COMMON ASSIGNMENT

Traffic Flow Management System (TFMS)

1 Question

Task 1 : Entity Identification and Attributes

Roads
- Road ID (PK)
- Road Name
- Length (meters
- Speed Limit (Km/h)

Intersections
- Intersection ID
- Intersection Name
- Latitude
- Longitude

Traffic Signals
- Signal ID
- Signal Status
- Timer
- Intersection ID (FK)

Traffic Data
- Traffic Data ID (PK)
- Timestamp
- Speed
- CongestionLevel
- Road ID (FK)

Task 2 : Relationship Modelling

① Road to Intersections:

* one road can connect to many intersections (one-to-many)
* one intersection can be connected by many roads (many-to-many through an intersection table).

M. Chanikya
19231193
DBMS CSA0563

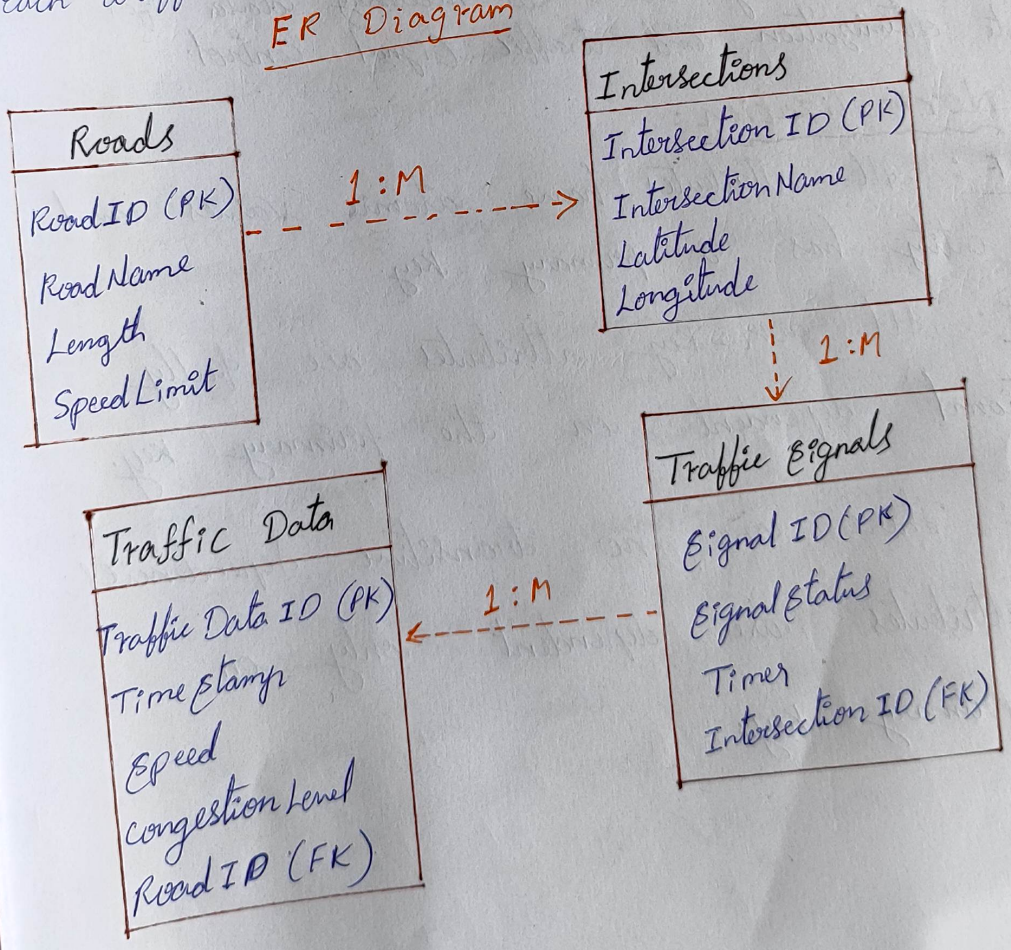
(2) Intersections to traffic Signals

* one intersection can have many traffic signals.
* one traffic signal is associated with one intersection (one-to-one)

(3) Roads to traffic Data

* one road can have multiple traffic data entries (one-to-many).
* Each traffic data entry is associated with one road (one-to-one)

ER Diagram



Task :4 Justification and Normalization

① Justification of Design choices

* Scalability: The design allows for the addition of new roads, intersections and traffic signals without major changes in database structure.
* Real-time Data processing: By separating real-time traffic data into its own entity, the system can efficiently handle frequent updates and queries.
* Efficient Traffic Management: The relationships between entities enable easy access to necessary data for route optimization and traffic signal control.

