1.)Traffic Flow Management System (TFMS) Scenario You are tasked with designing an Entity-Relationship (ER) diagram for a Traffic Flow Management System (TFMS) used in a city to optimize traffic routes, manage intersections, and control traffic signals. The TFMS aims to enhance transportation efficiency by utilizing real-time data from sensors and historical traffic patterns. The city administration has decided to implement a TFMS to address growing traffic congestion issues. The system will integrate real-time data from traffic sensors, cameras, and historical traffic patterns to provide intelligent traffic management solutions. Key functionalities include: 1. Road Network Management: o Roads: The city has a network of roads, each identified by a unique RoadID. Roads have attributes such as RoadName, Length (in meters), and SpeedLimit (in km/h). 2. Intersection Control: o Intersections: These are key points where roads meet and are crucial for traffic management. Each intersection is uniquely identified by IntersectionID and has attributes like IntersectionName and geographic Coordinates (Latitude, Longitude). 3. Traffic Signal Management: o Traffic Signals: Installed at intersections to regulate traffic flow. Each signal is identified by SignalID and has attributes such as SignalStatus (Green, Yellow, Red) indicating current state and Timer (countdown to next change). 4. Real-Time Data Integration: o Traffic Data: Real-time data collected from sensors includes TrafficDataID, Timestamp, Speed (average speed on the road), and CongestionLevel (degree of traffic congestion). 5. Functionality Requirements: o Route Optimization: Algorithms will be implemented to suggest optimal routes based on current traffic conditions. o Traffic Signal Control: Adaptive control algorithms will adjust signal timings dynamically based on real-time traffic flow and congestion data. o Historical Analysis: The system will store historical traffic data for analysis and planning future improvements. ER Diagram Design Requirements 1. Entities and Attributes: o Clearly define entities (Roads, Intersections, Traffic Signals, Traffic Data) and their attributes based on the scenario provided. o Include primary keys (PK) and foreign keys (FK) where necessary to establish relationships between entities. 2. Relationships: o Illustrate relationships between entities (e.g., Roads connecting to Intersections, Intersections hosting Traffic Signals). o Specify cardinality (one-to-one, one-to-many, many-to-many) and optionality constraints (mandatory vs. optional relationships). 3. Normalization Considerations: o Discuss how you would ensure the ER diagram adheres to normalization principles (1NF, 2NF, 3NF) to minimize redundancy and improve data integrity. 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. 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). 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. Deliverables 1. ER Diagram: A well-drawn ER diagram that accurately reflects the structure and relationships of the TFMS database. 2. Entity Definitions: Clear definitions of entities and their attributes, supporting the ER diagram. 3. Relationship Descriptions: Detailed descriptions of relationships with cardinality and optionality constraints. 4. Justification Document: A document explaining design choices, normalization considerations, and how the ER diagram supports TFMS functionalitie

ANSWER:

**Task 1: Entity Identification and Attributes**

Based on the scenario provided for the Traffic Flow Management System (TFMS), the following entities and their attributes can be identified:

1. **Roads**
   * Attributes: RoadID (PK), RoadName, Length, SpeedLimit
2. **Intersections**
   * Attributes: IntersectionID (PK), IntersectionName, Latitude, Longitude
3. **Traffic Signals**
   * Attributes: SignalID (PK), SignalStatus (Green, Yellow, Red), Timer (countdown to next change), IntersectionID (FK)
4. **Traffic Data**
   * Attributes: TrafficDataID (PK), Timestamp, Speed, CongestionLevel, RoadID (FK)

**Task 2: Relationship Modeling**

* **Roads <-> Intersections**
  + Relationship: One road connects to many intersections.
  + Cardinality: One-to-many (1

) from Roads to Intersections.

* + Optionality: Mandatory from Roads to Intersections (a road must connect to at least one intersection).
* **Intersections <-> Traffic Signals**
  + Relationship: Each intersection hosts multiple traffic signals.
  + Cardinality: One-to-many (1

) from Intersections to Traffic Signals.

* + Optionality: Optional from Intersections to Traffic Signals (an intersection may not have any traffic signals initially).
* **Roads <-> Traffic Data**
  + Relationship: Traffic data is collected for various roads.
  + Cardinality: One-to-many (1

) from Roads to Traffic Data.

* + Optionality: Optional from Roads to Traffic Data (traffic data may not be available for all roads at all times).

