DATABASE MANAGEMENT SYSTEMS

Name: G . Lakshmi Mahesh Kumar

Register number: 192311177 Course code: CSA0564

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.

Answers:

Task 1: Entity Identification and Attributes

Entities and their Attributes:

1. Roads:

- RoadID(PK): Unique identifier for each road

- RoadName: Name of the road

- Length: Length of the road in meters

- SpeedLimit: Maximum speed limit in km/h

2. Intersections:

- IntersectionID(PK): Unique identifier for each intersection

- IntersectionName: Name of the intersection

- Latitude: Geographic latitude of the intersection

- Longitude: Geographic longitude of the intersection

3. Traffic Signals:

- SignalID(PK): Unique identifier for each traffic signal

- SignalStatus: Current status of the signal (Green, Yellow, Red)

- Timer: Countdown timer to the next signal change

- IntersectionID(FK): Foreign key referring to the IntersectionID in Intersections

4. Traffic Data:

- TrafficDataID (PK): Unique identifier for each traffic data entry

- Timestamp: Date and time when the data was collected

- Speed: Average speed on the road

- CongestionLevel: Degree of traffic congestion

- RoadID(FK): Foreign key referring to the RoadID in Roads

Task 2: Relationship Modeling

Relationships and their Cardinality:

1. Roads to Intersections:

- Relationship: Roads intersect at Intersections

- Cardinality: Many-to-Many (A road can be part of multiple intersections, and an intersection can be connected by multiple roads)

- Optionality: Mandatory on both sides (each intersection must be connected by roads and each road must connect to intersections)

2. Intersections to Traffic Signals:

- Relationship: Intersections have Traffic Signals

- Cardinality: One-to-Many (An intersection can have multiple traffic signals, but a traffic signal belongs to only one intersection)

- Optionality: Mandatory for Traffic Signals (each signal must be at an intersection), Optional for Intersections (an intersection may not have traffic signals at all times)

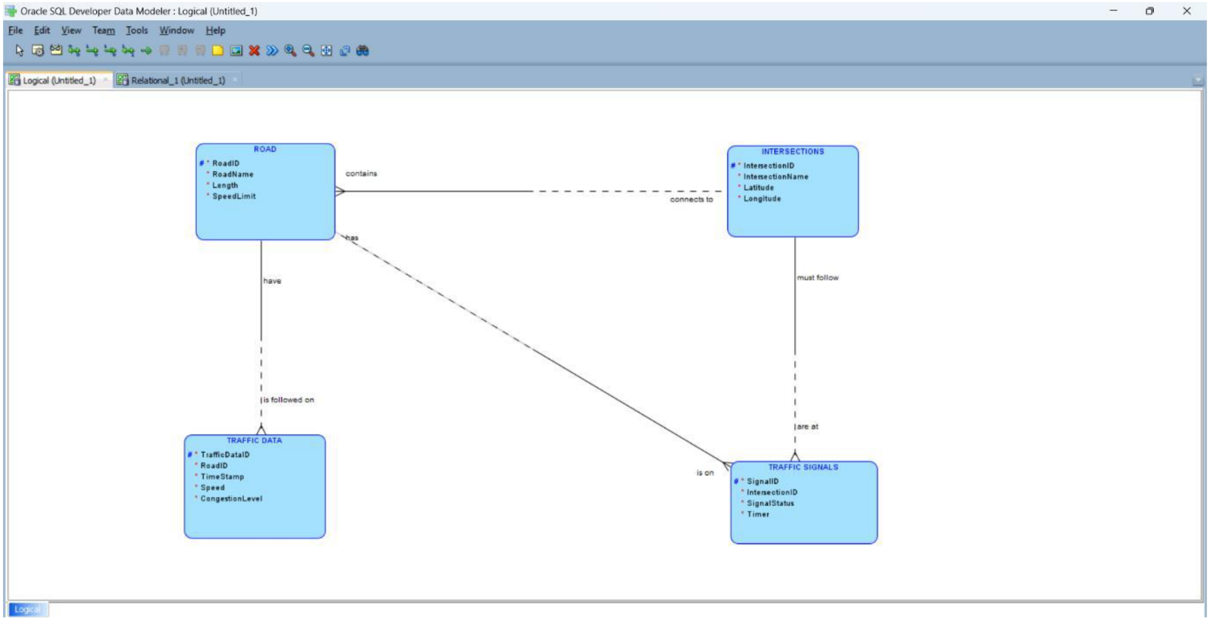
3. Traffic Data to Roads:

- Relationship: Traffic Data is collected for Roads

- Cardinality: Many-to-One (Multiple traffic data records can be associated with a single road)

- Optionality: Mandatory for Traffic Data (each data record must be associated with a road), Optional for Roads (a road might not have recent traffic data)

Task 3: ER Diagram Design



- Roads are connected to Intersections through a many-to-many relationship.

- Intersections are connected to Traffic Signals through a one-to-many relationship.

- Traffic Data is connected to Roads through a many-to-one relationship.

Task 4: Justification and Normalization

Justification:

1. Scalability and Real-Time Data Processing:

- The design allows for the addition of new roads, intersections, and traffic signals without affecting existing data.

- Traffic data is collected and stored in a way that supports real-time updates, ensuring that traffic conditions can be managed dynamically.

2. Efficient Traffic Management:

- The relationships ensure that traffic signals are managed at intersections and that traffic data is accurately linked to specific roads, facilitating better traffic management and route optimization.

