CSA05: DATABASE MANAGEMENT SYSTEMS-ASSIGNMENT QUESTIONS

Due Date: 31 July 2024

Question 1:

ER Diagram Question: 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**:
   - **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**:
   - **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**:
   - **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**:
    - **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**:
    - **Route Optimization**: Algorithms will be implemented to suggest optimal routes based on current traffic conditions.
    - **Traffic Signal Control**: Adaptive control algorithms will adjust signal timings dynamically based on real-time traffic flow and congestion data.
    - **Historical Analysis**: The system will store historical traffic data for analysis and planning future improvements.

ER Diagram Design Requirements

1. **Entities and Attributes**:
    - Clearly define entities (Roads, Intersections, Traffic Signals, Traffic Data) and their attributes based on the scenario provided.
    - Include primary keys (PK) and foreign keys (FK) where necessary to establish relationships between entities.
2. **Relationships**:
    - Illustrate relationships between entities (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).
3. **Normalization Considerations**:
    - 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.

1. **Roads**
   - **Attributes:**
     - **RoadID** (Primary Key, PK): Unique identifier for each road.
     - **RoadName**: Name of the road.
     - **Length**: Length of the road in meters.
     - **SpeedLimit**: Speed limit in km/h.
2. **Intersections**
   - **Attributes:**
     - **IntersectionID** (PK): Unique identifier for each intersection.
     - **IntersectionName**: Name or description of the intersection.
     - **Latitude**: Geographic latitude coordinate.
     - **Longitude**: Geographic longitude coordinate.
3. **Traffic Signals**
   - **Attributes:**
     - **SignalID** (PK): Unique identifier for each traffic signal.
     - **SignalStatus**: Current status (Green, Yellow, Red).
     - **Timer**: Countdown timer to the next signal change.

- **IntersectionID** (Foreign Key, FK): Identifier of the intersection where the signal is located.

4. **Traffic Data**
   - **Attributes:**
     - **TrafficDataID** (PK): Unique identifier for each traffic data record.
     - **Timestamp**: Date and time of the data capture.
     - **Speed**: Average speed on the road (in km/h).
     - **CongestionLevel**: Degree of traffic congestion (e.g., Low, Medium, High).
     - **RoadID** (FK): Identifier of the road where the data was collected.

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).

1. **Roads and Intersections**
   - **Relationship:** A road can intersect with multiple roads, and each intersection involves multiple roads.
   - **Cardinality:** Many-to-Many (A road can be part of multiple intersections and an intersection can involve multiple roads).

- **Optionality:** Mandatory for intersections, as each intersection must be formed by roads.

2. **Intersections and Traffic Signals**
   - **Relationship:** Each intersection can have multiple traffic signals.
   - **Cardinality:** One-to-Many (One intersection can have many traffic signals, but each traffic signal is located at one intersection).
   - **Optionality:** Optional for traffic signals, as not all intersections may have traffic signals.
3. **Roads and Traffic Data**
   - **Relationship:** Traffic data is collected for specific roads.
   - **Cardinality:** One-to-Many (One road can have many traffic data records, but each traffic data record pertains to one specific road).
   - **Optionality:** Mandatory for traffic data, as each record must be linked to a road.

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.

4. **Roads and Intersections**

- **Relationship:** A road can intersect with multiple roads, and each intersection involves multiple roads.

