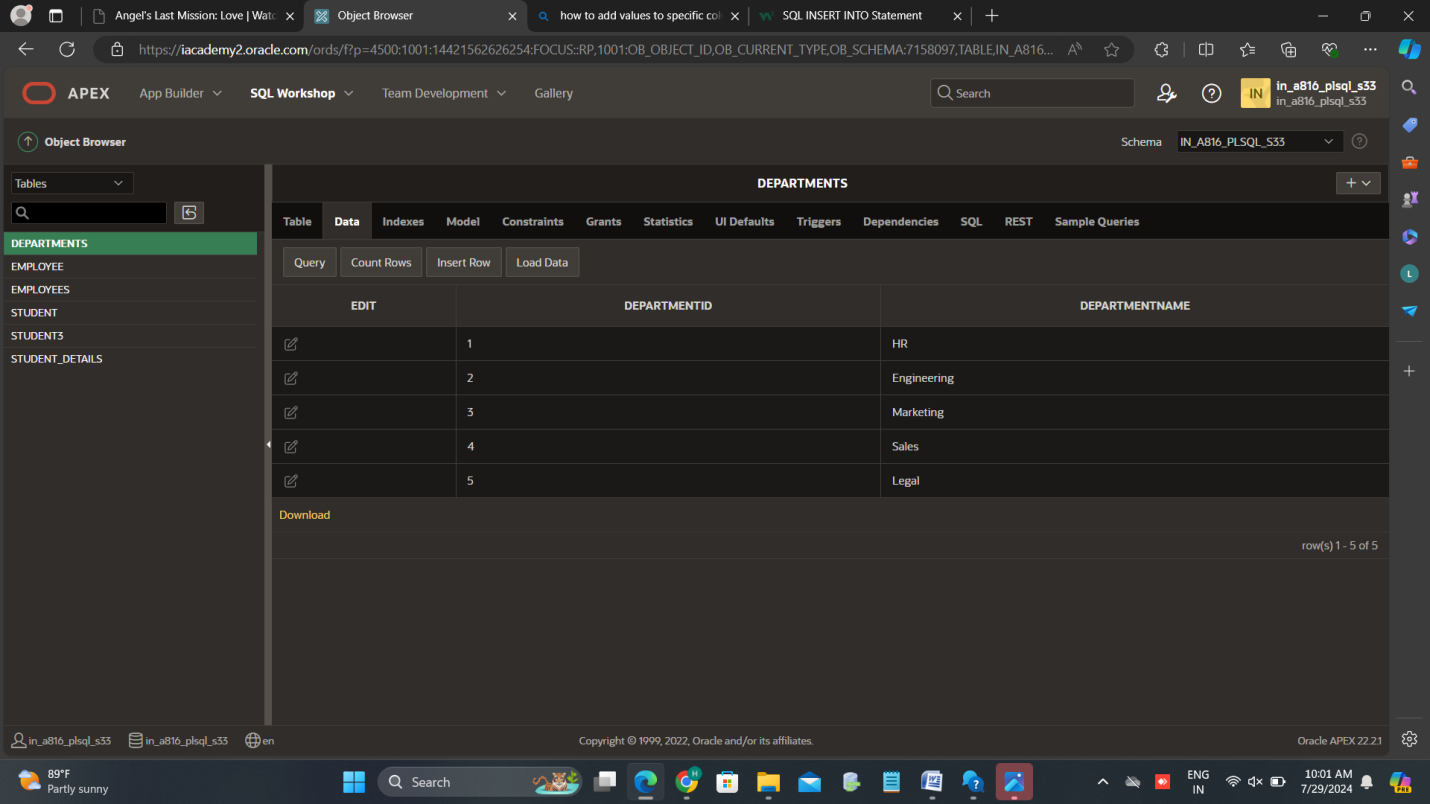
INSERT INTO Departments VALUES(1, 'HR');

INSERT INTO Departments VALUES(2, 'Engineering');

INSERT INTO Departments VALUES(3, 'Marketing');

INSERT INTO Departments VALUES(4, 'Sales');

INSERT INTO Departments VALUES(5, 'Legal');



QUERY FOR INSERTING VALUES INTO “EMPLOYEES’’

