Database Management System

PROJECT SUBMISSION

Railway Management System

TABLE OF CONTENT

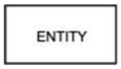
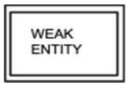
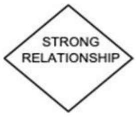
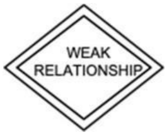
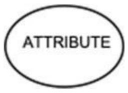
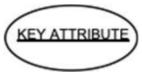
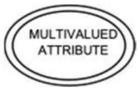
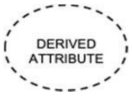
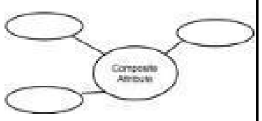
BASIC INTRODUCTION

1. **DATABASE MANAGEMENT SYSTEM:** A database management system (DBMS) is a software system that enables users to create, manage, and access databases. A database is a collection of data that is organized in a way that makes it easy to search, retrieve, update, and delete data. A DBMS provides a set of tools and functions for managing databases, including creating and modifying database schemas, storing and retrieving data, and controlling access to the database.

2. **RELATIONAL DATABASE MANAGEMENT SYSTEM:** A relational database management system (RDBMS) is a type of database management system (DBMS) that is based on the relational model of data. The relational model organizes data into one or more tables, where each table consists of rows and columns. Each column in a table represents a specific attribute of the data, and each row represents a specific instance or record.

3. **ER DIAGRAM:** An Entity-Relationship (ER) diagram is a type of diagram used to represent the relationships between entities in a database. The ER diagram is a graphical representation of the entities and their relationships to each other. It is a high-level view of the database schema, showing the main entities, attributes, and relationships. In an ER diagram, entities are represented as rectangles, and the relationships between entities are represented as diamonds. Each entity has a set of attributes, which are represented as ovals or circles within the entity rectangle.

4. TERMINOLOGIES AND SYMBOLS OF ER DIAGRAM:

SYMBOL	DESCRIPTION
	This is a basic entity that is represented by a rectangle with its name inside.
	This is an entity that cannot solely be identified with its attributes (due to the absence of a primary key). It inherits the identifier of its parent entity and often integrates it with a partial key.
	A strong relationship is depicted by a single rhombus with its name inside. In this, an entity is independent, that is, its primary key for any child doesn't contain the primary key of the linked entity.
	A weak relationship is depicted by a double rhombus with the name inside. In this, the child is. dependent on the parent entity as its primary key would contain a component of the parent's primary key.
	A basic attribute is represented by a single oval with its name written inside.
	This is a special attribute that is used to uniquely identify an entity. It is represented by an oval with its name underlined.
	These are the attributes that can have multiple values (like the Name attribute can have First and Last name) and are represented by a double oval.
	A derived attribute might not be physically present in the database and could be logically derived from any other attribute (Represented by a dotted oval).
	Composite attributes are those attributes which are composed of many other simple attributes.
—	This depicts that not all the entities in the set are a part of the relationship and is depicted by a single line.
═	This means that all the entities in the set are in a relationship and are depicted by a double line.

RAILWAY MANAGEMENT SYSTEM

1. Introduction including Requirement Analysis

A Railway Management System is a complex and large-scale system that handles a wide range of data related to trains, stations, schedules, routes, passengers, and more. To efficiently manage such a system, a Database Management System (DBMS) is essential. In this project, we will design and implement a DBMS for a Railway Management System.

The objective of the project is to create a database that can store and manage information related to trains, stations, schedules, routes, and passengers. The database will be able to handle queries related to train schedules, availability of tickets, passenger information, and more.

The database will consist of multiple tables, each with its own set of attributes. The tables will be linked to each other through foreign keys to maintain data integrity and ensure that the information is consistent across all tables.

Some of the key features of the Railway Management System DBMS include

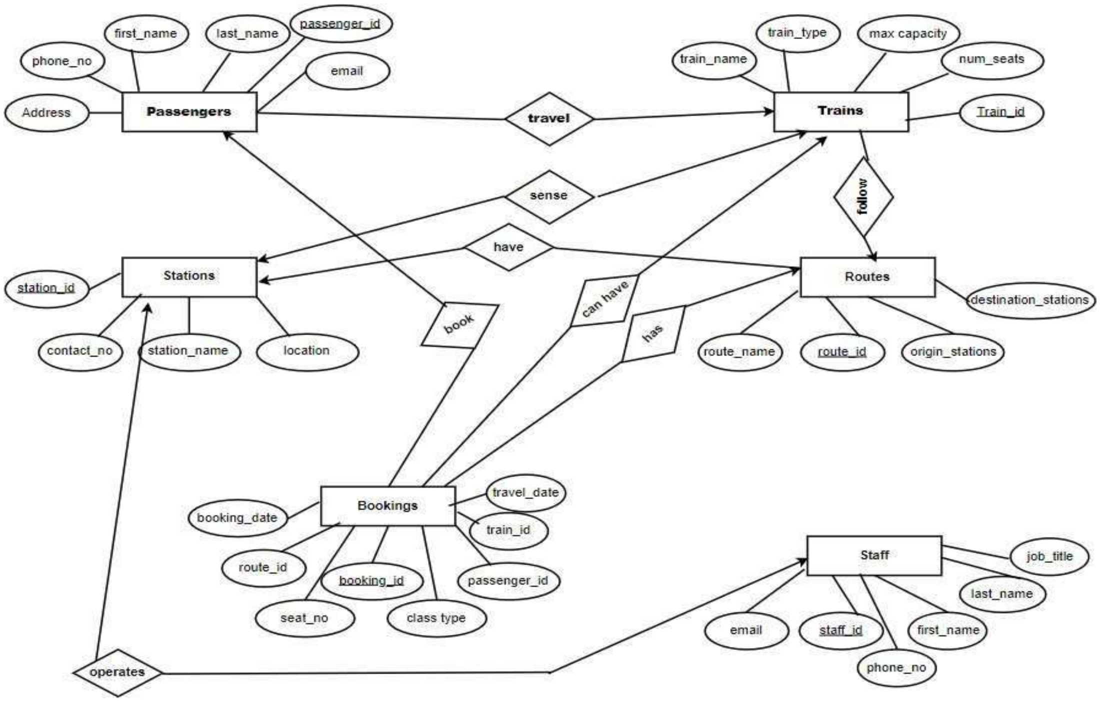
1. Train Management: This module will manage the trains and their schedules. It will include information such as train name, train number, departure time, arrival time, the route it takes, and more.
2. Station Management: This module will manage the stations and their details. It will include information such as station name, station code, location, and more.
3. Route Management: This module will manage the routes taken by trains. It will include information such as the list of stations on the route, distance between stations, and more.
4. Ticket Management: This module will manage the tickets purchased by passengers. It will include information such as the passenger’s name, train name, date of travel, seat number, and more.
5. Passenger Management: This module will manage the passenger details. It will include information such as passenger name, address, phone number, and more

The Railway Management System DBMS will be designed using a Relational Database Management System (RDBMS) such as MySQL or Oracle. The system will be developed using a combination of SQL queries, stored procedures, triggers, and views.

