INDIAN INSTITUTE OF TECHNOLOGY PATNA

Department of Computer Science and Engineering

CS2202 - Database and Data Warehousing

PROJECT REPORT

Indian Railway Ticket Reservation System

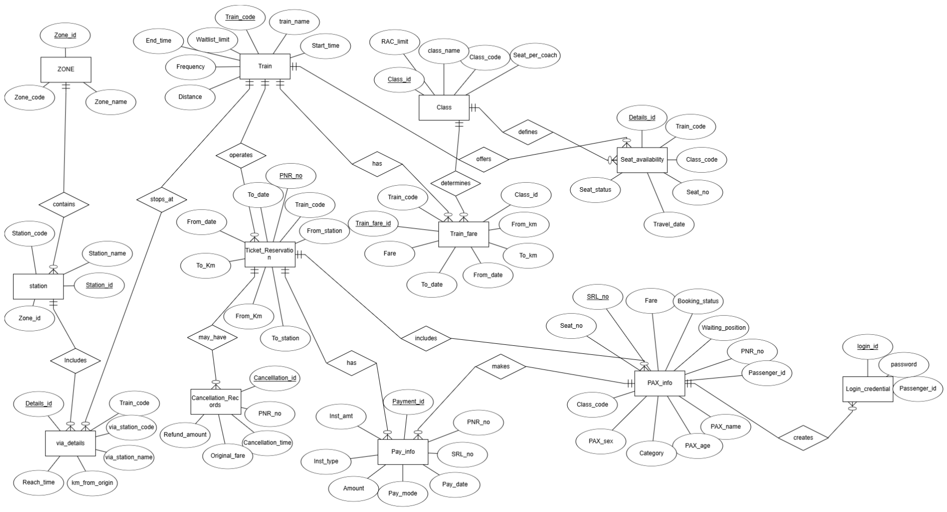
Contributors:

Pragya Mahajan
Mihika
Priyanshi Agrawal

Contents

1 1. ER Diagram with Descriptions

1.1 ER Diagram



1.2 Description of Entities and Relationships

- **Zone**: Represents different railway zones.
  - Attributes: Zone_id (PK), Zone_name, Zone_code.
- **Station**: Represents railway stations.
  - Attributes: Station_id (PK), Station_code, Station_name, Zone_id (FK).
  - Relationship: Each station belongs to a zone.
- **Train**: Represents train details.
  - Attributes: Train_code (PK), Train_name, Start_time, End_time, Distance, Frequency, Waitlist_limit.
- **Ticket_Reservation**: Represents ticket reservations.
  - Attributes: PNR_no (PK), Train_code (FK), From_station, To_station, From_Km, To_Km, From_date, To_date.
  - Relationship: Each reservation is linked to a train.
- **PAX_info**: Represents passenger information.
  - Attributes: SRL_no (PK), PNR_no (FK), PAX_Name, PAX_age, Category, PAX_sex, Class_code, Seat_no, Fare, Booking_status, Waitlist_position, Passenger_id (Unique).
  - Relationship: Each passenger is linked to a ticket reservation.
- **Pay_info**: Represents payment information.
  - Attributes: Payment_id (PK), PNR_no (FK), SRL_no (FK), Pay_date, Pay_mode, Amount, Inst_type, Inst_amt.
  - Relationship: Each payment is linked to a ticket reservation and passenger.
- **Login_credential**: Represents login credentials for passengers.
  - Attributes: login_id (PK), password, Passenger_id (FK).
  - Relationship: Each login credential is linked to a passenger.
- **Train_fare**: Represents fare details for trains.

  - Attributes: Train_fare_id (PK), Train_code (FK), Class_id, From_Km, To_Km, From_date, To_date, Fare.
  - Relationship: Each fare is linked to a train.

- **Class**: Represents different classes of travel.
  - Attributes: Class_id (PK), Class_code, Class_name, Seat_per_coach, RAC_limit.
- **Seat_availability**: Represents seat availability for trains.
  - Attributes: Details_id (PK), Train_code (FK), Class_code, Seat_No, travel_date, Seat_Status.
  - Relationship: Each seat availability record is linked to a train.
- **Via_details**: Represents details of stations a train passes through.
  - Attributes: Details_id (PK), Train_code (FK), Via_station_code, Via_station_name, Km_from_origin, Reach_time.
  - Relationship: Each via detail is linked to a train.
- **Cancellation_Records**: Represents records of ticket cancellations.
  - Attributes: Cancellation_id (PK), PNR_no (FK), Cancellation_time, Original_fare, Refund_amount.
  - Relationship: Each cancellation record is linked to a ticket reservation.

1.3 ER Diagram Notation

The notation can be broken down as:

- | = exactly one (mandatory)
- o = zero (optional)
- { = many
- ||–o{ represents a "one-to-many" relationship

For example, in `Zone ||--o{ Station : contains`, this means:

- Each Zone must contain zero or many Stations.
- Each Station must belong to exactly one Zone.