**Task 3: ER Diagram Design**

**ER Diagram Components:**

* **Entities**: Roads, Intersections, Traffic Signals, Traffic Data.
* **Attributes**: Clearly labeled within each entity box.
* **Relationships**: Lines connecting entities with appropriate cardinality indicators (1, M) and optionalities (0..1, 1).

**Primary Keys (PK) and Foreign Keys (FK)**:

* Primary keys are denoted with (PK) next to the attribute.
* Foreign keys are shown with (FK) next to the attribute that references the primary key in another table.

**Task 4: Justification and Normalization**

**Design Choices Justification:**

* **Scalability**: Designed to handle growth in the number of roads, intersections, and traffic signals dynamically.
* **Real-time Data Processing**: Enables efficient capture and utilization of real-time traffic data for dynamic traffic management.
* **Efficient Traffic Management**: Supports route optimization and adaptive traffic signal control.

**Normalization Considerations:**

* **1NF (First Normal Form)**: Attributes are atomic; each attribute contains only a single value.
* **2NF (Second Normal Form)**: No partial dependencies; every non-key attribute is fully functionally dependent on the primary key.
* **3NF (Third Normal Form)**: No transitive dependencies; all non-key attributes are dependent only on the primary key.

**Normalization Application:**

* Ensures data integrity and reduces redundancy by organizing data into well-structured tables with minimal data duplication.
* Reduces update anomalies and supports efficient querying of historical and real-time traffic data.

This ER diagram and associated design considerations provide a robust foundation for implementing the TFMS, facilitating effective traffic management through integration of real-time data and historical analysis.

Top of Form

Bottom of Form

2.) Question 2: SQL 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. 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 > Subsubcategory). 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. 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. 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. 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.

ANSWER:

**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.

**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 Query:**

sql

Copy code

WITH DepartmentAvgSalary 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

DepartmentAvgSalary

ORDER BY

AvgSalary DESC

LIMIT 3;

**Explanation:**

* **CTE (Common Table Expression)** named DepartmentAvgSalary: This calculates the average salary (AvgSalary) for each department (DepartmentID, DepartmentName). The LEFT JOIN ensures that all departments are included, even those without employees (where AVG(e.Salary) would result in NULL).
* The main query then selects DepartmentID, DepartmentName, and AvgSalary from DepartmentAvgSalary, ordered by AvgSalary in descending order (DESC), and limits the result to the top 3 departments using LIMIT 3.

**Question 2: Retrieving Hierarchical Category Paths**

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

**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 Query:**

sql

Copy code

WITH RECURSIVE CategoryPath AS (

SELECT

CategoryID,

CategoryName,

CAST(CategoryName AS VARCHAR(1000)) AS Path

FROM

Categories

WHERE

ParentCategoryID IS NULL

UNION ALL

SELECT

c.CategoryID,

c.CategoryName,

CONCAT(cp.Path, ' > ', c.CategoryName)

FROM

Categories c

JOIN

CategoryPath cp ON c.ParentCategoryID = cp.CategoryID

)

SELECT

CategoryID,

CategoryName,

Path AS HierarchicalPath

FROM

CategoryPath;

**Explanation:**

* **Recursive CTE CategoryPath**:
  + The initial part selects root categories (ParentCategoryID IS NULL) and assigns their own name as the starting path (Path).
  + The recursive part (UNION ALL) joins Categories with CategoryPath where ParentCategoryID matches CategoryID, appending the current category name to the existing path (Path).
* The final SELECT statement retrieves CategoryID, CategoryName, and the constructed Path which represents the hierarchical path from root to each category.

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

**Task:** 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 Query:**

sql

Copy code

SELECT

m.MonthName,

COUNT(DISTINCT o.CustomerID) AS CustomerCount

FROM

Months m

LEFT JOIN

Orders o ON m.MonthNumber = MONTH(o.OrderDate) AND YEAR(o.OrderDate) = YEAR(CURRENT\_DATE)

GROUP BY

m.MonthName

ORDER BY

m.MonthNumber;

**Explanation:**

* **Month Table (Months)**: Contains all months (MonthName, MonthNumber) ensuring all months are included, even if no orders were placed.
* **LEFT JOIN**: Joins Months with Orders on matching month and year (MONTH(o.OrderDate) and YEAR(o.OrderDate)), counting distinct CustomerID.
* **COUNT(DISTINCT o.CustomerID)**: Counts distinct customers per month.
* **Handling Zero Customer Counts**: LEFT JOIN ensures months without orders still appear in the result set, and COUNT(DISTINCT ...) handles zero customer counts naturally by returning 0 for such months.