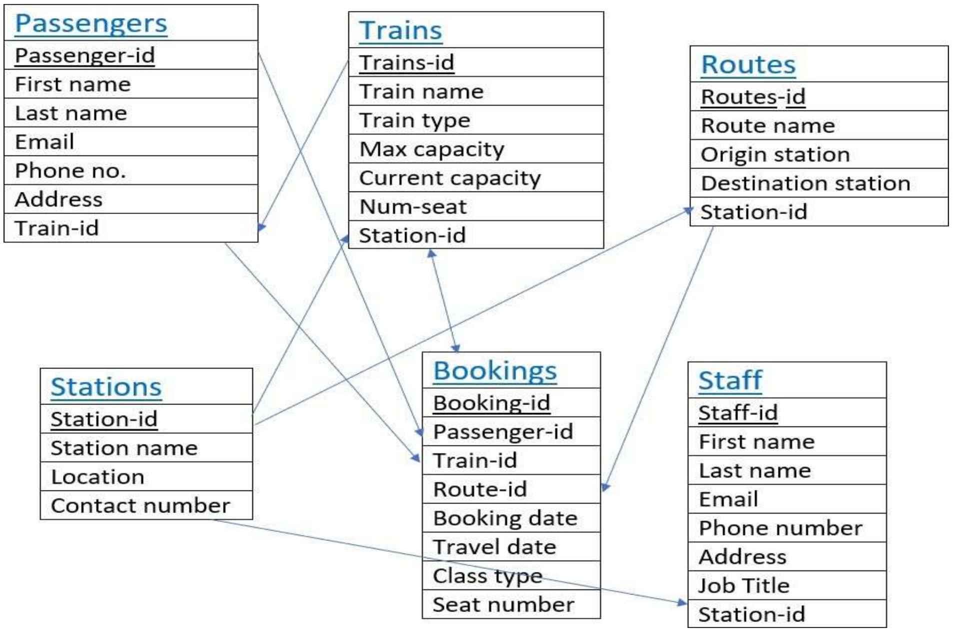
The final deliverables of the project will include a detailed database schema, a set of SQL queries to create and populate the database, and a user-friendly interface to access the database. The interface will allow users to perform tasks such as booking tickets, checking train schedules, and viewing passenger information.

Overall, the Railway Management System DBMS will provide an efficient and reliable way to manage the vast amounts of data associated with railway operations, making it easier for both railway staff and passengers to interact with the system.

2. **ER Diagram:**



3. **ER to Tables:**



Passengers
Passenger-id
First name
Last name
Email
Phone no.
Address
Train-id

Trains
Trains-id
Train name
Train type
Max capacity
Current capacity
Num-seat
Station-id

Routes
Routes-id
Route name
Origin station
Destination station
Station-id

Stations
Station-id
Station name
Location
Contact number

Bookings
Booking-id
Passenger-id
Train-id
Route-id
Booking date
Travel date
Class type
Seat number

Staff
Staff-id
First name
Last name
Email
Phone number
Address
Job Title
Station-id

4. Normalization:

The given database schema can be normalized to eliminate data redundancy and improve data integrity. We can follow the normalization process to achieve this:

First Normal Form (1NF):

All columns must contain atomic (indivisible) values.

No repeating groups or arrays.

Each record must be unique.

We can see that all the tables already satisfy 1NF.

Second Normal Form (2NF):

All the tables are in 2NF.

In the original schema, the primary keys were correctly defined in each table to ensure that every non-key attribute in the table is fully functionally dependent on the primary key.

Third Normal Form (3NF):

The Routes table is not in 3NF because it has a transitive dependency between the route_id and the origin_station/destination_station. To normalize the table, we can create a new table called Stations and move the origin_station and destination_station attributes from the Routes table to the Stations table. Original tables are already in 3NF, except for the Bookings table which has a transitive dependency, as class_type and num_seats depend on train_id which is not the primary key. To eliminate this, we can create a new table called Train_Details with train_id, class_type, max_capacity, and num_seats as attributes, and make train_id the primary key. Then, we can remove the class_type and num_seats attributes from the Bookings table, and add a foreign key to the Train_Details table.

The final normalized tables in 3NF would be:

- ***Passengers*** (passenger_id [PK], first_name, last_name, email, phone_number, address)
- ***Trains*** (train_id [PK], train_name, train_type)
- ***Train_Details*** (train_id [FK], class_type [FK], max_capacity, num_seats)
- ***Routes*** (route_id [PK], route_name)
- ***Stations*** (station_id [PK], station_name, location, contact_number)
- ***Route_Stations*** (route_id [FK], station_id [FK], stop_number [PK])
- ***Bookings*** (booking_id [PK], passenger_id [FK], train_id [FK], route_id [FK], booking_date, travel_date, class_type [FK], seat_number)
- ***Class_Types*** (class_type [PK], class_description)

The *Passengers*, *Trains*, *Routes*, and *Stations* tables are straightforward, with each row representing an individual entity and its attributes.

The *Train_Details* table holds information about each train's capacity and number of seats available for each class type. It has a foreign key to the *Trains* table and another foreign key to the *Class_Types* table.

The *Route_Stations* table establishes a many-to-many relationship between *Routes* and *Stations*, with an additional attribute indicating the order of stops on the route.

The *Bookings* table has foreign keys to *Passengers*, *Trains*, *Routes*, and *Class_Types* tables, representing the details of each booking made by a passenger.

Finally, the *Class_Types* table holds information about the available class types for each train. It has a primary key of C*lass_Types* and a class_description attribute.

By splitting the original tables into these normalized tables, we have eliminated any potential anomalies that could occur due to transitive dependencies, partial dependencies, or repeating groups, and ensured that the data is structured in a way that supports flexible querying and management.

5. OUTPUTS:

a) Procedure-

1. *CREATE OR REPLACE PROCEDURE delete_route(route_id IN INT) ASBEGIN*
   DELETE FROM Routes WHERE route_id = route_id;

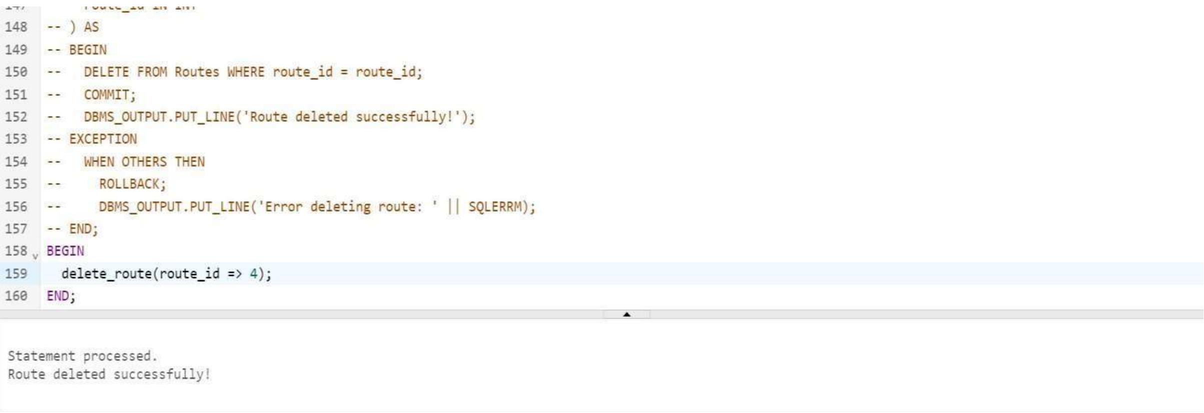
   COMMIT;

   DBMS_OUTPUT.PUT_LINE('Route deleted successfully!');

   EXCEPTION WHEN OTHERS THEN ROLLBACK;

   DBMS_OUTPUT.PUT_LINE('Error deleting route: ' || SQLERRM);

   END;



```
-- ) AS
-- BEGIN
--   DELETE FROM Routes WHERE route_id = route_id;
--   COMMIT;
--   DBMS_OUTPUT.PUT_LINE('Route deleted successfully!');
-- EXCEPTION
--   WHEN OTHERS THEN
--     ROLLBACK;
--     DBMS_OUTPUT.PUT_LINE('Error deleting route: ' || SQLERRM);
-- END;
BEGIN
  delete_route(route_id => 4);
END;
```

```
Statement processed.
Route deleted successfully!
```

2. *CREATE OR REPLACE PROCEDURE add_station(station_id IN INT, station_name IN VARCHAR2, location IN VARCHAR2, contact_number IN VARCHAR2) AS BEGIN INSERT INTO Stations (station_id, station_name, location, contact_number)*
VALUES (station_id, station_name, location, contact_number);

