25/7/24

Assignment - 3.

CSA0563

Data Base Management System.

1. ER Diagram Question : Traffic Flow Management System (TFMS)

ER Diagram Design Requirements :

i. Entities and Attributes.

ii. Relationships.

iii. Normalization Consideration.

Task 1: Entity Identification and Attributes.

1. Roads :

- Attributes :
  - i. Road_Id (Primary key)
  - ii. RoadName
  - iii. Length
  - iv. Speedlimit

2. Intersections.

- Attributes:
  - i. Intersection_Id (PK)
  - ii. IntersectionName
  - iii. Latitude
  - iv. Longitude

3. Traffic Signals.

- Attributes :
  - i. SignalID (PK)
  - ii. Signalstatus (Green, yellow, Red)
  - iii. Timer

- Foreign Key :
  - i. IntersectionID (FK) - to Indicate Intersection the Signal belongs to

4. Traffic Data :

- Attributes :
  - i. TrafficDataID (PK)
  - ii. Timestamp
  - iii. Speed
  - iv. Congestionlevel.

- Foreign Key :
  - i. RoadID (FK) - to Indicate which road the traffic data Pertains to.

Soumiyan.J
192373014
CSE(DS).

Task 2 : Relationship Modeling.

- **Roads and Intersections Relationship :**
  - i. One road can connect to multiple Intersections (one - to - many)
  - ii. One Intersection cannot to multiple roads (one - to - many).

- **Intersections and Traffic Signals Relationship :**
  - i. One Intersection can host multiple traffic signals (one - to - many)
  - ii. Each traffic signal belongs to exactly one Intersection (mandatory relationship).

- **Traffic Data Relationship :**
  - i. Traffic data is associated with a specific road (one - to - many).

2. Question 1:

Top 3 Departments with Highest Average Salary.

Task: Write SQL query to find the top 3 departments with the highest average salary of employees. Ensure departments with no employees shows an average salary of NULL.

```
WITH DeptAvgSalary 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
)
SELECT
    DepartmentID,
    DepartmentName, AvgSalary
FROM DeptAvgSalary
ORDER BY AvgSalary DESC
LIMIT 3;
```

Question 2: Retrieving Hierarchical Category Paths.

Task: Write SQL query using recursive Common Table Expression (CTE) to retrieve all categories along with their full hierarchical path.

```
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 Category_ID, CategoryName, Path AS HierarchicalPath
FROM Category Paths;
```

Questions 3: Total Distinct Customers by Month.

```
SELECT DATE_FORMAT(OrderDate, '%Y-%M') AS MonthName,
       COUNT(DISTINCT CustomerID) AS CustomerCount
FROM Orders
WHERE YEAR(OrderDate) = YEAR(CURRENT_DATE)
GROUP BY MONTH(OrderDate)
ORDER BY MONTH(OrderDate);
```

Question 4: Finding closest Locations.

Task: 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.

```
SELECT LocationID, LocationName, Latitude, Longitude,
    SQRT(POW(Latitude - :given_lat, 2) + POW(Longitude - :given_
                                    long, 2)) AS Distance
FROM Locations
ORDER BY Distance LIMIT 5;
```

Question 5: optimizing query for order table

Task: 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.

```
SELECT ORDERID, OrderDate, customerID, TotalAmount
FROM Orders
WHERE OrderDate >= DATE_SUB(CURRENT_DATE, INTERVAL 7 DAY)
ORDER BY OrderDate DESC;
```

3). Question 1:

Handling Division operation.

Task: Write an SQL block to perform a division operation where the divisor is obtained from user input. Handle the ZERO_DIVIDE exception gracefully with an appropriate error message.

```
DECLARE
    numerator NUMBER := 100;
    denominator NUMBER;
    result NUMBER;
BEGIN
    denominator := &user_input;
    BEGIN
        result := numerator / denominator;
        DBMS_OUTPUT.PUT_LINE('Result: ' || result);
    EXCEPTION
        WHEN ZERO_DIVIDE THEN
            DBMS_OUTPUT.PUT_LINE('Error: Division by zero is
                                    not allowed!');
    END;
END;
```

Question 2 :

Updating Rows for ALL.

