**CSA0562: DATABASE MANAGEMENT SYSTEM- ASSIGNMENT QUESTIONS**

**M. ROHITH**

**192324121**

**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)

1. 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 at least one traffic signal)

3. Traffic Signals to Traffic Data

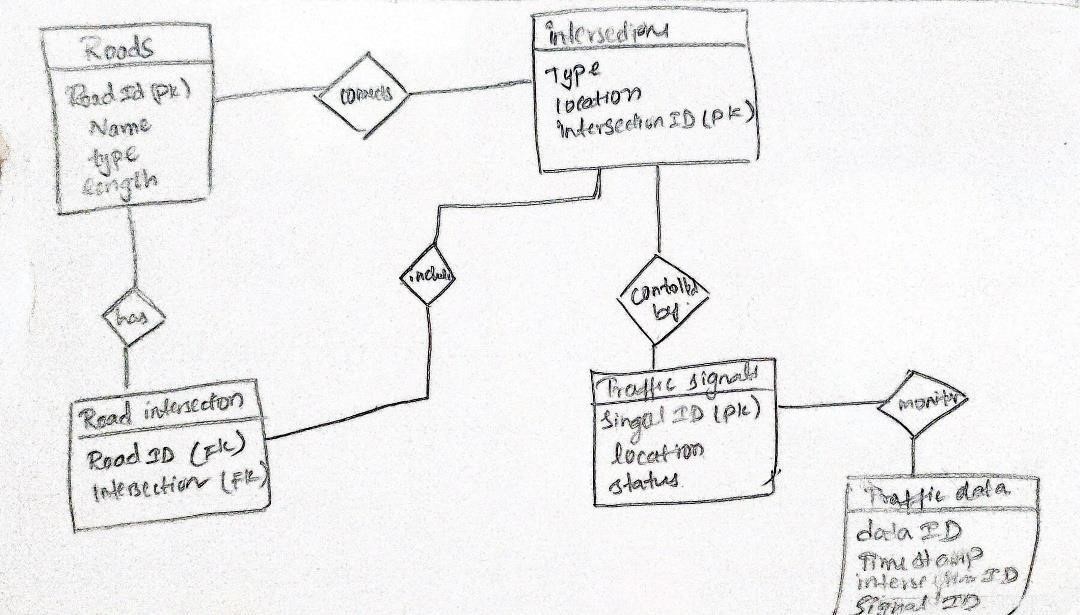
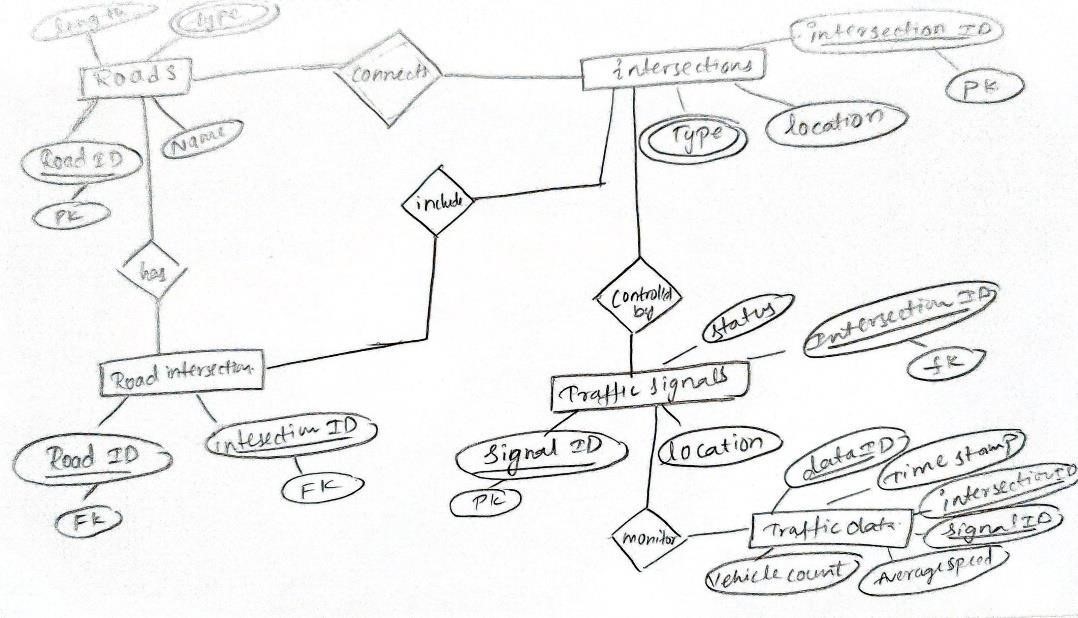
* Relationship: Traffic data is collected from sensors related to traffic signals.
* Cardinality: One-to-Many
* Optionality: Optional (traffic data may not always be available for every signal)