—

2 2. Relational Schema and Normalization Process

2.1 Relational Schema

```
CREATE TABLE Zone (
    Zone_id VARCHAR(10) PRIMARY KEY,
    Zone_name VARCHAR(50),
    Zone_code VARCHAR(10)
);

CREATE TABLE Station (
    Station_id VARCHAR(10) PRIMARY KEY,
    Station_code VARCHAR(10),
    Station_name VARCHAR(100),
    Zone_id VARCHAR(10),
    FOREIGN KEY (Zone_id) REFERENCES Zone(Zone_id) ON DELETE CASCADE
);

CREATE TABLE Train (
```

```sql
    Train_code VARCHAR(10) PRIMARY KEY,
    Train_name VARCHAR(100),
    Start_time TIME,
    End_time TIME,
    Distance INT,
    Frequency VARCHAR(20),
    Waitlist_limit INT DEFAULT 100
);

CREATE TABLE Ticket_Reservation (
    PNR_no VARCHAR(15) PRIMARY KEY,
    Train_code VARCHAR(10),
    From_station VARCHAR(50),
    To_station VARCHAR(50),
    From_Km INT,
    To_Km INT,
    From_date DATE,
    To_date DATE,
    FOREIGN KEY (Train_code) REFERENCES Train(Train_code) ON DELETE CASCADE
);

CREATE TABLE PAX_info (
    SRL_no INT PRIMARY KEY AUTO_INCREMENT,
    PNR_no VARCHAR(15),
    PAX_Name VARCHAR(100),
    PAX_age INT CHECK (PAX_age > 0),
    Category VARCHAR(20),
    PAX_sex ENUM('M', 'F', 'Other'),
    Class_code VARCHAR(15),
    Seat_no VARCHAR(10),
    Fare DECIMAL(10,2),
    Booking_status ENUM('Confirmed', 'RAC', 'Waitlist', 'Cancelled') DEFAULT 'Confirmed',
    Waitlist_position INT NULL,
    Passenger_id VARCHAR(15) UNIQUE,
    FOREIGN KEY (PNR_no) REFERENCES Ticket_Reservation(PNR_no) ON DELETE CASCADE
);

CREATE TABLE Pay_info (
    Payment_id INT PRIMARY KEY AUTO_INCREMENT,
    PNR_no VARCHAR(15),
    SRL_no INT,
    Pay_date DATE,
    Pay_mode ENUM('Credit␣Card', 'Debit␣Card', 'UPI', 'Net␣Banking', 'Cash'),
    Amount DECIMAL(10,2),
    Inst_type ENUM('Online', 'Counter'),
    Inst_amt DECIMAL(10,2),
    FOREIGN KEY (PNR_no) REFERENCES Ticket_Reservation(PNR_no) ON DELETE CASCADE,
    FOREIGN KEY (SRL_no) REFERENCES PAX_info(SRL_no) ON DELETE CASCADE
);

CREATE TABLE Login_credential (
    login_id VARCHAR(50) PRIMARY KEY,
    password VARCHAR(255),
    Passenger_id VARCHAR(15),
    FOREIGN KEY (Passenger_id) REFERENCES PAX_info(Passenger_id) ON DELETE CASCADE
);

CREATE TABLE Train_fare (
    Train_fare_id INT PRIMARY KEY AUTO_INCREMENT,
    Train_code VARCHAR(10),
    Class_id VARCHAR(10),
    From_Km INT,
    To_Km INT,
    From_date DATE,
    To_date DATE,
    Fare DECIMAL(10,2),
    FOREIGN KEY (Train_code) REFERENCES Train(Train_code) ON DELETE CASCADE
);

CREATE TABLE Class (
    Class_id VARCHAR(10) PRIMARY KEY,
    Class_code VARCHAR(10),
    Class_name VARCHAR(50),
    Seat_per_coach INT,
    RAC_limit INT DEFAULT 10
);

CREATE TABLE Seat_availability (
    Details_id INT PRIMARY KEY AUTO_INCREMENT,
    Train_code VARCHAR(10),
    Class_code VARCHAR(10),
    Seat_No INT,
    travel_date DATE NOT NULL,
    Seat_Status ENUM('Available', 'Unavailable', 'Booked', 'RAC'),
    FOREIGN KEY (Train_code) REFERENCES Train(Train_code) ON DELETE CASCADE
);

CREATE TABLE Via_details (
    Details_id INT PRIMARY KEY AUTO_INCREMENT,
    Train_code VARCHAR(10),
    Via_station_code VARCHAR(10),
    Via_station_name VARCHAR(100),
    Km_from_origin INT,
    Reach_time TIME,
    FOREIGN KEY (Train_code) REFERENCES Train(Train_code) ON DELETE CASCADE
);

CREATE TABLE Cancellation_Records (
    Cancellation_id INT PRIMARY KEY AUTO_INCREMENT,
    PNR_no VARCHAR(15),
    Cancellation_time DATETIME,
    Original_fare DECIMAL(10,2),
    Refund_amount DECIMAL(10,2),
    FOREIGN KEY (PNR_no) REFERENCES Ticket_Reservation(PNR_no)
);
```

2.2 Normalization Process

- **1NF (First Normal Form)**: Each table has a primary key, and all attributes contain atomic values.
- **2NF (Second Normal Form)**: All non-key attributes are fully functionally dependent on the primary key.
- **3NF (Third Normal Form)**: No transitive dependencies exist; all attributes are dependent only on the primary key.

—

3 3. SQL Queries, Procedures, Functions, Indices, Triggers

3.1 SQL Queries

- **Q1: PNR status tracking for a given ticket.**

```
SELECT
    tr.PNR_no,
    tr.Train_code,
    t.Train_name,
    tr.From_station,
    tr.To_station,
    tr.From_date,
    tr.To_date,
    pi.SRL_no,
    pi.PAX_Name,
    pi.PAX_age,
    pi.PAX_sex,
    pi.Category,
    pi.Class_code,
    pi.Seat_no,
    pi.Fare,
    pi.Booking_status,
    pi.Waitlist_position,
    tf.Fare AS Fare_from_table,
    sa.Seat_Status,
    cr.Cancellation_time,
    cr.Original_fare,
    cr.Refund_amount
FROM Ticket_Reservation tr
JOIN Train t ON tr.Train_code = t.Train_code
LEFT JOIN PAX_info pi ON tr.PNR_no = pi.PNR_no
LEFT JOIN Seat_availability sa ON sa.Train_code = tr.Train_code
                               AND sa.travel_date = tr.From_date
                               AND sa.Class_code = pi.Class_code
                               AND sa.Seat_No = pi.Seat_no
LEFT JOIN Train_fare tf ON tf.Train_code = tr.Train_code
                        AND tf.From_Km = tr.From_Km
                        AND tf.To_Km = tr.To_Km
                        AND tf.Class_id = pi.Class_code
                        AND tr.From_date BETWEEN tf.From_date AND tf.To_date
LEFT JOIN Cancellation_Records cr ON cr.PNR_no = tr.PNR_no
WHERE tr.PNR_no = 'YOUR_PNR_HERE';
```