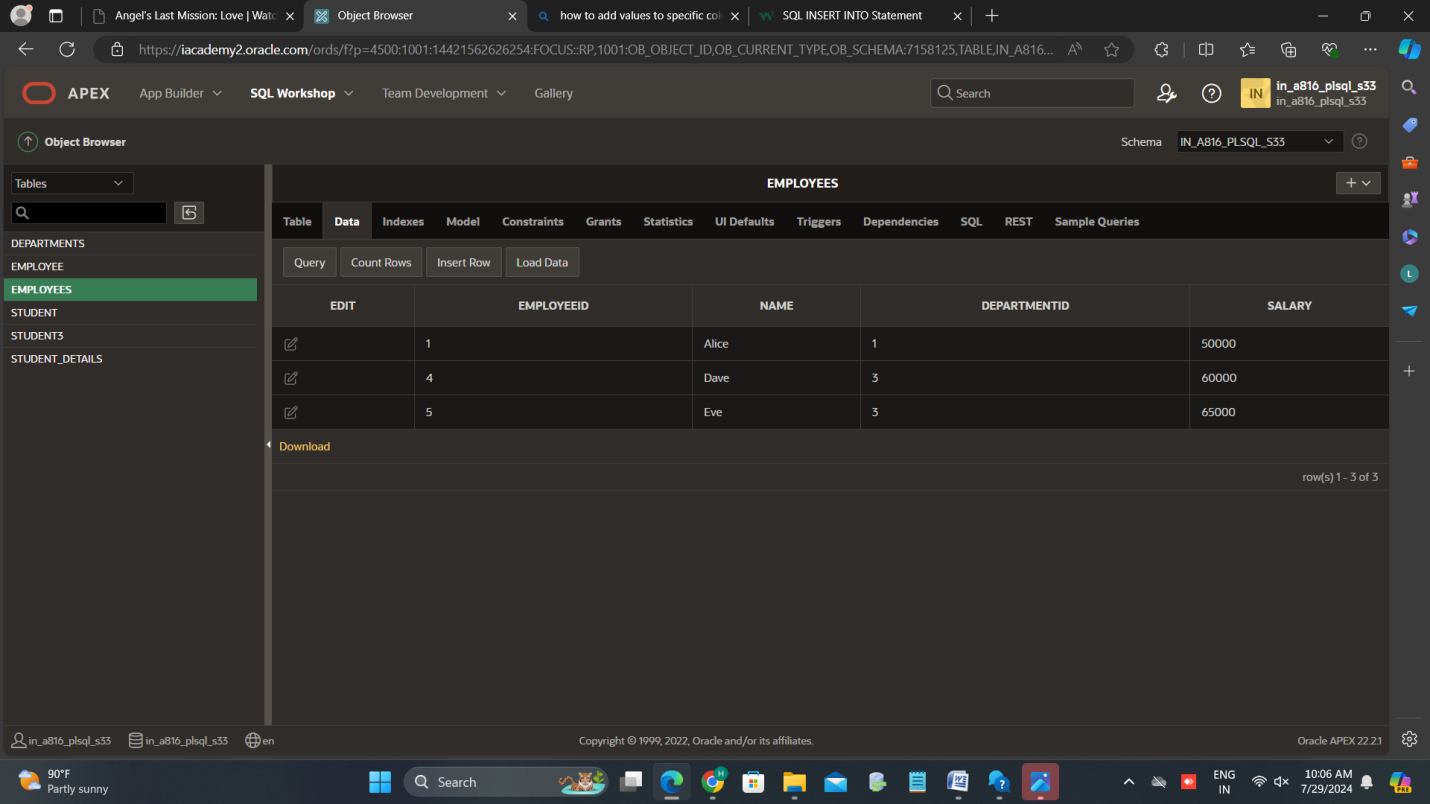
INSERT INTO Employees VALUES(1, 'Alice', 1, 50000);

INSERT INTO Employees VALUES(2, 'Bob', 2, 70000),

INSERT INTO Employees VALUES(3, 'Charlie', 2, 80000);

INSERT INTO Employees VALUES(4, 'Dave', 3, 60000);

INSERT INTO Employees VALUES(5, 'Eve', 3, 65000);



SQL query to find the top 3 departments with the highest average salary of employees