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

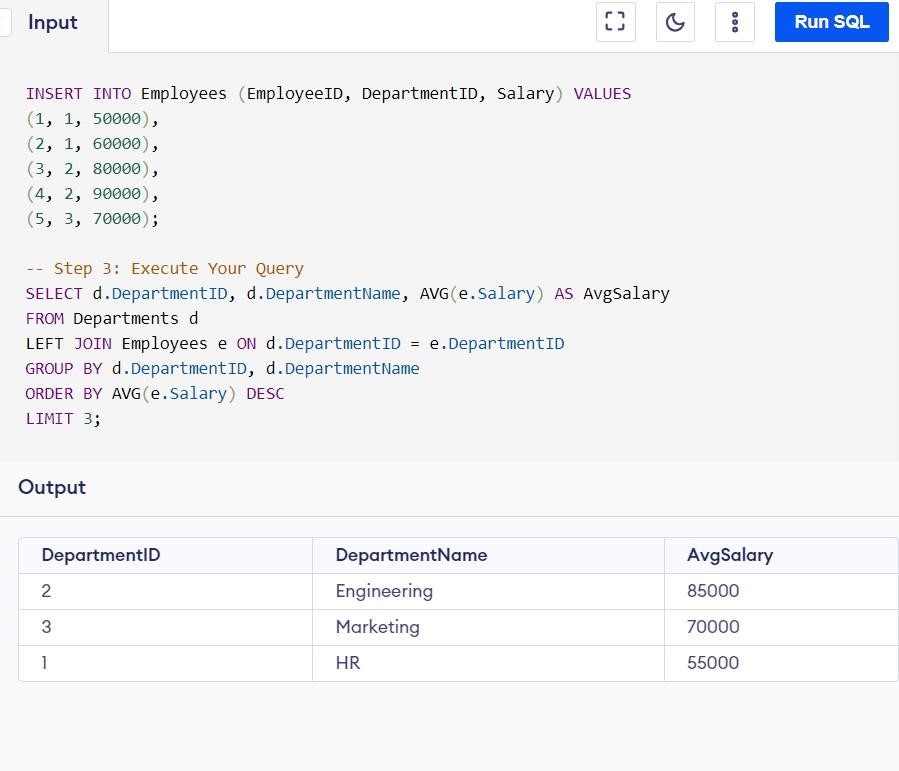
GROUP BY d.DepartmentID, d.DepartmentName

