Question 2:

Create Required tables and insert data to perform the below operations.

Department table:

```
1  CREATE TABLE Departments (
2      DepartmentID INT PRIMARY KEY,
3      DepartmentName VARCHAR(255)
4  );
```

Results | Explain | Describe | Saved SQL | History

```
Table created.

0.01 seconds
```

Insert data:

```
1  SELECT * FROM Departments;
```

Results | Explain | Describe | Saved SQL | History

DEPARTMENTID	DEPARTMENTNAME
1	HR
3	Engineering
4	Marketing
2	Finance

4 rows returned in 0.01 seconds Download

Employees Table:

```sql
CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY,
    EmployeeName VARCHAR(255),
    DepartmentID INT,
    Salary DECIMAL(10, 2),
    FOREIGN KEY (DepartmentID) REFERENCES Departments(DepartmentID)
);
```

Results | Explain | Describe | Saved SQL | History

```
Table created.

0.02 seconds
```

Insert data:

```sql
SELECT * FROM Employees;
```

Results | Explain | Describe | Saved SQL | History

EMPLOYEEID	EMPLOYEENAME	DEPARTMENTID	SALARY
3	Charlie	2	55000
4	David	3	70000
1	Alice	1	50000
2	Bob	2	60000
5	Eve	3	65000

5 rows returned in 0.01 seconds Download

Categories Table:

```sql
CREATE TABLE Categories (
    CategoryID INT PRIMARY KEY,
    CategoryName VARCHAR(255),
    ParentCategoryID INT,
    FOREIGN KEY (ParentCategoryID) REFERENCES Categories(CategoryID)
);
```

Results | Explain | Describe | Saved SQL | History

```
Table created.
```

0.01 seconds

Insert data:

```sql
SELECT * FROM Categories;
```

Results | Explain | Describe | Saved SQL | History

CATEGORYID	CATEGORYNAME	PARENTCATEGORYID
1	Electronics	-
2	Computers	1
5	Smartphones	1
3	Laptops	2
4	Desktops	2

5 rows returned in 0.00 seconds Download

And also create location customers and order tables.

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.

```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
    AvgSalary DESC
FETCH FIRST 3 ROWS ONLY;
```

Results | Explain | Describe | Saved SQL | History

DEPARTMENTID	DEPARTMENTNAME	AVGSALARY
4	Marketing	-
3	Engineering	67500
2	Finance	57500

3 rows returned in 0.01 seconds Download

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).

Deliverables:

1. SQL query that uses recursive CTE to fetch CategoryID, CategoryName, and hierarchical path.
2. Explanation of how the recursive CTE works to traverse the hierarchical data.

```sql
SELECT
    CategoryID,
    CategoryName,
    SYS_CONNECT_BY_PATH(CategoryName, ' > ') AS Path
FROM
    Categories
START WITH
    ParentCategoryID IS NULL
CONNECT BY
    PRIOR CategoryID = ParentCategoryID;
```

Results | Explain | Describe | Saved SQL | History

CATEGORYID	CATEGORYNAME	PATH
1	Electronics	> Electronics
2	Computers	> Electronics > Computers
3	Laptops	> Electronics > Computers > Laptops
4	Desktops	> Electronics > Computers > Desktops
5	Smartphones	> Electronics > Smartphones

5 rows returned in 0.01 seconds Download

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

```sql
WITH Months AS (
        SELECT ADD_MONTHS(TRUNC(SYSDATE, 'YEAR'), LEVEL - 1) AS MonthStart
FROM DUAL
    CONNECT BY LEVEL <= 12
)
SELECT
    TO_CHAR(MonthStart, 'Month') AS MonthName,
    COALESCE(
(SELECT COUNT(DISTINCT CustomerID)
FROM Orders
WHERE OrderDate >= MonthStart
    AND OrderDate < ADD_MONTHS(MonthStart, 1)),
    0
) AS CustomerCount
FROM
    Months
ORDER BY
    MonthStart;
```

Results | Explain | Describe | Saved SQL | History

MONTHNAME	CUSTOMERCOUNT
January	1
February	2
March	2
April	0
May	0

in_a813_plsql_s13 in_a813_plsql_s13 en Ora

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.

```sql
SELECT
    LocationID,
    LocationName,
    Latitude,
    Longitude,
    (6371 * 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;
```

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

```sql
SELECT
    OrderID,
    CustomerID,
    OrderDate,
    TotalAmount
FROM
    Orders
WHERE
    OrderDate >= CURRENT_DATE - INTERVAL '7 days'
ORDER BY
    OrderDate DESC;
```

Question 3:

PL/SQL Questions

Question 1: Handling Division Operation

Task:

1. Write a PL/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.

Deliverables:

1. PL/SQL block that performs the division operation and handles exceptions.
2. Explanation of error handling strategies implemented.

