CSA0562: DATABASE MANAGEMENT SYSTEM

ASSIGNMENT QUESTIONS

M. Varun Teja
192372083

Question 1:

ER Diagram Question:

Traffic Flow Management System (TFMS) ER Diagram

TASKS

Task 1: Entity Identification and Attributes

Entities and Attributes:

1. ROADS

Attributes:

- RoadID (PK)
- RoadName
- Length (meters)
- SpeedLimit (km/h)

2. INTERSECTIONS

- Attributes:
  - IntersectionID (PK)
  - IntersectionName
  - Latitude
  - Longitude

3. TRAFFIC SIGNALS

- Attributes:
  - SignalID (PK)

- SignalStatus (Green, Yellow, Red)

- Timer

4. TRAFFIC DATA

- Attributes:

  - TrafficDataID (PK)

  - Timestamp

  - Speed (average speed on the road)

  - CongestionLevel

Task 2: Relationship Modeling

Relationships:

1. Roads to Intersections

- Relationship: A road can be part of multiple intersections, and an intersection is formed by multiple roads.

- Cardinality: Many-to-Many

- Optionality: Mandatory (each intersection must be associated with at least one road)

2. Intersections to Traffic Signals

- Relationship: Each intersection can have multiple traffic signals.

- Cardinality: One-to-Many

- Optionality: Mandatory (each intersection must have atleast one traffic signal)

3. Traffic Signals to Traffic Data

- Relationship: Traffic data is collected from sensors related to traffic signals.

- Cardinality: One-to-Many

- Optionality: Optional (traffic data may not always be available for every signal)

4. Roads to Traffic Data

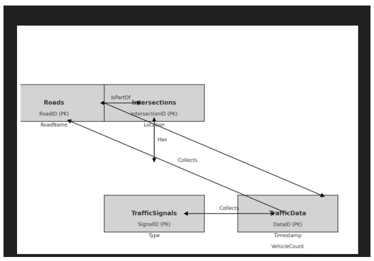
- Relationship:Traffic data is collected for each road.

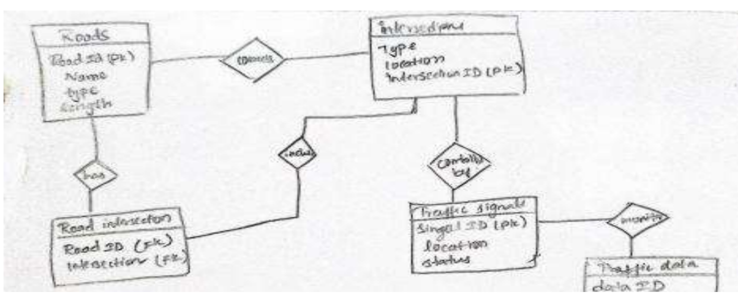
- Cardinality:One-to-Many

- Optionality: Optional (traffic data may not always be available for every road

Task 3: ER Diagram Design

Here's a simplified ER Diagram:





- Roads
  - RoadID (PK)

- RoadName
- Length
- SpeedLimit

- Intersections
  - IntersectionID (PK)
  - IntersectionName
  - Latitude
  - Longitude

- Traffic Signals
  - SignalID (PK)
  - SignalStatus
  - Timer
  - IntersectionID (FK)

- Traffic Data
  - TrafficDataID (PK)
  - Timestamp
  - Speed
  - CongestionLevel
  - RoadID (FK)
  - SignalID (FK)

Relationships:

1. Roads to Intersections:

   - Many-to-Many (through a junction table,e.g., RoadIntersection)

2. Intersections to Traffic Signals:

   - One-to-Many (1 Intersection can have multiple Traffic Signals)

3. Traffic Signals to Traffic Data:

   - One-to-Many (1 Traffic Signal can have multiple Traffic Data records)

4. Roads to Traffic Data:

   - One-to-Many (1 Road can have multiple Traffic Data records)

Task 4: Justification and Normalization

1. Normalization Principles:

- 1NF (First Normal Form): Each table has a primary key, and attributes are atomic.
- 2NF (Second Normal Form): All non-key attributes are fully functional dependent on the primary key.
- 3NF (Third Normal Form): No transitive dependency (attributes are not dependent on other non-key attributes).

2. Design Justification:

- Scalability: The design supports adding new roads, intersections, and traffic signals without major schema changes.
- Real-Time Data Processing: Traffic Data entity captures real-time data for analysis and integration into traffic management algorithms.
- Efficient Traffic Management: The relationships and attributes facilitate efficient retrieval and manipulation of data for route optimization and signal control.