Normalization Considerations:

1. 1NF (First Normal Form):

- Each table has a primary key, and attributes are atomic, ensuring no repeating groups or arrays.

2. 2NF (Second Normal Form):

- All non-key attributes are fully functionally dependent on the primary key. For example, Traffic Data attributes depend solely on the TrafficDataID, and not on other attributes.

3. 3NF (Third Normal Form):

- No transitive dependencies exist. For instance, Traffic Data attributes do not depend on non-key attributes of Roads.

Conclusion:

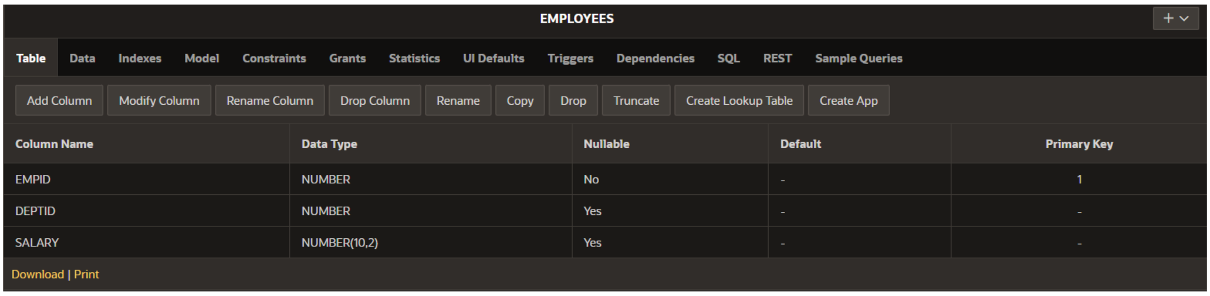
The ER diagram and associated design ensure data integrity, minimize redundancy, and support the key functionalities of the TFMS. The structure allows for efficient real-time data processing and supports future scalability as the city's traffic management needs evolve.

Question 2:

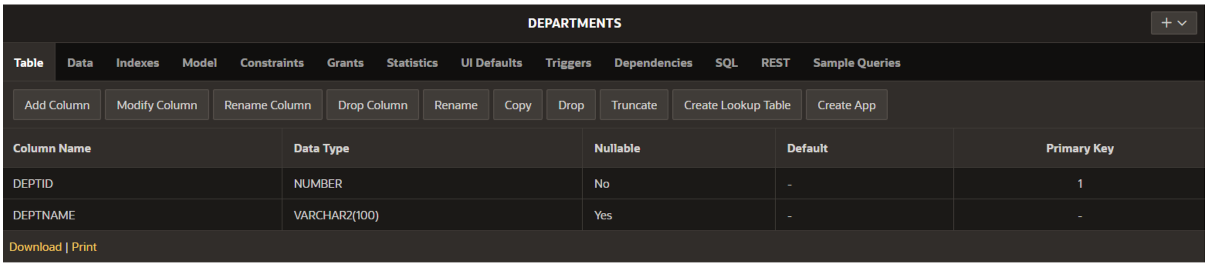
Question 1: Top 3 Departments with Highest Average Salary

Step 1: Create Tables

```
CREATE TABLE Departments (

DeptID INT PRIMARY KEY,

DeptName VARCHAR(100)

);

CREATE TABLE Employees (

EmpID INT PRIMARY KEY,

DeptID INT,

Salary DECIMAL(10, 2),

FOREIGN KEY (DeptID) REFERENCES Departments(DeptID)

);
```



Column Name	Data Type	Nullable	Default	Primary Key
EMPID	NUMBER	No	-	1
DEPTID	NUMBER	Yes	-	-
SALARY	NUMBER(10,2)	Yes	-	-



Column Name	Data Type	Nullable	Default	Primary Key
DEPTID	NUMBER	No	-	1
DEPTNAME	VARCHAR2(100)	Yes	-	-

Step 2: Insert Sample Data

```
INSERT INTO Departments (DeptID, DeptName) VALUES (1, 'HR');

INSERT INTO Departments (DeptID, DeptName) VALUES (2, 'Engineering');

INSERT INTO Departments (DeptID, DeptName) VALUES (3, 'Sales');

INSERT INTO Departments (DeptID, DeptName) VALUES (4, 'Marketing');
```

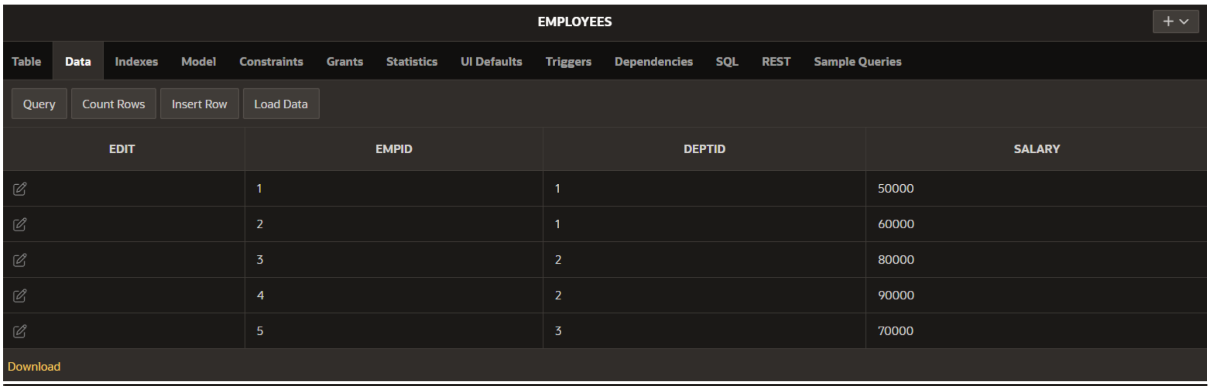
INSERT INTO Employees (EmpID, DeptID, Salary) VALUES (1, 1, 50000);