SELECT

d.DepartmentID,

d.DepartmentName,

AVG(e.Salary) AS AvgSalary

FROM

Departments d

LEFT JOIN

Employees e ON d.DepartmentID = e.DepartmentID

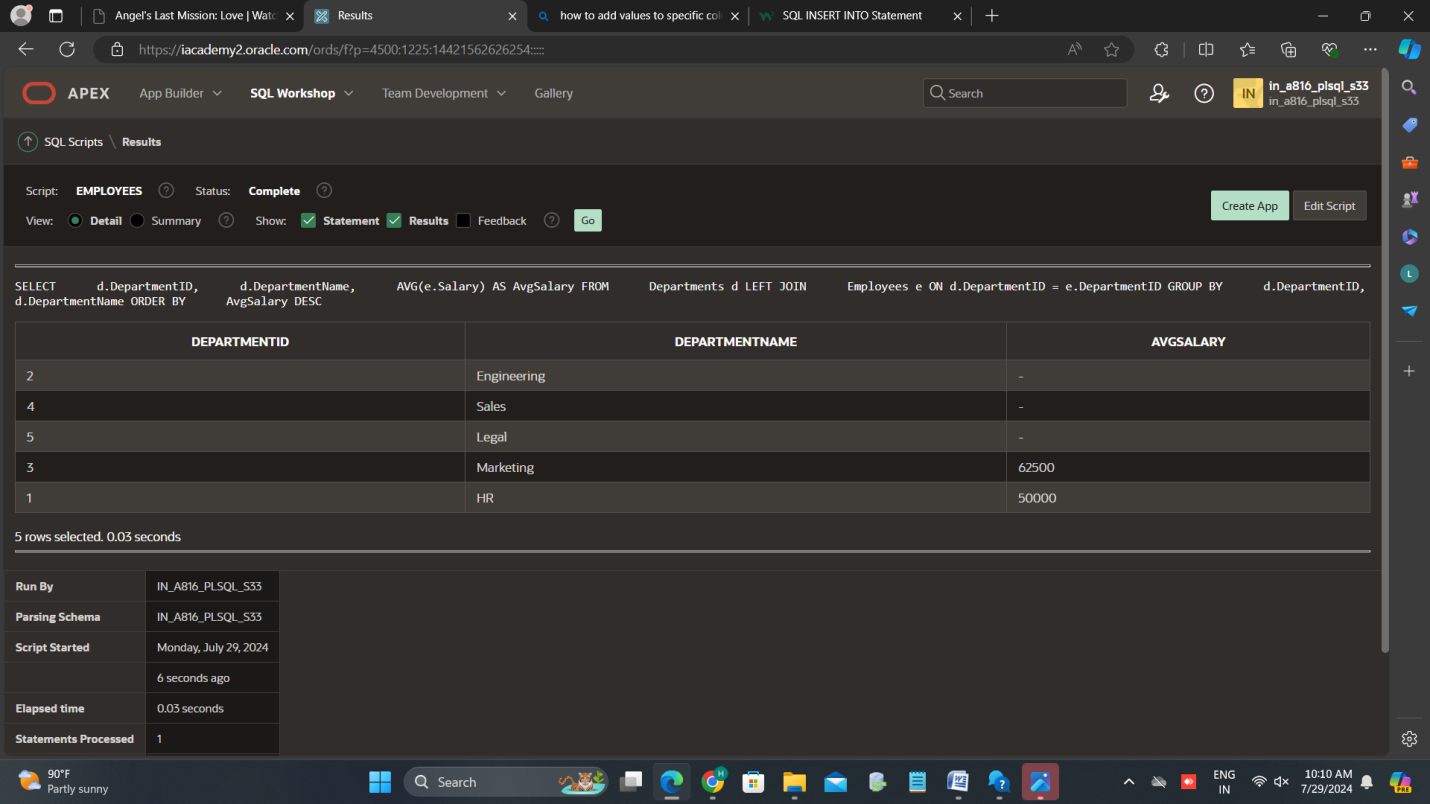
GROUP BY

d.DepartmentID,

d.DepartmentName

ORDER BY

AvgSalary DESC;



