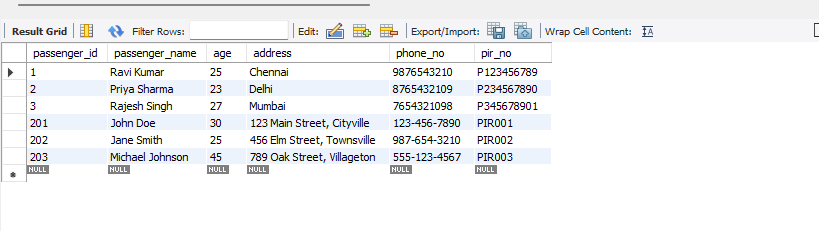
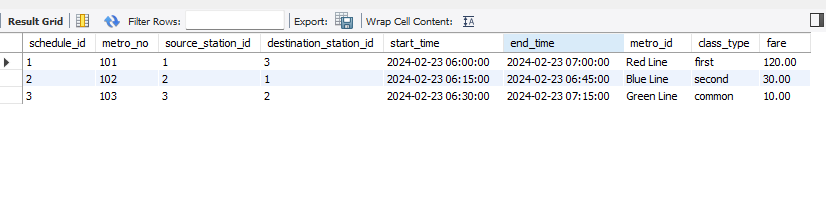
**DBMS PROJECT COMMANDS  
  
  
-- Selecting all passengers**

SELECT \* FROM Passenger;  
  


**-- Selecting schedule details with metro information**

SELECT s.\*, m.metro\_id, m.class\_type, m.fare

FROM Schedule s

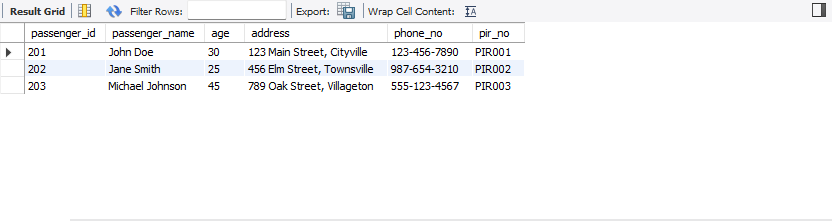
JOIN Metro m ON s.metro\_no = m.metro\_no;  
  


**-- Selecting passengers who booked a ticket with status 'confirmed'**

SELECT p.\*

FROM Passenger p

JOIN Booking b ON p.passenger\_id = b.passenger\_id

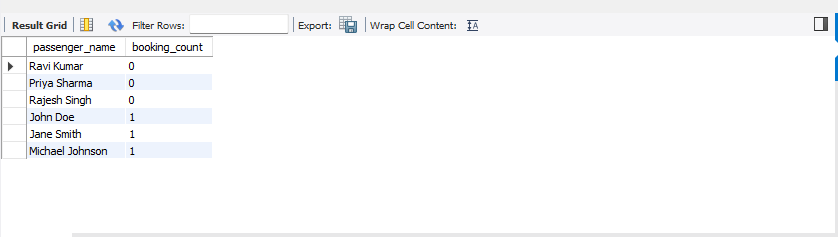
WHERE b.status = 'confirmed';  
  


**-- Counting the number of bookings for each passenger**

SELECT p.passenger\_name, COUNT(b.booking\_id) AS booking\_count

FROM Passenger p

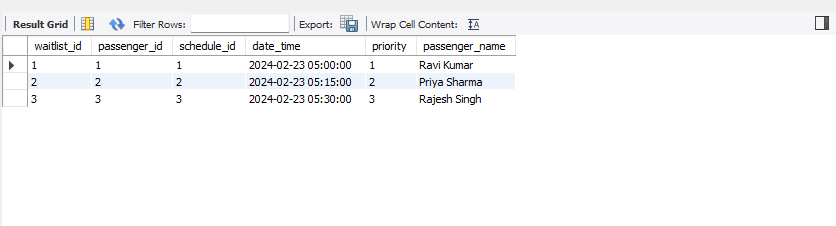
LEFT JOIN Booking b ON p.passenger\_id = b.passenger\_id

GROUP BY p.passenger\_id;  


