CS2202 Mini Project: Indian Railway Ticket Reservation System

GROUP MEMBERS –

1)ANISH RAGHASHETTY – 2301CS05

2)AAYUSH SHETH – 2301CS02

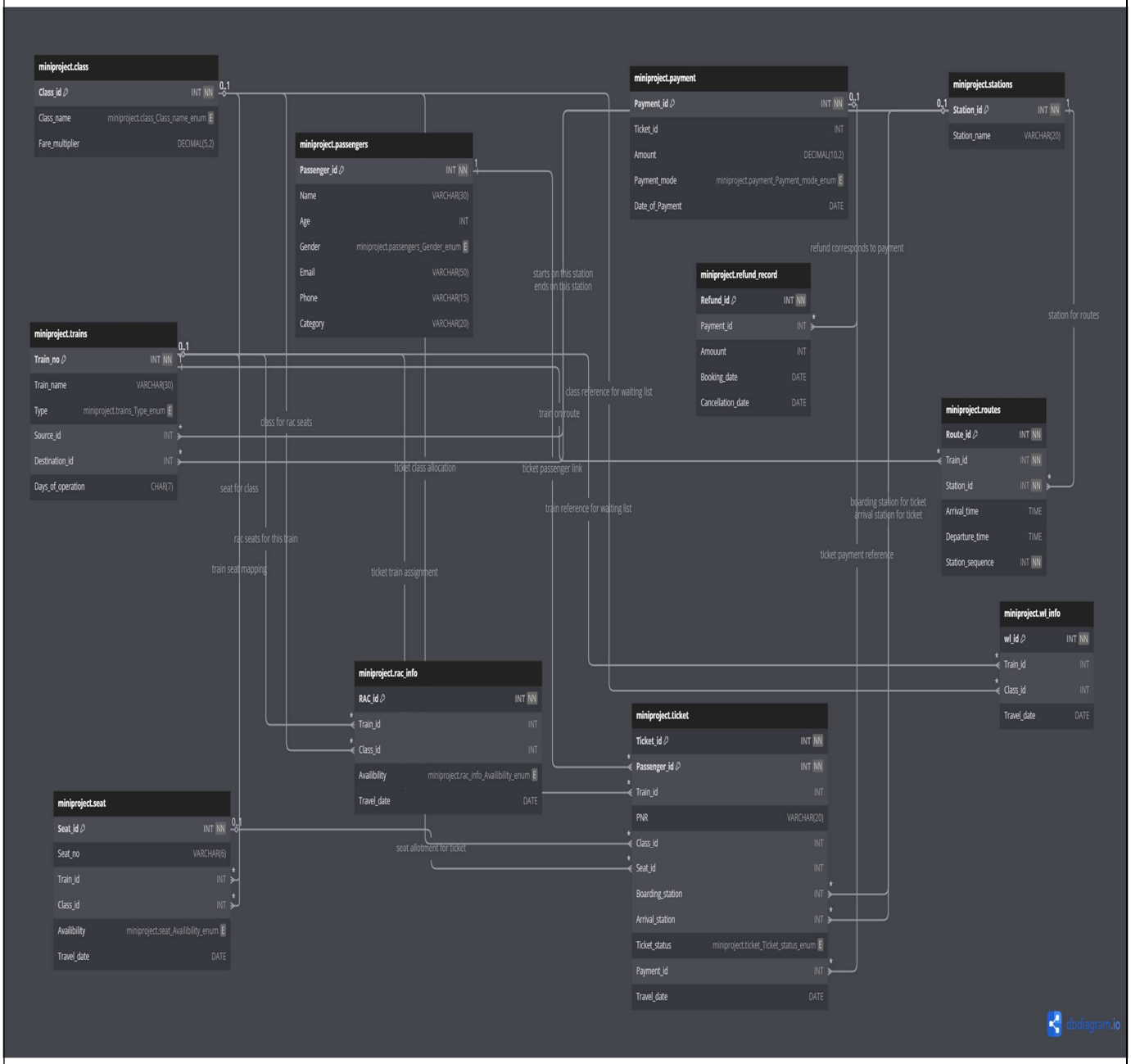
3)ARYAN PHAD – 2301CS09

Project Overview

This project is a relational database implementation of an Indian Railway Ticket Reservation System. It simulates key functionalities of a railway reservation system, including ticket booking (for multiple classes), seat availability tracking, PNR status inquiry, RAC (Reservation Against Cancellation) and waiting list management, cancellation handling, and refund processing.

The project is implemented entirely in MySQL and is developed/tested using MySQL Workbench.

ER DIAGRAM WITH DESCRIPTIONS



RELATIONAL SCHEMA AND NORMALIZATION:

The relational schema for the given project involves several interconnected tables, each representing different entities and their attributes in the context of train ticket booking. Below is the detailed relational schema for the project:

1. Stations Table

- **Station_id** (Primary Key): A unique identifier for each station. It helps to identify each station in the database.
- **Station_name**: The name of the station (e.g., "New York", "Mumbai Central"). This is used to identify and display the station name.

2. Class Table

- **Class_id** (Primary Key): A unique identifier for each class type (e.g., Sleeper, 3AC). This helps to identify different classes available on the trains.
- **Class_name**: The name of the class, such as 'General', 'Sleeper', '3AC', '2AC', or '1AC'.
- **Fare_multiplier**: The multiplier for calculating the fare of a seat in this class relative to the base fare. For example, '2AC' has a higher fare multiplier than 'Sleeper'.