- **Q2: Train schedule lookup for a given train.**

```
SELECT
    t.Train_code,
    t.Train_name,
    t.Start_time,
    t.End_time,
    v.Via_station_name,
    v.Km_from_origin,
    v.Reach_time
FROM
    Train t
LEFT JOIN
    Via_details v ON t.Train_code = v.Train_code
WHERE
    t.Train_code = 'TRAIN_CODE'
ORDER BY
    v.Km_from_origin;
```

- **Q3: Available seats query for a specific train, date, and class.**

```
SELECT
```

```
        Train_code, Seat_No, Class_code, Seat_Status
    FROM Seat_availability
    WHERE
        Train_code = '12321'
        AND travel_date = '2023-10-01'
        AND Class_code = '1A'
        AND Seat_Status = 'Available'
    ORDER BY Seat_No;
```

- **Q4: List all passengers traveling on a specific train on a given date.**

```
    SELECT
        tr.PNR_no,
        pi.Passenger_id,
        pi.PAX_Name,
        pi.PAX_age,
        pi.PAX_sex,
        pi.Category,
        pi.Class_code,
        pi.Seat_no,
        pi.Booking_status,
        pi.Waitlist_position
    FROM Ticket_Reservation tr
    JOIN PAX_info pi ON tr.PNR_no = pi.PNR_no
    WHERE
        tr.Train_code = '12321'
        AND tr.From_date = '2023-10-01'
        AND pi.Booking_status='Confirmed'
    ORDER BY pi.Class_code, pi.Seat_no;
```

- **Q5: Retrieve all waitlisted passengers for a particular train.**

```
    SELECT
        p.PAX_Name,
        p.Waitlist_position,
        tr.From_station,
        tr.To_station,
        tr.From_date
    FROM
        PAX_info p
    JOIN
        Ticket_Reservation tr ON p.PNR_no = tr.PNR_no
    WHERE
        tr.Train_code = '12302'
        AND p.Booking_status = 'Waitlist'
    ORDER BY
        p.Waitlist_position;
```

- **Q6: Find total amount that needs to be refunded for cancelling a train.**

```
    -- Step 1: PNR-wise refund breakdown
    SELECT
        cr.PNR_no,
        pi.PAX_Name,
        cr.Original_fare,
        cr.Refund_amount
    FROM Cancellation_Records cr
    JOIN Ticket_Reservation tr ON cr.PNR_no = tr.PNR_no
    JOIN PAX_info pi ON cr.PNR_no = pi.PNR_no
    WHERE tr.Train_code = '12320'
```

```
AND tr.From_date='2023-10-01'
ORDER BY cr.PNR_no;

-- Step 2: Total refund for the train
SELECT
    tr.Train_code,
    SUM(cr.Refund_amount) AS Total_Refund_Amount
FROM Cancellation_Records cr
JOIN Ticket_Reservation tr ON cr.PNR_no = tr.PNR_no
WHERE tr.Train_code = '12320'
AND tr.From_date='2023-10-01'
GROUP BY tr.Train_code;
```

- **Q7: Total revenue generated from ticket bookings over a specified period.**

```
WITH DailyRevenue AS (
    SELECT
        DATE(pi.Pay_date) AS payment_date,
        SUM(pi.Amount) AS daily_revenue,
        COUNT(DISTINCT pi.PNR_no) AS tickets_sold,
        p.Class_code,
        t.Train_name
    FROM
        Pay_info pi
    JOIN
        PAX_info p ON pi.PNR_no = p.PNR_no AND pi.SRL_no = p.SRL_no
    JOIN
        Ticket_Reservation tr ON pi.PNR_no = tr.PNR_no
    JOIN
        Train t ON tr.Train_code = t.Train_code
    WHERE
        pi.Pay_date BETWEEN '2023-06-01' AND '2023-12-31'
    GROUP BY
        DATE(pi.Pay_date), p.Class_code, t.Train_name
)

-- Detailed daily breakdown
SELECT
    payment_date,
    Class_code,
    Train_name,
    daily_revenue,
    tickets_sold,
    (SELECT SUM(daily_revenue) FROM DailyRevenue) AS total_revenue_generated
FROM
    DailyRevenue
UNION ALL

-- Summary row for total
SELECT
    NULL AS payment_date,
    'TOTAL' AS Class_code,
    NULL AS Train_name,
    SUM(daily_revenue) AS daily_revenue,
    SUM(tickets_sold) AS tickets_sold,
    SUM(daily_revenue) AS total_revenue_generated
FROM
    DailyRevenue
ORDER BY
    CASE WHEN payment_date IS NULL THEN 2 ELSE 1 END,
    payment_date,
    daily_revenue DESC;
```

- **Q8: Cancellation records with refund status.**

```
SELECT
    cr.PNR_no,
    cr.Cancellation_time,
    cr.Original_fare,
    cr.Refund_amount,
    CASE
        WHEN cr.Refund_amount = 0 THEN 'No␣Refund'
        WHEN cr.Refund_amount = cr.Original_fare THEN 'Full␣Refund'
        ELSE 'Partial␣Refund'
    END AS Refund_Status
FROM Cancellation_Records cr
ORDER BY cr.Cancellation_time DESC;
```

- **Q9: Find the busiest route based on passenger count.**

```
SELECT
    tr.From_station,
    tr.To_station,
    COUNT(pi.SRL_no) AS Passenger_Count
FROM Ticket_Reservation tr
JOIN PAX_info pi ON tr.PNR_no = pi.PNR_no
GROUP BY tr.From_station, tr.To_station
ORDER BY Passenger_Count DESC
LIMIT 1;
```