INSERT INTO Employees (EmpID, DeptID, Salary) VALUES (2, 1, 60000);

INSERT INTO Employees (EmpID, DeptID, Salary) VALUES (3, 2, 80000);

INSERT INTO Employees (EmpID, DeptID, Salary) VALUES (4, 2, 90000);

INSERT INTO Employees (EmpID, DeptID, Salary) VALUES (5, 3, 70000);



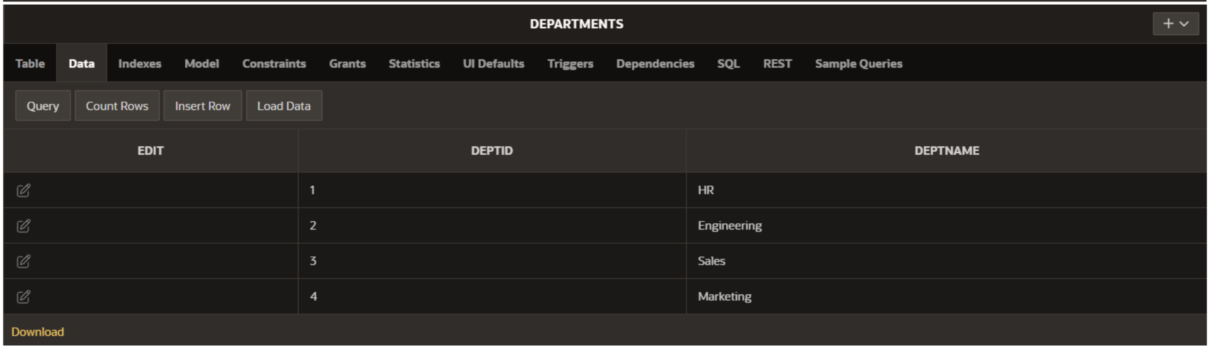
EMPLOYEES

Table | Data | Indexes | Model | Constraints | Grants | Statistics | UI Defaults | Triggers | Dependencies | SQL | REST | Sample Queries

Query | Count Rows | Insert Row | Load Data

EDIT	EMPID	DEPTID	SALARY
	1	1	50000
	2	1	60000
	3	2	80000
	4	2	90000
	5	3	70000

Download



DEPARTMENTS

Table | Data | Indexes | Model | Constraints | Grants | Statistics | UI Defaults | Triggers | Dependencies | SQL | REST | Sample Queries

Query | Count Rows | Insert Row | Load Data

EDIT	DEPTID	DEPTNAME
	1	HR
	2	Engineering
	3	Sales
	4	Marketing

Download

Step 3: Write the SQL Query

SELECT

d.DeptID,

d.DeptName,

AVG(e.Salary) AS AvgSalary

FROM

Departments d

LEFT JOIN

Employees e ON d.DeptID = e.DeptID

GROUP BY

d.DeptID, d.DeptName

ORDER BY

AvgSalary DESC

FETCH FIRST 3 ROWS ONLY;

SELECT d.DeptID, d.DeptName, AVG(e.Salary) AS AvgSalary FROM Departments d LEFT JOIN Employees e ON d.DeptID = e.DeptID GROUP BY d.DeptID, d.DeptName ORDER BY AvgSalary DESC FETCH FIRST 3 ROWS ONLY

DEPTID	DEPTNAME	AVGSALARY
4	Marketing	-
2	Engineering	85000
3	Sales	70000

3 rows selected. 0.01 seconds

Explanation:

- Departments with No Employees: `LEFT JOIN` ensures that departments with no employees are included with `NULL` for `AvgSalary`.

- Average Salary Calculation: `AVG(e.Salary)` computes the average salary for each department.

- Result Limitation: `FETCH FIRST 3 ROWS ONLY` limits the results to the top 3 departments by average salary.

Question 2: Retrieving Hierarchical Category Paths

Step 1: Create Table

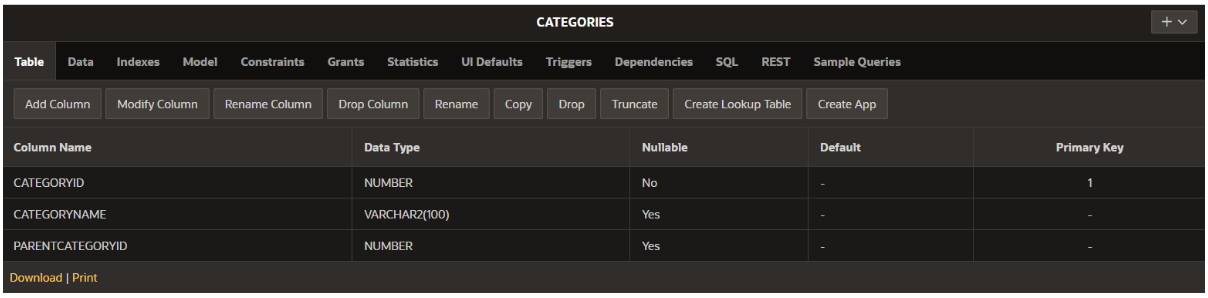
CREATE TABLE Categories (

CategoryID INT PRIMARY KEY,

CategoryName VARCHAR(100),

ParentCategoryID INT

);