3. Trains Table

- **Train_no** (Primary Key): A unique identifier for each train (e.g., train number).
- **Train_name**: The name of the train (e.g., "Rajdhani Express").
- **Type**: Specifies the type of train, such as 'Passenger', 'Express', or 'Premium', indicating its services and facilities.
- **Source_id** (Foreign Key): The Station_id of the source station (departure station) for the train.
- **Destination_id** (Foreign Key): The Station_id of the destination station (arrival station) for the train.
- **Days_of_operation**: A string representing the days of operation, where each character (Y/N) corresponds to a day of the week, starting from Sunday.

4. Passengers Table

- **Passenger_id** (Primary Key): A unique identifier for each passenger.
- **Name**: The name of the passenger.
- **Age**: The age of the passenger.
- **Gender**: The gender of the passenger. It can be 'Male', 'Female', or 'Other'.
- **Email**: The email address of the passenger. It is unique to each passenger.

- **Phone**: The phone number of the passenger.
- **Category**: The category of the passenger, which may be used for discounts (e.g., 'Senior Citizen', 'Student').

5. Payment Table

- **Payment_id** (Primary Key): A unique identifier for each payment transaction.
- **Ticket_id** (Foreign Key): A reference to the Ticket_id in the ticket table, representing the ticket that is being paid for.
- **Amount**: The total amount paid for the ticket.
- **Payment_mode**: The mode of payment used by the passenger, such as 'Offline', 'Credit Card', 'Debit Card', 'UPI', or 'Net Banking'.
- **Date_of_Payment**: The date when the payment was made.

6. Ticket Table

- **Ticket_id** (Primary Key): A unique identifier for each ticket.
- **Passenger_id** (Primary Key): A reference to the passenger who holds the ticket. The combination of Ticket_id and Passenger_id ensures that a passenger can only have one ticket entry per trip.
- **Train_id** (Foreign Key): A reference to the Train_no in the trains table, indicating which train the ticket is for.
- **PNR**: The Passenger Name Record (PNR) is a unique number assigned to each ticket for identification.
- **Class_id** (Foreign Key): A reference to the class in which the passenger is traveling (e.g., '3AC').
- **Seat_id** (Foreign Key): A reference to the seat assigned to the passenger, helping to identify which seat is booked.
- **Boarding_station** (Foreign Key): A reference to the Station_id of the boarding station (from where the passenger boards the train).
- **Arrival_station** (Foreign Key): A reference to the Station_id of the arrival station (where the passenger's journey ends).
- **Ticket_status**: The status of the ticket. It could be 'Confirmed', 'Cancelled', 'RAC' (Reservation Against Cancellation), or 'Waiting'.
- **Payment_id** (Foreign Key): A reference to the payment made for this ticket.
- **Travel_date**: The date of the train journey.

7. Routes Table

- **Route_id** (Primary Key): A unique identifier for each route.
- **Train_id** (Foreign Key): A reference to the Train_no in the trains table, indicating which train follows this route.
- **Station_id** (Foreign Key): A reference to the Station_id in the stations table, identifying a station along the route.
- **Arrival_time**: The time at which the train arrives at a particular station.
- **Departure_time**: The time at which the train departs from a particular station.
- **Station_sequence**: The sequence number indicating the order in which the stations are visited on the route (e.g., Station 1 is the source station, Station 2 is the next, etc.).

8. RAC Info Table

- **RAC_id** (Primary Key): A unique identifier for each RAC entry.
- **Train_id** (Foreign Key): A reference to the Train_no in the trains table, indicating which train this RAC entry applies to.
- **Class_id** (Foreign Key): A reference to the class type for the RAC seat.
- **Availability**: The availability status of the RAC seat, which can either be 'Available' or 'Booked'.
- **Travel_date**: The date for which the RAC seat is reserved.

9. Seat Table

- **Seat_id** (Primary Key): A unique identifier for each seat in a train.
- **Seat_no**: The seat number (e.g., 'A1', 'B2', etc.).
- **Train_id** (Foreign Key): A reference to the Train_no in the trains table, indicating which train the seat belongs to.
- **Class_id** (Foreign Key): A reference to the Class_id in the class table, specifying which class the seat belongs to.
- **Availability**: Indicates whether the seat is 'Booked' or 'Available'.
- **Travel_date**: The date for which the seat is reserved.

10. Waiting List (wl_info) Table

- **wl_id** (Primary Key): A unique identifier for each waiting list entry.
- **Train_id** (Foreign Key): A reference to the Train_no in the trains table, indicating the train for which a waiting list exists.
- **Class_id** (Foreign Key): A reference to the Class_id in the class table, specifying the class for which the waiting list is maintained.
- **Travel_date**: The date for which the waiting list is created.

11. Refund Record Table

- **Refund_id** (Primary Key): A unique identifier for each refund entry.
- **Payment_id** (Foreign Key): A reference to the Payment_id in the payment table, indicating the payment that is being refunded.
- **Amount**: The amount refunded to the passenger.
- **Booking_date**: The date when the original booking was made.
- **Cancellation_date**: The date when the ticket was cancelled.

Normalization Process

Normalization is a systematic approach used in relational database design to eliminate data redundancy, ensure logical data storage, and avoid anomalies during data operations such as insertion, update, and deletion. The schema for the Railway Reservation System project has been carefully structured following normalization principles to enhance data integrity, reduce duplication, and optimize query performance. The normalization process applied to this schema is described below:

---

1. First Normal Form (1NF)

- **Definition:**
  A relation is in **1NF** if all the attributes contain atomic (indivisible) values, and there are no repeating groups or arrays within a table. Each cell should store a single value, and each record should be unique.
- **Application:**
  In the provided schema:
  - Every table has uniquely identifiable primary keys.
  - All attributes contain atomic values.
  - For example, in the **trains** table, days_of_operation is stored as a single string (like YNNYNYN), where each character represents a specific day of operation — an atomic value in this context.
  - In the **passengers** table, each attribute like name, age, gender, email, phone, and category contains a single indivisible value per record.
  - Similarly, the **seat** table contains atomic values like seat_no, availability, class_id, and travel_date, with no multi-valued or repeating groups.