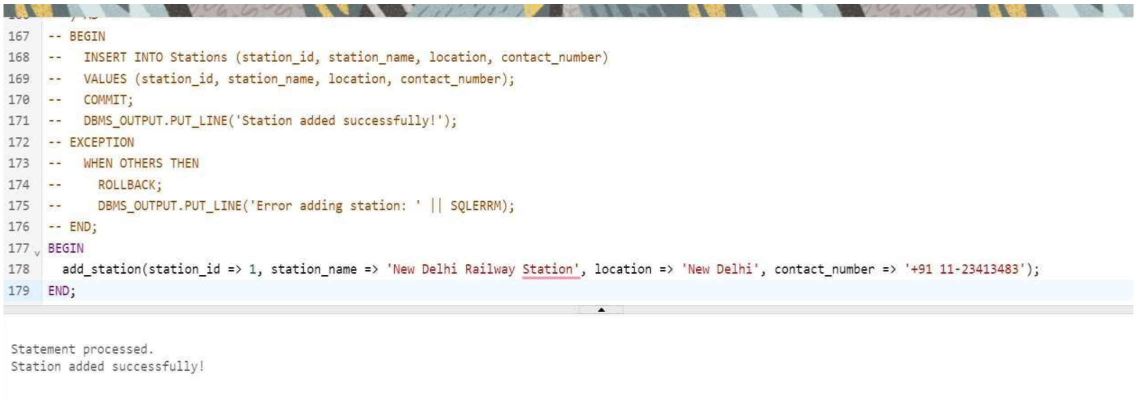
   COMMIT;

   DBMS_OUTPUT.PUT_LINE('Station added successfully!');

   EXCEPTION WHEN OTHERS THEN ROLLBACK;

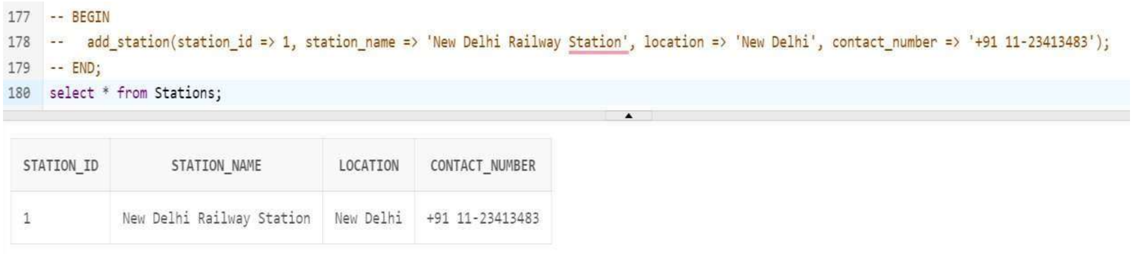
   DBMS_OUTPUT.PUT_LINE('Error adding station: ' || SQLERRM);

   END;



```
-- BEGIN
--   INSERT INTO Stations (station_id, station_name, location, contact_number)
--   VALUES (station_id, station_name, location, contact_number);
--   COMMIT;
--   DBMS_OUTPUT.PUT_LINE('Station added successfully!');
-- EXCEPTION
--   WHEN OTHERS THEN
--     ROLLBACK;
--     DBMS_OUTPUT.PUT_LINE('Error adding station: ' || SQLERRM);
-- END;
BEGIN
  add_station(station_id => 1, station_name => 'New Delhi Railway Station', location => 'New Delhi', contact_number => '+91 11-23413483');
END;
```

```
Statement processed.
Station added successfully!
```



```
-- BEGIN
--   add_station(station_id => 1, station_name => 'New Delhi Railway Station', location => 'New Delhi', contact_number => '+91 11-23413483');
-- END;
select * from Stations;
```

STATION_ID	STATION_NAME	LOCATION	CONTACT_NUMBER
1	New Delhi Railway Station	New Delhi	+91 11-23413483

3. CREATE OR REPLACE PROCEDURE update_job_title(staff_id IN NUMBER, new_job_title IN VARCHAR2) IS BEGIN

UPDATE Staff SET job_title = new_job_title WHERE staff_id = staff_id;

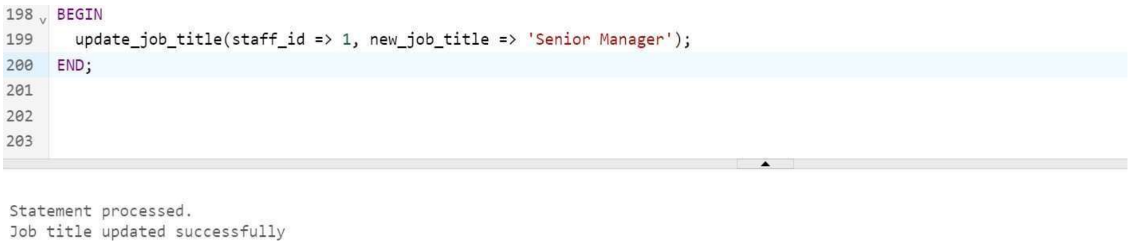
COMMIT;

DBMS_OUTPUT.PUT_LINE('Job title updated successfully');

EXCEPTION WHEN NO_DATA_FOUND THEN DBMS_OUTPUT.PUT_LINE('No staff member with ID ' || staff_id || ' found');

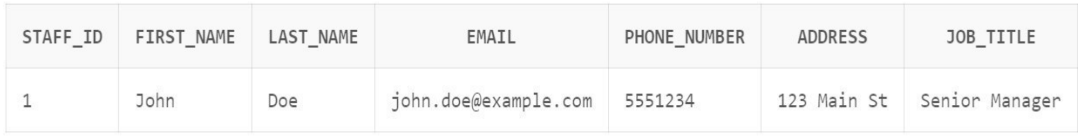
WHEN OTHERS THEN DBMS_OUTPUT.PUT_LINE('An error occurred: ' || SQLCODE || ' - ' || SQLERRM);

END;



```
BEGIN
  update_job_title(staff_id => 1, new_job_title => 'Senior Manager');
END;
```

```
Statement processed.
Job title updated successfully
```



STAFF_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	ADDRESS	JOB_TITLE
1	John	Doe	john.doe@example.com	5551234	123 Main St	Senior Manager

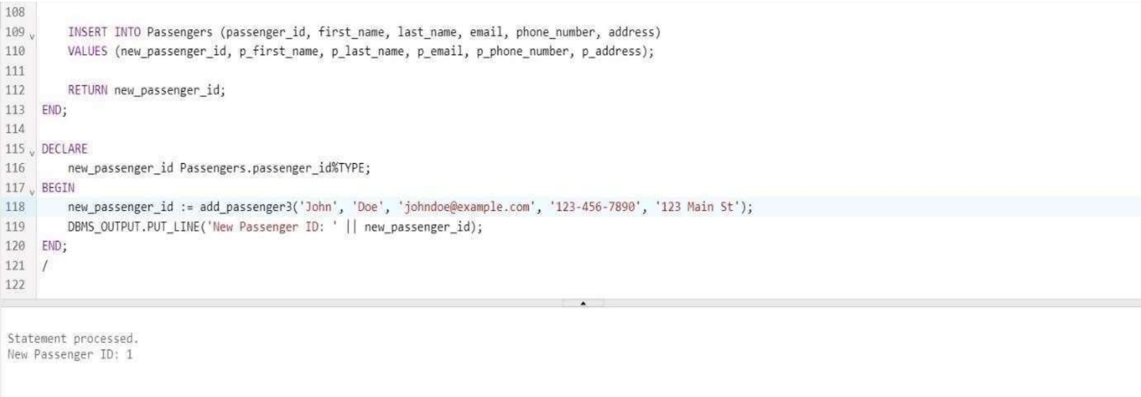
b) Functions-

1. *CREATE OR REPLACE FUNCTION add_passenger1(p_first_name IN Passengers.first_name%TYPE, p_last_name IN Passengers.last_name%TYPE, p_email IN Passengers.email%TYPE, p_phone_number IN Passengers.phone_number%TYPE, p_address IN Passengers.address%TYPE) RETURN Passengers.passenger_id%TYPE IS new_passenger_id Passengers.passenger_id%TYPE;*
   BEGIN

INSERT INTO Passengers (first_name, last_name, email, phone_number, address) VALUES (p_first_name, p_last_name, p_email, p_phone_number, p_address) RETURNING passenger_id INTO new_passenger_id;