② Normalization:

* 1NF: All attributes have atomic values and each entity has a primary key.
* 2NF: All non-key attributes are fully functional dependent on the primary key.
* 3NF: There are no transitive dependencies; all attributes are dependent only on the primary key.

Deliverables

1. **ER Diagram**: As shown above in textual form, which should be converted into a visual representation using ER diagram tool.

2. **Easy Definitions**: Clear definitions as provided in task 1.

3. **Relationship descriptions**: Detailed relationships as described in task 2.

4. **Justification Document**: A document explaining the design choices, normalization considerations, and how the ER diagram supports TFMS functionalities.

Question (2)

① Top 3 Departments with highest average Salary

```
SELECT DepartmentID, DepartmentName, AVG(Salary) AS AvgSalary
FROM Departments d
LEFT JOIN Employees e ON d.DepartmentID = e.DepartmentID
GROUP BY DepartmentID, DepartmentName
ORDER BY AvgSalary DESC
LIMIT 3;
```

Explanation:

* The "left join" ensures that all departments are included in the result set, even if they have no employees. For departments without employees, the "AVG(Salary)" function will return null.
* "Group by" is used to group the results by department.
* The "Order by "Avg Salary" DESC" clause sorts the departments by their average salary in descending order.
* LIMIT 3 restricts the output to top 3 departments.

② Retrieving Hierarchical Category Paths

```
WITH RECURSIVE CategoryPath 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) AS Path
    FROM Categories C
    JOIN CategoryPath CP ON C.ParentCategoryID = CP.CategoryID
)
SELECT CategoryID, CategoryName, Path
    FROM CategoryPath;
```

Explanation:

* The recursive CTE 'CategoryPath' starts by selecting all top-level categories (those without a parent).
* The UNION ALL clause recursively joins each category with its parent building the hierarchical path by concatenating parents Path with child category names.
* The base case of the recursion selects categories with 'ParentCategoryID is NULL'.
* The recursive case joins the 'Categories' tables with the 'CategoryPath' CTE, appending the child category name to the Path.

③ Total Distinct customers by Month

```
WITH MONTHS AS ( SELECT DATE_FORMAT (DATE_ADD('2024-01-01',
    INTERVAL seq MONTH), '%M') AS MonthName, seq AS MonthNumber
    FROM seq_0_to_11
),
Purchases AS (
    SELECT DISTINCT CustomerID, MONTH(OrderDate) AS OrderMonth
    FROM orders
    WHERE YEAR(OrderDate) = YEAR(CURDATE())
)
SELECT m.MonthName, COUNT(P.customerID) AS customerCount
FROM Months m
LEFT JOIN Purchases P ON m.MonthNumber + 1 = P.OrderMonth
GROUP BY m.MonthName
ORDER BY m.MonthNumber;
```

Explanation:

* The 'Months' CTE generates a list of month names for the current year.
* The 'Purchases' CTE retrieves distinct customer IDs and their order month.
* The LEFT JOIN b/w Months and Purchases ensures all months are added even those with no customer activity.
* COUNT(P.customerID) counts distinct customers per month, returning 0 for months with no purchases.

④ Finding Closest Locations

```
SELECT LocationID, LocationName, Latitude, Longitude,
(6371 * acos(cos(radians(@given_latitude)) * cos(radians(latitude)) AS Distance
FROM Locations ORDER BY Distance LIMIT 5;
```

⑤ Optimizing Query for order table

```
SELECT orderID, customerID, orderDate, TotalAmount
FROM orders WHERE orderDate >= CURDATE() - INTERVAL 7 DAY
ORDER BY orderDate DESC;
```

Explanation:

* The distance calculation is based one ractius of the earth 6371 Km.
* Locations are ordered by calculated distance and the top 5 closest locations are selected using 'LIMIT 5'.

Discussion:

* Indexing: Ensure there is an index on 'orderDate' DAY' to filter orders from last 7 days.
* Query Rewriting: The query uses 'CURDATE() - INTERVAL 7

Performance Considerations

* The 'ORDER BY orderDate DESC' clause ensures the results are sorted by the most recent orders first, which is often needed in practical applications.
* By directly using the indexed column in the 'WHER clause, the query minimizes the need for full table scans, significantly improving Performance for large datasets.