***Task 1: Database Creation and Table Setup***

1. ***Create a database named RailwayManagementDB.***

Ans = CREATE DATABASE RailwayManagementDB;

1. ***Create the following tables to track train information,*** schedules***, routes, passengers, and bookings:***

***o Trains: To store information about trains (TrainID, TrainName, TrainType, TotalSeats).***

***o Routes: To store information about routes (RouteID, StartStation, EndStation, Distance).***

***o Schedules: To store train schedules (ScheduleID, TrainID, RouteID, DepartureTime, ArrivalTime).***

***o Passengers: To store passenger information (PassengerID, FirstName, LastName, Age, Email).***

***o Bookings: To store booking details (BookingID, PassengerID, ScheduleID, BookingDate, SeatNumber).***

ans= -- Table: Trains

CREATE TABLE Trains (

TrainID INT,

TrainName VARCHAR(50),

TrainType VARCHAR(20),

TotalSeats INT

);

-- Table: Routes

CREATE TABLE Routes (

RouteID INT,

StartStation VARCHAR(50),

EndStation VARCHAR(50),

Distance INT

);

-- Table: Schedules

CREATE TABLE Schedules (

ScheduleID INT,

TrainID INT,

RouteID INT,

DepartureTime DATETIME,

ArrivalTime DATETIME

);

-- Table: Passengers

CREATE TABLE Passengers (

PassengerID INT,

FirstName VARCHAR(50),

LastName VARCHAR(50),

Age INT,

Email VARCHAR(100)

);

-- Table: Bookings

CREATE TABLE Bookings (

BookingID INT,

PassengerID INT,

ScheduleID INT,

BookingDate DATE,

SeatNumber INT

);

***3. Insert sample data into the tables for at least:***

***o 5 trains***

***o 3 routes***

***o 5 schedules***

***o 10 passengers***

***o 5 bookings***

ans = -- Inserting data into Trains table

INSERT INTO Trains VALUES (1, 'Rajdhani Express', 'Express', 300);

INSERT INTO Trains VALUES (2, 'Tejas Express', 'Superfast', 200);

INSERT INTO Trains VALUES (3, 'Shatabdi Express', 'Passenger', 250);

INSERT INTO Trains VALUES (4, 'Duronto Express', 'Superfast', 150);

INSERT INTO Trains VALUES (5, 'Garib Rath', 'Express', 350);

-- Inserting data into Routes table

INSERT INTO Routes VALUES (1, 'Delhi', 'Mumbai', 1400);

INSERT INTO Routes VALUES (2, 'Kolkata', 'Chennai', 1650);

INSERT INTO Routes VALUES (3, 'Jaipur', 'Ahmedabad', 650);

-- Inserting data into Schedules table

INSERT INTO Schedules VALUES (1, 1, 1, '2024-10-20 09:00:00', '2024-10-20 21:00:00');

INSERT INTO Schedules VALUES (2, 2, 2, '2024-10-21 08:30:00', '2024-10-21 22:00:00');

INSERT INTO Schedules VALUES (3, 3, 3, '2024-10-22 06:00:00', '2024-10-22 14:00:00');

INSERT INTO Schedules VALUES (4, 4, 1, '2024-10-23 10:00:00', '2024-10-23 22:00:00');

INSERT INTO Schedules VALUES (5, 5, 2, '2024-10-24 07:00:00', '2024-10-24 21:00:00');

-- Inserting data into Passengers table

INSERT INTO Passengers VALUES (1, 'Rajesh', 'Sharma', 45, 'rajesh.sharma@specialforce.com');

INSERT INTO Passengers VALUES (2, 'Priya', 'Mehra', 32, 'priya.mehra@specialforce.com');

INSERT INTO Passengers VALUES (3, 'Ankit', 'Verma', 29, 'ankit.verma@specialforce.com');

INSERT INTO Passengers VALUES (4, 'Kavita', 'Gupta', 40, 'kavita.gupta@specialforce.com');

INSERT INTO Passengers VALUES (5, 'Arun', 'Patel', 50, 'arun.patel@specialforce.com');

INSERT INTO Passengers VALUES (6, 'Neha', 'Joshi', 27, 'neha.joshi@specialforce.com');

INSERT INTO Passengers VALUES (7, 'Suresh', 'Nair', 33, 'suresh.nair@specialforce.com');

INSERT INTO Passengers VALUES (8, 'Pooja', 'Reddy', 36, 'pooja.reddy@specialforce.com');