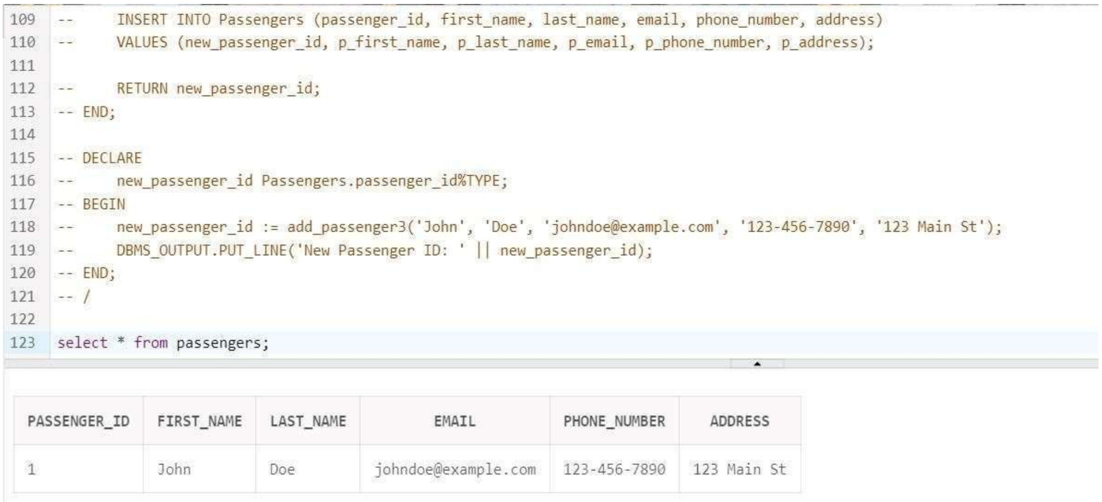
RETURN new_passenger_id;

END;



```

    INSERT INTO Passengers (passenger_id, first_name, last_name, email, phone_number, address)
    VALUES (new_passenger_id, p_first_name, p_last_name, p_email, p_phone_number, p_address);

    RETURN new_passenger_id;
END;

DECLARE
    new_passenger_id Passengers.passenger_id%TYPE;
BEGIN
    new_passenger_id := add_passenger3('John', 'Doe', 'johndoe@example.com', '123-456-7890', '123 Main St');
    DBMS_OUTPUT.PUT_LINE('New Passenger ID: ' || new_passenger_id);
END;
/

```

```
Statement processed.
New Passenger ID: 1
```



```
--     INSERT INTO Passengers (passenger_id, first_name, last_name, email, phone_number, address)
--     VALUES (new_passenger_id, p_first_name, p_last_name, p_email, p_phone_number, p_address);

--     RETURN new_passenger_id;
-- END;

-- DECLARE
--     new_passenger_id Passengers.passenger_id%TYPE;
-- BEGIN
--     new_passenger_id := add_passenger3('John', 'Doe', 'johndoe@example.com', '123-456-7890', '123 Main St');
--     DBMS_OUTPUT.PUT_LINE('New Passenger ID: ' || new_passenger_id);
-- END;
-- /

select * from passengers;
```

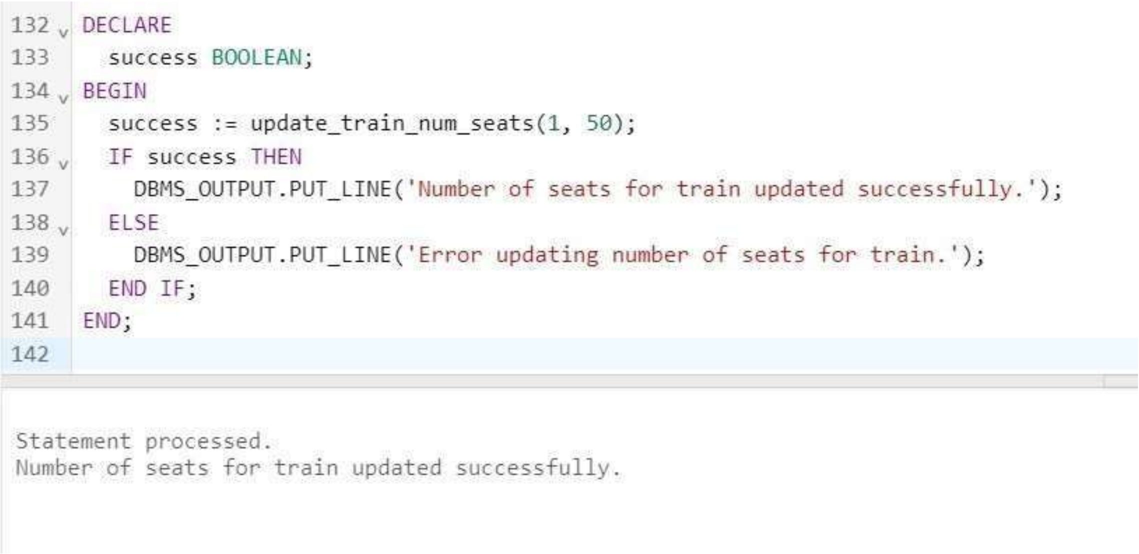
PASSENGER_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	ADDRESS
1	John	Doe	johndoe@example.com	123-456-7890	123 Main St

2. *CREATE OR REPLACE FUNCTION update_train_num_seats(train_id INT, new_num_seats INT) RETURN BOOLEAN AS BEGIN*
   UPDATE Trains

   SET num_seats = new_num_seats WHERE train_id = train_id;

   RETURN TRUE;

   END;



```
DECLARE
  success BOOLEAN;
BEGIN
  success := update_train_num_seats(1, 50);
  IF success THEN
    DBMS_OUTPUT.PUT_LINE('Number of seats for train updated successfully.');
  ELSE
    DBMS_OUTPUT.PUT_LINE('Error updating number of seats for train.');
  END IF;
END;
```

```
Statement processed.
Number of seats for train updated successfully.
```

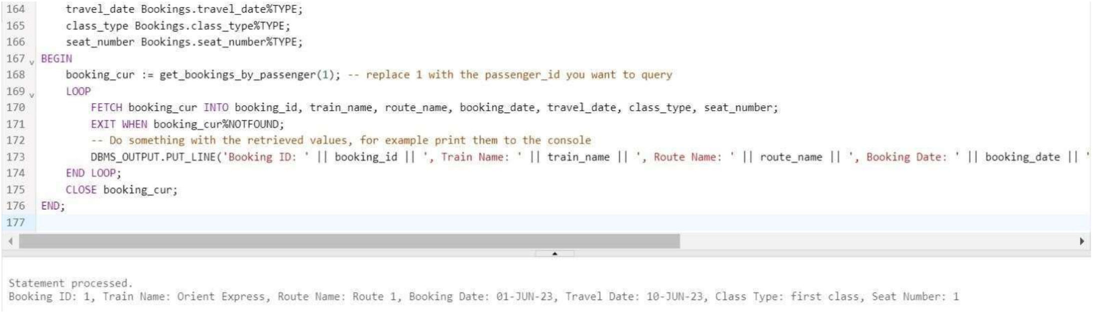
3. *CREATE OR REPLACE FUNCTION get_bookings_by_passenger(passenger_id INT) RETURN SYS_REFCURSOR AS booking_cur SYS_REFCURSOR;*
   BEGIN

   OPEN booking_cur FOR

   SELECT b.booking_id, t.train_name, r.route_name, b.booking_date, b.travel_date, b.class_type, b.seat_number FROM Bookings b JOIN Trains t ON b.train_id = t.train_id JOIN Routes r ON b.route_id = r.route_id WHERE b.passenger_id = passenger_id;

   RETURN booking_cur;

   END;