- **Q10: Generate an itemized bill for a ticket including all charges.**

```
SELECT
    pi.PAX_Name,
    pi.Class_code,
    pi.Seat_no,
    pi.Fare * 0.85 AS Base_Fare,
    pi.Fare * 0.1 AS Service_Charge,
    pi.Fare * 0.05 AS Convenience_Fee,
    pay.Amount AS Payment_Made,
    cr.Refund_amount AS Refund_Amount,
    (pi.Fare + (pi.Fare * 0.1) + (pi.Fare * 0.05) - COALESCE(cr.Refund_amount, 0)) AS
Total_Amount
FROM PAX_info pi
LEFT JOIN Pay_info pay ON pi.PNR_no = pay.PNR_no
LEFT JOIN Cancellation_Records cr ON pi.PNR_no = cr.PNR_no
WHERE pi.PNR_no = 'PNR62363373';
```

- **Q11: Occupancy rate**

```
SELECT
    sa.Train_code,
    sa.Class_code,
    sa.travel_date,
    COUNT(*) AS Total_Seats,
    SUM(CASE WHEN sa.Seat_Status IN ('Booked', 'RAC') THEN 1 ELSE 0 END)
    AS Booked_Seats,
    ROUND(SUM(CASE WHEN sa.Seat_Status IN ('Booked', 'RAC') THEN 1 ELSE 0 END) *100.0
      / COUNT(*),2)
    AS Occupancy_Percentage
FROM Seat_availability sa
```

```
    GROUP BY sa.Train_code, sa.Class_code, sa.travel_date
    ORDER BY sa.Train_code, sa.travel_date;
```

- **Q12: booking vs cancellation**

```
    SELECT
        tr.Train_code,
        COUNT(DISTINCT pi.SRL_no) AS Booked,
        SUM(CASE WHEN cr.PNR_no IS NOT NULL THEN 1 ELSE 0 END) AS Cancelled
    FROM Ticket_Reservation tr
    JOIN PAX_info pi ON tr.PNR_no = pi.PNR_no
    LEFT JOIN Cancellation_Records cr ON tr.PNR_no = cr.PNR_no
    GROUP BY tr.Train_code;
```

3.2 Stored Procedures and Triggers

Stored procedures, functions, and triggers are essential components of the database management system that enhance functionality and automate processes. Below are the details of the stored procedures, functions, and triggers used in the Railway Ticket Reservation System.

3.2.1 Stored Procedures

1. GenerateCompleteLoginCredentials This stored procedure generates unique login credentials for each passenger in the `PAX_info` table. It ensures that each login ID is unique and generates a secure password for each passenger.

```
DELIMITER //
CREATE PROCEDURE GenerateCompleteLoginCredentials()
BEGIN
    -- Declare all variables first
    DECLARE passenger_id_var VARCHAR(15);
    DECLARE pax_name_var VARCHAR(100);
    DECLARE pax_age_var INT;
    DECLARE first_name_var VARCHAR(100);
    DECLARE last_part_var VARCHAR(100);
    DECLARE login_id_var VARCHAR(200);
    DECLARE password_var VARCHAR(255);
    DECLARE total_count INT DEFAULT 0;
    DECLARE processed_count INT DEFAULT 0;
    DECLARE done INT DEFAULT FALSE;

    -- Declare cursor and handlers after all variables
    DECLARE cur CURSOR FOR SELECT Passenger_id, PAX_Name, PAX_age FROM PAX_info;
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

    -- Get total count for verification
    SELECT COUNT(*) INTO total_count FROM PAX_info;

    OPEN cur;

    read_loop: LOOP
        FETCH cur INTO passenger_id_var, pax_name_var, pax_age_var;
        IF done THEN
            LEAVE read_loop;
        END IF;

        SET processed_count = processed_count + 1;

        -- Extract first name (everything before first space)
        SET first_name_var = SUBSTRING_INDEX(pax_name_var, '␣', 1);
        -- Handle cases where there might not be a last name
        IF LOCATE('␣', pax_name_var) = 0 THEN
            SET last_part_var = '';
        ELSE
            SET last_part_var = SUBSTRING(pax_name_var, LOCATE('␣', pax_name_var) + 1);
        END IF;

        -- Generate login ID: firstname_lastname (lowercase with underscore)
        SET login_id_var = LOWER(
            CONCAT(
                REGEXP_REPLACE(first_name_var, '[^a-zA-Z0-9]', ''),
                '_',
                REGEXP_REPLACE(REPLACE(last_part_var, '␣', '_'), '[^a-zA-Z0-9_]', '')
            )
        );

        -- Ensure login_id is unique by appending passenger_id if needed
        SET @counter = 0;
        WHILE EXISTS (SELECT 1 FROM Login_credential WHERE login_id = login_id_var AND
    (@counter > 0 OR Passenger_id != passenger_id_var)) DO
            SET @counter = @counter + 1;
            SET login_id_var = CONCAT(
                LOWER(
                    CONCAT(
                        REGEXP_REPLACE(first_name_var, '[^a-zA-Z0-9]', ''),
                        '_',
                        REGEXP_REPLACE(REPLACE(last_part_var, '␣', '_'), '[^a-zA-Z0-9_]', '')
                    )
                ),
                @counter
            );
        END WHILE;

        -- Generate password: firstname + age + random characters (total 10-20 chars)
        SET @base = CONCAT(REGEXP_REPLACE(first_name_var, '[^a-zA-Z0-9]', ''), pax_age_var);
        SET @needed_length = 10 + FLOOR(RAND() * 11); -- Random length between 10-20
        SET @random_part = '';

        -- Generate random characters if needed
        IF LENGTH(@base) < @needed_length THEN
            SET @chars =
    'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789!@#$%^&*';
            WHILE LENGTH(@base) + LENGTH(@random_part) < @needed_length DO
                SET @random_part = CONCAT(@random_part,
                    SUBSTRING(@chars, FLOOR(1 + RAND() * LENGTH(@chars)), 1));
            END WHILE;
        END IF;

        SET password_var = CONCAT(@base, @random_part);

        -- Insert into Login_credential table
        INSERT INTO Login_credential (login_id, password, Passenger_id)
        VALUES (login_id_var, SHA2(password_var, 256), passenger_id_var)
        ON DUPLICATE KEY UPDATE
            login_id = VALUES(login_id),
            password = VALUES(password);

    END LOOP;

    CLOSE cur;

    -- Verification output
    SELECT CONCAT('Processed␣', processed_count, '␣of␣', total_count, '␣records') AS
    verification;
END //
DELIMITER ;
```