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

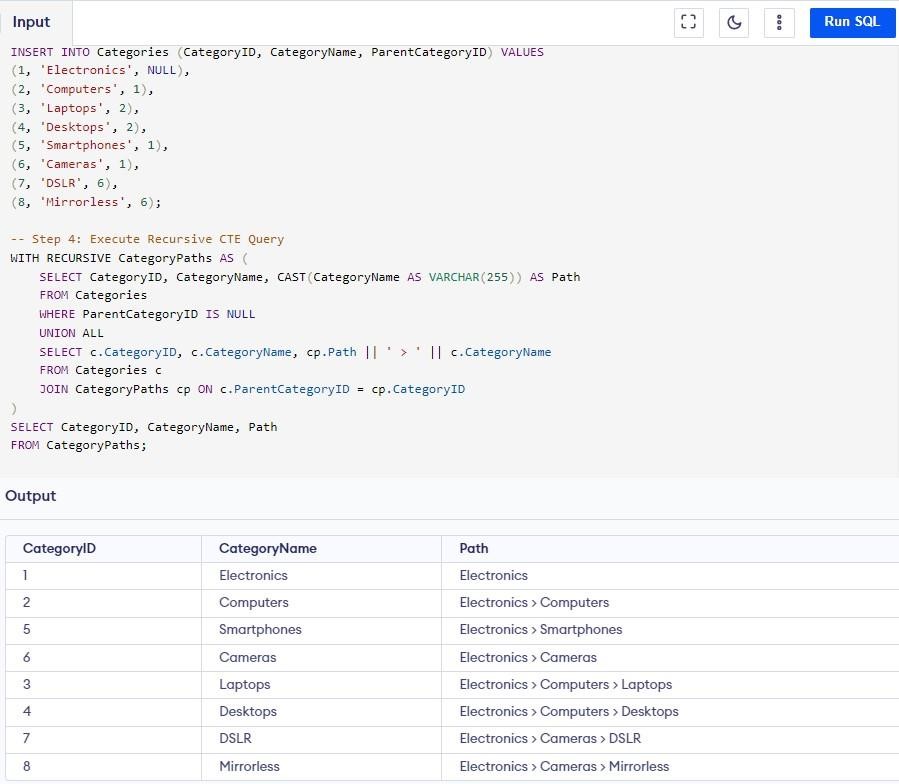
JOIN CategoryPaths cp ON c.ParentCategoryID = cp.CategoryID

)