---

2. Second Normal Form (2NF)

- **Definition:**
  A relation is in **2NF** if it is already in 1NF and every non-prime attribute is fully

functionally dependent on the entire primary key — meaning there are no partial dependencies on a part of a composite primary key.

- **Application:**
  In this schema:
  - Most tables use a **single-column primary key** (like ticket_id in the **ticket** table, passenger_id in the **passengers** table, seat_id in **seat** table, etc.), so by structure, no partial dependencies can exist.
  - For example, in the **ticket** table, attributes such as pnr, train_id, class_id, seat_id, boarding_station, arrival_station, ticket_status, payment_id, and travel_date all depend entirely on the ticket_id.
  - In the **class** table, class_name and fare_multiplier are fully functionally dependent on class_id.
  - The **routes** table uses route_id as the primary key, and attributes like train_id, station_id, arrival_time, departure_time, and station_sequence depend entirely on route_id.
  - No partial dependencies exist in any of the tables, satisfying 2NF.

---

3. Third Normal Form (3NF)

- **Definition:**
  A relation is in **3NF** if it is in 2NF and no transitive dependencies exist — meaning non-prime attributes are not dependent on other non-prime attributes.
- **Application:**
  In the schema:
  - In the **ticket** table, there are no transitive dependencies. train_id references the **trains** table, and class_id references the **class** table directly. No attribute within the table is indirectly dependent through another non-prime attribute.
  - In the **seat** table, attributes like seat_no, train_id, class_id, availability, and travel_date are directly dependent on seat_id.
  - In the **payment** table, amount, payment_mode, and date_of_payment are directly dependent on payment_id without relying on other non-key attributes.
  - The **refund_record** table stores refund_id as the primary key, and payment_id, amount, booking_date, and cancellation_date are directly dependent on it without transitive dependency.
  - In the **rac_info** and **wl_info** tables, attributes such as train_id, class_id, availability (for RAC), and travel_date are directly related to their respective primary keys (rac_id, wl_id) and don't depend on other non-prime attributes.
  - The **routes** table ensures no transitive dependencies by relating train_id, station_id, arrival_time, departure_time, and station_sequence directly to route_id.

Sample data Summary

1. Class Table

Class_id	Class_name	Fare_multiplier
1	General	100.00
2	Sleeper	150.75
3	3AC	250.05
4	2AC	350.02
5	1AC	500.09

count(*)
5

2. Passengers Table

Passenger_id	Name	Age	Gender	Email	Phone	Category
1001	Rajesh Kumar	35	Male	rajesh.kumar@gmail....	9876543210	
1002	Priya Sharma	28	Female	priya.sharma@gmail.c...	9876543211	
1003	Amit Singh	42	Male	amit.singh@gmail.com	9876543212	
1004	Neha Patel	19	Female	neha.patel@gmail.com	9876543213	Student

count(*)
32

3. Payment Table (will get populated when you call the book_tickets procedure)

Payment_id	Ticket_id	Amount	Payment_mode	Date_of_Payment
1	1	7080.00	Credit Card	2025-04-16
2	2	7080.00	Credit Card	2025-04-16

count(*)
4

4. RAC_info table

RAC_id	Train_id	Class_id	Availibility	Travel_date
1	12001	1	Available	2025-04-13
2	12001	1	Available	2025-04-13
3	12001	1	Available	2025-04-13
4	12001	1	Available	2025-04-13
5	12001	1	Available	2025-04-13

count(*)
3025

5. Refund_record Table

(will get populated when you call the cancel_ticket procedure)

Refund_id	Payment_id	Amount	Booking_date	Cancellation_date
7609870	1	1170	2025-04-16	2025-04-16

6. Routes Table

Route_id	Train_id	Station_id	Arrival_time	Departure_time	Station_sequence
101	12001	1	NULL	06:00:00	1
102	12001	12	08:30:00	08:35:00	2
103	12001	8	11:30:00	NULL	3

count(*)
56

7. Seat Table

Seat_id	Seat_no	Train_id	Class_id	Availibility	Travel_date
1	GEN-01	12001	1	Available	2025-04-13
2	GEN-02	12001	1	Available	2025-04-13
3	GEN-03	12001	1	Available	2025-04-13

count(*)
3025

8. Stations Table

Station_id	Station_name
1	Delhi
2	Mumbai Central
3	Chennai Central
4	Kolkata

count(*)
22

9. Ticket Table

(will get populated when you call the book_tickets procedure)

Ticket_id	Passenger_id	Train_id	PNR	Class_id	Seat_id	Boarding_station	Arrival_station	Ticket_status	Payment_id	Travel_date
1	1001	12001	PNR952016	4	416	1	8	Confirmed	1	2025-04-14
1	1002	12001	PNR952016	4	417	1	8	Confirmed	1	2025-04-14
1	1003	12001	PNR952016	4	418	1	8	Confirmed	1	2025-04-14