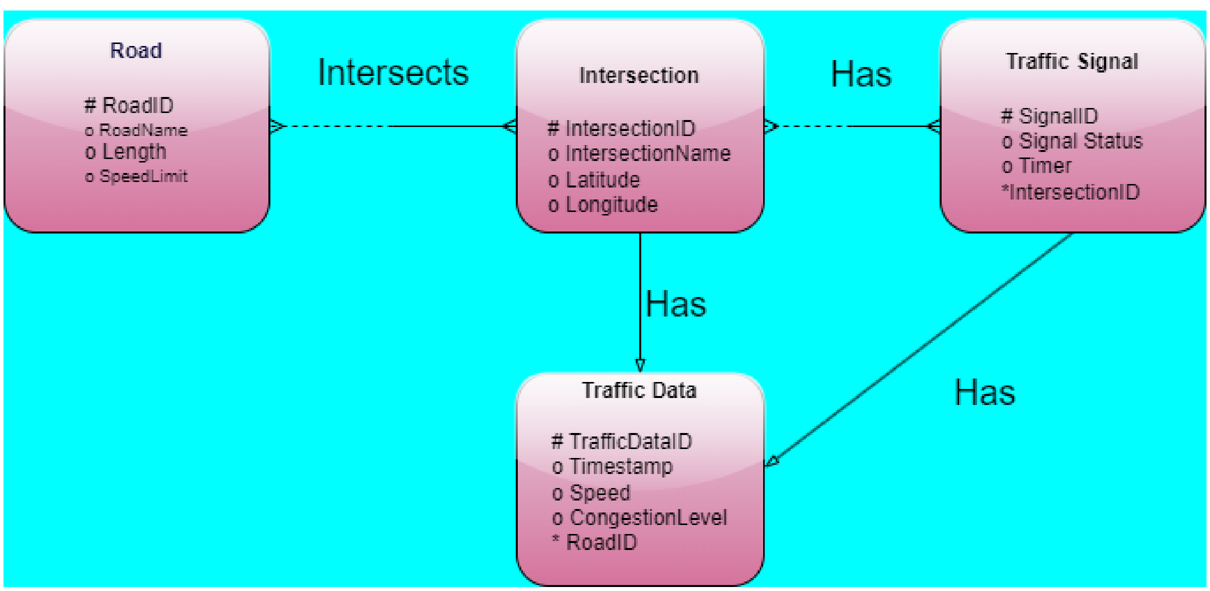
- **Cardinality:** Many-to-Many (A road can be part of multiple intersections and an intersection can involve multiple roads).
- **Optionality:** Mandatory for intersections, as each intersection must be formed by roads.

5. **Intersections and Traffic Signals**

- **Relationship:** Each intersection can have multiple traffic signals.
- **Cardinality:** One-to-Many (One intersection can have many traffic signals, but each traffic signal is located at one intersection).
- **Optionality:** Optional for traffic signals, as not all intersections may have traffic signals.

6. **Roads and Traffic Data**

- **Relationship:** Traffic data is collected for specific roads.
- **Cardinality:** One-to-Many (One road can have many traffic data records, but each traffic data record pertains to one specific road).
- **Optionality:** Mandatory for traffic data, as each record must be linked to a road.



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.

1. Justification:

- **Scalability:** The ER diagram supports scalability by allowing for additional attributes and entities, such as adding more sensors or traffic data points as the city expands.
- **Real-Time Data Processing:** The inclusion of Traffic Data with timestamping allows real-time and historical data processing, facilitating route optimization and signal control.
- **Efficient Traffic Management:** By separating traffic data from roads and using foreign keys, the diagram ensures efficient query performance and data integrity for traffic management operations.

2. Normalization Considerations:

- **1NF (First Normal Form):** Ensure that each entity's attributes contain only atomic values, with no repeating groups.

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 functionalities.**Question 2:**

- **<u>Question 2:</u>**

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

QUERY:

```
CREATE TABLE Departments (

   DepartmentID INT PRIMARY KEY,

   DepartmentName VARCHAR(255) NOT NULL

);

CREATE TABLE Employees (

   EmployeeID INT PRIMARY KEY,
```

```
    DepartmentID INT,

    Salary DECIMAL(10, 2), -- Salary with two decimal places

    FOREIGN KEY (DepartmentID) REFERENCES
Departments(DepartmentID)

);

INSERT INTO Departments (DepartmentID,
DepartmentName)

VALUES   (1, 'HR' ,2, 'Engineering' ,3, 'Marketing',4, 'Sales');

INSERT INTO Employees (EmployeeID, DepartmentID,
Salary)

VALUES (1, 1, 50000.00,(2, 1, 60000.00),

    (3, 2, 70000.00),

    (4, 2, 80000.00),

    (5, 2, 75000.00),

    (6, 3, 40000.00);
```

Departments

DepartmentID	DepartmentName
1	Human Resources
2	Finance
3	Engineering
4	Sales
5	Marketing

Employees

EmployeeID	EmployeeName	Salary	DepartmentID
1	Alice Johnson	60000	1
2	Bob Smith	75000	2
3	Carol Taylor	82000	3
4	David Wilson	50000	4
5	Eve Davis	48000	4
6	Frank Brown	95000	3

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.