INSERT INTO Passengers VALUES (9, 'Vikram', 'Singh', 42, 'vikram.singh@specialforce.com');

INSERT INTO Passengers VALUES (10, 'Aarti', 'Desai', 25, 'aarti.desai@specialforce.com');

-- Inserting data into Bookings table

INSERT INTO Bookings VALUES (1, 1, 1, '2024-10-10', 12);

INSERT INTO Bookings VALUES (2, 2, 1, '2024-10-11', 34);

INSERT INTO Bookings VALUES (3, 3, 2, '2024-10-12', 56);

INSERT INTO Bookings VALUES (4, 4, 3, '2024-10-13', 18);

INSERT INTO Bookings VALUES (5, 5, 4, '2024-10-14', 22);

***Task 2: Add Constraints After Data Insertion (Strictly write after data insertion)***

***1. Add a Primary Key to each table.***

***2. Add Foreign Keys to establish relationships between:***

***o Schedules and Trains (on TrainID).***

***o Schedules and Routes (on RouteID).***

***o Bookings and Passengers (on PassengerID).***

***o Bookings and Schedules (on ScheduleID).***

Ans ***=***

ALTER TABLE Trains ADD PRIMARY KEY (TrainID);

ALTER TABLE Routes ADD PRIMARY KEY (RouteID);

ALTER TABLE Schedules ADD PRIMARY KEY (ScheduleID);

ALTER TABLE Passengers ADD PRIMARY KEY (PassengerID);

ALTER TABLE Bookings ADD PRIMARY KEY (BookingID);

Foreign key:

ALTER TABLE Schedules

ADD CONSTRAINT fk\_train

FOREIGN KEY (TrainID) REFERENCES Trains(TrainID);

ALTER TABLE Schedules

ADD CONSTRAINT fk\_route

FOREIGN KEY (RouteID) REFERENCES Routes(RouteID);

ALTER TABLE Bookings

ADD CONSTRAINT fk\_passenger

FOREIGN KEY (PassengerID) REFERENCES Passengers(PassengerID);

ALTER TABLE Bookings

ADD CONSTRAINT fk\_schedule

FOREIGN KEY (ScheduleID) REFERENCES Schedules(ScheduleID);

***Task 3: Joins and Queries***

1. ***Query 1: Write a query to retrieve the train name, route details, and schedule for all trains using an INNER JOIN between the Trains, Routes, and Schedules tables.***

Ans = SELECT

T.TrainName,

R.StartStation, R.EndStation, R.Distance,

S.DepartureTime, S.ArrivalTime

FROM

Trains T

INNER JOIN

Schedules S ON T.TrainID = S.TrainID

INNER JOIN

Routes R ON S.RouteID = R.RouteID;

1. ***Query 2: Write a query to retrieve all trains that don't have any bookings using a LEFT JOIN between the Trains and Bookings tables.***

ans = SELECT

T.TrainName

FROM

Trains T

LEFT JOIN

Schedules S ON T.TrainID = S.TrainID

LEFT JOIN

Bookings B ON S.ScheduleID = B.ScheduleID

WHERE

B.BookingID IS NULL;

1. ***Query 3: Write a query to find all passengers who have booked seats for trains traveling a distance of more than 500 km using a RIGHT JOIN and subquery.***

Ans = SELECT

P.FirstName, P.LastName, R.Distance

FROM

Passengers P

RIGHT JOIN

Bookings B ON P.PassengerID = B.PassengerID

JOIN

Schedules S ON B.ScheduleID = S.ScheduleID

JOIN

Routes R ON S.RouteID = R.RouteID

WHERE

R.Distance > 500;

1. ***Query 4: Write a query to list all train schedules, even if there are no passengers booked, using an OUTER JOIN.***

***Ans =***

SELECT

T.TrainName,

S.DepartureTime, S.ArrivalTime,

B.BookingID, P.FirstName, P.LastName

FROM

Schedules S

LEFT JOIN

Trains T ON S.TrainID = T.TrainID

LEFT JOIN

Bookings B ON S.ScheduleID = B.ScheduleID

LEFT JOIN

Passengers P ON B.PassengerID = P.PassengerID;

***Task 4: Normalization***

1. ***Normalize the tables to the 3rd Normal Form (3NF) to eliminate redundancy and ensure data integrity.***

***Task 5: Sub Queries***

1. ***Query 5: Write a query to calculate the total number of passengers for each train route.***