10.Trains Table

Train_no	Train_name	Type	Source_id	Destination_id	Days_of_operation
12001	Shatabdi Express	Premium	1	8	YYNYNYN
12002	Rajdhani Express	Premium	1	2	YYYYYYY
12003	Duronto Express	Premium	3	4	YYNYYNN
12004	Garib Rath	Express	5	6	NYYNYYY

count(*)
20

LIST OF ALL THE POTENTIAL QUERIES

Query1 ---- PNR status tracking for a given ticket :

```
DELIMITER //
CREATE PROCEDURE CheckPNRStatus(
  IN p_pnr VARCHAR(10)
)
BEGIN
  SELECT t.PNR, p.Name, p.Age, p.Gender, tr.Train_name,
      s1.Station_name AS Boarding_station,
      s2.Station_name AS Arrival_station,
      c.Class_name,
      COALESCE(se.Seat_no, 'Not Assigned') AS Seat_no,
      t.Ticket_status,
      t.Travel_date,
      CASE
        WHEN t.Ticket_status = 'Waitlisted' THEN
          (SELECT COUNT(*) + 1 FROM ticket
           WHERE Train_id = t.Train_id AND Class_id = t.Class_id
           AND Ticket_status = 'Waitlisted' AND Travel_date = t.Travel_date
           AND Ticket_id < t.Ticket_id)
        ELSE NULL
      END AS Waitlist_Position
  FROM ticket t
  JOIN passengers p ON t.Passenger_id = p.Passenger_id
  JOIN trains tr ON t.Train_id = tr.Train_no
  JOIN stations s1 ON t.Boarding_station = s1.Station_id
  JOIN stations s2 ON t.Arrival_station = s2.Station_id
  JOIN class c ON t.Class_id = c.Class_id
  LEFT JOIN seat se ON t.Seat_id = se.Seat_id
  WHERE t.PNR = p_pnr
  ORDER BY p.Name;
END //
DELIMITER ;
```

EXAMPLE USE :

```
call CheckPNRStatus('PNR792699');
```

PNR	Name	Age	Gender	Train_name	Boarding_station	Arrival_station	Class_name	Seat_no	Ticket_status	Travel_date
PNR792699	Neha Patel	19	Female	Shatabdi Express	Delhi	Jaipur	2AC	2AC-03	Confirmed	2025-04-14
PNR792699	Priya Sharma	28	Female	Shatabdi Express	Delhi	Jaipur	2AC	2AC-02	Confirmed	2025-04-14
PNR792699	Rajesh Kumar	35	Male	Shatabdi Express	Delhi	Jaipur	2AC	2AC-01	Confirmed	2025-04-14

Query 2 ---- Train schedule lookup for a given train.

```
CREATE PROCEDURE GetTrainSchedule(
  IN p_train_id INT
)
BEGIN
  -- Get basic train information
  SELECT t.Train_no, t.Train_name, t.Type,
      CASE
        WHEN SUBSTRING(t.Days_of_operation, 1, 1) = 'Y' THEN 'Mon, ' ELSE '' END +
        CASE WHEN SUBSTRING(t.Days_of_operation, 2, 1) = 'Y' THEN 'Tue, ' ELSE '' END +
        CASE WHEN SUBSTRING(t.Days_of_operation, 3, 1) = 'Y' THEN 'Wed, ' ELSE '' END +
        CASE WHEN SUBSTRING(t.Days_of_operation, 4, 1) = 'Y' THEN 'Thu, ' ELSE '' END +
        CASE WHEN SUBSTRING(t.Days_of_operation, 5, 1) = 'Y' THEN 'Fri, ' ELSE '' END +
        CASE WHEN SUBSTRING(t.Days_of_operation, 6, 1) = 'Y' THEN 'Sat, ' ELSE '' END +
        CASE WHEN SUBSTRING(t.Days_of_operation, 7, 1) = 'Y' THEN 'Sun' ELSE '' END AS
Running_Days
  FROM trains t
  WHERE t.Train_no = p_train_id;

  -- Get detailed schedule
  SELECT r.Station_sequence AS Stop_Number,
      s.Station_name,
      r.Arrival_time,
      r.Departure_time,
      CASE
        WHEN r.Arrival_time IS NULL THEN 'Source Station'
        WHEN r.Departure_time IS NULL THEN 'Destination Station'
        ELSE 'Intermediate Station'
      END AS Station_Type,
      TIMEDIFF(r.Departure_time, r.Arrival_time) AS Halt_Time
  FROM routes r
  JOIN stations s ON r.Station_id = s.Station_id
  WHERE r.Train_id = p_train_id
  ORDER BY r.Station_sequence;
END
```

EXAMPLE USE :

```
call GetTrainSchedule(12001);
```

Stop_Number	Station_name	Arrival_time	Departure_time	Station_Type	Halt_Time
1	Delhi	NULL	06:00:00	Source Station	NULL
2	Chandigarh	08:30:00	08:35:00	Intermediate Station	00:05:00
3	Jaipur	11:30:00	NULL	Destination Station	NULL

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

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `GetAvailableSeats`(
  IN p_train_id INT,
  IN p_class_id VARCHAR(10),
  IN p_travel_date DATE
)
BEGIN
  -- Query available confirmed seats from 'seats'
  SELECT 'CONFIRMED' AS category, COUNT(*) AS seats
  FROM seat
  WHERE Train_id = p_train_id
   AND Class_id = p_class_id
   AND Travel_date = p_travel_date
   AND Availibility = 'Available'
   UNION ALL
  -- Query available RAC seats from 'rac'
  SELECT 'RAC' AS category, COUNT(*) AS seats
  FROM rac_info
  WHERE Train_id = p_train_id
   AND Class_id = p_class_id
   AND Travel_date = p_travel_date
   AND Availibility = 'Available';
END
```

EXAMPLE USE:

call GetAvailableSeats(12001,1,'2025-04-14');

	category	seats
▸	CONFIRMED	5
	RAC	5

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