**Question 2: Retrieving Hierarchical Category Paths**

**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.

-- Query to find the top 3 departments with the highest average salary

SELECT

d.DepartmentID,

d.DepartmentName,

AVG(e.Salary) AS AvgSalary

FROM

Departments d

LEFT JOIN

Employees e ON d.DepartmentID = e.DepartmentID

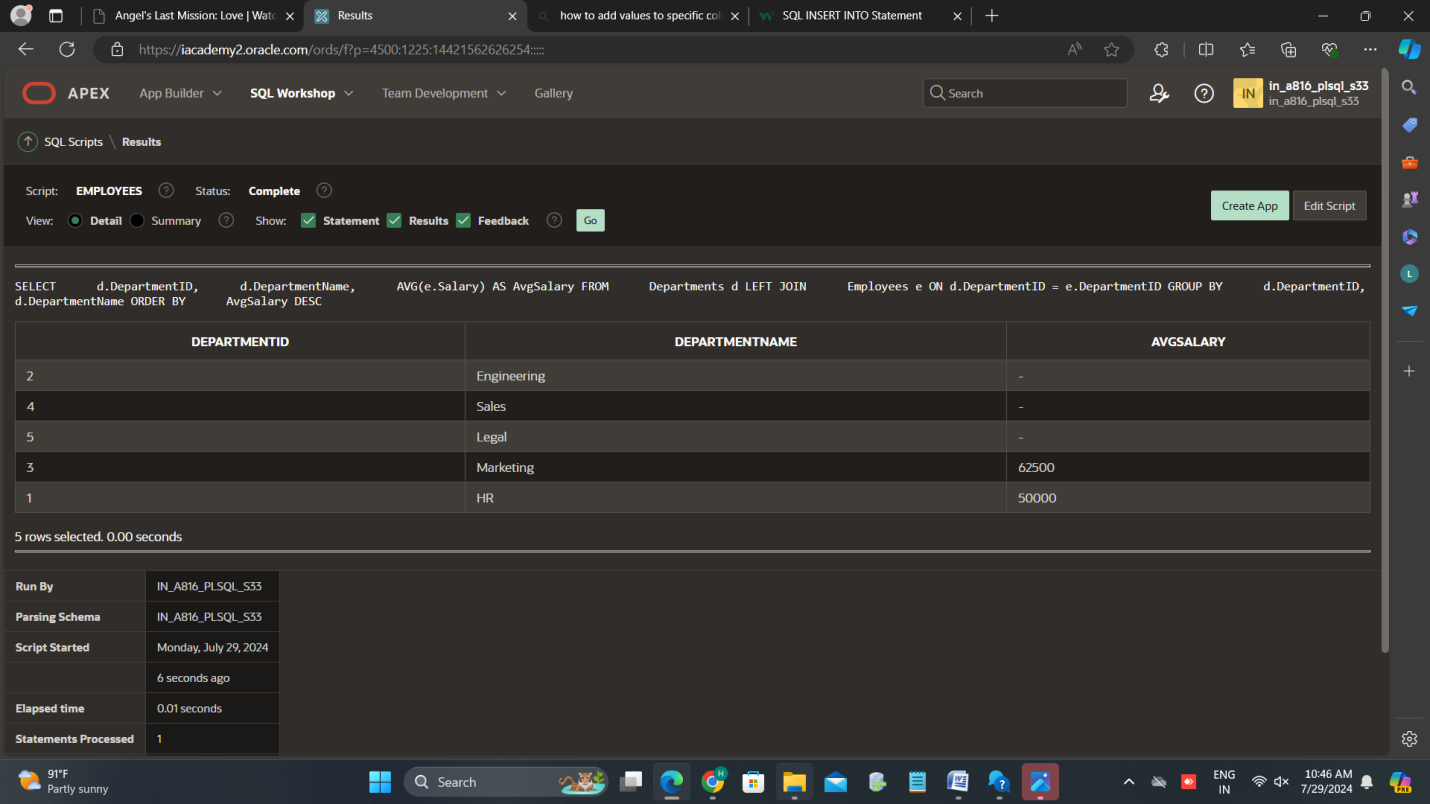
GROUP BY

d.DepartmentID,

d.DepartmentName

ORDER BY

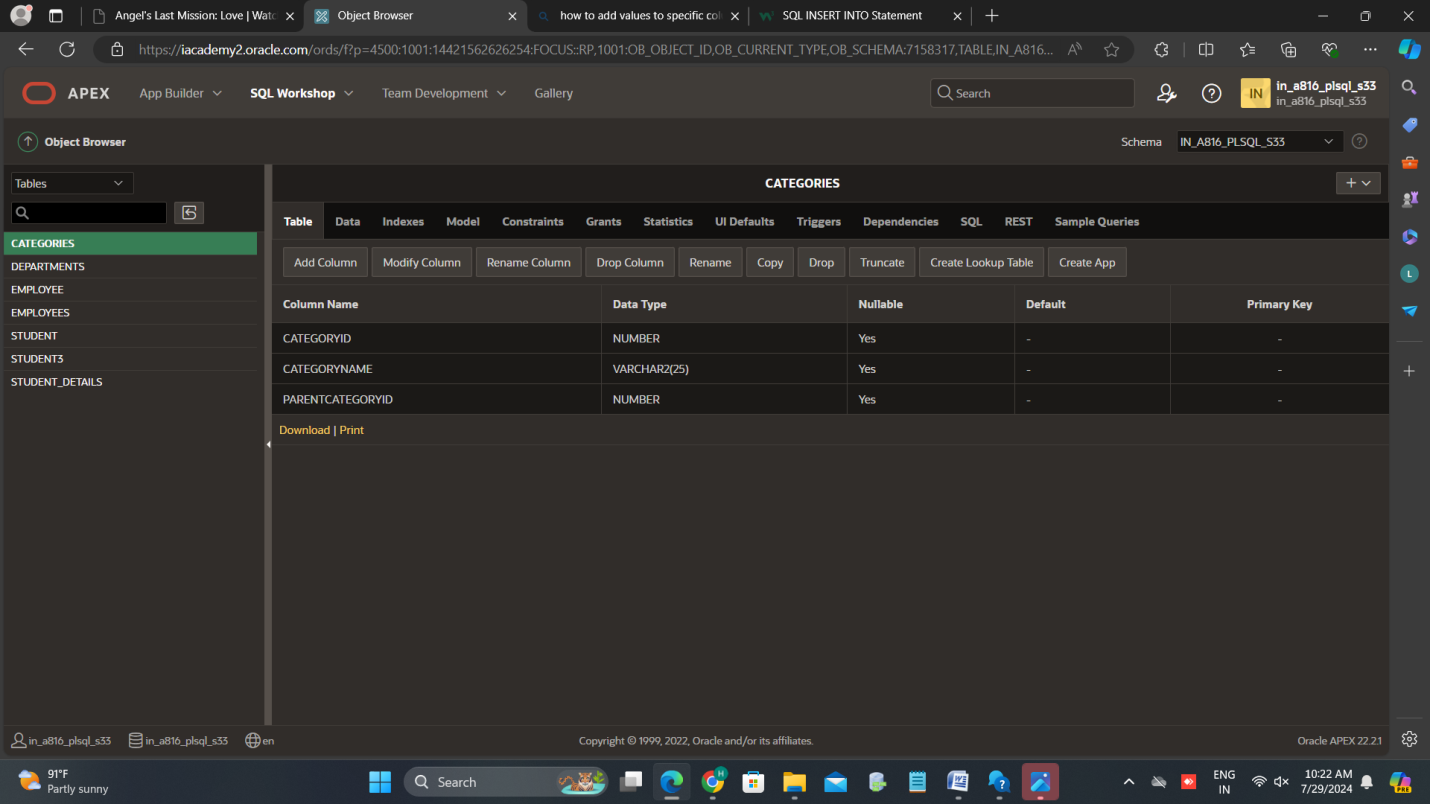
AvgSalary DESC;



2.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).

CREATING SQL QUERY FOR RETRIVE ALL CATEGORIES

CREATE TABLE Categories (CategoryID INT,CategoryName VARCHAR(25),ParentCategoryID INT);



INSERTING VALUES INTO “”CATEGORIES’’

INSERT INTO Categories VALUES(1, 'Electronics', NULL);

INSERT INTO Categories VALUES(2, 'Computers', 1);

INSERT INTO Categories VALUES(3, 'Laptops', 2);