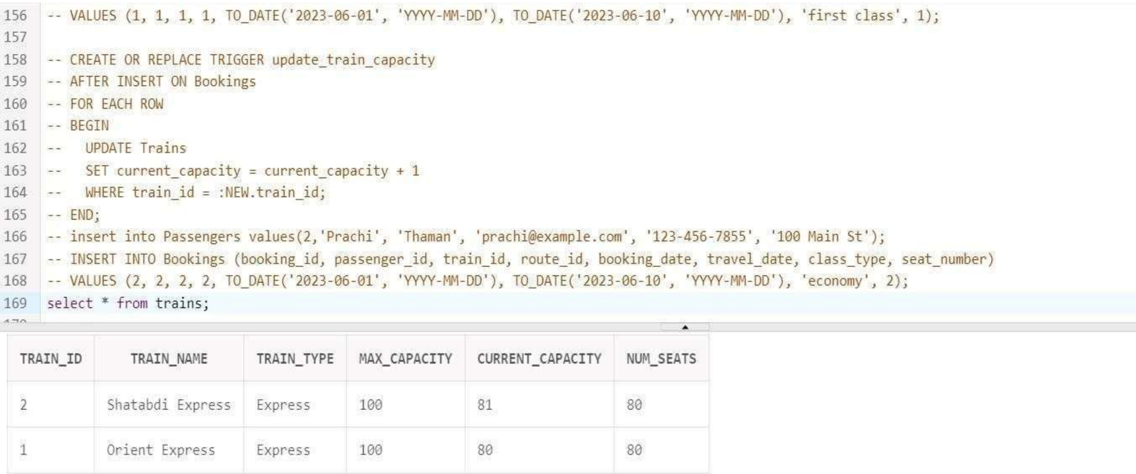


```
    travel_date Bookings.travel_date%TYPE;
    class_type Bookings.class_type%TYPE;
    seat_number Bookings.seat_number%TYPE;
BEGIN
    booking_cur := get_bookings_by_passenger(1); -- replace 1 with the passenger_id you want to query
    LOOP
        FETCH booking_cur INTO booking_id, train_name, route_name, booking_date, travel_date, class_type, seat_number;
        EXIT WHEN booking_cur%NOTFOUND;
        -- Do something with the retrieved values, for example print them to the console
        DBMS_OUTPUT.PUT_LINE('Booking ID: ' || booking_id || ', Train Name: ' || train_name || ', Route Name: ' || route_name || ', Booking Date: ' || booking_date || '
    END LOOP;
    CLOSE booking_cur;
END;
```

```
Statement processed.
Booking ID: 1, Train Name: Orient Express, Route Name: Route 1, Booking Date: 01-JUN-23, Travel Date: 10-JUN-23, Class Type: first class, Seat Number: 1
```

c) Triggers-

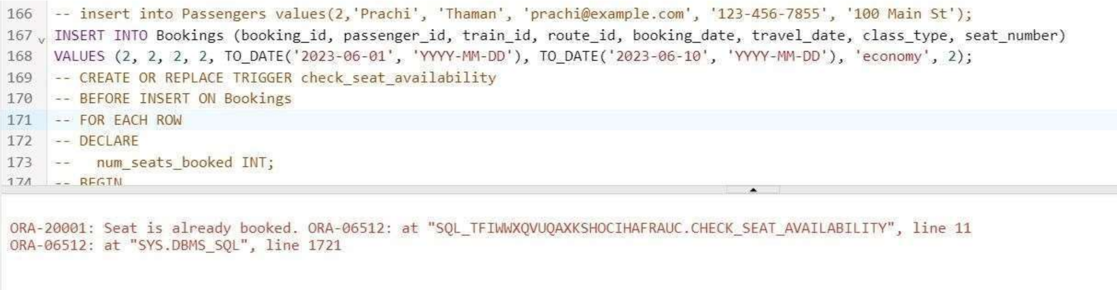
1.
   ```
   CREATE OR REPLACE TRIGGER update_train_capacity AFTER INSERT ON
   Bookings FOR EACH ROW
    BEGIN

    UPDATE Trains

    SET current_capacity = current_capacity + 1 WHERE train_id = :NEW.train_id;

    END;
   ```



```
-- VALUES (1, 1, 1, 1, TO_DATE('2023-06-01', 'YYYY-MM-DD'), TO_DATE('2023-06-10', 'YYYY-MM-DD'), 'first class', 1);

-- CREATE OR REPLACE TRIGGER update_train_capacity
-- AFTER INSERT ON Bookings
-- FOR EACH ROW
-- BEGIN
--   UPDATE Trains
--   SET current_capacity = current_capacity + 1
--   WHERE train_id = :NEW.train_id;
-- END;
-- insert into Passengers values(2,'Prachi', 'Thaman', 'prachi@example.com', '123-456-7855', '100 Main St');
-- INSERT INTO Bookings (booking_id, passenger_id, train_id, route_id, booking_date, travel_date, class_type, seat_number)
-- VALUES (2, 2, 2, 2, TO_DATE('2023-06-01', 'YYYY-MM-DD'), TO_DATE('2023-06-10', 'YYYY-MM-DD'), 'economy', 2);
select * from trains;
```

TRAIN_ID	TRAIN_NAME	TRAIN_TYPE	MAX_CAPACITY	CURRENT_CAPACITY	NUM_SEATS
2	Shatabdi Express	Express	100	81	80
1	Orient Express	Express	100	80	80

2.
   ```
   CREATE OR REPLACE TRIGGER check_seat_availability BEFORE INSERT ON
   Bookings FOR EACH ROW DECLARE num_seats_booked INT;
    BEGIN

   SELECT COUNT(*) INTO num_seats_booked FROM Bookings WHERE train_id =
   :NEW.train_id AND travel_date = :NEW.travel_date AND seat_number =
   :NEW.seat_number;

   IF num_seats_booked > 0 THEN

    RAISE_APPLICATION_ERROR(-20001, 'Seat is already booked.');

    END IF;

    END;
   ```



```
-- insert into Passengers values(2,'Prachi', 'Thaman', 'prachi@example.com', '123-456-7855', '100 Main St');
INSERT INTO Bookings (booking_id, passenger_id, train_id, route_id, booking_date, travel_date, class_type, seat_number)
VALUES (2, 2, 2, 2, TO_DATE('2023-06-01', 'YYYY-MM-DD'), TO_DATE('2023-06-10', 'YYYY-MM-DD'), 'economy', 2);
-- CREATE OR REPLACE TRIGGER check_seat_availability
-- BEFORE INSERT ON Bookings
-- FOR EACH ROW
-- DECLARE
--   num_seats_booked INT;
-- BEGIN
```

```
ORA-20001: Seat is already booked. ORA-06512: at "SQL_TFIWWXQVUQAXKSHOCIHAFRAUC.CHECK_SEAT_AVAILABILITY", line 11
ORA-06512: at "SYS.DBMS_SQL", line 1721
```

3. *CREATE OR REPLACE TRIGGER check_train_capacity BEFORE INSERT ON Bookings FOR EACH ROW DECLARE max_capacity INT;*
   current_capacity INT;

BEGIN

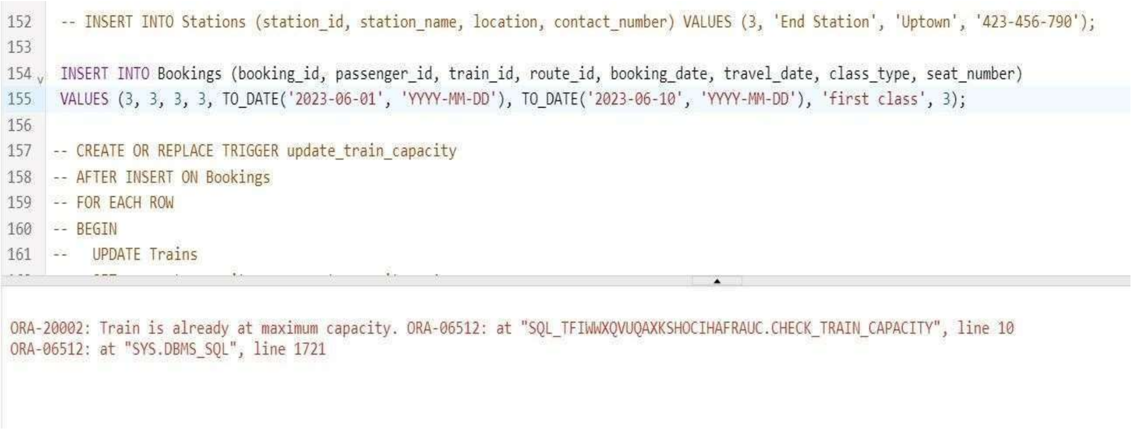
SELECT max_capacity, current_capacity INTO max_capacity, current_capacity FROM Trains WHERE train_id = :NEW.train_id;