2. ProcessCancellation This stored procedure manages the cancellation of tickets and calculates refund amounts based on the time of cancellation relative to the journey date.

```
DELIMITER //
CREATE PROCEDURE ProcessCancellation(
    IN p_pnr_no VARCHAR(15),
    IN p_cancellation_time DATETIME,
    OUT p_refund_amount DECIMAL(10,2)
)
BEGIN
    DECLARE v_train_code VARCHAR(10);
    DECLARE v_departure_date DATE;
    DECLARE v_rac_limit INT;
    DECLARE v_passenger_count INT;
    DECLARE v_cancelled_count INT DEFAULT 0;
    DECLARE v_original_fare DECIMAL(10,2);
    DECLARE v_status_message VARCHAR(100);
    DECLARE done INT DEFAULT FALSE;
    DECLARE class_code VARCHAR(15);

    -- Cursor for handling multiple classes
    DECLARE cur CURSOR FOR
        SELECT DISTINCT Class_code FROM PAX_info WHERE PNR_no = p_pnr_no;
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

    -- Get basic booking details
    SELECT
        tr.Train_code,
        tr.From_date,
        c.RAC_limit,
        COUNT(p.Passenger_id),
        SUM(p.Fare),
        t.Start_time
    INTO
        v_train_code,
        v_departure_date,
        v_rac_limit,
        v_passenger_count,
        v_original_fare,
        @start_time
    FROM Ticket_Reservation tr
    JOIN Train t ON tr.Train_code = t.Train_code
    JOIN PAX_info p ON tr.PNR_no = p.PNR_no
    JOIN Class c ON p.Class_code = c.Class_code
    WHERE tr.PNR_no = p_pnr_no
    GROUP BY tr.Train_code, tr.From_date, c.RAC_limit, t.Start_time;

    -- Check if PNR exists
    IF v_train_code IS NULL THEN
        SET v_status_message = 'Invalid␣PNR';
        SET p_refund_amount = 0;
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Invalid␣PNR';
    END IF;

    -- Get refund amount based on cancellation time relative to journey
    SET p_refund_amount = CASE
        WHEN TIMESTAMPDIFF(HOUR, p_cancellation_time, CONCAT(v_departure_date, '␣',
    @start_time)) >= 48 THEN v_original_fare * 0.80
        WHEN TIMESTAMPDIFF(HOUR, p_cancellation_time, CONCAT(v_departure_date, '␣',
    @start_time)) >= 24 THEN v_original_fare * 0.50
        WHEN TIMESTAMPDIFF(HOUR, p_cancellation_time, CONCAT(v_departure_date, '␣',
    @start_time)) >= 12 THEN v_original_fare * 0.25
        ELSE 0
    END;

    -- Mark all tickets as cancelled and free seats
    UPDATE PAX_info p
    SET
        p.Booking_status = 'Cancelled',
        p.Seat_no = NULL,
        p.Waitlist_position = NULL
    WHERE p.PNR_no = p_pnr_no
    AND p.Booking_status != 'Cancelled';

    -- Get count of cancelled tickets
    SELECT ROW_COUNT() INTO v_cancelled_count;

    -- Free up confirmed seats
    UPDATE Seat_availability sa
    JOIN PAX_info p ON sa.Seat_No = SUBSTRING(p.Seat_no, 2)
    SET sa.Seat_Status = 'Available'
    WHERE sa.Train_code = v_train_code
    AND sa.Class_code IN (SELECT Class_code FROM PAX_info WHERE PNR_no = p_pnr_no)
    AND p.PNR_no = p_pnr_no
    AND p.Booking_status = 'Cancelled'
    AND p.Seat_no IS NOT NULL
    AND sa.travel_date = v_departure_date;

    -- Record cancellation (one record per PNR)
    INSERT INTO Cancellation_Records (
        PNR_no,
        Cancellation_time,
        Original_fare,
        Refund_amount
    ) VALUES (
        p_pnr_no,
        p_cancellation_time,
        v_original_fare,
        p_refund_amount
    );

    -- Process upgrades if any confirmed seats were cancelled
    IF EXISTS (
        SELECT 1 FROM PAX_info
        WHERE PNR_no = p_pnr_no
        AND Booking_status = 'Cancelled'
        AND Seat_no IS NOT NULL
    ) THEN
        -- Call upgrade for each affected class
        OPEN cur;
        read_loop: LOOP
            FETCH cur INTO class_code;
            IF done THEN
                LEAVE read_loop;
            END IF;
            CALL ProcessUpgrades(v_train_code, class_code, v_rac_limit);
        END LOOP;
        CLOSE cur;
    END IF;
```

```
    -- Return cancellation summary
    SELECT
        p_pnr_no AS PNR_Number,
        v_cancelled_count AS Tickets_Cancelled,
        v_original_fare AS Original_Fare,
        p_refund_amount AS Refund_Amount,
        CONCAT(v_cancelled_count, '␣ticket(s)␣cancelled.␣Refund␣amount:␣', p_refund_amount) AS
    Status_Message;
END //
DELIMITER ;
```