CATEGORIES

Table | Data | Indexes | Model | Constraints | Grants | Statistics | UI Defaults | Triggers | Dependencies | SQL | REST | Sample Queries

Add Column | Modify Column | Rename Column | Drop Column | Rename | Copy | Drop | Truncate | Create Lookup Table | Create App

Column Name	Data Type	Nullable	Default	Primary Key
CATEGORYID	NUMBER	No	-	1
CATEGORYNAME	VARCHAR2(100)	Yes	-	-
PARENTCATEGORYID	NUMBER	Yes	-	-

Download | Print

Step 2: Insert Sample Data

INSERT INTO Categories (CategoryID, CategoryName, ParentCategoryID) VALUES (1, 'Electronics', NULL);

INSERT INTO Categories (CategoryID, CategoryName, ParentCategoryID) VALUES (2, 'Computers', 1);

INSERT INTO Categories (CategoryID, CategoryName, ParentCategoryID) VALUES (3, 'Laptops', 2);

INSERT INTO Categories (CategoryID, CategoryName, ParentCategoryID) VALUES (4, 'Smartphones', 1);

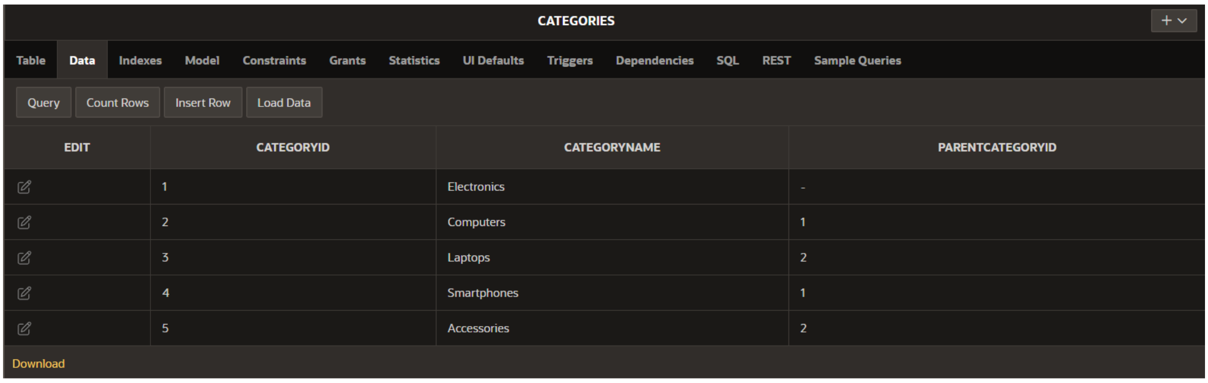
INSERT INTO Categories (CategoryID, CategoryName, ParentCategoryID) VALUES (5, 'Accessories', 2);

Step 3: Write the SQL Query

WITH RECURSIVE CategoryPaths AS (

SELECT

```
CategoryID,
CategoryName,
ParentCategoryID,
CategoryName AS Path
FROM
Categories
WHERE
ParentCategoryID IS NULL
UNION ALL
SELECT
c.CategoryID,
c.CategoryName,
c.ParentCategoryID,
CONCAT(cp.Path, ' > ', c.CategoryName) AS Path
FROM
Categories c
JOIN
CategoryPaths cp ON c.ParentCategoryID = cp.CategoryID
)
SELECT
CategoryID,
CategoryName,
Path
FROM
CategoryPaths
ORDER BY
Path;
```



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

Explanation:

- Recursive CTE: `CategoryPaths` starts from root categories and recursively joins child categories to build paths.

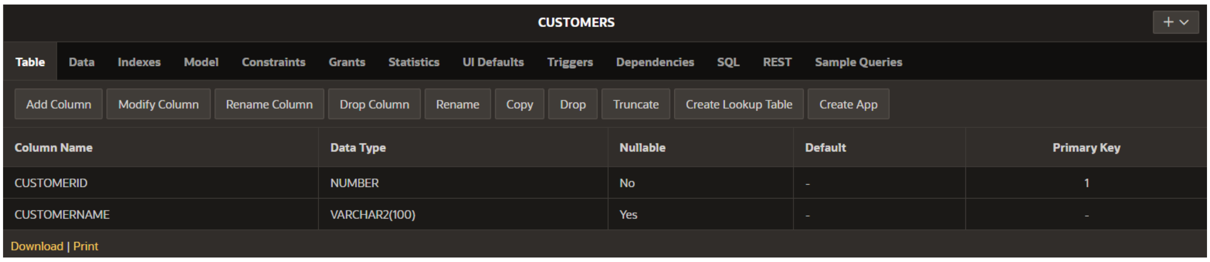
- Base Case: Initial selection includes categories with `NULL` for `ParentCategoryID`.

- Recursive Case: Continues to build paths by joining parent categories.

