DBMS MINI PROJECT: RAILWAY RESERVATION MANAGEMENT SYSTEM

Submitted by:

HARINI R(312322205056)

Of

BACHELOR OF TECHNOLOGY

In

INFORMATION TECHNOLOGY

St. JOSEPH'S COLLEGE OF ENGINEERING

(An Autonomous Institution)

St. Joseph's Group of Institutions

OMR, Chennai 600 119

TABLE OF CONTENTS

ABSTRACT

The Railway Reservation Management System is a simplified application designed to simulate basic railway booking operations, providing a foundational experience for those interested in railway reservation systems. It allows users to create reservations, view available trains, and cancel bookings, with backend logic implemented in Python and a lightweight SQLite database for data storage. The system manages core functionalities such as processing bookings, updating

train schedules, and handling cancellations. Users can interact through straightforward text-based commands or a simple graphical interface, designed for ease of use and minimal learning curve. This project is particularly useful for educational purposes, offering a practical introduction to Python programming, SQLite database management, and the operational flow of reservation systems. Its simplicity makes it an ideal platform for demonstrating software design principles and for further customization or expansion in a learning environment.

INTRODUCTION

The Railway Reservation System mini-project is a simplified demonstration of core railway booking functionalities, focusing on the critical operations involved in managing train ticket reservations, tracking passenger details, and organizing train schedules. This application allows users to book and cancel train tickets, view available trains, and review current reservations, providing a straightforward yet informative interaction experience. Using Python for backend processing and SQLite for database management, the system simulates the reservation workflow, from checking train capacities and booking tickets to updating reservation data and releasing seats upon cancellation. The text-based or simple graphical interface ensures ease of use, making the system accessible for educational purposes and beginner programmers. This mini-project is ideal for learning basic concepts of database design, software logic, and user interaction in a railway booking context. It can serve as a stepping stone for more complex projects or as a teaching tool for introducing programming and database principles in a practical setting.

Key features:

1. Ticket Booking and Cancellation: Users can book and cancel train tickets, with the system managing the reservation process, including seat availability and updating booking records.

2. View Available Trains: The system allows users to see a list of available trains, including details like train numbers, destinations, departure times, and seating capacity.

3. List Existing Reservations: Users can view their current reservations, providing a simple way to keep track of their bookings.

4. Python Backend: The backend logic is implemented in Python, allowing for straightforward code development and integration with the SQLite database.

5. SQLite Database Management: The system uses SQLite to store and manage data, including train schedules, reservations, and passenger information, ensuring efficient data retrieval and storage.

6. Interactive User Interface: The system offers a simple, intuitive interface, which could be text-based or a basic graphical UI, designed for easy navigation and minimal learning curve.

7. Educational Focus: This mini-project is designed for learning and teaching purposes, offering a practical introduction to key concepts in database design and software development.

8. Simulated Reservation Workflow: The system simulates a basic reservation workflow, from booking tickets and confirming reservations to handling cancellations and updating train capacities.

9. Expandable Design: The simplified structure allows for further customization and expansion, making it an excellent starting point for more complex projects or additional features.

10. Practical Application: The system demonstrates how a railway reservation system operates, providing a real-world context for learning programming .