```
DELIMITER //
CREATE PROCEDURE GetPassengerList(
  IN p_train_id INT,
  IN p_travel_date DATE
)
BEGIN
```

```
  SELECT
    tk.Ticket_status AS Passenger_Type,
    tk.PNR,
    p.Name,
    p.Age,
    p.Gender,
    p.Category,
    c.Class_name,
    COALESCE(se.Seat_no, 'Not Assigned') AS Seat_no,
    s1.Station_name AS Boarding_station,
    s2.Station_name AS Arrival_station,
    pm.Amount AS Ticket_Amount
  FROM ticket tk
  JOIN passengers p ON tk.Passenger_id = p.Passenger_id
  JOIN class c ON tk.Class_id = c.Class_id
  LEFT JOIN seat se ON tk.Seat_id = se.Seat_id
  JOIN stations s1 ON tk.Boarding_station = s1.Station_id
  JOIN stations s2 ON tk.Arrival_station = s2.Station_id
  JOIN payment pm ON tk.Payment_id = pm.Payment_id
  WHERE tk.Train_id = p_train_id
  AND tk.Travel_date = p_travel_date
  ORDER BY tk.Ticket_status, p.Name;
END //

DELIMITER ;
```

EXAMPLE USE :

```
call GetPassengerList(12001,'2025-04-14');
```

Passenger_id	Name
1003	Amit Singh
1004	Neha Patel
1002	Priya Sharma
1001	Rajesh Kumar
1005	Suresh Verma

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

```
CREATE PROCEDURE GetWaitlistedPassengers(
  IN p_train_id INT
)
BEGIN
  SELECT
    passengers.Passenger_id,
```

```
    passengers.Name,
    ticket.Travel_date
  FROM ticket
  JOIN passengers ON ticket.Passenger_id = passengers.Passenger_id
  WHERE ticket.Train_id = p_train_id
   AND ticket.Ticket_status = 'Waiting'
  ORDER BY passengers.Name;
END
```

EXAMPLE USE:

```
call GetWaitlistedPassengers(12001);
```

Passenger_id	Name	Travel_Date
1011	Kiran Joshi	2025-04-14
1012	Ritu Malhotra	2025-04-14

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

```
CREATE PROCEDURE CalculateTrainCancellationRefund(
  IN p_train_no INT,
  IN p_travel_date DATE
)
BEGIN
  SELECT
    SUM(pm.Amount) AS Total_Refund_Amount
  FROM payment pm
  WHERE pm.Payment_id IN (
    SELECT DISTINCT tk.Payment_id
    FROM ticket tk
    WHERE tk.Train_id = p_train_no
     AND tk.Travel_date = p_travel_date
     AND tk.Ticket_status IN ('Confirmed', 'RAC', 'Waiting')
  );
END
```

EXAMPLE USE : For the given data :

Payment_id	Ticket_id	Amount	Payment_mode	Date_of_Payment
1	1	3480.00	Credit Card	2025-04-13
2	2	3600.00	Credit Card	2025-04-13

```
call CalculateTrainCancellationRefund(12001,'2025-04-14');
```

Total_Refund_Amount
7080.00

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

```
DELIMITER //

CREATE PROCEDURE CalculateTotalRevenue(
  IN p_start_date DATE,
  IN p_end_date DATE
)
BEGIN
  SELECT
    SUM(Amount) AS Total_Revenue
  FROM payment
  WHERE Date_of_Payment BETWEEN p_start_date AND p_end_date;
END //

DELIMITER ;

EXAMPLE USE:
call CalculateTotalRevenue('2025-04-13','2025-04-16');
```

Total_Revenue
14160.00

Query 8 ---- Find the busiest train based on passenger count.

```
CREATE PROCEDURE FindBusiestTrain()
BEGIN
  -- Find the train with the maximum number of unique ticket-passenger pairs
  SELECT Train_id, COUNT(DISTINCT Ticket_id, Passenger_id) AS Total_Pairs
  FROM ticket
  GROUP BY Train_id
  ORDER BY Total_Pairs DESC
  LIMIT 1;
END

EXAMPLE USE:
call FindBusiestTrain();
```

Train_id	Total_Pairs
12001	12

Query 9 ---- Cancellation records with refund status.

```
Select * from refund_record;
```

EXAMPLE USE:

	Refund_id	Payment_id	Amount	Booking_date	Cancellation_date
▶	7609870	1	1170	2025-04-16	2025-04-16

OTHER INTERESTING QUERIES

Query 10 ---- Get list of trains running on a specific date

```
CREATE PROCEDURE get_trains_by_date(IN travel_date DATE)
BEGIN
  SELECT DISTINCT Train_no, Train_name
  FROM trains
  WHERE SUBSTRING(days_of_operation, DAYOFWEEK(travel_date), 1) = 'Y';
END
```

```
6 •  call get_trains_by_date('2025-04-11');
7
```

esult Grid | Filter Rows: | Export:

Train_no	Train_name
12001	Shatabdi Express
12002	Rajdhani Express
12004	Garib Rath
12005	Chennai Mail
12006	Mumbai Express
12007	Konkan Kanya
12009	Deccan Express
12010	Varanasi Express
12013	Kerala Express
12014	Chandigarh Express
12015	Bhopal Express
12016	Patna Vadodara Ex...
12017	Ganga Express
12019	Himalayan Queen

Query 11 – Get the day of week in which maximum trains run.