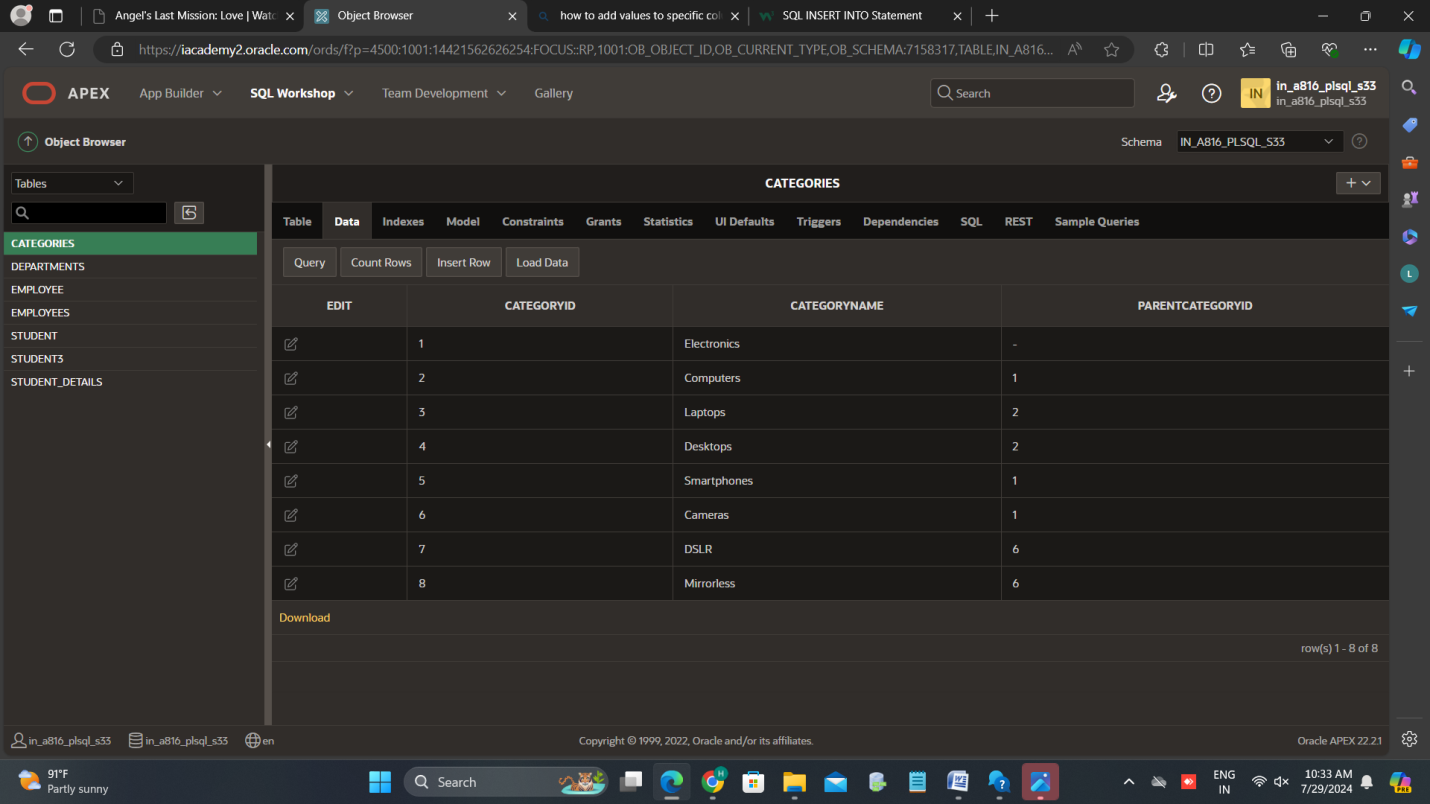
INSERT INTO Categories VALUES(4, 'Desktops', 2);

INSERT INTO Categories VALUES(5, 'Smartphones', 1);

INSERT INTO Categories VALUES(6, 'Cameras', 1);

INSERT INTO Categories VALUES(7, 'DSLR', 6);

INSERT INTO Categories VALUES(8, 'Mirrorless', 6);

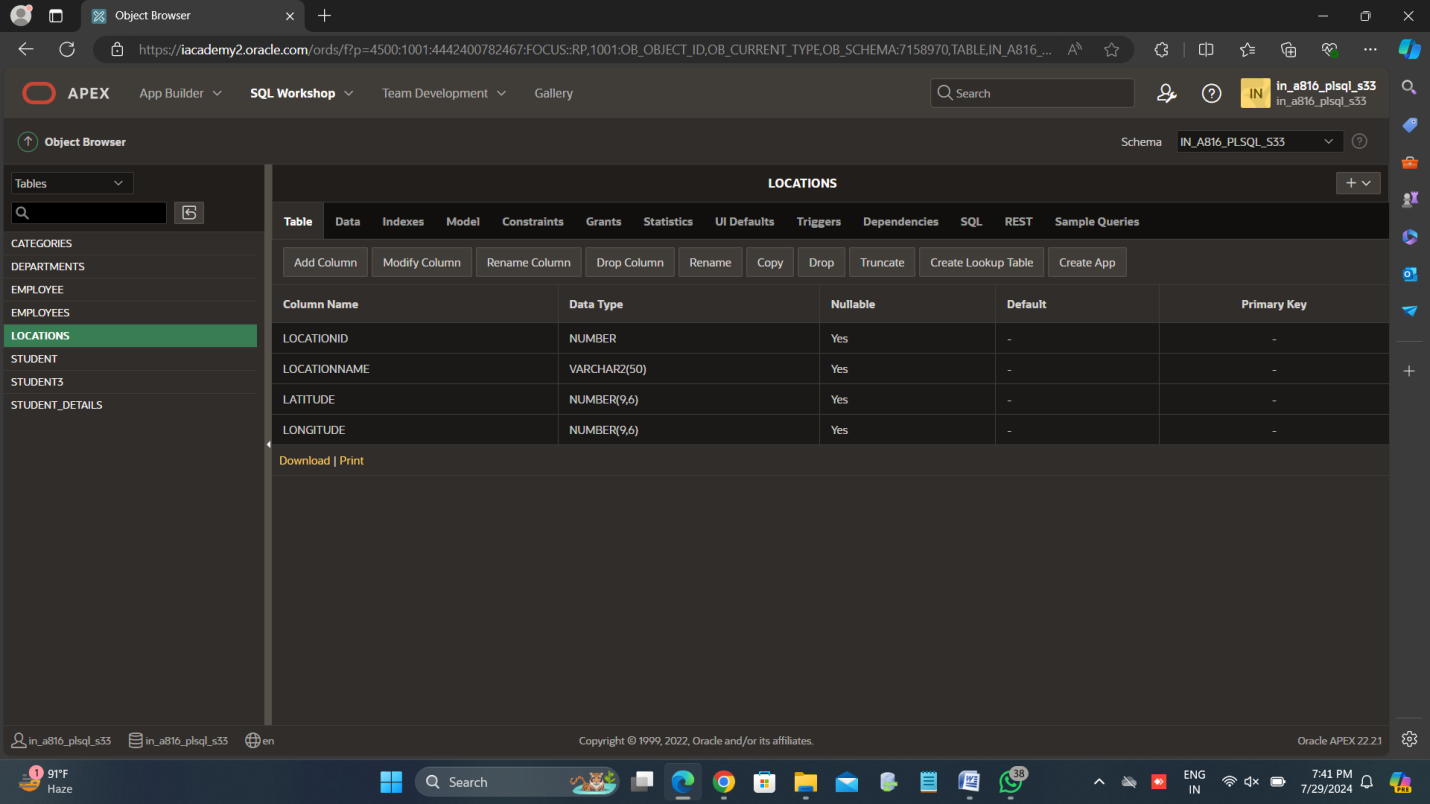


3.Design a SQL query to find the total number of distinct customers who made a purchase in each month of the current year. Ensure months with no customer activity show a count of 0.

