AIRLINE RESERVATION SYSTEM

A PROJECT REPORT

Submitted by
G.MIDHUN CHAKRAVARTHY YADAV
[Reg. No. RA2211003011289]
T. SRI RANGASUTHAN [Reg. No. RA2211003011319]
M.VISHNUPRASANTH [Reg. No. RA2211003011326]

Under the Guidance of

Dr. R. