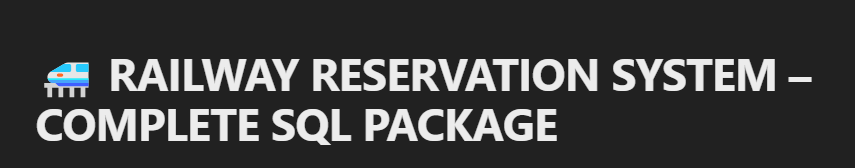
****

**Tables:**

CREATE TABLE Passenger (

PassengerID INT PRIMARY KEY AUTO\_INCREMENT,

Name VARCHAR(100) NOT NULL,

Age INT CHECK (Age > 0),

Gender ENUM('Male', 'Female', 'Other') NOT NULL,

ContactNumber VARCHAR(15) NOT NULL,

Email VARCHAR(100),

ConcessionCategory ENUM('Senior Citizen', 'Student', 'Disabled', 'None') DEFAULT 'None'

);

CREATE TABLE Station (

StationID INT PRIMARY KEY AUTO\_INCREMENT,

StationName VARCHAR(100) NOT NULL,

Location VARCHAR(100)

);

CREATE TABLE Train (

TrainID INT PRIMARY KEY AUTO\_INCREMENT,

TrainName VARCHAR(100) NOT NULL,

TrainType ENUM('Express', 'Superfast', 'Mail', 'Local') NOT NULL,

SourceStationID INT,

DestinationStationID INT,

FOREIGN KEY (SourceStationID) REFERENCES Station(StationID),

FOREIGN KEY (DestinationStationID) REFERENCES Station(StationID)

);

CREATE TABLE Route (

RouteID INT PRIMARY KEY AUTO\_INCREMENT,

TrainID INT,

StationID INT,

ArrivalTime TIME,

DepartureTime TIME,

StopNumber INT,

FOREIGN KEY (TrainID) REFERENCES Train(TrainID),

FOREIGN KEY (StationID) REFERENCES Station(StationID)

);

CREATE TABLE Schedule (

ScheduleID INT PRIMARY KEY AUTO\_INCREMENT,

TrainID INT,

Date DATE,

Status ENUM('Running', 'Cancelled') DEFAULT 'Running',

FOREIGN KEY (TrainID) REFERENCES Train(TrainID)

);

CREATE TABLE Class (

ClassID INT PRIMARY KEY AUTO\_INCREMENT,

ClassName ENUM('Sleeper', 'AC 3-Tier', 'AC 2-Tier', 'First Class') NOT NULL,

FarePerKM DECIMAL(6,2) NOT NULL

);

CREATE TABLE Seat (

SeatID INT PRIMARY KEY AUTO\_INCREMENT,

TrainID INT,

ClassID INT,

SeatNumber VARCHAR(10) NOT NULL,

FOREIGN KEY (TrainID) REFERENCES Train(TrainID),

FOREIGN KEY (ClassID) REFERENCES Class(ClassID)

);

CREATE TABLE Ticket (

TicketID INT PRIMARY KEY AUTO\_INCREMENT,

PassengerID INT,

ScheduleID INT,

ClassID INT,

BookingStatus ENUM('Confirmed', 'RAC', 'WL') NOT NULL,

SeatID INT, -- Can be NULL for WL/RAC

PNR VARCHAR(20) UNIQUE NOT NULL,

BookingDate DATE NOT NULL,

FOREIGN KEY (PassengerID) REFERENCES Passenger(PassengerID),

FOREIGN KEY (ScheduleID) REFERENCES Schedule(ScheduleID),

FOREIGN KEY (ClassID) REFERENCES Class(ClassID),

FOREIGN KEY (SeatID) REFERENCES Seat(SeatID)

);

CREATE TABLE Payment (

PaymentID INT PRIMARY KEY AUTO\_INCREMENT,

TicketID INT UNIQUE,

PaymentMode ENUM('Online', 'Counter') NOT NULL,

Amount DECIMAL(10,2) NOT NULL,

PaymentDate DATE NOT NULL,

RefundStatus ENUM('Yes', 'No') DEFAULT 'No',

FOREIGN KEY (TicketID) REFERENCES Ticket(TicketID)

);