**-- Finding passengers on waitlist with their priority**

SELECT w.\*, p.passenger\_name

FROM Waitlist w

JOIN Passenger p ON w.passenger\_id = p.passenger\_id;  


**Finding passengers who booked a ticket for a schedule**

SELECT passenger\_name

FROM Passenger

WHERE passenger\_id IN (

SELECT passenger\_id

FROM Booking

WHERE schedule\_id IN (

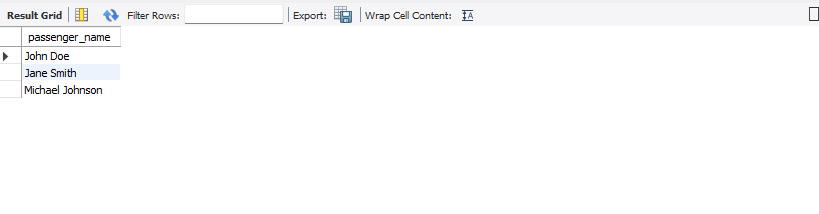
SELECT schedule\_id

FROM Schedule

JOIN Metro ON Schedule.metro\_no = Metro.metro\_no

WHERE fare > 10.00

)

);  


**-- Finding the maximum fare among all schedules**

SELECT MAX(fare) AS max\_fare

FROM Schedule

JOIN Metro ON Schedule.metro\_no = Metro.metro\_no;

**  
  
  
  
  
  
  
  
  
  
-- Creating a view to show confirmed bookings**

CREATE VIEW ConfirmedBookings AS

SELECT \* FROM Booking WHERE status = 'confirmed';

**-- Querying the view**

SELECT \* FROM ConfirmedBookings;

**View to show the details of passengers along with their bookings**

CREATE VIEW PassengerBookingsDetails AS

SELECT p.passenger\_id, p.passenger\_name, p.age, p.address, p.phone\_no,

b.booking\_id, b.date\_time AS booking\_time, b.ticket\_no, b.class\_type, b.seat\_no, b.status,

s.schedule\_id, s.start\_time AS departure\_time, s.end\_time AS arrival\_time,

m.metro\_id, m.class\_type AS metro\_class, m.fare AS metro\_fare

FROM Passenger p

JOIN Booking b ON p.passenger\_id = b.passenger\_id

JOIN Schedule s ON b.schedule\_id = s.schedule\_id

JOIN Metro m ON s.metro\_no = m.metro\_no;

**-- View to show the details of stations along with their routes**

CREATE VIEW StationRoutes AS

SELECT s.station\_id, s.station\_name,

r.route\_id, r.schedule\_id,

s2.station\_id AS next\_station\_id, s2.station\_name AS next\_station\_name

FROM Station s

JOIN Route r ON s.station\_id = r.station\_id

JOIN Station s2 ON r.stop\_no + 1 = s2.station\_id;

**Subquery to find passengers who have bookings with waiting status**

SELECT passenger\_name

FROM Passenger

WHERE passenger\_id IN (

SELECT passenger\_id

FROM Booking

WHERE status = 'waiting'

);

**-- Subquery to find schedules that have no bookings**

SELECT schedule\_id

FROM Schedule

WHERE schedule\_id NOT IN (

SELECT DISTINCT schedule\_id

FROM Booking);

**-- Join to retrieve passenger details along with their bookings and associated metro details**

SELECT p.\*, b.booking\_id, b.date\_time, b.ticket\_no, b.class\_type, b.seat\_no, b.status,

m.metro\_id, m.class\_type AS metro\_class, m.fare AS metro\_fare

FROM Passenger p

JOIN Booking b ON p.passenger\_id = b.passenger\_id

JOIN Schedule s ON b.schedule\_id = s.schedule\_id

JOIN Metro m ON s.metro\_no = m.metro\_no;

**-- Join to retrieve station names for a given route**

SELECT r.route\_id, r.schedule\_id, r.station\_id, r.stop\_no,

s.station\_name

FROM Route r

JOIN Station s ON r.station\_id = s.station\_id;