Question 2:

SQL Queries

Question 1: Top 3 Departments with Highest Average Salary

```sql
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 AVG(e.Salary) DESC

FETCH FIRST 3 ROWS ONLY;
```

Explanation:

- `LEFT JOIN` ensures departments with no employees show NULL for `AvgSalary`.

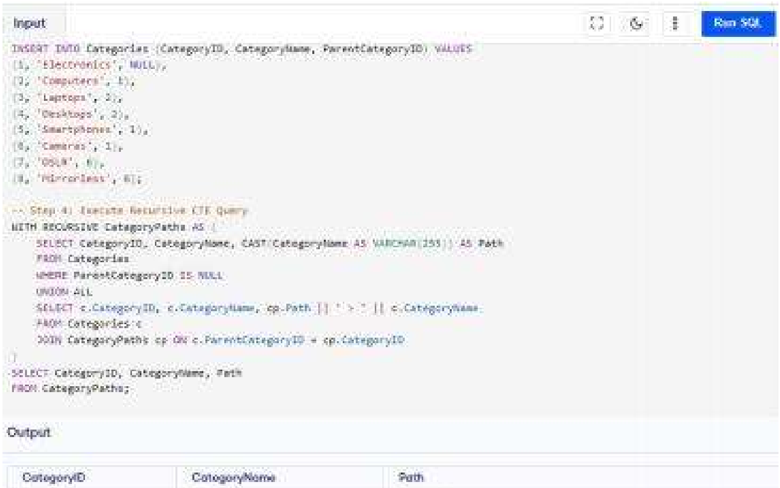
- `GROUP BY` groups data by department.

- `ORDER BY` sorts departments by average salary in descending order.

- `FETCH FIRST 3 ROWS ONLY` limits the result to the top 3 departments.

Question 2: Retrieving Hierarchical Category Paths

```sql
WITH RECURSIVE CategoryPaths AS (
  SELECT CategoryID, CategoryName, CAST(CategoryName AS VARCHAR(255)) AS Path
  FROM Categories
  WHERE ParentCategoryID IS NULL
  UNION ALL
  SELECT c.CategoryID, c.CategoryName, CONCAT(cp.Path, ' > ',c.CategoryName)
  FROM Categories c
  JOIN CategoryPaths cp ON c.ParentCategoryID = cp.CategoryID
)
SELECT CategoryID, CategoryName, Path
FROM CategoryPaths;
```



Explanation:

- `WITH RECURSIVE` defines a CTE that recursively builds the hierarchical path.

- The `UNION ALL` combines the base case with recursive case results.

- `CONCAT` builds the path from parent to child.

Question 3: Total Distinct Customers by Month

```sql
SELECT TO_CHAR(purchase_date, 'Month') AS MonthName,

    COUNT(DISTINCT customer_id) AS CustomerCount

FROM Purchases

WHERE EXTRACT(YEAR FROM purchase_date) = EXTRACT(YEAR FROM CURRENT_DATE)

GROUP BY TO_CHAR(purchase_date, 'Month')

ORDER BY TO_DATE(TO_CHAR(purchase_date, 'Month'), 'Month') ASC;
```

Explanation:

- `TO_CHAR` converts dates to month names.

- `COUNT(DISTINCT customer_id)` counts unique customers.

- `EXTRACT` ensures only the current year's data is considered.

- `ORDER BY` sorts by month.

Question 4: Finding Closest Locations

```sql
SELECT LocationID, LocationName, Latitude, Longitude,
    (3959 * acos(cos(radians(:latitude)) * cos(radians(Latitude)) *
    cos(radians(Longitude) - radians(:longitude)) + sin(radians(:latitude)) *
    sin(radians(Latitude)))) AS Distance
FROM Locations
ORDER BY Distance
FETCH FIRST 5 ROWS ONLY;
```

Explanation:

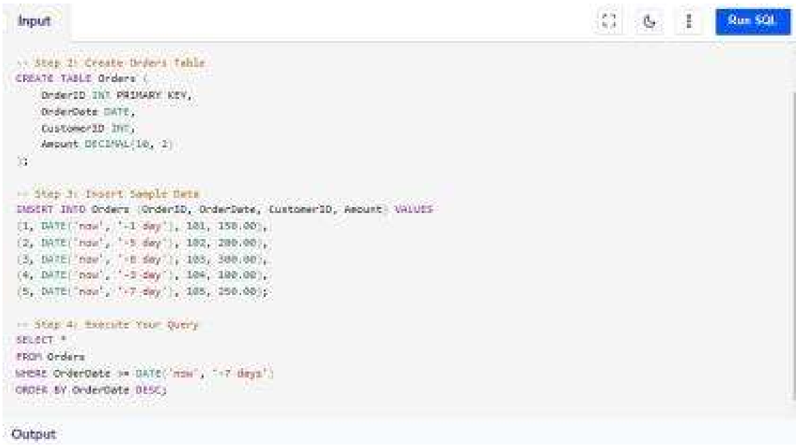
- Haversine formula calculates distance between points.