IF current_capacity >= max_capacity THEN

RAISE_APPLICATION_ERROR(-20002, 'Train is already at maximum capacity.');

END IF;

END;



```
 -- INSERT INTO Stations (station_id, station_name, location, contact_number) VALUES (3, 'End Station', 'Uptown', '423-456-790');

 INSERT INTO Bookings (booking_id, passenger_id, train_id, route_id, booking_date, travel_date, class_type, seat_number)
 VALUES (3, 3, 3, 3, TO_DATE('2023-06-01', 'YYYY-MM-DD'), TO_DATE('2023-06-10', 'YYYY-MM-DD'), 'first class', 3);

-- CREATE OR REPLACE TRIGGER update_train_capacity
-- AFTER INSERT ON Bookings
-- FOR EACH ROW
-- BEGIN
--   UPDATE Trains
```

```
ORA-20002: Train is already at maximum capacity. ORA-06512: at "SQL_TFIWWXQVUQAXKSHOCIHAFRAUC.CHECK_TRAIN_CAPACITY", line 10
ORA-06512: at "SYS.DBMS_SQL", line 1721
```

d) Cursors-

1. *CREATE OR REPLACE PROCEDURE get_passengers_bookings IS CURSOR passengers_cursor IS*
   *SELECT * FROM Passengers;*

passenger Passengers%ROWTYPE;

*CURSOR bookings_cursor (p_id IN NUMBER) IS SELECT * FROM Bookings WHERE passenger_id = p_id;*

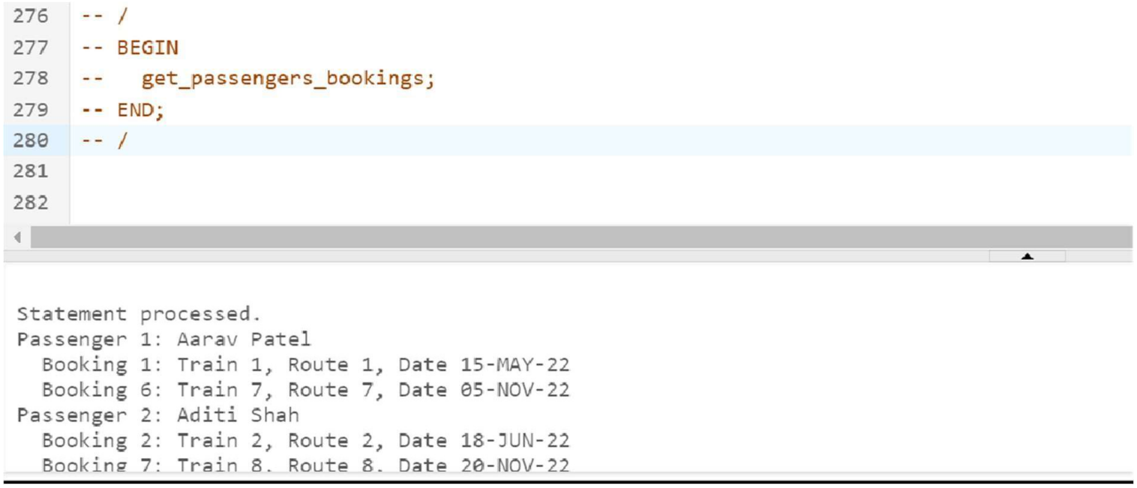
booking Bookings%ROWTYPE;

BEGIN

FOR passenger IN passengers_cursor LOOP DBMS_OUTPUT.PUT_LINE('Passenger ' || passenger.passenger_id || ': ' || passenger.first_name || ' ' || passenger.last_name);

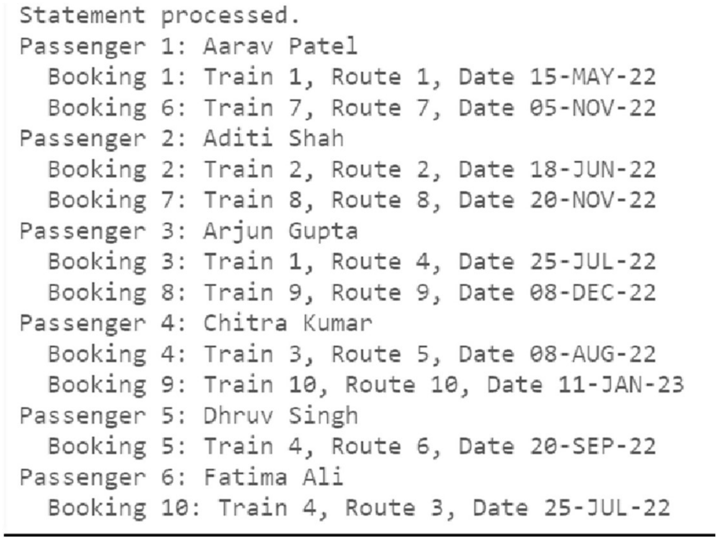
OPEN bookings_cursor(passenger.passenger_id);

```
    LOOP

    FETCH bookings_cursor  INTO  booking;

    EXIT WHEN bookings_cursor%NOTFOUND;

    DBMS_OUTPUT.PUT_LINE('      Booking ' || booking.booking_id || ': Train ' ||
booking.train_id || ', Route ' || booking.route_id || ', Date ' || booking.travel_date);

    END LOOP;

    CLOSE bookings_cursor;

  END LOOP;

  END;
```



```
-- /
-- BEGIN
--   get_passengers_bookings;
-- END;
-- /
```

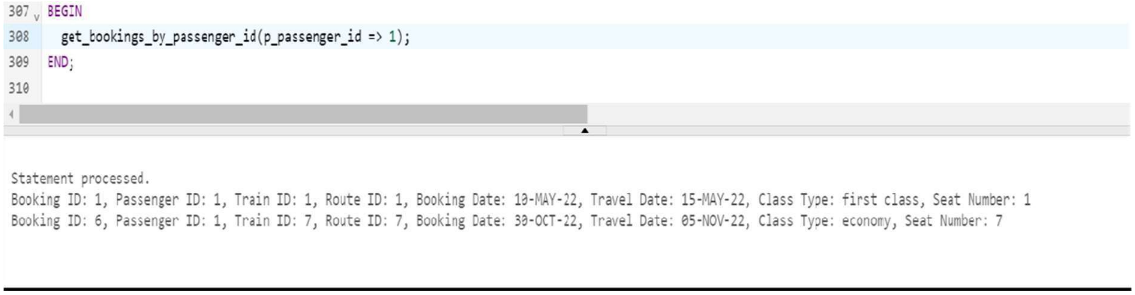
```
Statement processed.
Passenger 1: Aarav Patel
  Booking 1: Train 1, Route 1, Date 15-MAY-22
  Booking 6: Train 7, Route 7, Date 05-NOV-22
Passenger 2: Aditi Shah
  Booking 2: Train 2, Route 2, Date 18-JUN-22
  Booking 7: Train 8, Route 8, Date 20-NOV-22
```



```
Statement processed.
Passenger 1: Aarav Patel
  Booking 1: Train 1, Route 1, Date 15-MAY-22
  Booking 6: Train 7, Route 7, Date 05-NOV-22
Passenger 2: Aditi Shah
  Booking 2: Train 2, Route 2, Date 18-JUN-22
  Booking 7: Train 8, Route 8, Date 20-NOV-22
Passenger 3: Arjun Gupta
  Booking 3: Train 1, Route 4, Date 25-JUL-22
  Booking 8: Train 9, Route 9, Date 08-DEC-22
Passenger 4: Chitra Kumar
  Booking 4: Train 3, Route 5, Date 08-AUG-22
  Booking 9: Train 10, Route 10, Date 11-JAN-23
Passenger 5: Dhruv Singh
  Booking 5: Train 4, Route 6, Date 20-SEP-22
Passenger 6: Fatima Ali
  Booking 10: Train 4, Route 3, Date 25-JUL-22
```