3. BookTicket This stored procedure handles the entire booking flow, from validating station details to confirming the booking, calculating fares, assigning seats, and processing payments.

```
DELIMITER //

CREATE PROCEDURE BookTicket(
    IN p_train_code VARCHAR(10),
    IN p_from_station VARCHAR(50),
    IN p_to_station VARCHAR(50),
    IN p_from_date DATE,
    IN p_passenger_name VARCHAR(100),
    IN p_passenger_age INT,
    IN p_passenger_gender ENUM('M', 'F', 'Other'),
    IN p_passenger_category VARCHAR(20),
    IN p_class_code VARCHAR(15),
    IN p_payment_mode ENUM('Credit␣Card', 'Debit␣Card', 'UPI', 'Net␣Banking', 'Cash'),
    IN p_inst_type ENUM('Online', 'Counter'),
    OUT p_status_message VARCHAR(100),
    OUT p_booking_status ENUM('Confirmed', 'RAC', 'Waitlist', 'Cancelled'),
    OUT p_waitlist_position INT
)
BEGIN
    DECLARE v_pnr_no VARCHAR(15);
    DECLARE v_from_km INT;
    DECLARE v_to_km INT;
    DECLARE v_total_fare DECIMAL(10,2) DEFAULT 0;
    DECLARE v_seat_no INT;
    DECLARE v_passenger_id VARCHAR(15);
    DECLARE v_fare DECIMAL(10,2);
    DECLARE v_payment_id INT;
    DECLARE v_srl_no INT;
    DECLARE v_seat_prefix CHAR(1);
    DECLARE v_available_seats INT DEFAULT 0;
    DECLARE v_rac_seats INT DEFAULT 0;
    DECLARE v_current_waitlist INT DEFAULT 0;
    DECLARE v_booking_status ENUM('Confirmed', 'RAC', 'Waitlist','Cancelled') DEFAULT
    'Confirmed';
    DECLARE v_waitlist_position INT DEFAULT 0;
    DECLARE v_km_rate DECIMAL(10,2);
    DECLARE v_distance INT;
    DECLARE v_seat_exists INT DEFAULT 0;
    DECLARE v_seats_per_coach INT;

    -- Configuration parameters
    DECLARE v_rac_limit INT DEFAULT 10; -- Number of RAC seats per train/class
    DECLARE v_waitlist_limit INT DEFAULT 100; -- Maximum waitlist positions

    -- Get number of seats per coach from Class table
    SELECT Seat_per_coach INTO v_seats_per_coach
```

```
FROM Class
WHERE Class_code = p_class_code;

-- Validate from and to stations
IF p_from_station = p_to_station THEN
    SET p_status_message = 'From␣and␣To␣stations␣cannot␣be␣the␣same';
    SET p_booking_status = 'Cancelled';
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'From␣and␣To␣stations␣cannot␣be␣identical';
END IF;

-- Generate PNR number (format: PNR + random 8 digits)
SET v_pnr_no = CONCAT('PNR', LPAD(FLOOR(RAND() * 100000000), 8, '0'));

-- Get distance information with proper validation
SELECT vd1.Km_from_origin, vd2.Km_from_origin
INTO v_from_km, v_to_km
FROM Via_details vd1
JOIN Via_details vd2 ON vd1.Train_code = vd2.Train_code
WHERE vd1.Train_code = p_train_code
AND vd1.Via_station_code = p_from_station
AND vd2.Via_station_code = p_to_station
AND vd2.Km_from_origin > vd1.Km_from_origin
LIMIT 1;

-- Validate distance information
IF v_from_km IS NULL OR v_to_km IS NULL THEN
    SET p_status_message = 'Invalid␣station␣codes␣or␣route␣information';
    SET p_booking_status = 'Cancelled';
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Could␣not␣determine␣route␣distance';
END IF;

-- Calculate distance
SET v_distance = v_to_km - v_from_km;

-- Validate calculated distance
IF v_distance <= 0 THEN
    SET p_status_message = 'Invalid␣route␣distance␣calculated';
    SET p_booking_status = 'Cancelled';
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Invalid␣route␣distance';
END IF;

-- IMPROVED FARE CALCULATION
-- Try exact match first
SELECT Fare INTO v_fare
FROM Train_fare
WHERE Train_code = p_train_code
AND Class_id = (SELECT Class_id FROM Class WHERE Class_code = p_class_code)
AND From_Km <= v_from_km
AND To_Km >= v_to_km
ORDER BY (To_Km - From_Km) ASC  -- Prefer more specific ranges
LIMIT 1;

-- If no exact match, find closest fare
IF v_fare IS NULL THEN
    SELECT Fare INTO v_fare
    FROM Train_fare
    WHERE Train_code = p_train_code
    AND Class_id = (SELECT Class_id FROM Class WHERE Class_code = p_class_code)
    ORDER BY ABS(From_Km - v_from_km) + ABS(To_Km - v_to_km) ASC
    LIMIT 1;
END IF;

-- Final fallback if no fare could be determined
```

```
    IF v_fare IS NULL THEN
        SET v_fare = 1000.00; -- Default fare
        SET p_status_message = CONCAT(IFNULL(p_status_message, ’’),
                                   ’␣No␣fare␣found.␣Used␣default␣fare.’);
    END IF;

    -- Validate fare
    IF v_fare <= 0 THEN
        SET v_fare = 1000.00;
        SET p_status_message = CONCAT(IFNULL(p_status_message, ’’),
                                   ’␣Invalid␣fare.␣Used␣default.’);
    END IF;

    -- Apply concessions
    SET v_fare = v_fare *
        CASE p_passenger_category
            WHEN ’student’ THEN 0.75
            WHEN ’senior’ THEN 0.60
            WHEN ’disabled’ THEN 0.50
            WHEN ’child’ THEN 0.50
            ELSE 1.00
        END;

    -- Create ticket reservation with validated distance values
    INSERT INTO Ticket_Reservation (
        PNR_no, Train_code, From_station, To_station,
        From_Km, To_Km, From_date, To_date
    ) VALUES (
        v_pnr_no, p_train_code, p_from_station, p_to_station,
        v_from_km, v_to_km, p_from_date, p_from_date
    );

    -- Get seat prefix based on class
    SET v_seat_prefix =
        CASE p_class_code
            WHEN ’1A’ THEN ’A’
            WHEN ’2A’ THEN ’B’
            WHEN ’3A’ THEN ’C’
            WHEN ’SL’ THEN ’S’
            WHEN ’CC’ THEN ’D’
            WHEN ’EC’ THEN ’E’
            WHEN ’2S’ THEN ’F’
            WHEN ’FC’ THEN ’G’
            ELSE ’H’
        END;

    -- Check if seats exist for this train/class/date
    SELECT COUNT(*) INTO v_seat_exists
    FROM Seat_availability
    WHERE Train_code = p_train_code
    AND Class_code = p_class_code
    AND travel_date = p_from_date;