Question 3: Total Distinct Customers by Month

Step 1: Create Tables

```
CREATE TABLE Customers (

CustomerID INT PRIMARY KEY,

CustomerName VARCHAR(100)

);

CREATE TABLE Purchases (

PurchaseID INT PRIMARY KEY,

CustomerID INT,

PurchaseDate DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

CREATE TABLE CalendarMonths (

MonthNumber INT PRIMARY KEY,

MonthName VARCHAR(20)

);
```



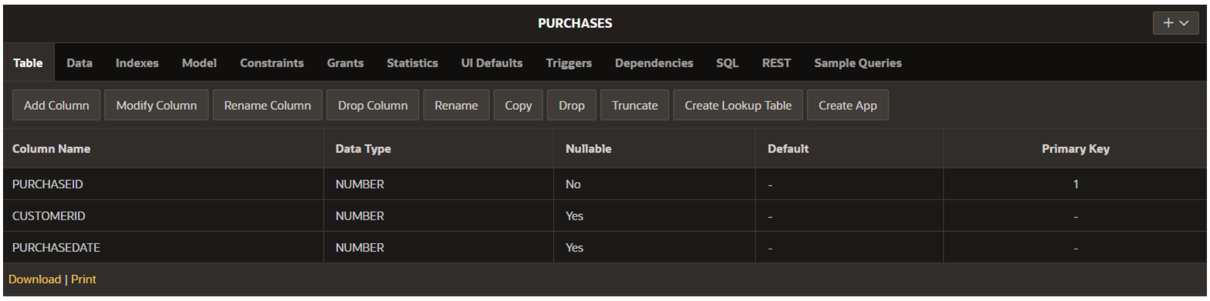
CUSTOMERS

Table | Data | Indexes | Model | Constraints | Grants | Statistics | UI Defaults | Triggers | Dependencies | SQL | REST | Sample Queries

Add Column | Modify Column | Rename Column | Drop Column | Rename | Copy | Drop | Truncate | Create Lookup Table | Create App

Column Name	Data Type	Nullable	Default	Primary Key
CUSTOMERID	NUMBER	No	-	1
CUSTOMERNAME	VARCHAR2(100)	Yes	-	-

Download | Print



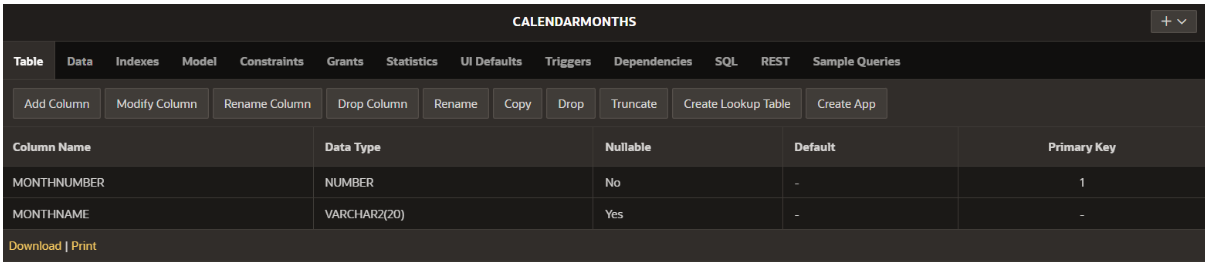
PURCHASES

Table | Data | Indexes | Model | Constraints | Grants | Statistics | UI Defaults | Triggers | Dependencies | SQL | REST | Sample Queries

Add Column | Modify Column | Rename Column | Drop Column | Rename | Copy | Drop | Truncate | Create Lookup Table | Create App

Column Name	Data Type	Nullable	Default	Primary Key
PURCHASEID	NUMBER	No	-	1
CUSTOMERID	NUMBER	Yes	-	-
PURCHASEDATE	NUMBER	Yes	-	-

Download | Print



CALENDARMONTHS

Table | Data | Indexes | Model | Constraints | Grants | Statistics | UI Defaults | Triggers | Dependencies | SQL | REST | Sample Queries

Add Column | Modify Column | Rename Column | Drop Column | Rename | Copy | Drop | Truncate | Create Lookup Table | Create App

Column Name	Data Type	Nullable	Default	Primary Key
MONTHNUMBER	NUMBER	No	-	1
MONTHNAME	VARCHAR2(20)	Yes	-	-

Download | Print

Step 2: Insert Sample Data

INSERT INTO Customers (CustomerID, CustomerName) VALUES (1, 'Alice');

INSERT INTO Customers (CustomerID, CustomerName) VALUES (2, 'Bob');

INSERT INTO Customers (CustomerID, CustomerName) VALUES (3, 'Charlie');

INSERT INTO Purchases (PurchaseID, CustomerID, PurchaseDate) VALUES (1, 1, '2024-07-05');

INSERT INTO Purchases (PurchaseID, CustomerID, PurchaseDate) VALUES (2, 2, '2024-07-15');

INSERT INTO Purchases (PurchaseID, CustomerID, PurchaseDate) VALUES (3, 1, '2024-08-20');

INSERT INTO CalendarMonths (MonthNumber, MonthName) VALUES (1, 'January');

INSERT INTO CalendarMonths (MonthNumber, MonthName) VALUES (2, 'February');

INSERT INTO CalendarMonths (MonthNumber, MonthName) VALUES (3, 'March');

INSERT INTO CalendarMonths (MonthNumber, MonthName) VALUES (4, 'April');

INSERT INTO CalendarMonths (MonthNumber, MonthName) VALUES (5, 'May');

INSERT INTO CalendarMonths (MonthNumber, MonthName) VALUES (6, 'June');

INSERT INTO CalendarMonths (MonthNumber, MonthName) VALUES (7, 'July');