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

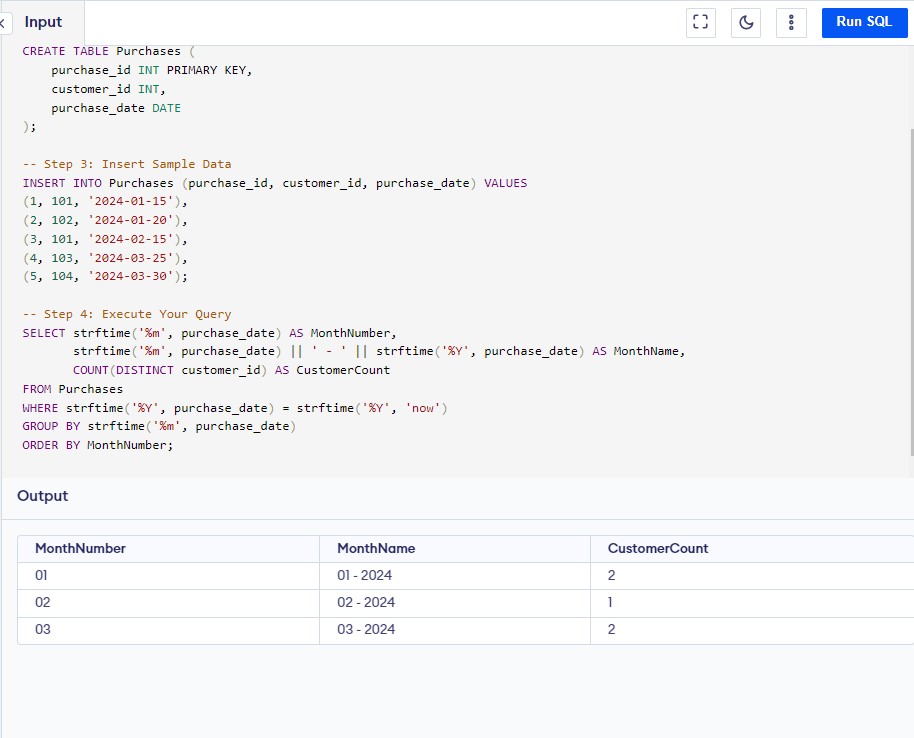
FROM Purchases

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

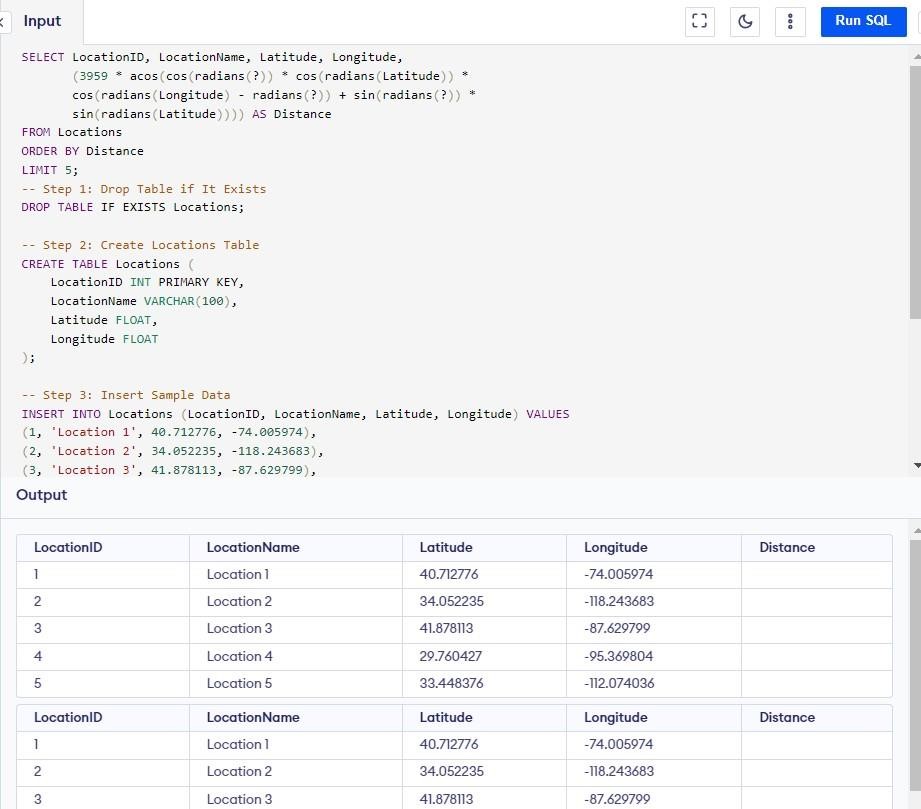
FROM Locations

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

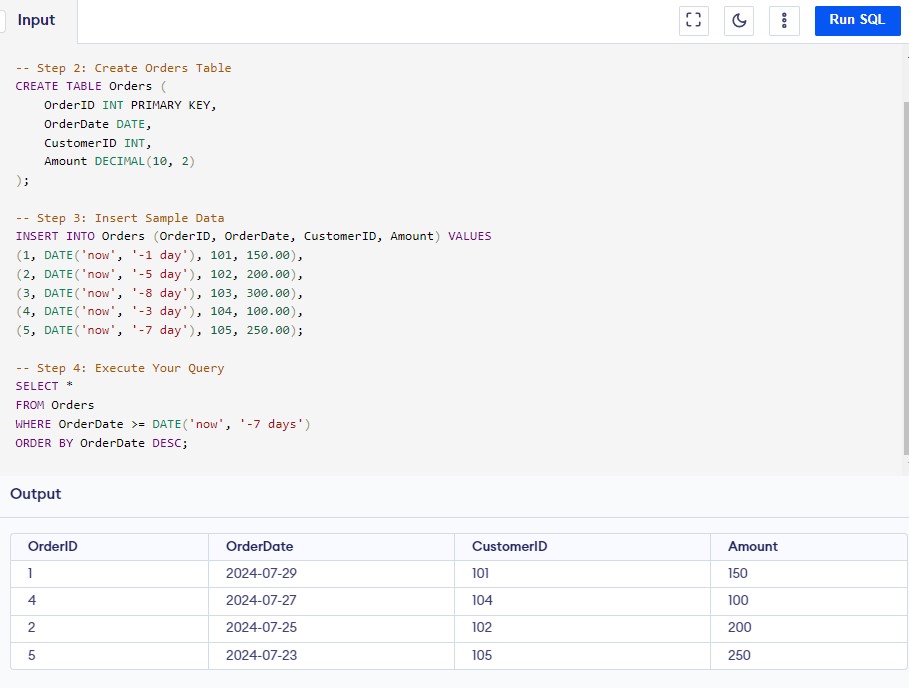
\*

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

DECLA

RE 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;

EXCEPTI

ON

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

DECLA

RE

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 them as a cursor.

## Question 4: Using Cursor Variables and Dynamic SQL

```plsql

DECLA

RE

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.