4.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.

CREATING A SQL TABLE “LOCATIONS”

CREATE TABLE Locations (LocationID INT,LocationName VARCHAR(50),Latitude DECIMAL(9, 6),Longitude DECIMAL(9, 6));



INSERTING VALUES INTO THE TABLE “LOCATIONS”

INSERT INTO Locations VALUES(1, 'Location A', 40.712776, -74.005974);

INSERT INTO Locations VALUES(2, 'Location B', 34.052235, -118.243683);

INSERT INTO Locations VALUES(3, 'Location C', 51.507351, -0.127758);

INSERT INTO Locations VALUES(4, 'Location D', 48.856613, 2.352222);

INSERT INTO Locations VALUES(5, 'Location E', 35.689487, 139.691711);

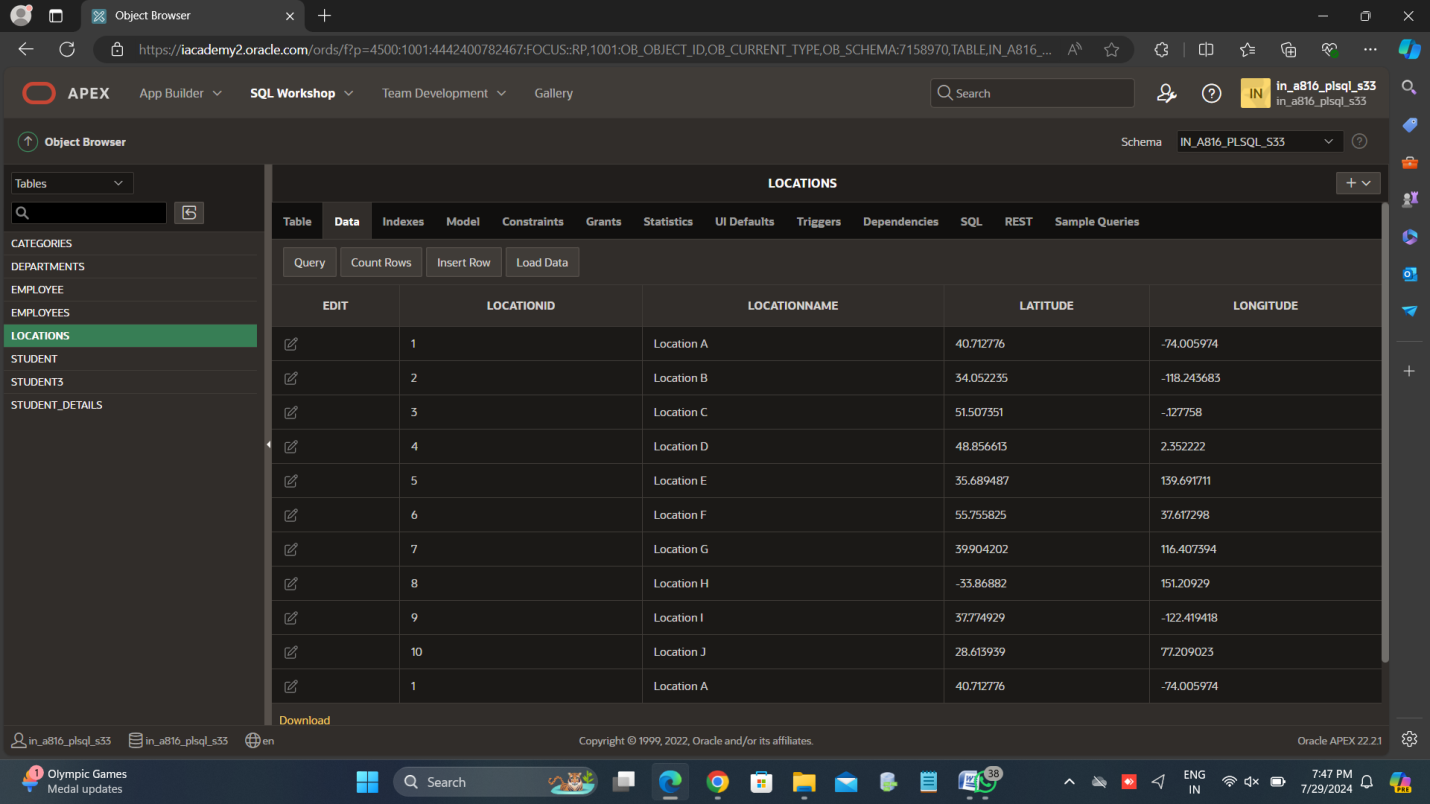
INSERT INTO Locations VALUES(6, 'Location F', 55.755825, 37.617298);

INSERT INTO Locations VALUES(7, 'Location G', 39.904202, 116.407394);

INSERT INTO Locations VALUES(8, 'Location H', -33.868820, 151.209290);

INSERT INTO Locations VALUES(9, 'Location I', 37.774929, -122.419418);

INSERT INTO Locations VALUES(10, 'Location J', 28.613939, 77.209023);



5.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.