- `:latitude` and `:longitude` are input parameters.

- `ORDER BY Distance` sorts locations by proximity.

Question 5: Optimizing Query for Orders Table

```sql
SELECT *
FROM Orders
WHERE OrderDate >= SYSDATE - INTERVAL '7' DAY
ORDER BY OrderDate DESC;
```



Explanation:

- `SYSDATE - INTERVAL '7' DAY` retrieves orders from the last 7 days.

- `ORDER BY OrderDate DESC` sorts by the most recent orders.

--

Question 3:

PL/SQL Questions

Question 1: Handling Division Operation

```plsql

DECLARE

   divisor NUMBER := &divisor_input;

   dividend NUMBER := &dividend_input;

   result NUMBER;

BEGIN

   IF divisor = 0 THEN
```

```
      DBMS_OUTPUT.PUT_LINE('Error: Division by zero is not allowed.');

   ELSE

      result := dividend / divisor;

      DBMS_OUTPUT.PUT_LINE('Result: ' || result);

   END IF;

EXCEPTION

   WHEN ZERO_DIVIDE THEN

      DBMS_OUTPUT.PUT_LINE('Error: Division by zero.');

END;
```

Explanation:

- Handles division by zero using an `IF` statement and `ZERO_DIVIDE` exception.

- `DBMS_OUTPUT.PUT_LINE` displays results or error messages.

Question 2: Updating Rows with FORALL

```plsql

DECLARE

   TYPE emp_id_array IS TABLE OF Employees.EmployeeID%TYPE;

   TYPE salary_array IS TABLE OF NUMBER;

   l_emp_ids emp_id_array := emp_id_array(101, 102, 103);

   l_salaries salary_array := salary_array(500, 600, 700);

BEGIN

   FORALL i IN INDICES OF l_emp_ids

      UPDATE Employees

      SET Salary = Salary + l_salaries(i)

      WHERE EmployeeID = l_emp_ids(i);

   COMMIT;
```

```
END;
```

Explanation:

- `FORALL` is used for bulk updates, enhancing performance by reducing context switches between SQL and PL/SQL.

Question 3: Implementing Nested Table Procedure

```plsql
CREATE OR REPLACE PROCEDURE GetEmployeesByDept(p_dept_id IN NUMBER,p_employees OUT SYS_REFCURSOR) AS

BEGIN

  OPEN p_employees FOR

    SELECT * FROM Employees WHERE DepartmentID = p_dept_id;

END;
```

Explanation:

- A procedure that retrieves employees based on department ID and returns themasa cursor.

Question 4: Using Cursor Variables and Dynamic SQL

```plsql

DECLARE

  TYPE emp_ref_cursor IS REF CURSOR;

  l_emp_cursor emp_ref_cursor;

  l_salary_threshold NUMBER := &salary_threshold;

BEGIN

  OPEN l_emp_cursor FOR

    'SELECT EmployeeID, FirstName, LastName
```

```
      FROM Employees
      WHERE Salary > :1'
     USING l_salary_threshold;

  -- Use l_emp_cursor as needed
  CLOSE l_emp_cursor;
END;
```

Explanation:

- Demonstrates use of REF CURSOR and dynamic SQL to query employees based on a salary threshold.

Question 5: Designing P

ipelined Function for Sales Data

```plsql
CREATE OR REPLACE FUNCTION get_sales_data(p_month IN NUMBER,p_year IN NUMBER)
RETURN sales_data_tab_type PIPELINED AS
BEGIN
  FOR rec IN (
    SELECT OrderID, CustomerID, OrderAmount
    FROM Orders
    WHERE EXTRACT(MONTH FROM OrderDate) = p_month
     AND EXTRACT(YEAR FROM OrderDate) = p_year
  ) LOOP
    PIPE ROW (rec);
  END LOOP;
END;
```

```
```

Explanation:

- `PIPELINED` function allows efficient processing of large datasets by returning rows incrementally.

DELIVERABLES

1. ER Diagram:

   - Provides a visual representation of the TFMS entities, attributes, and relationships.

2. Entity Definitions:

   - Clear descriptions of each entity and their attributes.

3. Relationship Descriptions:

   - Details of relationships between entities, including cardinality and optionality.

4. Justification Document:

   - Explanation of design choices, normalization adherence, and considerations for efficiency and scalability.