```
WITH day_counts AS (
 SELECT
  1 AS day_no, 'Sunday' AS day_name, SUM(SUBSTRING(days_of_operation,1,1) = 'Y') AS
train_count
 FROM miniproject.trains
 UNION ALL
 SELECT
  2, 'Monday', SUM(SUBSTRING(days_of_operation,2,1) = 'Y')
 FROM miniproject.trains
 UNION ALL
 SELECT
  3, 'Tuesday', SUM(SUBSTRING(days_of_operation,3,1) = 'Y')
 FROM miniproject.trains
 UNION ALL
 SELECT
  4, 'Wednesday', SUM(SUBSTRING(days_of_operation,4,1) = 'Y')
 FROM miniproject.trains
 UNION ALL
 SELECT
  5, 'Thursday', SUM(SUBSTRING(days_of_operation,5,1) = 'Y')
 FROM miniproject.trains
 UNION ALL
 SELECT
  6, 'Friday', SUM(SUBSTRING(days_of_operation,6,1) = 'Y')
 FROM miniproject.trains
 UNION ALL
 SELECT
  7, 'Saturday', SUM(SUBSTRING(days_of_operation,7,1) = 'Y')
 FROM miniproject.trains
),
max_count AS ( SELECT MAX(train_count) AS max_trains FROM day_counts)
SELECT day_name, train_count
FROM day_counts, max_count
WHERE train_count = max_trains;
```

EXAMPLE USE :

day_name	train_count
Monday	18

SQL Procedure for Booking Tickets :

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `book_tickets`(IN p_travel_date DATE, IN
p_passenger_ids VARCHAR(255), IN p_class_id INT, IN p_boarding_station_id INT, IN
p_arrival_station_id INT, IN p_train_id INT, IN p_payment_mode VARCHAR(50))
BEGIN
  DECLARE v_passenger_id, v_passenger_count, v_seat_id, v_boarding_sequence,
v_arrival_sequence, v_station_difference, v_ticket_id, v_payment_id, v_available_count,
v_rac_count, v_waiting_count, v_has_seat_entries, i, v_offset INT DEFAULT 0;
  DECLARE v_remaining VARCHAR(255);
  DECLARE v_fare_multiplier, v_passenger_amount, v_total_amount DECIMAL(10,2) DEFAULT
0;
  DECLARE v_category, v_ticket_status, v_seat_no VARCHAR(50);
  DECLARE v_current_date DATE;

  SET v_current_date=CURDATE();
  SELECT COUNT(*) INTO v_has_seat_entries FROM seat WHERE Train_id=p_train_id AND
Travel_date=p_travel_date AND Class_id=p_class_id;
  IF v_has_seat_entries=0 THEN SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT='No seat
entries available'; END IF;

  SELECT Fare_multiplier INTO v_fare_multiplier FROM class WHERE Class_id=p_class_id;
  SELECT Station_sequence INTO v_boarding_sequence FROM routes WHERE
Train_id=p_train_id AND Station_id=p_boarding_station_id;
  SELECT Station_sequence INTO v_arrival_sequence FROM routes WHERE Train_id=p_train_id
AND Station_id=p_arrival_station_id;
  IF v_boarding_sequence IS NULL OR v_arrival_sequence IS NULL THEN SIGNAL SQLSTATE
'45000' SET MESSAGE_TEXT='Invalid boarding/arrival station'; END IF;
  IF v_boarding_sequence>=v_arrival_sequence THEN SIGNAL SQLSTATE '45000' SET
MESSAGE_TEXT='Boarding station must precede arrival'; END IF;

  SET v_station_difference=v_arrival_sequence-v_boarding_sequence;
  SELECT COUNT(*) INTO v_available_count FROM seat WHERE Train_id=p_train_id AND
Class_id=p_class_id AND Travel_date=p_travel_date AND Availibility='Available';
  SELECT COUNT(*) INTO v_rac_count FROM RAC_info WHERE Train_id=p_train_id AND
Class_id=p_class_id AND Travel_date=p_travel_date;
  SELECT COUNT(*) INTO v_waiting_count FROM WL_info WHERE Train_id=p_train_id AND
Class_id=p_class_id AND Travel_date=p_travel_date;

  SET @v_common_pnr=CONCAT('PNR',LPAD(FLOOR(RAND()*1000000),6,'0'));

  CREATE TEMPORARY TABLE IF NOT EXISTS temp_passengers(id INT);
  SET v_remaining=p_passenger_ids;
  WHILE LENGTH(v_remaining)>0 DO
    SET v_passenger_count=v_passenger_count+1;
    IF LOCATE(',',v_remaining)>0 THEN
```

```
      SET v_passenger_id=SUBSTRING(v_remaining,1,LOCATE(',',v_remaining)-1);
      SET v_remaining=SUBSTRING(v_remaining,LOCATE(',',v_remaining)+1);
    ELSE
      SET v_passenger_id=v_remaining;
      SET v_remaining='';
    END IF;
    INSERT INTO temp_passengers VALUES(v_passenger_id);
  END WHILE;

  IF v_passenger_count>(v_available_count+(5-v_rac_count)+1000) THEN DROP TEMPORARY
TABLE temp_passengers; SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT='Not enough seats';
END IF;

  CREATE TEMPORARY TABLE IF NOT EXISTS temp_ticket_details(Passenger_id INT, seat_id INT,
ticket_status VARCHAR(20));
  SET i=1;
  WHILE i<=v_passenger_count DO
    SET v_offset=i-1;
    SELECT id INTO v_passenger_id FROM temp_passengers LIMIT v_offset,1;
    SELECT Category INTO v_category FROM passengers WHERE
Passenger_id=v_passenger_id;
    SET v_passenger_amount=500+(v_station_difference*v_fare_multiplier);
    IF v_category IS NOT NULL THEN SET v_passenger_amount=v_passenger_amount*0.9; END
IF;
    SET v_passenger_amount=FLOOR(v_passenger_amount);
    SET v_total_amount=v_total_amount+v_passenger_amount;

    IF v_available_count>0 THEN
      SELECT Seat_id,Seat_no INTO v_seat_id,v_seat_no FROM seat WHERE Train_id=p_train_id
AND Class_id=p_class_id AND Travel_date=p_travel_date AND Availibility='Available' LIMIT 1;
      SET v_ticket_status='Confirmed';
      SET v_available_count=v_available_count-1;
      UPDATE seat SET Availibility='Booked' WHERE Seat_id=v_seat_id;
    ELSE
      SELECT COUNT(*) INTO v_rac_count FROM RAC_info WHERE Train_id=p_train_id AND
Class_id=p_class_id AND Travel_date=p_travel_date AND Availibility='Booked';
      IF 5-v_rac_count>0 THEN
        SELECT RAC_id INTO v_seat_id FROM RAC_info WHERE Train_id=p_train_id AND
Class_id=p_class_id AND Travel_date=p_travel_date AND Availibility='Available' LIMIT 1;
        SET v_ticket_status='RAC';
        SET v_seat_no=CONCAT('RAC',v_seat_id);
        UPDATE RAC_info SET Availibility='Booked' WHERE RAC_id=v_seat_id;
      ELSE
        SET v_ticket_status='Waiting';
        SET v_seat_id=NULL;
        SET v_waiting_count=v_waiting_count+1;
        SET v_seat_no=CONCAT('WL',v_waiting_count);
```

```
            INSERT INTO WL_info(Train_id,Class_id,Travel_date)
VALUES(p_train_id,p_class_id,p_travel_date);
          END IF;END IF;
       INSERT INTO temp_ticket_details VALUES(v_passenger_id,v_seat_id,v_ticket_status);
       SET i=i+1;
    END WHILE;

    SELECT IFNULL(MAX(Ticket_id),0)+1 INTO v_ticket_id FROM ticket;
    SET v_payment_id=v_ticket_id;
    INSERT INTO payment
VALUES(v_payment_id,v_ticket_id,v_total_amount,p_payment_mode,v_current_date);
    INSERT INTO
ticket(Ticket_id,Passenger_id,Train_id,PNR,Class_id,Seat_id,Boarding_station,Arrival_station,Ti
cket_status,Payment_id,Travel_date)
    SELECT
v_ticket_id,Passenger_id,p_train_id,@v_common_pnr,p_class_id,seat_id,p_boarding_station_id
,p_arrival_station_id,ticket_status,v_payment_id,p_travel_date FROM temp_ticket_details;

    SELECT @v_common_pnr AS PNR,v_passenger_count AS
Total_Tickets_Booked,v_total_amount AS Total_Amount_Charged;
    DROP TEMPORARY TABLE temp_passengers;
    DROP TEMPORARY TABLE temp_ticket_details;
END
```