INSERT INTO CalendarMonths (MonthNumber, MonthName) VALUES (8, 'August');

INSERT INTO CalendarMonths (MonthNumber, MonthName) VALUES (9, 'September');

INSERT INTO CalendarMonths (MonthNumber, MonthName) VALUES (10, 'October');

INSERT INTO CalendarMonths (MonthNumber, MonthName) VALUES (11, 'November');

INSERT INTO CalendarMonths (MonthNumber, MonthName) VALUES (12, 'December');

CATEGORIES

Table | Data | Indexes | Model | Constraints | Grants | Statistics | UI Defaults | Triggers | Dependencies | SQL | REST | Sample Queries

Query | Count Rows | Insert Row | Load Data

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

Download

PURCHASES

Table | Data | Indexes | Model | Constraints | Grants | Statistics | UI Defaults | Triggers | Dependencies | SQL | REST | Sample Queries

Query | Count Rows | Insert Row | Load Data

EDIT	PURCHASEID	CUSTOMERID	PURCHASEDATE
	1	1	5072024
	2	2	15072024
	3	1	20082024

Download

CALENDARMONTHS

Table | Data | Indexes | Model | Constraints | Grants | Statistics | UI Defaults | Triggers | Dependencies | SQL | REST | Sample Queries

Query | Count Rows | Insert Row | Load Data

EDIT	MONTHNUMBER	MONTHNAME
	1	January
	2	February
	3	March
	4	April
	5	May
	6	June
	7	July
	8	August
	9	September
	10	October
	11	November
	12	December

Step 3: Write the SQL Query

WITH MonthlyPurchases AS (

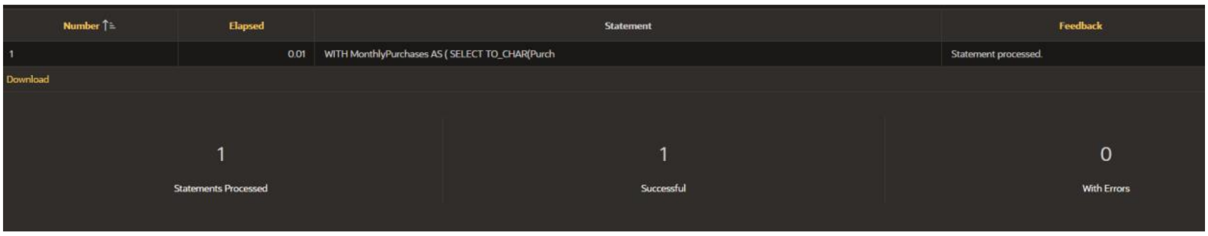
SELECT

TO_CHAR(p.PurchaseDate, 'MM') AS MonthNumber,

TO_CHAR(p.PurchaseDate, 'Month') AS MonthName,

COUNT(DISTINCT p.CustomerID) AS CustomerCount

```
FROM
Purchases p
WHERE
EXTRACT(YEAR FROM p.PurchaseDate) = EXTRACT(YEAR FROM SYSDATE)
GROUP BY
TO_CHAR(p.PurchaseDate, 'MM'), TO_CHAR(p.PurchaseDate, 'Month')
)
SELECT
cm.MonthName,
COALESCE(mp.CustomerCount, 0) AS CustomerCount
FROM
CalendarMonths cm
LEFT JOIN
MonthlyPurchases mp ON cm.MonthNumber = mp.MonthNumber
ORDER BY
cm.MonthNumber;
```



Explanation:

- Calendar Table: `CalendarMonths` ensures all months are covered.

- Distinct Customer Counts: `MonthlyPurchases` calculates distinct customers per month.

- Including Zero Counts: `LEFT JOIN` and `COALESCE` ensure all months are included, even those with zero activity.

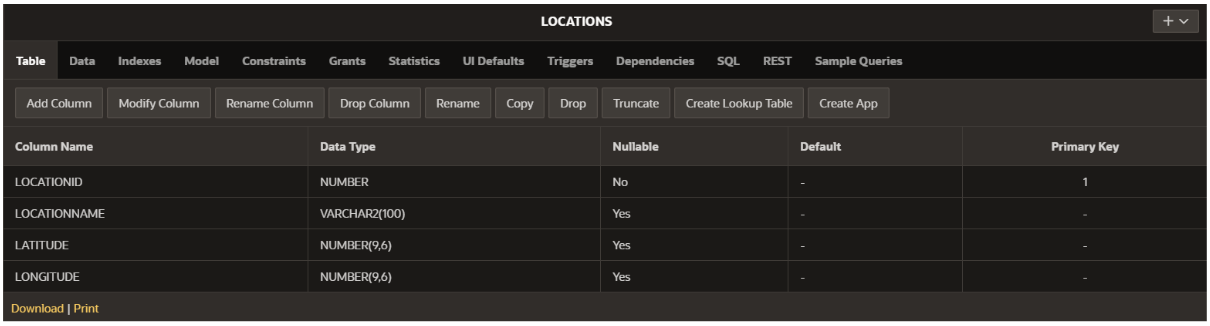
Question 4: Finding Closest Locations

Step 1: Create Table

```
CREATE TABLE Locations (
LocationID INT PRIMARY KEY,
LocationName VARCHAR(100),
Latitude DECIMAL(9, 6),
```

Longitude DECIMAL(9, 6)

);