**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.

**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 Query:**

sql

Copy code

SELECT TOP 5

LocationID,

LocationName,

Latitude,

Longitude,

SQRT(POWER(Latitude - @givenLatitude, 2) + POWER(Longitude - @givenLongitude, 2)) AS Distance

FROM

Locations

ORDER BY

Distance;

**Explanation:**

* **Mathematical Approach**: Uses the Euclidean distance formula (SQRT(POWER(Latitude - @givenLatitude, 2) + POWER(Longitude - @givenLongitude, 2))) to calculate the distance between each location and the given point (@givenLatitude, @givenLongitude).
* **ORDER BY Distance**: Sorts locations by calculated distance in ascending order.
* **TOP 5**: Limits the result set to the closest 5 locations.

**Question 5: Optimizing Query for Orders 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.

**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.

**Optimized SQL Query:**

sql

Copy code

SELECT

OrderID,

OrderDate,

CustomerID,

TotalAmount

FROM

Orders

WHERE

OrderDate >= DATEADD(DAY, -7, CURRENT\_DATE)

ORDER BY

OrderDate DESC;

**Optimization Strategies:**

* **Indexing**: Ensure there is an index on OrderDate to facilitate fast retrieval of records within the specified date range.
* **Query Rewriting**: Simplify the WHERE clause condition to use a direct date comparison (OrderDate >= ...) rather than complex date functions which can be costly.
* **Performance Testing**: Evaluate query performance using database query execution plans and adjust indexing strategies based on performance metrics.

This optimized query efficiently retrieves recent orders by leveraging indexing and straightforward date comparisons, ensuring optimal performance even with a large Orders table.

Top of Form

Bottom of Form

3.) 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. 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. 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. 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. 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. Evaluation 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%

ANSWER:

**Task:** 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.

**PL/SQL Block:**

sql

Copy code

DECLARE

v\_dividend NUMBER := 100;

v\_divisor NUMBER;

v\_result NUMBER;

BEGIN

-- Get divisor from user input (simulate user input)

v\_divisor := &divisor\_input;

-- Perform division

v\_result := v\_dividend / v\_divisor;

-- Display result

DBMS\_OUTPUT.PUT\_LINE('Result of division: ' || v\_result);

EXCEPTION

WHEN ZERO\_DIVIDE THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Division by zero is not allowed.');

END;

/

**Explanation:**

* **Error Handling Strategy:**
  + The ZERO\_DIVIDE exception is anticipated because division by zero is not allowed in SQL.
  + Inside the exception block (WHEN ZERO\_DIVIDE), an appropriate error message is printed using DBMS\_OUTPUT.PUT\_LINE.
  + This ensures that if the user inputs zero as the divisor, the exception is caught, and a clear error message is displayed.

**Question 2: Updating Rows with FORALL**

**Task:** 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.

**PL/SQL Block:**

sql

Copy code

DECLARE

TYPE emp\_id\_array IS TABLE OF Employees.EmployeeID%TYPE;

TYPE salary\_inc\_array IS TABLE OF Employees.Salary%TYPE;

v\_emp\_ids emp\_id\_array := emp\_id\_array(101, 102, 103); -- Example employee IDs

v\_salary\_inc salary\_inc\_array := salary\_inc\_array(1000, 1500, 2000); -- Example salary increments

BEGIN

FORALL i IN 1..v\_emp\_ids.COUNT

UPDATE Employees

SET Salary = Salary + v\_salary\_inc(i)

WHERE EmployeeID = v\_emp\_ids(i);

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Salaries updated successfully.');

END;

/

**Description:**

* **FORALL Statement:**
  + FORALL allows bulk DML operations (like updates) to be performed more efficiently than traditional FOR loops.
  + It minimizes context switches between the PL/SQL and SQL engines, improving performance significantly for large datasets.
* **Arrays (emp\_id\_array, salary\_inc\_array):**
  + These arrays hold employee IDs and corresponding salary increments.
  + FORALL i IN 1..v\_emp\_ids.COUNT iterates over the arrays, updating salaries based on the indexed values.