Cancellation procedure for tickets:

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `cancel_ticket`(IN p_ticket_id INT)
BEGIN
    DECLARE v_ticket_id, v_seat_id, v_train_id, v_class_id, v_payment_id, v_refund_id,
v_passenger_count, v_cancelled_confirmed, v_cancelled_rac, v_cancelled_waiting, done INT
DEFAULT 0;
    DECLARE v_pnr VARCHAR(10);
    DECLARE v_travel_date, v_payment_date, v_today DATE;
    DECLARE v_payment_amount, v_refund_amount DECIMAL(10,2);

    DECLARE seat_cursor CURSOR FOR SELECT Seat_id FROM temp_ticket_details WHERE
Ticket_status='Confirmed';
    DECLARE rac_cursor CURSOR FOR SELECT Seat_id FROM rac_info WHERE
Train_id=v_train_id AND Class_id=v_class_id AND Travel_date=v_travel_date;

    START TRANSACTION;

    SELECT Train_id, Class_id, Travel_date, Payment_id, PNR, COUNT(*)
```

```
  INTO v_train_id, v_class_id, v_travel_date, v_payment_id, v_pnr, v_passenger_count
  FROM ticket WHERE Ticket_id=p_ticket_id GROUP BY Train_id, Class_id, Travel_date,
Payment_id, PNR;

  IF v_train_id IS NULL THEN SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT='Ticket not found';
ROLLBACK; END IF;

  SELECT Amount, Date_of_Payment INTO v_payment_amount, v_payment_date FROM
payment WHERE Payment_id=v_payment_id;

  SET v_today=CURDATE(),
    v_refund_amount=CASE
      WHEN DATEDIFF(v_travel_date,v_today)>=5 THEN v_payment_amount
      WHEN DATEDIFF(v_travel_date,v_today)>=2 THEN v_payment_amount*0.5
      WHEN DATEDIFF(v_travel_date,v_today)>=1 THEN v_payment_amount*0.25
      ELSE 0
    END,
    v_refund_id=FLOOR(5000000+RAND()*4000000);

  INSERT INTO refund_record (Refund_id, Payment_id, Amount, Booking_date,
Cancellation_date)
  VALUES (v_refund_id, v_payment_id, v_refund_amount, v_payment_date, v_today);

  DROP TABLE IF EXISTS temp_ticket_details;
  CREATE TABLE temp_ticket_details AS SELECT Ticket_id, Passenger_id, Seat_id, Ticket_status
FROM ticket WHERE PNR=v_pnr;