CREATE VIEW MetroSchedules AS

SELECT sc.schedule\_id, sc.start\_time, sc.end\_time,

sc.source\_station\_id, ss.station\_name AS source\_station\_name,

sc.destination\_station\_id, ds.station\_name AS destination\_station\_name,

m.metro\_id, m.class\_type AS metro\_class, m.fare AS metro\_fare

FROM Schedule sc

JOIN Station ss ON sc.source\_station\_id = ss.station\_id

JOIN Station ds ON sc.destination\_station\_id = ds.station\_id

JOIN Metro m ON sc.metro\_no = m.metro\_no;

: CREATE VIEW WaitlistedPassengers AS

SELECT w.waitlist\_id, w.passenger\_id, p.passenger\_name,

w.schedule\_id, sc.start\_time, sc.end\_time,

w.date\_time AS waitlist\_time, w.priority

FROM Waitlist w

JOIN Passenger p ON w.passenger\_id = p.passenger\_id

JOIN Schedule sc ON w.schedule\_id = sc.schedule\_id;

CREATE VIEW ScheduleRoutes AS

SELECT sc.schedule\_id, sc.start\_time, sc.end\_time,

r.route\_id, r.station\_id, r.stop\_no,

s.station\_name

FROM Schedule sc

JOIN Route r ON sc.schedule\_id = r.schedule\_id

JOIN Station s ON r.station\_id = s.station\_id;

CREATE VIEW PassengerBookingsRoutes AS

SELECT p.passenger\_id, p.passenger\_name,

b.booking\_id, b.date\_time AS booking\_time, b.ticket\_no, b.class\_type, b.seat\_no, b.status,

sr.schedule\_id, sr.start\_time AS departure\_time, sr.end\_time AS arrival\_time,

sr.route\_id, sr.station\_id, sr.stop\_no,

s.station\_name

FROM Passenger p

JOIN Booking b ON p.passenger\_id = b.passenger\_id

JOIN ScheduleRoutes sr ON b.schedule\_id = sr.schedule\_id

JOIN Station s ON sr.station\_id = s.station\_id

WHERE b.status = 'confirmed';

CREATE VIEW WaitlistedPassengers AS

SELECT w.waitlist\_id, w.passenger\_id, p.passenger\_name,

w.schedule\_id, sc.start\_time, sc.end\_time,

w.date\_time AS waitlist\_time, w.priority

FROM Waitlist w

JOIN Passenger p ON w.passenger\_id = p.passenger\_id

JOIN Schedule sc ON w.schedule\_id = sc.schedule\_id;  
  
**trigger**DELIMITER //

CREATE TRIGGER CheckBookingAvailability

AFTER INSERT ON Booking

FOR EACH ROW

BEGIN

DECLARE total\_bookings INT;

DECLARE max\_capacity INT;

-- Get the total number of bookings for the schedule

SELECT COUNT(\*) INTO total\_bookings

FROM Booking

WHERE schedule\_id = NEW.schedule\_id;

-- Get the maximum capacity for the schedule

SELECT COUNT(\*) INTO max\_capacity

FROM Route

WHERE schedule\_id = NEW.schedule\_id;

-- If the total number of bookings exceeds the maximum capacity, insert into the Waitlist table

IF total\_bookings >= max\_capacity THEN

INSERT INTO Waitlist (passenger\_id, schedule\_id, date\_time, priority)

VALUES (NEW.passenger\_id, NEW.schedule\_id, NOW(), 1);

END IF;

END //

DELIMITER ;  


DELIMITER //

CREATE TRIGGER UpdateBookingFare

AFTER UPDATE ON Metro

FOR EACH ROW

BEGIN

UPDATE Booking

SET fare = NEW.fare

WHERE schedule\_id IN (

SELECT schedule\_id

FROM Schedule

WHERE metro\_no = NEW.metro\_no

);

END //

DELIMITER ;  
  
  
  
  
**CONSTRAINTS**  
  
ALTER TABLE metro

ADD CONSTRAINT UC\_metro\_id UNIQUE (metro\_id);

ALTER TABLE Passenger

ADD CONSTRAINT CHK\_AdultOnly CHECK (age >= 18);

ALTER TABLE Station

ADD CONSTRAINT PK\_Station PRIMARY KEY (station\_id);

ALTER TABLE Route

ADD CONSTRAINT FK\_Schedule FOREIGN KEY (schedule\_id) REFERENCES Schedule(schedule\_id);

**PITFALLS**

pitfall in our database

🔯 information about metro trains, including their metro\_id. If there are duplicate metro\_id values, it could lead to data redundancy and inconsistency.