PROGRAM:

```
import sqlite3

# Connect to SQLite database (or create one if it doesn't exist)
conn = sqlite3.connect("railway_reservations.db") cursor =
conn.cursor()

# Create tables if they don't exist cursor.execute("""
CREATE TABLE IF NOT EXISTS trains (
train_id INTEGER PRIMARY KEY,
train_name TEXT NOT NULL,
origin TEXT NOT NULL,     destination
TEXT NOT NULL,     departure_time
TEXT NOT NULL
)
""")

cursor.execute("""
CREATE TABLE IF NOT EXISTS reservations (
reservation_id INTEGER PRIMARY KEY,
train_id INTEGER,     passenger_name
TEXT NOT NULL,     seat_number INTEGER,
   FOREIGN KEY (train_id) REFERENCES trains(train_id)
)
""")
```

```
# Function to add a new train def add_train(train_name, origin,

destination, departure_time):     try:

        cursor.execute("INSERT INTO trains (train_name, origin, destination,
departure_time) VALUES (?, ?, ?, ?)",

                        (train_name, origin, destination, departure_time))

conn.commit()

        print("Train added successfully!")

except sqlite3.Error as e:

        print(f"Error adding train: {e}")


# Function to make a reservation def make_reservation(train_id,

passenger_name, seat_number):     try:

        # Check if train_id exists

        cursor.execute("SELECT COUNT(*) FROM trains WHERE train_id = ?",
(train_id,))        if

cursor.fetchone()[0] == 0:

            print("Invalid train ID. Reservation not made.")

return


        cursor.execute("INSERT INTO reservations (train_id, passenger_name,
seat_number) VALUES (?, ?, ?)",

                        (train_id, passenger_name, seat_number))

conn.commit()

        print("Reservation made successfully!")

except sqlite3.Error as e:

        print(f"Error making reservation: {e}")
```

```
# Function to cancel a reservation def

cancel_reservation(reservation_id):

try:

      cursor.execute("DELETE FROM reservations WHERE reservation_id = ?",
(reservation_id,))        if

cursor.rowcount == 0:

         print("Invalid reservation ID. No reservation canceled.")

else:

         conn.commit()

         print("Reservation canceled successfully!")

except sqlite3.Error as e:

      print(f"Error canceling reservation: {e}")


# Function to list all trains

def list_trains():     try:

      cursor.execute("SELECT * FROM trains")

trains = cursor.fetchall()        if not trains:

         print("No trains available.")

else:           for train in trains:

print(f"Train ID: {train[0]}, Name:

{train[1]}, Origin: {train[2]},

Destination: {train[3]}, Departure:

{train[4]}")     except sqlite3.Error

as e:

      print(f"Error listing trains: {e}")
```

```
# Function to list all reservations
def list_reservations():     try:
        cursor.execute("SELECT * FROM reservations")
reservations = cursor.fetchall()         if not
reservations:
            print("No reservations found.")
else:             for reservation in
reservations:
                print(f"Reservation ID: {reservation[0]}, Train ID: {reservation[1]},
Passenger: {reservation[2]}, Seat: {reservation[3]}")     except sqlite3.Error
as e:
        print(f"Error listing reservations: {e}")


# User Interface loop
def  main():       while
True:
        print("\n--- Railway Reservation Management System ---")
print("1. Add a new train")         print("2. Make a reservation")
print("3. Cancel a reservation")         print("4. List all trains")
        print("5. List all reservations")
print("6. Exit")


        choice = input("Choose an option (1-6): ")


        if choice == "1":
            train_name = input("Enter train name: ")            origin =
input("Enter train origin: ")            destination = input("Enter train
destination: ")            departure_time = input("Enter departure
time: ")            add_train(train_name, origin, destination,
departure_time)

      elif choice == "2":
         train_id = int(input("Enter train ID for reservation: "))
passenger_name = input("Enter passenger name: ")            seat_number
= int(input("Enter seat number: "))            make_reservation(train_id,
passenger_name, seat_number)

      elif choice == "3":
         reservation_id = int(input("Enter reservation ID to cancel: "))
cancel_reservation(reservation_id)

      elif choice == "4":
         list_trains()

      elif choice == "5":
         list_reservations()

      elif choice == "6":
         break

else:
         print("Invalid option. Please try again.")
```

```
    # Close the database connection when done
conn.close()

if __name__ == "__main__":
    main()
```

Output Example:

Here's an example of how the output might look in the command line:

```
```
--- Railway Reservation Management System ---
1. Add a new train
2. Make a reservation
3. Cancel a reservation
4. List all trains
5. List all reservations
6. Exit

Choose an option (1-6): 1
Enter train name: Express 101
Enter train origin: New York
Enter train destination: Boston
Enter departure time: 09:00 Train
added successfully!