  SELECT
    SUM(CASE WHEN Ticket_status='Confirmed' THEN 1 ELSE 0 END),
    SUM(CASE WHEN Ticket_status='RAC' THEN 1 ELSE 0 END),
    SUM(CASE WHEN Ticket_status='Waiting' THEN 1 ELSE 0 END)
  INTO v_cancelled_confirmed, v_cancelled_rac, v_cancelled_waiting FROM
temp_ticket_details;

  OPEN seat_cursor;
  read_seat: LOOP
    FETCH seat_cursor INTO v_seat_id;
    IF done THEN LEAVE read_seat; END IF;
    UPDATE seat SET Availability='Available' WHERE Seat_id=v_seat_id;
  END LOOP;
  CLOSE seat_cursor;

  OPEN rac_cursor;
  read_rac: LOOP
    FETCH rac_cursor INTO v_seat_id;
    IF done THEN LEAVE read_rac; END IF;
    UPDATE rac_info SET Availability='Available' WHERE Seat_id=v_seat_id AND
Train_id=v_train_id AND Class_id=v_class_id AND Travel_date=v_travel_date;
```

```
    END LOOP;
    CLOSE rac_cursor;
    DELETE FROM wl_info WHERE Train_id=v_train_id AND Class_id=v_class_id AND
Travel_date=v_travel_date;
    DELETE FROM ticket WHERE PNR=v_pnr;
    DELETE FROM payment WHERE Payment_id=v_payment_id;
    COMMIT;
    SELECT p_ticket_id AS Cancelled_Ticket_ID, v_pnr AS Cancelled_PNR, v_passenger_count AS
Passengers_Affected,
        v_refund_amount AS Refund_Amount, v_refund_id AS Refund_ID,
        v_cancelled_confirmed AS Confirmed_Tickets_Cancelled, v_cancelled_rac AS
RAC_Tickets_Cancelled,
        v_cancelled_waiting AS Waiting_Tickets_Cancelled; END
```

Example of Booking and Cancellation of tickets:

Calling the booking procedure to book 4 tickets

```
CALL book_tickets(
    '2025-04-18',                -- Travel date
    '1004,1005,1006,1007',           -- Passenger IDs (Rajesh, Priya, Neha)
    1,                           -- Class ID (2AC)
    5,                           -- Boarding station ID (Delhi)1
    6,                           -- Arrival station ID (Jaipur)
    12004,                       -- Train ID (Shatabdi Express)
    'Offline'                 -- Payment mode
);
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

PNR	Total_Tickets_Booked	Total_Amount_Charged
PNR058479	4	2340.00

4 tickets have been booked with same PNR and Confirmed Status

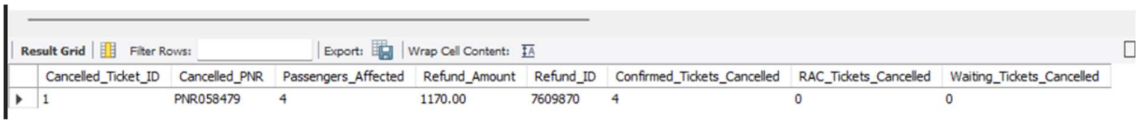
Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

Ticket_id	Passenger_id	Train_id	PNR	Class_id	Seat_id	Boarding_station	Arrival_station	Ticket_status	Payment_id	Travel_date
1	1004	12004	PNR058479	1	2001	5	6	Confirmed	1	2025-04-18
1	1005	12004	PNR058479	1	2002	5	6	Confirmed	1	2025-04-18
1	1006	12004	PNR058479	1	2003	5	6	Confirmed	1	2025-04-18
1	1007	12004	PNR058479	1	2004	5	6	Confirmed	1	2025-04-18
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Calling the Cancel ticket procedure

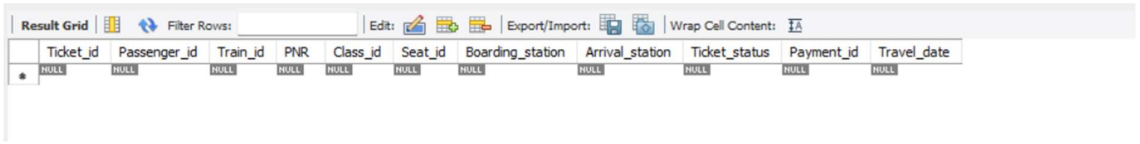
```
call cancel_ticket(1);
```

Cancel Report



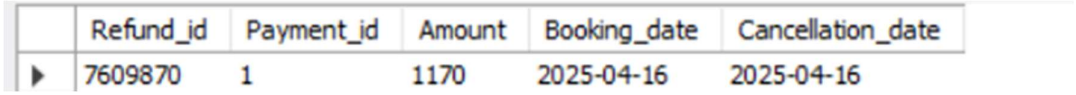
Cancelled_Ticket_ID	Cancelled_PNR	Passengers_Affected	Refund_Amount	Refund_ID	Confirmed_Tickets_Cancelled	RAC_Tickets_Cancelled	Waiting_Tickets_Cancelled
1	PNR058479	4	1170.00	7609870	4	0	0

Tickets have been removed from Ticket table



Ticket_id	Passenger_id	Train_id	PNR	Class_id	Seat_id	Boarding_station	Arrival_station	Ticket_status	Payment_id	Travel_date
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Refund table has been updated



Refund_id	Payment_id	Amount	Booking_date	Cancellation_date
7609870	1	1170	2025-04-16	2025-04-16