2. CREATE OR REPLACE PROCEDURE get_bookings_by_passenger_id (p_passenger_id IN INT) IS CURSOR c_bookings IS SELECT * FROM bookings WHERE passenger_id = p_passenger_id;

```
  v_booking_id bookings.booking_id%TYPE;
 v_passenger_id bookings.passenger_id%TYPE;
 v_train_id  bookings.train_id%TYPE;
 v_route_id bookings.route_id%TYPE;
   v_booking_date bookings.booking_date%TYPE;
  v_travel_date bookings.travel_date%TYPE;
  v_class_type bookings.class_type%TYPE;
  v_seat_number bookings.seat_number%TYPE;
  BEGIN
 OPEN c_bookings;
 LOOP
 FETCH c_bookings INTO v_booking_id, v_passenger_id, v_train_id, v_route_id, v_booking_date, v_travel_date, v_class_type, v_seat_number;
 EXIT WHEN c_bookings%NOTFOUND;
DBMS_OUTPUT.PUT_LINE('Booking ID: ' || v_booking_id || ', Passenger ID: ' || v_passenger_id || ', Train ID: ' || v_train_id || ', Route ID: ' || v_route_id || ',  Booking Date: ' || v_booking_date || ', Travel Date: ' || v_travel_date || ', Class Type: ' || v_class_type || ', Seat Number: ' || v_seat_number);
 END LOOP;
CLOSE c_bookings;
 END;
```



```
BEGIN
  get_bookings_by_passenger_id(p_passenger_id => 1);
END;
```

```
Statement processed.
Booking ID: 1, Passenger ID: 1, Train ID: 1, Route ID: 1, Booking Date: 10-MAY-22, Travel Date: 15-MAY-22, Class Type: first class, Seat Number: 1
Booking ID: 6, Passenger ID: 1, Train ID: 7, Route ID: 7, Booking Date: 30-OCT-22, Travel Date: 05-NOV-22, Class Type: economy, Seat Number: 7
```

e) Exception Handling-

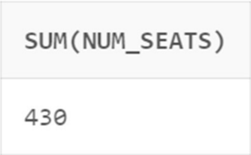
```
1. DECLARE
    -- declare variables

     v_passenger_name Passengers.first_name%TYPE;

     v_train_name Trains.train_name%TYPE;

     v_error_msg VARCHAR2(200);

   BEGIN

    -- get input values

    v_passenger_name := '&Enter_Passenger_Name';

    v_train_name := '&Enter_Train_Name';

      -- try to insert a new booking

    BEGIN

  INSERT INTO Bookings (passenger_id, train_id, route_id, booking_date, travel_date, class_type, seat_number) SELECT p.passenger_id, t.train_id, r.route_id, SYSDATE, SYSDATE + INTERVAL '1' DAY, 'economy', 1 FROM Passengers p JOIN Trains t ON t.train_name = v_train_name JOIN Routes r ON r.origin_station = 'Station A' AND r.destination_station = 'Station B' WHERE p.first_name = v_passenger_name AND t.current_capacity < t.max_capacity AND NOT EXISTS (SELECT 1 FROM Bookings b WHERE b.train_id = t.train_id AND b.travel_date = SYSDATE +  INTERVAL  '1' DAY AND b.seat_number = 1 );

DBMS_OUTPUT.PUT_LINE('Booking created successfully.');

EXCEPTION WHEN  NO_DATA_FOUND  THEN  v_error_msg := 'Passenger or train
not found.';

 DBMS_OUTPUT.PUT_LINE(v_error_msg);

 WHEN TOO_MANY_ROWS THEN v_error_msg := 'Multiple passengers or trains
found.';

 DBMS_OUTPUT.PUT_LINE(v_error_msg);

 WHEN OTHERS THEN v_error_msg := 'An error occurred while creating booking.';

 DBMS_OUTPUT.PUT_LINE(v_error_msg);

  END;

END;
```

```
2. DECLARE
  p_id Passengers.passenger_id%TYPE := 100; -- example passenger ID
p_name Passengers.first_name%TYPE := 'John'; -- example passenger name
BEGIN
  INSERT INTO Passengers (passenger_id, first_name, last_name, email, phone_number, address) VALUES (p_id, p_name, 'Doe', 'johndoe@example.com', '555-1234', '123 Main St');
EXCEPTION WHEN OTHERS THEN DBMS_OUTPUT.PUT_LINE('Error: ' || SQLCODE || ' - ' || SQLERRM);
END;
```

SQL Queries:

1. Retrieve the total number of seats for all trains in the Trains table:

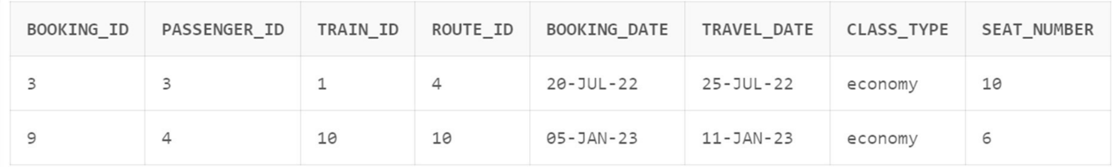
SELECT SUM(num_seats) FROM Trains;



SUM(NUM_SEATS)
430

2. Retrieve the passenger IDs and booking dates for all bookings in the Bookings table that have a class type of 'economy':

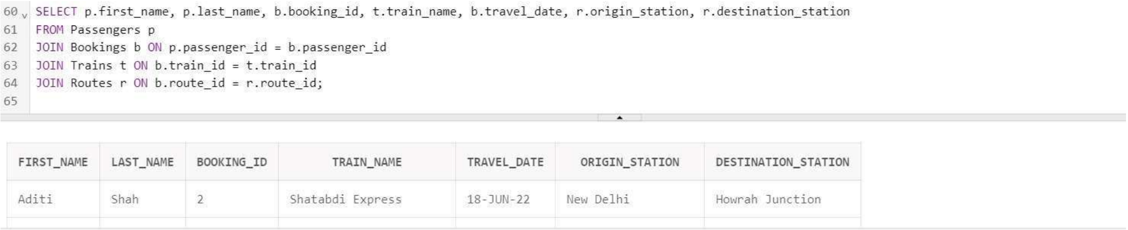
select * from bookings where class_type='economy';



BOOKING_ID	PASSENGER_ID	TRAIN_ID	ROUTE_ID	BOOKING_DATE	TRAVEL_DATE	CLASS_TYPE	SEAT_NUMBER
3	3	1	4	20-JUL-22	25-JUL-22	economy	10
9	4	10	10	05-JAN-23	11-JAN-23	economy	6

3. Retrieve the passenger names and their corresponding bookings, including the train name, travel date, and origin and destination stations:

SELECT p.first_name, p.last_name, b.booking_id, t.train_name, b.travel_date, r.origin_station, r.destination_station FROM Passengers p JOIN Bookings b ON p.passenger_id = b.passenger_id JOIN Trains t ON b.train_id = t.train_id JOIN Routes r ON b.route_id = r.route_id;