Column Name	Data Type	Nullable	Default	Primary Key
LOCATIONID	NUMBER	No	-	1
LOCATIONNAME	VARCHAR2(100)	Yes	-	-
LATITUDE	NUMBER(9,6)	Yes	-	-
LONGITUDE	NUMBER(9,6)	Yes	-	-

Step 2: Insert Sample Data

INSERT INTO Locations (LocationID, LocationName, Latitude, Longitude) VALUES (1, 'Location A', 40.730610, -73.935242);

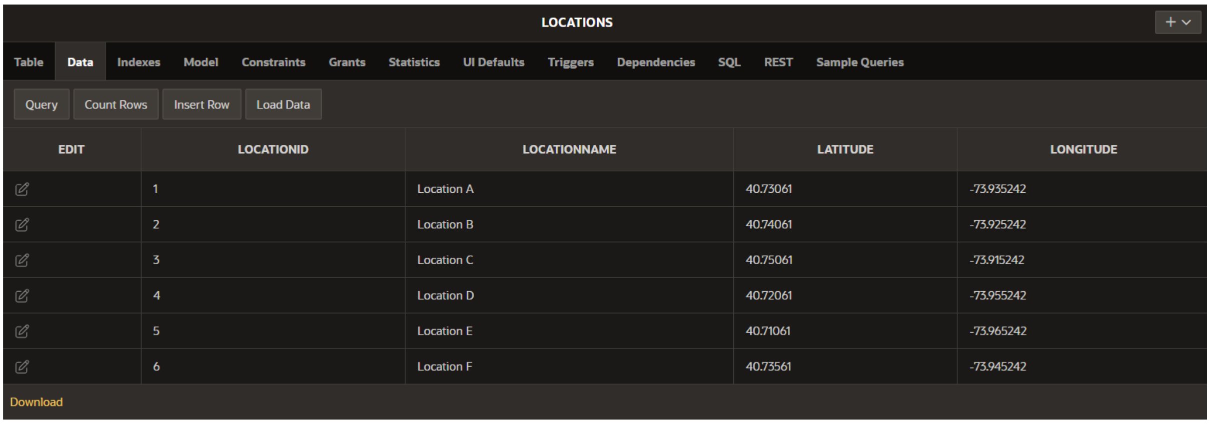
INSERT INTO Locations (LocationID, LocationName, Latitude, Longitude) VALUES (2, 'Location B', 40.740610, -73.925242);

INSERT INTO Locations (LocationID, LocationName, Latitude, Longitude) VALUES (3, 'Location C', 40.750610, -73.915242);

INSERT INTO Locations (LocationID, LocationName, Latitude, Longitude) VALUES (4, 'Location D', 40.720610, -73.955242);

INSERT INTO Locations (LocationID, LocationName, Latitude, Longitude) VALUES (5, 'Location E', 40.710610, -73.965242);

INSERT INTO Locations (LocationID, LocationName, Latitude, Longitude) VALUES (6, 'Location F', 40.735610, -73.945242);



EDIT	LOCATIONID	LOCATIONNAME	LATITUDE	LONGITUDE
	1	Location A	40.73061	-73.935242
	2	Location B	40.74061	-73.925242
	3	Location C	40.75061	-73.915242
	4	Location D	40.72061	-73.955242
	5	Location E	40.71061	-73.965242
	6	Location F	40.73561	-73.945242

Step 3: Write the SQL Query

DECLARE

v_Latitude DECIMAL(9, 6) := 40.730610; -- Example Latitude

v_Longitude DECIMAL(9, 6) := -73.935242; -- Example Longitude

BEGIN

SELECT

LocationID,

LocationName,

Latitude,

Longitude,

(6371 * ACOS(

COS(RADIANS(v_Latitude)) * COS(RADIANS(Latitude)) *

COS(RADIANS(Longitude) - RADIANS(v_Longitude)) +

SIN(RADIANS(v_Latitude)) * SIN(RADIANS(Latitude))

)) AS Distance

FROM

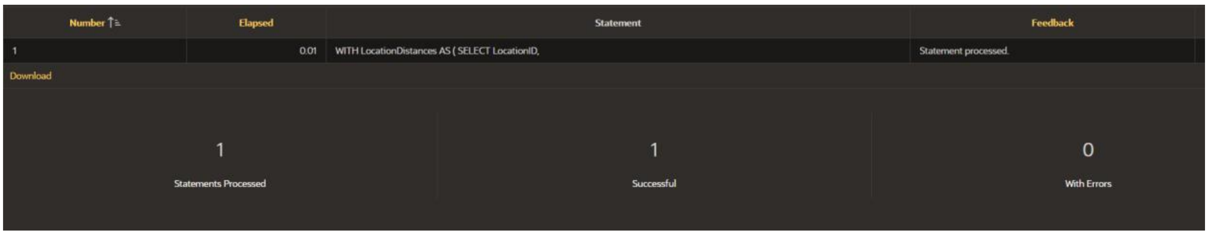
Locations

ORDER BY

Distance

FETCH FIRST 5 ROWS ONLY;

END;



Explanation:

- Distance Calculation: Uses the Haversine formula to compute distances based on latitude and longitude.

- Ordering and Limiting: Results are sorted by calculated distance, and the closest 5 locations are returned.