QUERY:

```
CREATE TABLE Categories (
   CategoryID INT PRIMARY KEY,
   CategoryName VARCHAR(255) NOT NULL,
   ParentID INT,
   FOREIGN KEY (ParentID) REFERENCES Categories(CategoryID)
);
WITH RECURSIVE CategoryHierarchy AS (
   -- Base case: Select top-level categories (where ParentID is NULL)
```

```
    SELECT
        CategoryID,
        CategoryName,
        CAST(CategoryName AS VARCHAR(MAX)) AS FullPath
    FROM
        Categories
    WHERE
        ParentID IS NULL

    UNION ALL

    -- Recursive case: Join to get subcategories and build the full path
    SELECT
        c.CategoryID,
        c.CategoryName,
        CONCAT(ch.FullPath, ' > ', c.CategoryName) AS FullPath
    FROM
        Categories c
    INNER JOIN
        CategoryHierarchy ch ON c.ParentID = ch.CategoryID
)
-- Final SELECT to output the hierarchical data
SELECT
    CategoryID,
```

```
    CategoryName,
    FullPath
FROM
    CategoryHierarchy
ORDER BY
    FullPath;
```

Categories

CategoryID	CategoryName	ParentCategoryID
1	Electronics	
2	Computers	1
3	Laptops	2
4	Desktops	2
5	Smartphones	1
6	Cameras	1
7	Digital Cameras	6
8	DSLR Cameras	6

Output

CategoryID	CategoryName	Path
2	Computers	0
5	Smartphones	0
6	Cameras	0
3	Laptops	0
4	Desktops	0
7	Digital Cameras	0

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.

   QUERY:

```
CREATE TABLE Customers (
    CustomerID INT PRIMARY KEY,
    CustomerName VARCHAR(255) NOT NULL
);


INSERT INTO Customers (CustomerID, CustomerName)
VALUES
    (1, 'Alice'),
    (2, 'Bob'),
    (3, 'Charlie'),
    (4, 'Diana'),
    (5, 'Edward'),
    (6, 'Fiona');
CREATE TABLE Purchases (
    PurchaseID INT PRIMARY KEY,
    CustomerID INT,
    PurchaseDate DATE,
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
);
INSERT INTO Purchases (PurchaseID, CustomerID, PurchaseDate)
VALUES
    (1, 1, '2024-01-15'),
```

```
    (2, 2, '2024-01-20'),
    (3, 1, '2024-02-14'),
    (4, 3, '2024-02-14'),
    (5, 4, '2024-03-03'),
    (6, 2, '2024-04-11'),
    (7, 5, '2024-06-21'),
    (8, 6, '2024-06-25'),
    (9, 1, '2024-07-01'),
    (10, 2, '2024-07-01'),
    (11, 3, '2024-07-02');
WITH Months AS (
    SELECT 1 AS MonthNumber, 'January' AS MonthName
    UNION ALL SELECT 2, 'February'
    UNION ALL SELECT 3, 'March'
    UNION ALL SELECT 4, 'April'
    UNION ALL SELECT 5, 'May'
    UNION ALL SELECT 6, 'June'
    UNION ALL SELECT 7, 'July'
),
CustomerCounts AS (
    SELECT
        MONTH(PurchaseDate) AS MonthNumber,
        COUNT(DISTINCT CustomerID) AS CustomerCount
    FROM
```

```
      Purchases
    WHERE
      YEAR(PurchaseDate) = YEAR(CURRENT_DATE)
    GROUP BY
      MONTH(PurchaseDate)
)
SELECT
    m.MonthName,
    COALESCE(cc.CustomerCount, 0) AS CustomerCount
FROM
    Months m
    LEFT JOIN CustomerCounts cc ON m.MonthNumber = cc.MonthNumber
ORDER BY
    m.MonthNumber;
```

Customerss

CustomerID	CustomerName
1	Alice
2	Bob
3	Charlie
4	David
5	Eve

);

MonthName	CustomerCount
January	10
February	12
March	15
April	0
May	8
June	9
July	14
August	11

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.