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

WITH MonthNames AS (

SELECT

'January' AS MonthName, 1 AS MonthNumber

UNION ALL SELECT 'February', 2

UNION ALL SELECT 'March', 3

UNION ALL SELECT 'April', 4

UNION ALL SELECT 'May', 5

UNION ALL SELECT 'June', 6

UNION ALL SELECT 'July', 7

UNION ALL SELECT 'August', 8

UNION ALL SELECT 'September', 9

UNION ALL SELECT 'October', 10

UNION ALL SELECT 'November', 11

UNION ALL SELECT 'December', 12

),

CustomerCounts AS (

SELECT

MONTH(OrderDate) AS MonthNumber,

COUNT(DISTINCT CustomerID) AS CustomerCount

FROM

Orders

WHERE

YEAR(OrderDate) = YEAR(CURRENT\_DATE)

GROUP BY

MONTH(OrderDate)

)

SELECT

m.MonthName,

COALESCE(c.CustomerCount, 0) AS CustomerCount

FROM

MonthNames m

LEFT JOIN

CustomerCounts c ON m.MonthNumber = c.MonthNumber

ORDER BY

m.MonthNumber;

**Question 4**: Finding Closest Locations SQL Query:

SQL QUERY :

CREATE TABLE Locations (

LocationID INT PRIMARY KEY,

LocationName VARCHAR(255),

Latitude DECIMAL(9, 6),

Longitude DECIMAL(9, 6));

INSERT INTO Locations (LocationID, LocationName, Latitude, Longitude) VALUES

(1, 'Location A', 40.712776, -74.005974),

(2, 'Location B', 34.052235, -118.243683),

(3, 'Location C', 41.878113, -87.629799),

(4, 'Location D', 29.760427, -95.369804),

(5, 'Location E', 51.507351, -0.127758),

(6, 'Location F', 48.856613, 2.352222),

(7, 'Location G', 35.689487, 139.691711);

SET @GivenLatitude = 40.730610;

SET @GivenLongitude = -73.935242;

SELECT

LocationID,

LocationName,

Latitude,

Longitude,

(

3959 \* ACOS(

COS(RADIANS(@GivenLatitude)) \* COS(RADIANS(Latitude)) \* COS(RADIANS(Longitude) - RADIANS(@GivenLongitude)) +

SIN(RADIANS(@GivenLatitude)) \* SIN(RADIANS(Latitude))

)

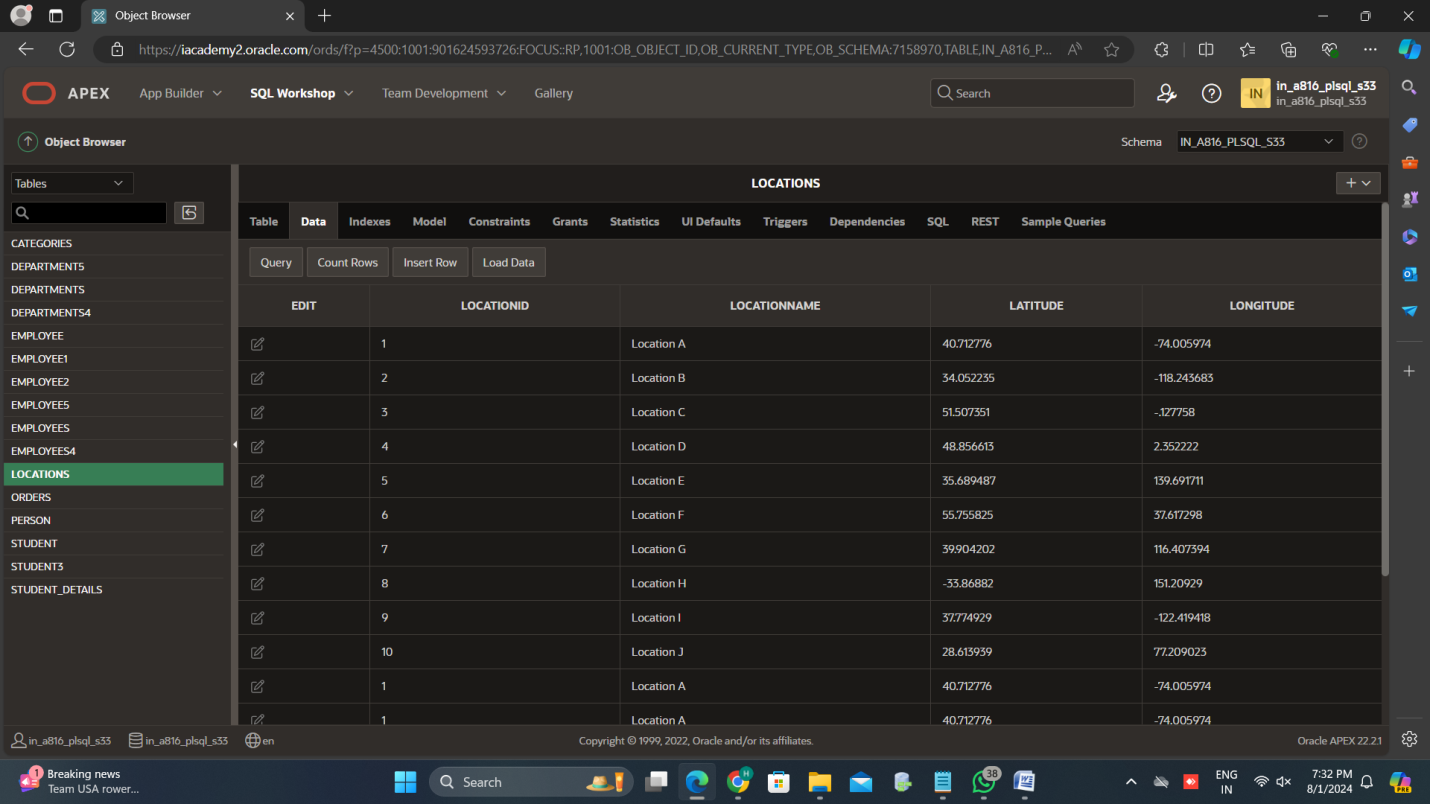
) AS Distance

FROM

Locations

ORDER BY

Distance;