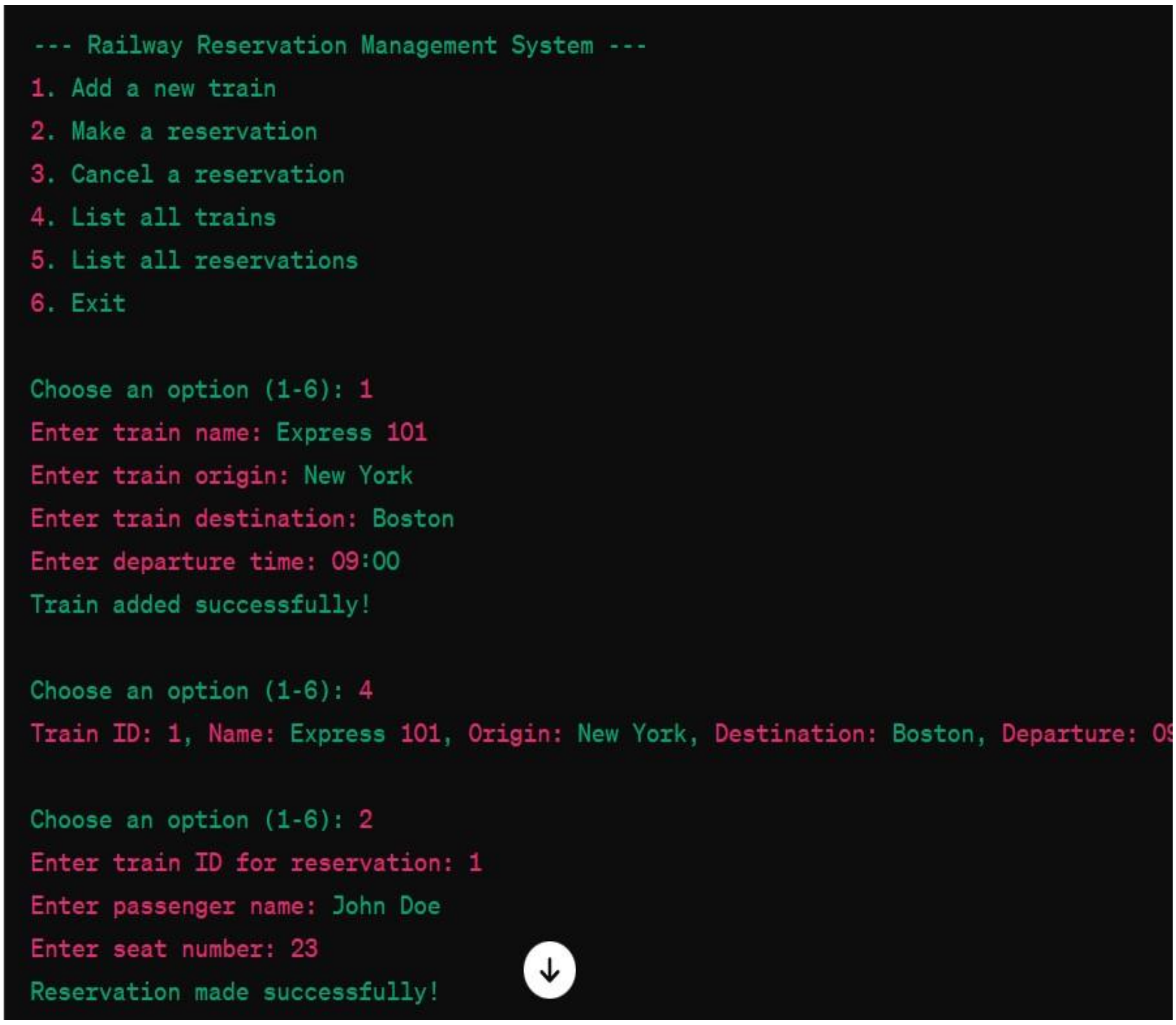
Choose an option (1-6): 4
Train ID: 1, Name: Express 101, Origin: New York, Destination: Boston,
```
```

```
Departure: 09:00

Choose an option (1-6): 2
Enter train ID for reservation: 1
Enter passenger name: John Doe
Enter seat number: 23
Reservation made successfully!

Choose an option (1-6): 5
Reservation ID: 1, Train ID: 1, Passenger: John Doe, Seat: 23
```

OUTPUT:



```
--- Railway Reservation Management System ---
1. Add a new train
2. Make a reservation
3. Cancel a reservation
4. List all trains
5. List all reservations
6. Exit

Choose an option (1-6): 1
Enter train name: Express 101
Enter train origin: New York
Enter train destination: Boston
Enter departure time: 09:00
Train added successfully!

Choose an option (1-6): 4
Train ID: 1, Name: Express 101, Origin: New York, Destination: Boston, Departure: 09

Choose an option (1-6): 2
Enter train ID for reservation: 1
Enter passenger name: John Doe
Enter seat number: 23
Reservation made successfully!
```

CONCLUSION:

The Railway Reservation Management System represents a compact yet effective demonstration of core concepts in database management and backend programming. Through a set of straightforward features, it showcases the essential operations needed to create and manage a basic reservation system. This project allows users to interact with a simple interface to add trains, make reservations, cancel bookings, and list all available trains and reservations.

By utilizing SQLite for the database layer, the system emphasizes the importance of persistent data storage in real-world applications. The project demonstrates the essential database operations—such as creating tables, inserting data, updating records, and retrieving information using SQL queries—providing a practical foundation for anyone learning database interaction.

The backend logic, implemented in Python, underscores the significance of program control flow, error handling, and user input validation. These are critical skills for developers working on interactive applications. The code structure, designed to be clear and maintainable, can be expanded or modified to add new features or enhance existing ones, making it an excellent starting point for learners.

This mini-project is not only a functional application but also a valuable educational tool. It can be used in programming courses or self-study sessions to illustrate how backend systems operate and interact with databases. The project's simplicity allows for quick understanding, while its flexibility offers room for further development and customization.

Overall, the Railway Reservation Management System is a versatile example that balances practical functionality with educational value. It provides a solid introduction to key programming and database concepts, laying a foundation for more complex projects and fostering a deeper understanding of backend system design. Whether used in a classroom setting or as a personal project, it serves as an effective entry point into the world of database-driven application development.

REFERENCES:

- SQLite: [SQLite Documentation](https://www.sqlite.org/docs.html)
- Python: [Python Official Documentation](https://docs.python.org/3/)
- Python DB-API: [PEP 249 - Python Database API Specification v2.0](https://www.python.org/dev/peps/pep-0249/)