**RL Tables:**

CREATE TABLE TicketPassenger (

TicketID INT,

PassengerID INT,

PRIMARY KEY (TicketID, PassengerID),

FOREIGN KEY (TicketID) REFERENCES Ticket(TicketID),

FOREIGN KEY (PassengerID) REFERENCES Passenger(PassengerID)

);

CREATE TABLE SeatAvailability (

ScheduleID INT,

SeatID INT,

IsBooked BOOLEAN DEFAULT FALSE,

PRIMARY KEY (ScheduleID, SeatID),

FOREIGN KEY (ScheduleID) REFERENCES Schedule(ScheduleID),

FOREIGN KEY (SeatID) REFERENCES Seat(SeatID)

);

CREATE TABLE TrainClass (

TrainID INT,

ClassID INT,

PRIMARY KEY (TrainID, ClassID),

FOREIGN KEY (TrainID) REFERENCES Train(TrainID),

FOREIGN KEY (ClassID) REFERENCES Class(ClassID)

);

CREATE TABLE RAC\_WL\_Status (

TicketID INT PRIMARY KEY,

QueueNumber INT NOT NULL,

CurrentStatus ENUM('RAC', 'WL') NOT NULL,

FOREIGN KEY (TicketID) REFERENCES Ticket(TicketID)

);

**Foreign Key Relationships Summary**

Table -> FK References

Train -> Station(SourceStationID, DestinationStationID)

Route ->Train(TrainID), Station(StationID)

Schedule -> Train(TrainID)

Seat -> Train(TrainID), Class(ClassID)

Ticket -> Passenger, Schedule, Class, Seat

Payment -> Ticket

**Procedures:**

-- 1. PNR status tracking for a given ticket

DELIMITER //

CREATE PROCEDURE GetPNRStatus(IN pnr VARCHAR(20))

BEGIN

SELECT PNR, BookingStatus, SeatID

FROM Ticket

WHERE PNR = pnr;

END //

DELIMITER ;

-- 2. Train schedule lookup for a given train

DELIMITER //

CREATE PROCEDURE GetTrainSchedule(IN train\_id INT)

BEGIN

SELECT s.ScheduleID, s.Date, s.Status

FROM Schedule s

WHERE s.TrainID = train\_id;

END //

DELIMITER ;

-- 3. Available seats query for a specific train, date and class

DELIMITER //

CREATE PROCEDURE GetAvailableSeats(

IN train\_id INT,

IN journey\_date DATE,

IN class\_id INT)

BEGIN

SELECT sa.SeatID

FROM SeatAvailability sa

JOIN Seat s ON sa.SeatID = s.SeatID

JOIN Schedule sch ON sa.ScheduleID = sch.ScheduleID

WHERE sch.TrainID = train\_id AND sch.Date = journey\_date

AND s.ClassID = class\_id AND sa.IsBooked = FALSE;

END //

DELIMITER ;

-- 4. List all passengers traveling on a specific train on a given date

DELIMITER //

CREATE PROCEDURE GetPassengersByTrainDate(

IN train\_id INT,

IN journey\_date DATE)

BEGIN

SELECT DISTINCT p.PassengerID, p.Name, p.Age, p.Gender

FROM Passenger p

JOIN Ticket t ON p.PassengerID = t.PassengerID

JOIN Schedule s ON t.ScheduleID = s.ScheduleID

WHERE s.TrainID = train\_id AND s.Date = journey\_date;

END //

DELIMITER ;

-- 5. Retrieve all waitlisted passengers for a particular train

DELIMITER //

CREATE PROCEDURE GetWaitlistedPassengers(IN train\_id INT)

BEGIN

SELECT p.PassengerID, p.Name, t.PNR

FROM Passenger p

JOIN Ticket t ON p.PassengerID = t.PassengerID

JOIN Schedule s ON t.ScheduleID = s.ScheduleID

WHERE s.TrainID = train\_id AND t.BookingStatus = 'WL';

END //

DELIMITER ;

-- 6. Find total amount to be refunded for cancelling a train

DELIMITER //

CREATE PROCEDURE GetTotalRefundForCancelledTrain(IN train\_id INT)

BEGIN

SELECT SUM(pay.Amount) AS TotalRefund

FROM Payment pay

JOIN Ticket t ON pay.TicketID = t.TicketID

JOIN Schedule s ON t.ScheduleID = s.ScheduleID

WHERE s.TrainID = train\_id AND s.Status = 'Cancelled' AND pay.RefundStatus = 'Yes';