Ans = SELECT R.RouteID, R.StartStation, R.EndStation, COUNT(B.PassengerID) AS TotalPassengers

FROM Routes R, Schedules S, Bookings B

WHERE R.RouteID = S.RouteID AND S.ScheduleID = B.ScheduleID

GROUP BY R.RouteID;

1. ***Query 6: Write a query to find the average number of passengers booked per train.***

Ans = SELECT T.TrainName, AVG(PassengerCount.Total) AS AvgPassengers

FROM Trains T

JOIN (

SELECT S.TrainID, COUNT(B.PassengerID) AS Total

FROM Schedules S

LEFT JOIN Bookings B ON S.ScheduleID = B.ScheduleID

GROUP BY S.TrainID

) AS PassengerCount ON T.TrainID = PassengerCount.TrainID

GROUP BY T.TrainID;

1. ***Query 7: Write a query to find the train with the highest number of bookings.***

Ans = SELECT T.TrainName

FROM Trains T

JOIN Schedules S ON T.TrainID = S.TrainID

JOIN Bookings B ON S.ScheduleID = B.ScheduleID

GROUP BY T.TrainID

ORDER BY COUNT(B.BookingID) DESC

LIMIT 1;

1. ***Query 8: Write a query to find the total seats booked per train route where the booking date is between 01-Jan-2023 and 31-Dec-2023.***

Ans = SELECT R.RouteID, COUNT(B.SeatNumber) AS TotalSeats

FROM Routes R, Schedules S, Bookings B

WHERE R.RouteID = S.RouteID AND S.ScheduleID = B.ScheduleID AND B.BookingDate BETWEEN '2023-01-01' AND '2023-12-31'

GROUP BY R.RouteID;

***5. Query 9: Write a query to list all bookings where the passenger's age is greater than 60.***

Ans = SELECT B.BookingID, P.FirstName, P.LastName, B.SeatNumber

FROM Bookings B

JOIN Passengers P ON B.PassengerID = P.PassengerID

WHERE P.Age > 60;

***Task 6: Stored Procedures and Functions***

1. ***Write a stored procedure to update the number of available seats in a train after a booking has been made.***

***Ans=***

DELIMITER //

CREATE PROCEDURE UpdateAvailableSeats(IN TrainID INT)

BEGIN

DECLARE TotalSeats INT;

DECLARE BookedSeats INT;

-- Get total seats of the train

SELECT TotalSeats INTO TotalSeats FROM Trains WHERE TrainID = TrainID;

-- Get the number of seats already booked

SELECT COUNT(SeatNumber) INTO BookedSeats

FROM Bookings B

JOIN Schedules S ON B.ScheduleID = S.ScheduleID

WHERE S.TrainID = TrainID;

-- Update available seats

UPDATE Trains

SET TotalSeats = TotalSeats - BookedSeats

WHERE TrainID = TrainID;

END //

DELIMITER ;

1. ***Write a function to calculate the total travel time (in hours) between two stations based on departure and arrival times from the Schedules table.***

***Ans =***

DELIMITER //

CREATE FUNCTION GetTravelTime(ScheduleID INT)

RETURNS DECIMAL(5,2)

DETERMINISTIC

BEGIN

DECLARE travel\_time DECIMAL(5,2);

-- Calculate the time difference between Departure and Arrival time

SELECT TIMESTAMPDIFF(HOUR, DepartureTime, ArrivalTime) INTO travel\_time

FROM Schedules

WHERE ScheduleID = ScheduleID;

RETURN travel\_time;

END //

DELIMITER ;

***Task 7: Views and Indexes***

1. ***Create a view named PassengerBookingsView that combines passenger details, train information, and booking details in one query.***

Ans = CREATE VIEW PassengerBookingsView AS

SELECT P.PassengerID, P.FirstName, P.LastName, T.TrainName, B.SeatNumber, B.BookingDate

FROM Passengers P

JOIN Bookings B ON P.PassengerID = B.PassengerID

JOIN Schedules S ON B.ScheduleID = S.ScheduleID

JOIN Trains T ON S.TrainID = T.TrainID;

1. ***Create an index on the Bookings table to improve the performance of queries filtering by BookingDate.***

Ans = CREATE INDEX idx\_booking\_date ON Bookings(BookingDate);

***Task 8: Temporary Tables***

1. ***Create a temporary table to store the schedule of all trains departing on a specific day (for example, 15-Oct-2023), and then query it.***