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

Question 2: Updating Rows with FORALL

Task:

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

Deliverables:

1. PL/SQL block that uses FORALL to update salaries efficiently.
2. Description of how FORALL improves performance for bulk updates.

```plsql
DECLARE
    TYPE emp_id_array IS TABLE OF employees.employee_id%TYPE;
    TYPE salary_inc_array IS TABLE OF NUMBER;
    emp_ids emp_id_array := emp_id_array(101, 102, 103);
    salary_incs salary_inc_array := salary_inc_array(500, 1000, 1500);
BEGIN
    FORALL i IN 1..emp_ids.COUNT
UPDATE employees
SET salary = salary + salary_incs(i)
WHERE employee_id = emp_ids(i);
    DBMS_OUTPUT.PUT_LINE('Salaries updated successfully.');
END;
```

Question 3: Implementing Nested Table Procedure

Task:

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

Deliverables:

1. PL/SQL procedure with nested table implementation.
2. Explanation of how nested tables are utilized and returned as output.

```plsql
CREATE OR REPLACE TYPE emp_table AS TABLE OF employees%ROWTYPE;

CREATE OR REPLACE PROCEDURE get_employees_by_dept (
    p_dept_id IN employees.department_id%TYPE,
    p_emp_list OUT emp_table
) IS
BEGIN
SELECT * BULK COLLECT INTO p_emp_list
FROM employees
WHERE department_id = p_dept_id;
END;
```

Question 4: Using Cursor Variables and Dynamic SQL

Task:

1. Write a PL/SQL block demonstrating the use of cursor variables (REF CURSOR) and dynamic SQL. Declare a cursor variable for querying EmployeeID, FirstName, and LastName based on a specified salary threshold.

Deliverables:

1. PL/SQL block that declares and uses cursor variables with dynamic SQL.
2. Explanation of how dynamic SQL is constructed and executed.

```
DECLARE
    TYPE ref_cursor IS REF CURSOR;
    c_ref_cursor ref_cursor;
    v_employee_id employees.employee_id%TYPE;
    v_first_name employees.first_name%TYPE;
    v_last_name employees.last_name%TYPE;
    v_salary_threshold NUMBER := &salary_threshold;
BEGIN
    OPEN c_ref_cursor FOR
            'SELECT employee_id, first_name, last_name FROM employees WHERE salary > ' || v_salary_threshold;
LOOP
    FETCH c_ref_cursor INTO v_employee_id, v_first_name, v_last_name;
    EXIT WHEN c_ref_cursor%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE('Employee ID: ' || v_employee_id || ', Name: ' || v_first_name || ' ' || v_last_name);
END LOOP;
    CLOSE c_ref_cursor;
END;
```

Question 5: Designing Pipelined Function for Sales Data

Task:

1. Design a pipelined PL/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.

Deliverables:

1. PL/SQL code for the pipelined function get_sales_data.
2. Explanation of how pipelined table functions improve data retrieval efficiency.

```sql
CREATE OR REPLACE TYPE sales_record AS OBJECT (
    order_id NUMBER,
    customer_id NUMBER,
    order_amount NUMBER
    );

    CREATE OR REPLACE TYPE sales_table AS TABLE OF sales_record;

    CREATE OR REPLACE FUNCTION get_sales_data (
        p_month IN NUMBER,
        p_year IN NUMBER
        ) RETURN sales_table PIPELINED IS
            v_sales_rec sales_record;
            BEGIN
                FOR rec IN (SELECT order_id, customer_id, order_amount
                            FROM sales
                            WHERE EXTRACT(MONTH FROM order_date) = p_month
                            AND EXTRACT(YEAR FROM order_date) = p_year)
                            LOOP
                                v_sales_rec := sales_record(rec.order_id, rec.customer_id, rec.order_amount);
                                PIPE ROW (v_sales_rec);
                                END LOOP;

                                RETURN;
                                END;

)
```

```sql
DECLARE
    TYPE ref_cursor IS REF CURSOR;
    c_ref_cursor ref_cursor;
    v_employee_id employees.employee_id%TYPE;
    v_first_name employees.first_name%TYPE;
    v_last_name employees.last_name%TYPE;
    v_salary_threshold NUMBER := &salary_threshold;
BEGIN
    OPEN c_ref_cursor FOR
        'SELECT employee_id, first_name, last_name FROM employees WHERE salary > ' || v_salary_threshold;
LOOP
    FETCH c_ref_cursor INTO v_employee_id, v_first_name, v_last_name;
    EXIT WHEN c_ref_cursor%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE('Employee ID: ' || v_employee_id || ', Name: ' || v_first_name || ' ' || v_last_name);
END LOOP;
    CLOSE c_ref_cursor;
END;
```

Rubrics

Criteria	Description	Percentage
Conceptual Understanding	Demonstrates clear understanding of the problem domain (e.g., traffic flow management for ER Diagram, data retrieval and manipulation for SQL/PLSQL).	25%
Technical Accuracy	Accuracy in designing the ER Diagram or writing SQL/PLSQL queries, ensuring they meet requirements and handle edge cases effectively.	30%
Documentation and Clarity	Quality of documentation, including clarity of explanations, use of appropriate terminology, and organization of diagrams or code.	25%
Design and Solution Justification	Justification of design choices (e.g., normalization in ER Diagram, query optimization in SQL/PLSQL) with clear reasoning and considerations for scalability or efficiency.	20%