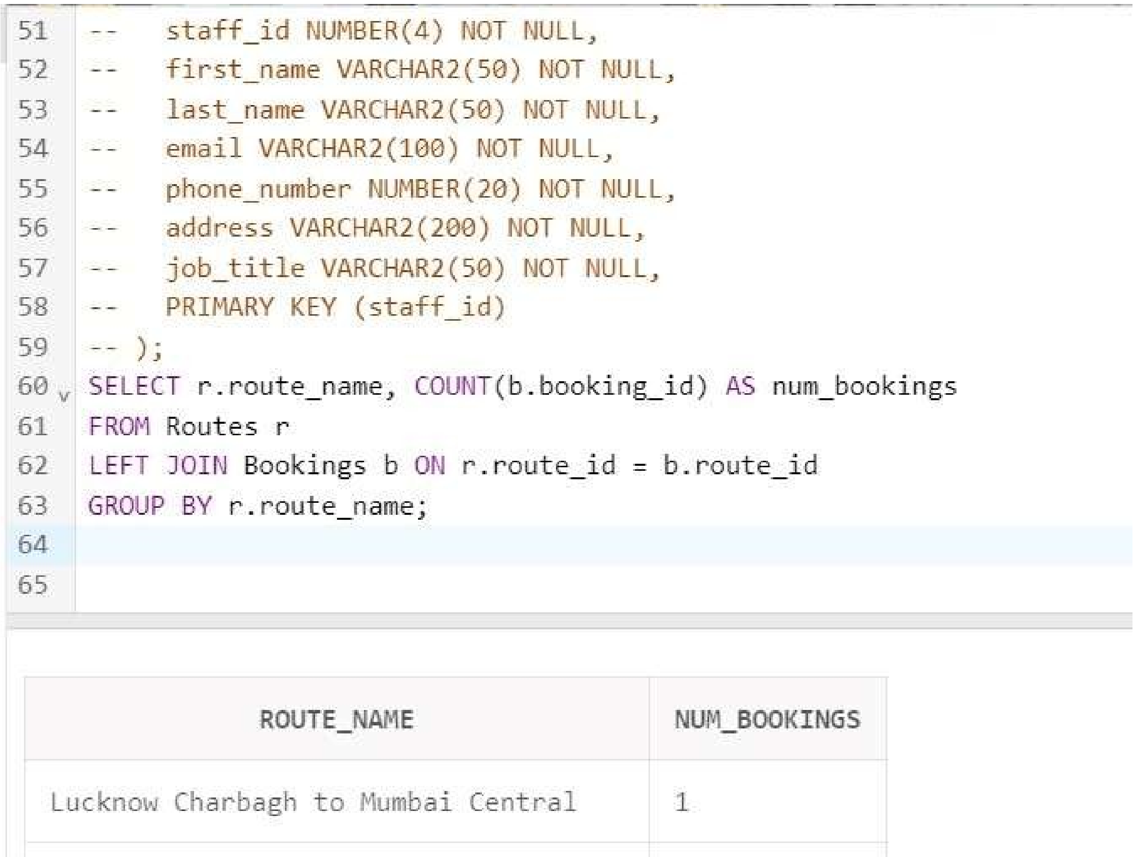


```
SELECT p.first_name, p.last_name, b.booking_id, t.train_name, b.travel_date, r.origin_station, r.destination_station
FROM Passengers p
JOIN Bookings b ON p.passenger_id = b.passenger_id
JOIN Trains t ON b.train_id = t.train_id
JOIN Routes r ON b.route_id = r.route_id;
```

FIRST_NAME	LAST_NAME	BOOKING_ID	TRAIN_NAME	TRAVEL_DATE	ORIGIN_STATION	DESTINATION_STATION
Aditi	Shah	2	Shatabdi Express	18-JUN-22	New Delhi	Howrah Junction

4. **Retrieve the route names and the number of bookings for each route:**

SELECT r.route_name, COUNT(b.booking_id) AS num_bookings
FROM Routes r
LEFT JOIN Bookings b ON r.route_id = b.route_id
GROUP BY r.route_name;

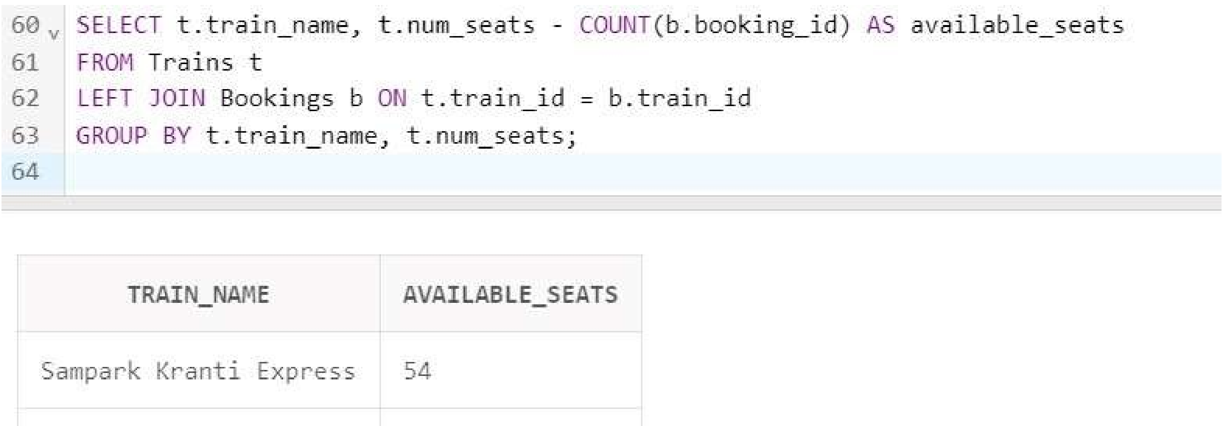


```
--   staff_id NUMBER(4) NOT NULL,
--   first_name VARCHAR2(50) NOT NULL,
--   last_name VARCHAR2(50) NOT NULL,
--   email VARCHAR2(100) NOT NULL,
--   phone_number NUMBER(20) NOT NULL,
--   address VARCHAR2(200) NOT NULL,
--   job_title VARCHAR2(50) NOT NULL,
--   PRIMARY KEY (staff_id)
-- );
SELECT r.route_name, COUNT(b.booking_id) AS num_bookings
FROM Routes r
LEFT JOIN Bookings b ON r.route_id = b.route_id
GROUP BY r.route_name;
```

ROUTE_NAME	NUM_BOOKINGS
Lucknow Charbagh to Mumbai Central	1

5. **Retrieve the train names and the number of available seats for each train:**

SELECT t.train_name, t.num_seats - COUNT(b.booking_id) AS available_seats
FROM Trains t
LEFT JOIN Bookings b ON t.train_id = b.train_id
GROUP BY t.train_name, t.num_seats;



```
SELECT t.train_name, t.num_seats - COUNT(b.booking_id) AS available_seats
FROM Trains t
LEFT JOIN Bookings b ON t.train_id = b.train_id
GROUP BY t.train_name, t.num_seats;
```

TRAIN_NAME	AVAILABLE_SEATS
Sampark Kranti Express	54

6. Conclusion:

In conclusion, our railway system management project using a database management system (DBMS) has been successfully implemented, providing a robust and efficient solution for managing the operations of a railway system. The project involves designing and implementing a database schema to represent the various entities involved in the system, including trains, stations, routes, and passengers.

The database management system used in the project allowed for efficient storage and retrieval of data, as well as effective management of the relationships between the different entities. The project also involved developing a user interface that allowed for easy and intuitive interaction with the database, facilitating tasks such as scheduling trains, booking tickets, and managing passenger information.

Overall, the railway system management project using a DBMS has demonstrated the importance and effectiveness of using modern database technologies in managing complex systems such as a railway network. The project provides a solid foundation for future development and expansion of the railway system, allowing for continued improvement in the efficiency, safety, and quality of service provided to passengers.

7. References:

1. W. Connolly and C. Begg, "Database Systems: A Practical Approach to Design, Implementation, and Management," 6th Edition, Pearson Education, 2014.

2. E. F. Codd, "A Relational Model of Data for Large Shared Data Banks," Communications of the ACM, vol. 13, no. 6, pp. 377-387, 1970.

3. A. Wahi, M. Shekhar, and A. Chen, "Railway operations and management: a review of current practices and future challenges," Journal of Intelligent Transportation Systems, vol. 22, no. 4, pp. 301-323, 2018.

4. S. M. Subhani and R. Shinde, "Railway reservation and management system using database management system," International Journal of Scientific Research in Computer Science, Engineering and Information Technology, vol. 4, no. 4, pp. 1094-1099, 2019.

Signature of Faculty