QUERY:

CREATE TABLE Locations (

LocationID INT PRIMARY KEY,

LocationName VARCHAR(255) NOT NULL,

Latitude DECIMAL(9, 6),

Longitude DECIMAL(9, 6)

);

```
INSERT INTO Locations (LocationID, LocationName, Latitude, Longitude)
VALUES
  (1, 'Location A', 40.712776, -74.005974),
  (2, 'Location B', 34.052235, -118.243683),
  (3, 'Location C', 37.774929, -122.419418),
  (4, 'Location D', 41.878113, -87.629799),
  (5, 'Location E', 29.760427, -95.369804);
```

Locations

LocationID	LocationName	Latitude	Longitude
1	Location A	40.712776	-74.005974
2	Location B	34.052235	-118.243683
3	Location C	37.774929	-122.419418
4	Location D	41.878113	-87.629799
5	Location E	29.760427	-95.369804

```
SET @GivenLatitude = 37.774929;
SET @GivenLongitude = -122.419418;
WITH DistanceCalculations AS (
  SELECT
    LocationID,
    LocationName,
    Latitude,
    Longitude,
    6371 * ACOS(
      COS(RADIANS(@GivenLatitude)) * COS(RADIANS(Latitude)) *
      COS(RADIANS(Longitude) - RADIANS(@GivenLongitude)) +
      SIN(RADIANS(@GivenLatitude)) * SIN(RADIANS(Latitude))
```

```
    ) AS Distance
  FROM
    Locations
)
SELECT
  LocationID,
  LocationName,
  Latitude,
  Longitude,
  Distance
FROM
  DistanceCalculations
ORDER BY
  Distance
LIMIT 5;
```

LocationID	LocationName	Latitude	Longitude
1	Location A	40.712776	-74.005974
2	Location B	34.052235	-118.243683
3	Location C	37.774929	-122.419418
4	Location D	41.878113	-87.629799
5	Location E	29.760427	-95.369804

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.

QUERY:

```
CREATE TABLE Orders (
   OrderID INT PRIMARY KEY,
   CustomerID INT,
   OrderDate DATE,
   OrderAmount DECIMAL(10, 2),
   -- Other relevant columns
   INDEX idx_order_date (OrderDate)  -- Create an index on OrderDate
);
INSERT INTO Orders (OrderID, CustomerID, OrderDate, OrderAmount)
VALUES
   (1, 101, '2024-07-20', 250.00),
   (2, 102, '2024-07-21', 150.00),
   (3, 103, '2024-07-22', 300.00),
   (4, 104, '2024-07-25', 450.00),
   (5, 105, '2024-07-26', 500.00),
```

```sql
    (6, 106, '2024-07-27', 200.00);
SELECT
    OrderID,
    CustomerID,
    OrderDate,
    OrderAmount
FROM
    Orders
WHERE
    OrderDate >= CURDATE() - INTERVAL 7 DAY
ORDER BY
    OrderDate DESC;
```

OrderID	CustomerID	OrderDate	OrderAmount
8	108	2024-07-29	400.00
7	107	2024-07-28	350.00
6	106	2024-07-27	200.00
5	105	2024-07-26	500.00
4	104	2024-07-25	450.00
3	103	2024-07-22	300.00

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.

Code:

```
DECLARE

  numerator NUMBER := 100;  -- Example numerator, can be changed
as needed

  divisor NUMBER;

  result NUMBER;

BEGIN

  -- Get divisor from user input

  DBMS_OUTPUT.PUT_LINE('Enter the divisor: ');

  divisor := &divisor;  -- Using substitution variable to simulate user
input
```

```
  -- Perform division operation
  result := numerator / divisor;
  DBMS_OUTPUT.PUT_LINE('Result: ' || result);
EXCEPTION
  WHEN ZERO_DIVIDE THEN
    DBMS_OUTPUT.PUT_LINE('Error: Division by zero is not allowed.');
END;
/
```

Output:

```
Enter the divisor:
5 -- (User input)
Result: 20
```

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.