**Question 3: Implementing Nested Table Procedure**

**Task:** 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.

**PL/SQL Procedure:**

sql

Copy code

CREATE OR REPLACE PROCEDURE GetEmployeesByDept(

p\_department\_id IN Departments.DepartmentID%TYPE,

p\_employee\_list OUT SYS.ODCINumberList

)

IS

BEGIN

SELECT EmployeeID

BULK COLLECT INTO p\_employee\_list

FROM Employees

WHERE DepartmentID = p\_department\_id;

END;

/

**Explanation:**

* **Nested Table (SYS.ODCINumberList):**
  + SYS.ODCINumberList is a built-in Oracle type for nested tables of numbers (in this case, EmployeeIDs).
  + BULK COLLECT INTO p\_employee\_list collects all EmployeeIDs that match the input p\_department\_id into the output parameter p\_employee\_list.
* **Procedure Usage:**
  + Call this procedure with a department ID, and it will populate p\_employee\_list with EmployeeIDs belonging to that department.

**Question 4: Using Cursor Variables and Dynamic SQL**

**Task:** 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.

**PL/SQL Block:**

sql

Copy code

DECLARE

TYPE emp\_cursor IS REF CURSOR;

v\_emp\_cursor emp\_cursor;

v\_min\_salary NUMBER := 50000; -- Example salary threshold

v\_sql\_stmt VARCHAR2(200);

v\_emp\_id Employees.EmployeeID%TYPE;

v\_first\_name Employees.FirstName%TYPE;

v\_last\_name Employees.LastName%TYPE;

BEGIN

-- Dynamic SQL statement

v\_sql\_stmt := 'SELECT EmployeeID, FirstName, LastName FROM Employees WHERE Salary >= :1';

-- Open cursor variable with dynamic SQL

OPEN v\_emp\_cursor FOR v\_sql\_stmt USING v\_min\_salary;

-- Fetch data from cursor

LOOP

FETCH v\_emp\_cursor INTO v\_emp\_id, v\_first\_name, v\_last\_name;

EXIT WHEN v\_emp\_cursor%NOTFOUND;

-- Process fetched data

DBMS\_OUTPUT.PUT\_LINE('Employee ID: ' || v\_emp\_id || ', Name: ' || v\_first\_name || ' ' || v\_last\_name);

END LOOP;

-- Close cursor variable

CLOSE v\_emp\_cursor;

END;

/

**Explanation:**

* **Cursor Variable (emp\_cursor):**
  + TYPE emp\_cursor IS REF CURSOR; declares a cursor variable.
  + OPEN v\_emp\_cursor FOR v\_sql\_stmt USING v\_min\_salary; opens the cursor variable with a dynamically constructed SQL statement (v\_sql\_stmt).
* **Dynamic SQL (v\_sql\_stmt):**
  + v\_sql\_stmt := 'SELECT EmployeeID, FirstName, LastName FROM Employees WHERE Salary >= :1'; defines the SQL query with a bind variable (:1) for the salary threshold.
  + USING v\_min\_salary binds the value of v\_min\_salary to the bind variable.
* **Execution:**
  + The cursor variable v\_emp\_cursor is fetched in a loop, processing each row fetched until v\_emp\_cursor%NOTFOUND.
  + DBMS\_OUTPUT.PUT\_LINE prints each EmployeeID, FirstName, and LastName retrieved by the cursor.

**Question 5: Designing Pipelined Function for Sales Data**

**Task:** 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.

**PL/SQL Pipelined Function:**

sql

Copy code

CREATE OR REPLACE FUNCTION get\_sales\_data(

p\_month NUMBER,

p\_year NUMBER

) RETURN SYS.ODCITabList PIPELINED

IS

v\_order\_id Orders.OrderID%TYPE;

v\_customer\_id Orders.CustomerID%TYPE;

v\_order\_amount Orders.OrderAmount%TYPE;

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;

RETURN;

END;

/

**Explanation:**

* **Pipelined Function (get\_sales\_data):**
  + RETURN SYS.ODCITabList PIPELINED indicates that the function returns a collection of rows (table) pipelined.
  + The function queries Orders for OrderID, CustomerID, and OrderAmount where OrderDate matches the input p\_month and p\_year.
  + PIPE ROW (rec); pipes each row fetched from the cursor (rec) into the pipelined table function result.
* **Efficiency Improvement:**
  + Pipelined functions improve efficiency by streaming rows directly to the caller as they are produced, rather than collecting all rows in memory first.
  + This reduces memory usage and improves performance, especially for large result sets.

These solutions provide comprehensive implementations and explanations for each PL/SQL task, ensuring clarity and adherence to best practices in PL/SQL development.

Top of Form

Bottom of Form