Student Name: ABHISHEK KUMAR **UID:24MCC20006**
Branch: MCA-CCD **Section/Group:1-A**
Semester: 1st **Date of Performance:30-10-24**
Subject Name: PL/SQL LAB. **Subject Code: 24CAP-602**

1.Aim:
The aim of this project is to develop a comprehensive Travel Database Manager using Python and SQLite, enabling users to effectively manage their travel itineraries. The application allows users to add, view, search, and delete trips while maintaining data integrity and offering a user-friendly interface.

2.Task to be Done:

 Database Design: Create an SQLite database and design a schema for storing trip information.

 Implement CRUD Operations: Develop functions to Create, Read, Update, and Delete trip records.

 User Interface: Build a command-line interface that allows users to interact with the database easily.

 Data Validation: Ensure proper validation for user inputs to maintain data integrity.

 Documentation: Comment the code and create a README file for instructions on usage.

3.Steps:

 Set Up Environment:

- Ensure Python is installed on your system. If not, download and install the latest version of Python.
- Install SQLite3 (usually comes bundled with Python).

 Create Database:

- Define a database schema that includes a trips table with relevant fields like id, destination, date, and budget.

 Define Functions:

- Implement functions for adding, viewing, searching, and deleting trips.
- Include error handling to manage incorrect inputs.

 User Interface Development:

- Create a menu-driven interface to facilitate user interaction with the program.

 Testing:

- Test each functionality to ensure that the application behaves as expected under various scenarios.

 Documentation:

- Write clear comments in the code and provide a README file explaining how to set up and run the application.

4.Code for experiment:

```
import sqlite3

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

# Create the 'trips' table if it doesn't exist
cursor.execute('''
CREATE TABLE IF NOT EXISTS trips (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    destination TEXT NOT NULL,
    date TEXT NOT NULL,
    budget REAL
)
''')
conn.commit()

def add_trip(destination, date, budget):
    cursor.execute('INSERT INTO trips (destination, date, budget) VALUES (?, ?, ?)',
                   (destination, date, budget))
    conn.commit()
    print(f"Trip to {destination} added successfully.")

def view_trips():
    cursor.execute('SELECT * FROM trips')
    trips = cursor.fetchall()
    if trips:
        for trip in trips:
            print(f"ID: {trip[0]}, Destination: {trip[1]}, Date: {trip[2]}, Budget: {trip[3]}")
    else:
        print("No trips found.")

def search_trip(destination):
    cursor.execute('SELECT * FROM trips WHERE destination LIKE ?', (f'%{destination}%',))
    trips = cursor.fetchall()
    if trips:
        for trip in trips:
            print(f"ID: {trip[0]}, Destination: {trip[1]}, Date: {trip[2]}, Budget: {trip[3]}")
    else:
        print(f"No trips found for destination '{destination}'.")
```

```
def delete_trip(trip_id):
    cursor.execute('DELETE FROM trips WHERE id = ?', (trip_id,))
    conn.commit()
    print(f"Trip with ID {trip_id} deleted.")

def show_menu():
    print("\nTravel Database Manager")
    print("1. Add a Trip")
    print("2. View All Trips")
    print("3. Search for a Trip")
    print("4. Delete a Trip")
    print("5. Exit")

while True:
    show_menu()
    choice = input("Enter your choice: ")

    if choice == '1':
        destination = input("Enter destination: ")
        date = input("Enter travel date (YYYY-MM-DD): ")
        budget = float(input("Enter budget: "))
        add_trip(destination, date, budget)

    elif choice == '2':
        view_trips()

    elif choice == '3':
        destination = input("Enter destination to search: ")
        search_trip(destination)

    elif choice == '4':
        trip_id = int(input("Enter trip ID to delete: "))
        delete_trip(trip_id)

    elif choice == '5':
        print("Exiting... Goodbye!")
        break

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

```
# Close the connection
conn.close()
```

5.Output: The output of the application varies based on user interactions. Here are somne output:-

```
Travel Database Manager
1. Add a Trip
2. View All Trips
3. Search for a Trip
4. Delete a Trip
5. Exit
Enter your choice:  1
Enter destination:  kedarnath
Enter travel date (YYYY-MM-DD):  2025-06-06
Enter budget:  10000
Trip to kedarnath added successfully.

Travel Database Manager
1. Add a Trip
2. View All Trips
3. Search for a Trip
4. Delete a Trip
5. Exit
Enter your choice:  1
Enter destination:  ujjain
Enter travel date (YYYY-MM-DD):  2024-10-30
Enter budget:  5000
Trip to ujjain added successfully.

Travel Database Manager
1. Add a Trip
2. View All Trips
3. Search for a Trip
4. Delete a Trip
5. Exit
Enter your choice:  3
Enter destination to search:  kedarnath
ID: 1, Destination: kedarnath, Date: 2025-06-06, Budget: 10000.0
ID: 2, Destination: kedarnath, Date: 2025-06-06, Budget: 10000.0

Travel Database Manager
1. Add a Trip
2. View All Trips
3. Search for a Trip
4. Delete a Trip
5. Exit
Enter your choice:  2
ID: 1, Destination: kedarnath, Date: 2025-06-06, Budget: 10000.0
```

```
ID: 2, Destination: kedarnath, Date: 2025-06-06, Budget: 10000.0

Travel Database Manager
1. Add a Trip
2. View All Trips
3. Search for a Trip
4. Delete a Trip
5. Exit
Enter your choice:  5
Exiting... Goodbye!
```

6.Conclusion:

The Travel Database Manager is a functional and user-friendly application that allows individuals to manage their travel plans effectively. By utilizing Python and SQLite, the project demonstrates the ease of CRUD operations on a database, providing a practical solution for tracking trips.

7.Learning Outcomes:

 Database Management: Gained experience in creating and managing a SQLite database using Python.

 CRUD Operations: Learned to implement CRUD operations effectively in an application.

 User Input Validation: Developed skills in validating user inputs to ensure data integrity.

 Command-Line Interfaces: Improved proficiency in building a command-line interface for user interaction.

 Error Handling: Enhanced ability to handle errors and exceptions in user inputs.

 Code Documentation: Learned the importance of commenting code and creating documentation for clarity and maintenance.

Teacher's Signature