Ans = CREATE TEMPORARY TABLE TempTrainSchedule AS

-> SELECT S.ScheduleID, T.TrainName, S.DepartureTime, S.ArrivalTime

-> FROM Schedules S

-> JOIN Trains T ON S.TrainID = T.TrainID

-> WHERE DATE(S.DepartureTime) = '2023-10-15'***;***

***Task 9: Cursors***

1. ***Write a basic cursor to iterate over the passengers who have booked more than 5 tickets.***

Ans = DELIMITER //

CREATE PROCEDURE CursorPassengers()

BEGIN

DECLARE done INT DEFAULT 0;

DECLARE p\_firstname VARCHAR(50);

DECLARE p\_lastname VARCHAR(50);

DECLARE cur CURSOR FOR

SELECT FirstName, LastName FROM Passengers P

WHERE (SELECT COUNT(\*) FROM Bookings B WHERE B.PassengerID = P.PassengerID) > 5;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

OPEN cur;

read\_loop: LOOP

FETCH cur INTO p\_firstname, p\_lastname;

IF done THEN

LEAVE read\_loop;

END IF;

-- Do something with each passenger

SELECT p\_firstname, p\_lastname;

END LOOP;

CLOSE cur;

END //

DELIMITER ;

***Task 10: ACID Properties and Transactions***

***1. Ensure the database follows ACID properties by using transactions:***

***o Begin a transaction before updating seat availability.***

***o Use a savepoint before deleting any booking records.***

***o Rollback if an error occurs, ensuring no changes are made to the database.***

***o Commit the transaction only after all changes have been successfully applied.***

Ans = START TRANSACTION;

-- Begin the transaction

SAVEPOINT BeforeUpdate;

-- Update seat availability

CALL UpdateAvailableSeats(1);

-- Simulate an error condition

-- ROLLBACK to savepoint if there’s any issue

ROLLBACK TO BeforeUpdate;

-- Commit the transaction after the updates are successful

COMMIT;

***Task 11: Triggers (Understand which trigger to use when and for what)***

1. ***Create a Trigger that automatically assigns a seat number to passengers when a booking is created.***

Ans = DELIMITER //

CREATE TRIGGER AssignSeatNumber

BEFORE INSERT ON Bookings

FOR EACH ROW

BEGIN

DECLARE max\_seat INT;

-- Get the maximum seat number booked for the same schedule

SELECT IFNULL(MAX(SeatNumber), 0) INTO max\_seat

FROM Bookings

WHERE ScheduleID = NEW.ScheduleID;

-- Assign the next available seat number

SET NEW.SeatNumber = max\_seat + 1;

END //

DELIMITER ;

1. ***Create a Trigger that updates the total available seats in the train after a booking is confirmed.***

Ans = DELIMITER //

CREATE TRIGGER UpdateTrainSeats

AFTER INSERT ON Bookings

FOR EACH ROW

BEGIN

-- Update available seats in the train after booking confirmation

CALL UpdateAvailableSeats((SELECT TrainID FROM Schedules WHERE ScheduleID = NEW.ScheduleID));

END //

DELIMITER ;

***Task 12: UNION and UNION ALL***

1. ***Write a query to combine the results of two queries that return passengers booked on trains for two different routes.***

Ans = SELECT P.FirstName, P.LastName, R.StartStation, R.EndStation

FROM Passengers P

JOIN Bookings B ON P.PassengerID = B.PassengerID

JOIN Schedules S ON B.ScheduleID = S.ScheduleID

JOIN Routes R ON S.RouteID = R.RouteID

WHERE R.RouteID = 1

UNION

SELECT P.FirstName, P.LastName, R.StartStation, R.EndStation

FROM Passengers P

JOIN Bookings B ON P.PassengerID = B.PassengerID

JOIN Schedules S ON B.ScheduleID = S.ScheduleID

JOIN Routes R ON S.RouteID = R.RouteID

WHERE R.RouteID = 2;

1. ***Write a query to combine the results of all bookings made on different dates.***

Ans = SELECT B.BookingID, B.BookingDate, P.FirstName, P.LastName

FROM Bookings B

JOIN Passengers P ON B.PassengerID = P.PassengerID

WHERE B.BookingDate = '2024-10-10'