🔯 Ensure that the metro\_id is unique (using the UNIQUE constraint) to prevent duplicate entries.

🔯 The fare column in the metro table should not be negative.

Implement a CHECK constraint to validate fare values.

🔯 Verify that foreign key constraints are correctly defined and enforced.

For example, the Booking table references both Passenger and Schedule. Ensure that passenger and schedule IDs exist before creating bookings.  
  
  
**FUNCTIONAL DEPENDENCY**

* 1. **metro\_no -> metro\_id, class\_type, fare**
* Ensure that metro\_no is the primary key in the metro table.
* Include attributes metro\_id, class\_type, and fare in the metro table.
* 2. s**chedule\_id -> metro\_no, source\_station\_id, destination\_station\_id, start\_time, end\_time**
* Make schedule\_id the primary key in the Schedule table.
* Include attributes metro\_no, source\_station\_id, destination\_station\_id, start\_time, and end\_time in the Schedule table.
* 3. **station\_id -> station\_name**
* Keep station\_id as the primary key in the Station table.
* Include attribute station\_name in the Station table.
* 4. **route\_id -> schedule\_id, station\_id, stop\_no**
* Ensure that route\_id is the primary key in the Route table.
* Include attributes schedule\_id, station\_id, and stop\_no in the Route table.
* 5. **passenger\_id -> passenger\_name, age, address, phone\_no, pir\_no**
* Make passenger\_id the primary key in the Passenger table.
* Include attributes passenger\_name, age, address, phone\_no, and pir\_no in the Passenger table.
* 6. **booking\_id -> passenger\_id, schedule\_id, date\_time, ticket\_no, class\_type, seat\_no, status**
* Make booking\_id the primary key in the Booking table.
* Include attributes passenger\_id, schedule\_id, date\_time, ticket\_no, class\_type, seat\_no, and status in the Booking table.
* 7. **waitlist\_id -> passenger\_id, schedule\_id, date\_time, priority**
* Ensure that waitlist\_id is the primary key in the Waitlist table.
* Include attributes passenger\_id, schedule\_id, date\_time, and priority in the Waitlist table.

**NORMALIZATION FORM**

**First Normal Form (1NF):**

* Each cell in the table should contain a single value, and there should be no repeating groups.
* All the tables in the given schema satisfy the First Normal Form.

**Second Normal Form (2NF):**

* The table should be in 1NF, and all non-key attributes should be fully dependent on the primary key.
* All the tables in the given schema satisfy the Second Normal Form.

**Third Normal Form (3NF):**

* The table should be in 2NF, and there should be no transitive dependencies between non-key attributes.
* The metro table satisfies the Third Normal Form as it has no transitive dependencies.
* The Station table satisfies the Third Normal Form as it has no transitive dependencies.
* The Passenger table satisfies the Third Normal Form as it has no transitive dependencies.
* The Schedule table satisfies the Third Normal Form as it has no transitive dependencies.
* The Route table satisfies the Third Normal Form as it has no transitive dependencies.
* The Booking table satisfies the Third Normal Form as it has no transitive dependencies.
* The Waitlist table satisfies the Third Normal Form as it has no transitive dependencies.

**Boyce-Codd Normal Form (BCNF):**

* The table should be in 3NF, and every determinant should be a candidate key.
* All the tables in the given schema satisfy the Boyce-Codd Normal Form.

**Fourth Normal Form (4NF)**:

* The table should be in BCNF, and there should be no multi-valued dependencies.
* All the tables in the given schema satisfy the Fourth Normal Form as there are no multi-valued dependencies present.

**Fifth Normal Form (5NF):**

* The table should be in 4NF, and every join dependency should be derived from the candidate keys.
* All the tables in the given schema satisfy the Fifth Normal Form as there are no join dependencies that are not derived from the candidate keys.