    -- If no seats exist, create them
    IF v_seat_exists = 0 THEN
        -- Create a temporary table to generate seat numbers
        CREATE TEMPORARY TABLE IF NOT EXISTS temp_seat_numbers (seat_num INT);

        -- Insert seat numbers from 1 to v_seats_per_coach
        SET @counter = 1;
        WHILE @counter <= v_seats_per_coach DO
            INSERT INTO temp_seat_numbers VALUES (@counter);
            SET @counter = @counter + 1;
        END WHILE;

        -- Insert new seats for the coach
        INSERT INTO Seat_availability (Train_code, Class_code, Seat_No, Seat_Status,
    travel_date)
        SELECT p_train_code, p_class_code,
               seat_num as Seat_No,
               'Available' as Seat_Status,
               p_from_date as travel_date
        FROM temp_seat_numbers;

        -- Drop the temporary table
        DROP TEMPORARY TABLE IF EXISTS temp_seat_numbers;

        -- Set available seats to the number of seats just created
        SET v_available_seats = v_seats_per_coach;
    ELSE
        -- Check available seats
        SELECT COUNT(*) INTO v_available_seats
        FROM Seat_availability
        WHERE Train_code = p_train_code
        AND Class_code = p_class_code
        AND Seat_Status = 'Available'
        AND travel_date = p_from_date;
    END IF;

    -- Check current RAC count
    SELECT COUNT(*) INTO v_rac_seats
    FROM PAX_info p
    JOIN Ticket_Reservation t ON p.PNR_no = t.PNR_no
    WHERE t.Train_code = p_train_code
    AND p.Class_code = p_class_code
    AND p.Booking_status = 'RAC';

    -- Check current waitlist count
    SELECT COUNT(*) INTO v_current_waitlist
    FROM PAX_info p
    JOIN Ticket_Reservation t ON p.PNR_no = t.PNR_no
    WHERE t.Train_code = p_train_code
    AND p.Class_code = p_class_code
    AND p.Booking_status = 'Waitlist';

    -- Determine booking status
    IF v_available_seats > 0 THEN
        -- Assign available seat
        SELECT Seat_No INTO v_seat_no
        FROM Seat_availability
        WHERE Train_code = p_train_code
        AND Class_code = p_class_code
        AND Seat_Status = 'Available'
        AND travel_date = p_from_date
        LIMIT 1;

        SET v_booking_status = 'Confirmed';
    ELSEIF v_rac_seats < v_rac_limit THEN
        -- Assign RAC status
        SET v_seat_no = NULL;
        SET v_booking_status = 'RAC';
    ELSEIF v_current_waitlist < v_waitlist_limit THEN
        -- Assign waitlist status
        SET v_seat_no = NULL;
        SET v_booking_status = 'Waitlist';
        SET v_waitlist_position = v_current_waitlist + 1;
    ELSE
        -- No more bookings accepted
        SET p_status_message = 'No␣seats␣available.␣Waitlist␣is␣full.';
        SET p_booking_status = 'Waitlist';
        SET p_waitlist_position = 0;
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'No␣seats␣available.␣Waitlist␣is␣full.';
    END IF;

    -- Generate passenger ID
    SET v_passenger_id = CONCAT(v_pnr_no, '_P01');

    -- Add passenger with booking status
    INSERT INTO PAX_info (
        PNR_no, PAX_Name, PAX_age, Category, PAX_sex,
        Class_code, Seat_no, Fare, Passenger_id,
        Booking_status, Waitlist_position
    ) VALUES (
        v_pnr_no,
        p_passenger_name,
        p_passenger_age,
        p_passenger_category,
        p_passenger_gender,
        p_class_code,
        CASE WHEN v_seat_no IS NOT NULL THEN CONCAT(v_seat_prefix, v_seat_no) ELSE NULL END,
        v_fare,
        v_passenger_id,
        v_booking_status,
        CASE WHEN v_booking_status = 'Waitlist' THEN v_waitlist_position ELSE NULL END
    );

    -- Get the serial number of the passenger
    SET v_srl_no = LAST_INSERT_ID();

    -- Update seat status if confirmed
    IF v_booking_status = 'Confirmed' THEN
        UPDATE Seat_availability
        SET Seat_Status = 'Booked'
        WHERE Train_code = p_train_code
        AND Class_code = p_class_code
        AND Seat_No = v_seat_no
        AND travel_date = p_from_date;
    END IF;

    -- Process payment
    INSERT INTO Pay_info (
        PNR_no, SRL_no, Pay_date, Pay_mode,
        Amount, Inst_type, Inst_amt
    ) VALUES (
        v_pnr_no,
        v_srl_no,
        CURDATE(),
        p_payment_mode,
        v_fare,
        p_inst_type,
        CASE WHEN p_inst_type = 'Counter' THEN ROUND(10 + RAND() * 20, 2) ELSE 0 END
    );

    -- Set payment ID
    SET v_payment_id = LAST_INSERT_ID();

    -- Set output parameters
    SET p_status_message = CONCAT('Booking␣', v_booking_status,
                                 CASE WHEN v_booking_status = 'Waitlist'
                                      THEN CONCAT('.␣Position:␣', v_waitlist_position)
                                      ELSE '' END);
    SET p_booking_status = v_booking_status;
    SET p_waitlist_position = CASE WHEN v_booking_status = 'Waitlist' THEN v_waitlist_position
    ELSE 0 END;

    -- Return booking information
    SELECT
        v_pnr_no AS PNR_Number,
        p_train_code AS Train_Code,
        (SELECT Train_name FROM Train WHERE Train_code = p_train_code) AS Train_Name,
        p_from_station AS From_Station,
        p_to_station AS To_Station,
        p_from_date AS Journey_Date,
        v_fare AS Total_Fare,
        1 AS Passenger_Count,
        v_payment_id AS Payment_ID,
        v_passenger_id AS Passenger_ID,
        CASE WHEN v_seat_no IS NOT NULL THEN CONCAT(v_seat_prefix, v_seat_no) ELSE NULL END AS
    Seat_Number,
        v_booking_status AS Booking_Status,
        CASE WHEN v_booking_status = 'Waitlist' THEN v_waitlist_position ELSE NULL END AS
    Waitlist_Position,
        p_status_message AS Status_Message;
END //

DELIMITER ;
```