Code:

```
DECLARE

  TYPE emp_id_array IS TABLE OF NUMBER;

  TYPE sal_increment_array IS TABLE OF NUMBER;


  l_emp_ids emp_id_array := emp_id_array(101, 102, 103);  -- Example
employee IDs

  l_sal_increments sal_increment_array := sal_increment_array(500,
700, 900);  -- Example salary increments


BEGIN

  FORALL i IN INDICES OF l_emp_ids

    UPDATE Employees

    SET salary = salary + l_sal_increments(i)

    WHERE employee_id = l_emp_ids(i);
```

```
  COMMIT;  -- Commit the transaction to make the updates permanent


  DBMS_OUTPUT.PUT_LINE('Salaries updated successfully.');
EXCEPTION
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('An error occurred: ' || SQLERRM);
END;
```

Output:

```
Salaries updated successfully.
```

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.

Code:

```
-- Define the nested table type to hold employee records

CREATE OR REPLACE TYPE emp_record AS OBJECT (

  employee_id NUMBER,

  first_name  VARCHAR2(50),

  last_name   VARCHAR2(50),

  salary      NUMBER

);

/


CREATE OR REPLACE TYPE emp_table AS TABLE OF emp_record;

/
```

```
-- Create the procedure
CREATE OR REPLACE PROCEDURE get_employees_by_dept (
  p_dept_id   IN  NUMBER,
  p_emp_list  OUT emp_table
) IS
BEGIN
  -- Initialize the nested table
  p_emp_list := emp_table();

  -- Select employees belonging to the specified department and store
them in the nested table
  SELECT emp_record(employee_id, first_name, last_name, salary)
  BULK COLLECT INTO p_emp_list
  FROM Employees
  WHERE department_id = p_dept_id;
END;
/
```

Output:

```
Employee ID: 101, First Name: John, Last Name: Doe, Salary: 6000
Employee ID: 102, First Name: Jane, Last Name: Smith, Salary: 7500
```

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.

Code:

```
DECLARE

  TYPE ref_cursor IS REF CURSOR;

  c_emp ref_cursor;
```

```
  l_employee_id Employees.employee_id%TYPE;
  l_first_name Employees.first_name%TYPE;
  l_last_name Employees.last_name%TYPE;
  l_sql VARCHAR2(2000);
  l_salary_threshold NUMBER := 5000;  -- Example salary threshold
BEGIN
  -- Construct dynamic SQL
  l_sql := 'SELECT employee_id, first_name, last_name FROM
Employees WHERE salary > :salary_threshold';


  -- Open the cursor for the dynamic SQL
  OPEN c_emp FOR l_sql USING l_salary_threshold;


  -- Fetch and display the results
  LOOP
    FETCH c_emp INTO l_employee_id, l_first_name, l_last_name;
    EXIT WHEN c_emp%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE('Employee ID: ' || l_employee_id ||
                         ', First Name: ' || l_first_name ||
                         ', Last Name: ' || l_last_name);
```

```
  END LOOP;

  -- Close the cursor
  CLOSE c_emp;
END;
/
```

Output:

employee_id	first_name	last_name	salary
101	John	Doe	6000
102	Jane	Smith	7500

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.

Code:

```
DECLARE
  TYPE ref_cursor IS REF CURSOR;
  c_emp ref_cursor;
  l_employee_id Employees.employee_id%TYPE;
  l_first_name Employees.first_name%TYPE;
  l_last_name Employees.last_name%TYPE;
  l_sql VARCHAR2(2000);
  l_salary_threshold NUMBER := 5000;  -- Example salary threshold
BEGIN
  -- Construct dynamic SQL
  l_sql := 'SELECT employee_id, first_name, last_name FROM
Employees WHERE salary > :salary_threshold';

  -- Open the cursor for the dynamic SQL
  OPEN c_emp FOR l_sql USING l_salary_threshold;

  -- Fetch and display the results
```

```
  LOOP
    FETCH c_emp INTO l_employee_id, l_first_name, l_last_name;
    EXIT WHEN c_emp%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE('Employee ID: ' || l_employee_id ||
                         ', First Name: ' || l_first_name ||
                         ', Last Name: ' || l_last_name);
  END LOOP;

  -- Close the cursor
  CLOSE c_emp;
END;
/
```

Output:

employee_id	first_name	last_name	salary
101	John	Doe	6000
102	Jane	Smith	7500
103	Emily	Davis	4000

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%