UNION ALL

SELECT B.BookingID, B.BookingDate, P.FirstName, P.LastName

FROM Bookings B

JOIN Passengers P ON B.PassengerID = P.PassengerID

WHERE B.BookingDate = '2024-10-11';

***Task 13: Copying Tables***

1. ***Copy the structure of the Passengers table into a new table named OldPassengers.***

Ans = CREATE TABLE OldPassengers LIKE Passengers;

1. ***Copy all the data from the Bookings table into another table named ArchivedBookings.***

Ans = CREATE TABLE ArchivedBookings AS

SELECT \* FROM Bookings;

***Additional Assignment: Keys Practice (You should know exact difference and how to use***

***them)***

***Duration: 15-20 Minutes***

***Scenario:***

***SpecialForce Private Limited has an Employee Management System that stores employee data. The***

***management system uses various keys to maintain data integrity and ensure fast retrieval of information. You***

***are tasked with identifying and implementing different types of keys in the database schema provided below.***

***Employee Table:***

***EmpID (PK) FirstName LastName Email PhoneNumber Department***

***101 Rajesh Sharma rajesh@specialforce.com 9876543210 HR***

***102 Priya Mehra priya@specialforce.com 8765432109 Finance***

***103 Ankit Verma ankit@specialforce.com 7654321098 IT***

***104 Kavita Gupta kavita@specialforce.com 6543210987 IT***

***105 Suresh Nair suresh@specialforce.com 5432109876 Sales***

***Assignment Tasks:***

***1. Super Key:***

***Identify all possible Super Keys from the Employee table. Remember, a Super Key is any combination***

***of columns that uniquely identifies each record.***

***2. Candidate Key:***

***Determine which of the Super Keys qualify as Candidate Keys. A Candidate Key is a minimal Super***

***Key, meaning it contains no unnecessary columns.***

***3. Primary Key:***

***What is the Primary Key for the Employee table? Explain why this key was chosen as the Primary Key.***

***4. Alternate Key:***

***Identify any Alternate Keys. Alternate Keys are Candidate Keys that were not chosen as the Primary***

***Key.***

***5. Composite Key:***

***Suppose the company wants to track employees working on multiple projects, where both the EmpID and***

***ProjectID are needed to uniquely identify records in a new EmployeeProjects table.***

***Define a Composite Key for this table using EmpID and ProjectID.***

***EmployeeProjects Table:***

***EmpID ProjectID***

***101 P01***

***102 P02***

***103 P03***

***104 P01***

***105 P03***

1. Super Key:

A Super Key is any combination of attributes that can uniquely identify a row in a table. It can contain more columns than necessary but must guarantee uniqueness.

* Super Keys for the Employee Table:
  + (EmpID)
  + (EmpID, FirstName)
  + (EmpID, LastName)
  + (EmpID, Email)
  + (EmpID, PhoneNumber)
  + (Email)
  + (PhoneNumber)

Any combination that includes EmpID, Email, or PhoneNumber will qualify as a Super Key because these fields can uniquely identify each employee.

2. Candidate Key:

A Candidate Key is a minimal Super Key. It contains only the necessary columns to uniquely identify a row. No column in a Candidate Key can be removed without losing the unique identification property.

* Candidate Keys for the Employee Table:
  + (EmpID)
  + (Email)
  + (PhoneNumber)

These are the minimal Super Keys since they are sufficient to uniquely identify a record, and none of the columns can be removed while still maintaining uniqueness.

3. Primary Key:

The Primary Key is the main key chosen to uniquely identify rows in a table. It cannot contain NULL values.

* Primary Key for the Employee Table: EmpID

Explanation: EmpID is chosen as the Primary Key because it is unique for every employee, and it's the most intuitive choice for identifying employees in the system. It is often system-generated and immutable, which ensures data integrity and performance during retrieval operations.

4. Alternate Key:

An Alternate Key is a Candidate Key that was not chosen as the Primary Key.

* Alternate Keys for the Employee Table:
  + Email
  + PhoneNumber

Since these could have been Candidate Keys but weren't chosen as the Primary Key, they are classified as Alternate Keys.

5. Composite Key:

A Composite Key is a key that consists of two or more columns used together to uniquely identify a record in a table. It is necessary when a single column is not sufficient for unique identification.

* Composite Key for EmployeeProjects Table: (EmpID, ProjectID)

The combination of EmpID and ProjectID is required to uniquely identify which employees are working on which projects. Neither EmpID nor ProjectID alone is sufficient, but together they ensure uniqueness.