Question 5: Optimizing Query for Orders Table

Step 1: Create Table

CREATE TABLE Orders (

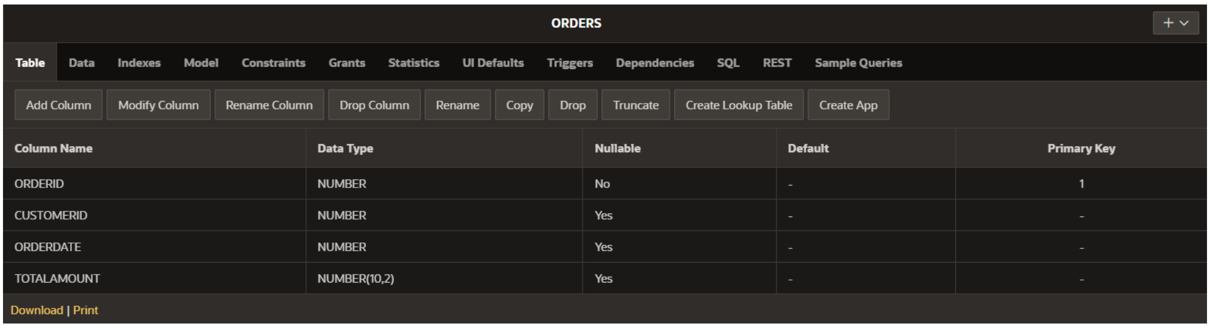
OrderID INT PRIMARY KEY,

CustomerID INT,

OrderDate DATE,

TotalAmount DECIMAL(10, 2)

);



Column Name	Data Type	Nullable	Default	Primary Key
ORDERID	NUMBER	No	-	1
CUSTOMERID	NUMBER	Yes	-	-
ORDERDATE	NUMBER	Yes	-	-
TOTALAMOUNT	NUMBER(10,2)	Yes	-	-

Step 2: Insert Sample Data

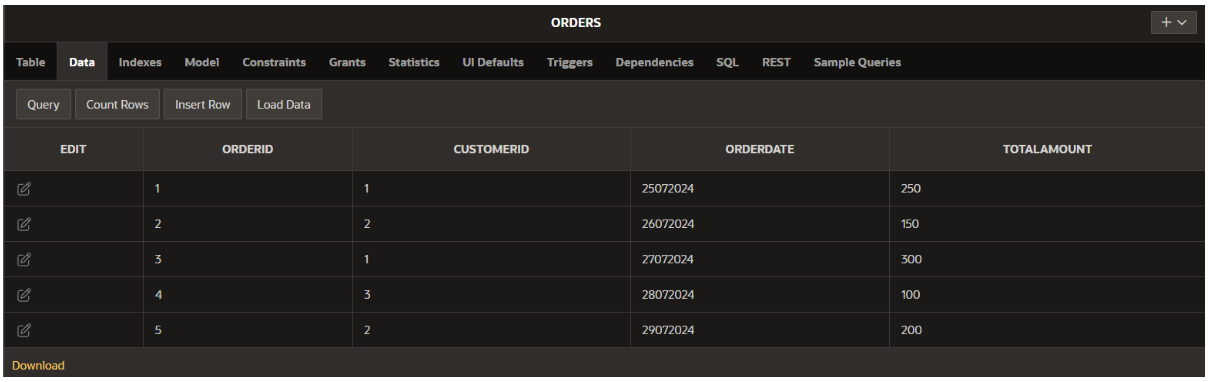
INSERT INTO Orders (OrderID, CustomerID, OrderDate, TotalAmount) VALUES (1, 1, '2024-07-25', 250.00);

INSERT INTO Orders (OrderID, CustomerID, OrderDate, TotalAmount) VALUES (2, 2, '2024-07-26', 150.00);

INSERT INTO Orders (OrderID, CustomerID, OrderDate, TotalAmount) VALUES (3, 1, '2024-07-27', 300.00);

INSERT INTO Orders (OrderID, CustomerID, OrderDate, TotalAmount) VALUES (4, 3, '2024-07-28', 100.00);

INSERT INTO Orders (OrderID, CustomerID, OrderDate, TotalAmount) VALUES (5, 2, '2024-07-29', 200.00);



EDIT	ORDERID	CUSTOMERID	ORDERDATE	TOTALAMOUNT
	1	1	25072024	250
	2	2	26072024	150
	3	1	27072024	300
	4	3	28072024	100
	5	2	29072024	200

Step 3: Write the SQL Query

SELECT

OrderID,

CustomerID,

OrderDate,

TotalAmount

FROM

Orders

WHERE

OrderDate >= CURRENT_DATE - INTERVAL '7' DAY

ORDER BY

OrderDate DESC;

```
SELECT      OrderID,      CustomerID,      TO_CHAR(TO_DATE(OrderDate, 'DDMMYYYY'), 'DDMMYYYY') AS OrderDate,      TotalAmount FROM      Orders WHERE      TO_DATE(OrderDate, 'DDMMYYYY') >= TRUNC(SYSDATE) - INTERVAL '7' DAY ORDER
BY      TO_DATE(OrderDate, 'DDMMYYYY') DESC
```

ORDERID	CUSTOMERID	ORDERDATE	TOTALAMOUNT
5	2	29072024	200
4	3	28072024	100
3	1	27072024	300
2	2	26072024	150
1	1	25072024	250

5 rows selected. 0.01 seconds

Optimization Strategies:

1. Indexing: Create an index on the `OrderDate` column to speed up queries filtering by date.

2. Query Rewriting: Using `CURRENT_DATE - INTERVAL '7' DAY` ensures efficient date range filtering.

3. Performance Monitoring: Regularly review execution plans and adjust indexes as needed based on query performance.