END //

DELIMITER ;

-- 7. Total revenue generated from ticket bookings over a specified period

DELIMITER //

CREATE PROCEDURE GetRevenue(IN start\_date DATE, IN end\_date DATE)

BEGIN

SELECT SUM(Amount) AS TotalRevenue

FROM Payment

WHERE PaymentDate BETWEEN start\_date AND end\_date AND RefundStatus = 'No';

END //

DELIMITER ;

-- 8. Cancellation records with refund status

DELIMITER //

CREATE PROCEDURE GetCancellationRecords()

BEGIN

SELECT t.TicketID, t.PNR, s.Status AS TrainStatus, pay.RefundStatus

FROM Ticket t

JOIN Schedule s ON t.ScheduleID = s.ScheduleID

JOIN Payment pay ON t.TicketID = pay.TicketID

WHERE s.Status = 'Cancelled';

END //

DELIMITER ;

-- 9. Find the busiest route based on passenger count

DELIMITER //

CREATE PROCEDURE GetBusiestRoute()

BEGIN

SELECT r.StationID, COUNT(t.TicketID) AS PassengerCount

FROM Route r

JOIN Schedule s ON r.TrainID = s.TrainID

JOIN Ticket t ON s.ScheduleID = t.ScheduleID

GROUP BY r.StationID

ORDER BY PassengerCount DESC

LIMIT 1;

END //

DELIMITER ;

-- 10. Generate an itemized bill for a ticket including all charges

DELIMITER //

CREATE PROCEDURE GetItemizedBill(IN ticket\_id INT)

BEGIN

SELECT t.TicketID, t.PNR, c.ClassName, c.FarePerKM, p.Amount AS TotalAmount,

pay.PaymentMode, pay.PaymentDate

FROM Ticket t

JOIN Class c ON t.ClassID = c.ClassID

JOIN Payment p ON t.TicketID = p.TicketID

JOIN Payment pay ON p.TicketID = pay.TicketID

WHERE t.TicketID = ticket\_id;

END //

DELIMITER ;

**Triggers:**

-- 1. Trigger: Mark seat as booked when a ticket is confirmed

DELIMITER //

CREATE TRIGGER AfterTicketInsert

AFTER INSERT ON Ticket

FOR EACH ROW

BEGIN

IF NEW.BookingStatus = 'Confirmed' THEN

UPDATE SeatAvailability

SET IsBooked = TRUE

WHERE SeatID = NEW.SeatID AND ScheduleID = NEW.ScheduleID;

END IF;

END;

//

DELIMITER ;

-- 2. Trigger: Auto-refund when a train is cancelled

DELIMITER //

CREATE TRIGGER AfterTrainCancellation

AFTER UPDATE ON Schedule

FOR EACH ROW

BEGIN

IF NEW.Status = 'Cancelled' AND OLD.Status != 'Cancelled' THEN

UPDATE Payment p

JOIN Ticket t ON p.TicketID = t.TicketID

SET p.RefundStatus = 'Yes'

WHERE t.ScheduleID = NEW.ScheduleID;

END IF;

END;

//

DELIMITER ;

-- 3. Trigger: Placeholder for auto-assign RAC/WL seat if a confirmed seat is cancelled

-- You must create a RAC\_WL\_Queue table and logic to support this.

-- Pseudo-code logic here, skip if no queue table exists.

-- 4. Trigger: Validate payment amount before insert

DELIMITER //

CREATE TRIGGER BeforePaymentInsert

BEFORE INSERT ON Payment

FOR EACH ROW

BEGIN

IF NEW.Amount <= 0 THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Payment amount must be greater than 0';

END IF;

END;

//

DELIMITER ;

-- 5. Trigger: Prevent double booking of the same seat

DELIMITER //

CREATE TRIGGER PreventDoubleBooking

BEFORE INSERT ON Ticket

FOR EACH ROW

BEGIN

IF NEW.BookingStatus = 'Confirmed' AND EXISTS (

SELECT 1 FROM Ticket

WHERE SeatID = NEW.SeatID AND ScheduleID = NEW.ScheduleID AND BookingStatus = 'Confirmed'

) THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Seat already booked';

END IF;

END;

//

DELIMITER ;