Task : Use the FORALL statement to update multiple rows in the Employees table based on arrays of employee IDs and Salary increments.

```
DECLARE
    TYPE emp_id_array IS TABLE OF Employees.EmployeeID%TYPE;
    TYPE Salary_inc_array IS TABLE OF Employees.Salary%TYPE;
    emp_ids emp_id_array := emp_id_array(101,102,103);
    Salary_incs Salary_inc_array := Salary_inc_array(1000,1500,2000);
BEGIN
    FORALL i IN . emp_ids.COUNT
        UPDATE Employees
        SET Salary = Salary + Salary_incs(i)
        WHERE EmployeeID = emp_ids(i);
    COMMIT;
    DBMS_OUTPUT.PUT_LINE ('Salary updates applied successfully');
END;
```

Question 3:

Implementing Nested Table Procedure.

Task : Implement a SQL procedure that accepts a departmentID as input, retrieves employees belonging to the department, stores them in a nested table type, and returns this collection as an output parameter.

```
CREATE OR REPLACE PROCEDURE GetEmployeesByDept(
    P_department_id IN Departments.DepartmentID%TYPE,
    P_deptyee_list OUT SYS_REFCURSOR
)
AS
    TYPE emp_list_type IS TABLE OF Employees%ROWTYPE;
    l_emp_list emp_list_type := emp_list_type();
BEGIN
    SELECT *
    BULK COLLECT INTO l_emp_list
    FROM Employees
    WHERE DepartmentID = P_department_id;

    OPEN P_employee_list FOR
        SELECT *
        FROM TABLE (l_emp_list);
END;
```

Question 4 :

Using cursor Variables and Dynamic SQL.

Task : Write a SQL block demonstrating the use of cursor variables (REF CURSOR) and dynamic SQL. Declare a cursor variable for querying EmployeeID, FFirstName, and LastName based on a specified Salary threshold.

```
DECLARE
   TYPE emp_ref_cursor IS REF CURSOR;
   emp_cursor emp_ref_cursor;
   v_sql VARCHAR2(200);
   v_min_salary NUMBER := 50000;
   v_emp_id Employees.EmployeeID%TYPE;
   v_first_name Employees.firstname%TYPE;
   v_Last_name Employees.LastName%TYPE;

BEGIN
   v_sql := 'SELECT EmployeeID, firstName, LastName FROM Employees
                        WHERE Salary >= :min_salary';
OPEN emp_cursor FOR v_sql USING v_min_salary;

LOOP
   FETCH emp_cursor INTO v_emp_id, v_first_name, v_last_name;
   EXIT WHEN emp_cursor%NOTFOUND;
   DBMS_OUTPUT.PUT_LINE('Employee ID:' || v_emp id || ',Name:' ||
                        V_first_name || ' ' || V_last_name);

END LOOP;
CLOSE emp_cursor;
END;
```

Question 5 :

Designing pipelined function for sales Data

Task : Design a pipelined SQL function get_sales_data that retrieves sales data for a given month and year. The function should return a table of records containing OrderID, CustomerID, and OrderAmount for orders placed in the specified month and year.

```
CREATE OR REPLACE FUNCTION get_sales_data(
   P_month NUMBER,
   P_year NUMBER
) RETURN SYS_REFCURSOR PIPELINED
AS
   TYPE sales_data_rec IS RECORD(
      OrderID Orders.OrderID%TYPE,
      CustomerID Orders.CustomerID%TYPE,
      OrderAmount orders.Total Amount%TYPE
   );
   v_sales_data sales_data_rec;
BEGIN
   FOR rec IN(
      SELECT orderID, customerID, TotalAmount
      FROM orders
      WHERE EXTRACT(YEAR FROM OrderDate) = P_month
      AND EXTRACT(YEAR From Orderdate) = P_year
   )LOOP
      v_sales_data.orderID := rec.orderID;
      v_sales_data.CustomerID := rec.customerID;
      v_sales_data.orderAmount := rec.TotalAmount;
      PIPE ROW(v_sales_data);
END LOOP;
   RETURN;
END;
```