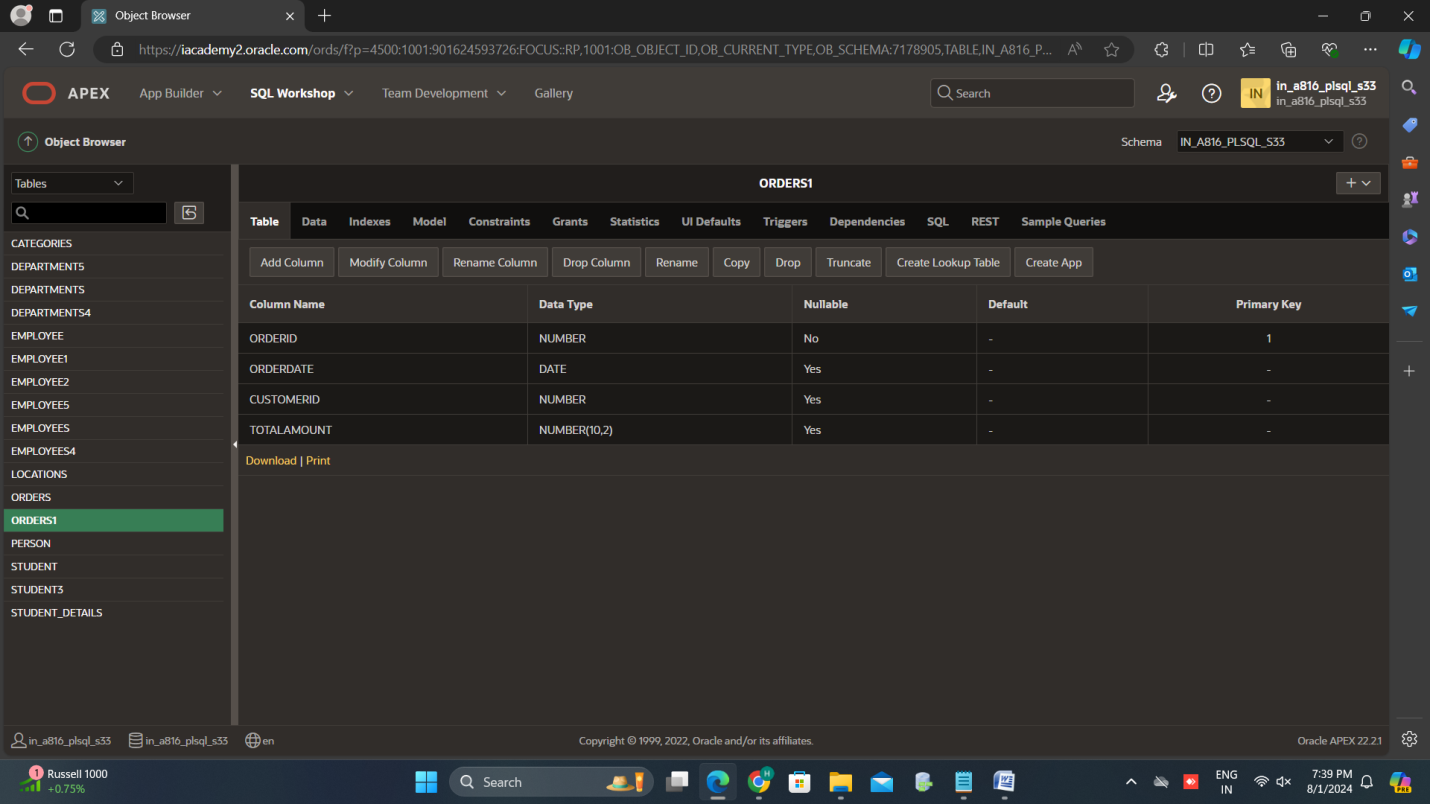


**Question 5:** Optimizing Query for Orders Table

QUESTION: 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.

1.CREATING A TABLE “ORDERS”

CREATE TABLE Orders1 (OrderID INT PRIMARY KEY,OrderDate DATE,CustomerID INT,TotalAmount DECIMAL(10, 2));



2.INSERTING VALUES INTO THE TABLE “ORDERS”

SQL QUERY:

INSERT INTO Orders1 (OrderID, OrderDate, CustomerID, TotalAmount) VALUES

(1, '2024-07-25', 101, 150.75),

(2, '2024-07-26', 102, 200.50),

(3, '2024-07-27', 103, 300.00),

(4, '2024-07-28', 104, 50.25),

(5, '2024-07-29', 105, 400.60),

(6, '2024-07-30', 106, 250.40),

(7, '2024-07-31', 107, 100.00),

(8, '2024-08-01', 108, 500.00),

(9, '2024-08-02', 109, 320.20),

(10, '2024-08-03', 110, 220.75);

**Optimized SQL Query**

SELECT

OrderID,

OrderDate,

CustomerID,

TotalAmount

FROM

Orders

WHERE

OrderDate >= CURRENT\_DATE - INTERVAL 7 DAY

ORDER BY

OrderDate DESC;