3.2.2 Triggers

1. insert_seat_availability_from_pax This trigger manages seat availability in the Seat_availability table whenever a new passenger is added to the PAX_info table. It ensures that the correct number of seats are available for each train and class combination.

```


DELIMITER //
CREATE TRIGGER insert_seat_availability_from_pax
BEFORE INSERT ON PAX_info
FOR EACH ROW
BEGIN
    DECLARE seat_count INT;
    DECLARE i INT DEFAULT 1;
    DECLARE existing_seats INT;
    DECLARE journey_date DATE;
    DECLARE v_train_code VARCHAR(10);

    -- Get journey date and train code from Ticket_Reservation
    SELECT From_date, Train_code INTO journey_date, v_train_code
    FROM Ticket_Reservation
    WHERE PNR_no = NEW.PNR_no;

    -- Get seat count from Class table
    SELECT Seat_per_coach INTO seat_count
    FROM Class
    WHERE Class_code = NEW.Class_code;

    -- Check if availability already exists
    SELECT COUNT(*) INTO existing_seats
    FROM Seat_availability
    WHERE Train_code = v_train_code
      AND Class_code = NEW.Class_code
      AND travel_date = journey_date;

    -- If no seats exist for this train-class-date combination
    IF existing_seats = 0 THEN
        -- Delete any existing seats (in case they were created with wrong count)
        DELETE FROM Seat_availability
        WHERE Train_code = v_train_code
        AND Class_code = NEW.Class_code
        AND travel_date = journey_date;

        -- Insert correct number of seats
        WHILE i <= seat_count DO
            INSERT INTO Seat_availability (
                Train_code, Class_code, Seat_No, travel_date, Seat_Status
            ) VALUES (
                v_train_code,
                NEW.Class_code,
                i,
                journey_date,
                'Available'
            );
            SET i = i + 1;
        END WHILE;

        UPDATE Seat_availability SET Seat_Status='Booked' WHERE Train_Code=v_train_code
                                                    AND Class_code=NEW.Class_code
                                                    AND Seat_No=1
                                                    AND travel_date=journey_date;
    END IF;

    -- Update seat status if booking is confirmed
    IF NEW.Booking_status = 'Confirmed' AND NEW.Seat_no IS NOT NULL THEN
        UPDATE Seat_availability
        SET Seat_Status = 'Booked'
        WHERE Train_Code = v_train_code
        AND Class_code = NEW.Class_code
        AND Seat_No = CAST(SUBSTRING(NEW.Seat_no, 2) AS UNSIGNED)
        AND travel_date = journey_date;
    END IF;
END //
DELIMITER ;
```

Trigger Functionality - The trigger is executed before a new record is inserted into the `PAX_info` table. - It checks if there are existing seats for the specified train and class on the journey date. - If no seats exist, it creates the required number of seats in the `Seat_availability` table. - It updates the seat status to 'Booked' if the booking is confirmed.

3.3 Conclusion

The Railway Ticket Reservation System simulates real-world railway operations by integrating a relational database with stored procedures, functions, and triggers. From fare calculation to passenger upgrades and cancellations, the system emphasizes:

1. Data integrity through relational constraints.
2. Business rule automation through triggers and procedures.
3. Real-time seat status updates ensuring accurate bookings.

Through this project, the design demonstrates the power of SQL Server-based automation and logical abstraction, allowing complex scenarios like RAC handling, seat allocation, and dynamic refund processing to be managed with minimal manual intervention.

3.4 Future Enhancements

As part of future improvements, the following features are proposed:

- Real-time seat availability and status sync for mobile and web interfaces.
- Payment gateway integration for automated payment processing.
- A dedicated administrative panel for schedule and train management.
- Enhanced data analytics for predictive fare suggestions.

3.5 Acknowledgments

This project was made possible through the collective guidance, technical insight, and academic resources provided by:

- Academic Mentors and Instructors
- Open-source community contributions
- Team members: **Pragya Mahajan**, **Mihika**, and **Priyanshi Agrawal**

Their combined efforts helped turn this theoretical model into a working database system.

3.6 Contact

For queries or feedback, please feel free to reach out to the project contributors:

- Pragya Mahajan — pragya_2302cs05@iitp.ac.in
- Mihika — mihika_2301cs31@iitp.ac.in
- Priyanshi Agrawal — priyanshi_2301cs90@iitp.ac.in

GitHub Repository: Project Link

Thank you for exploring the Railway Ticket Reservation System!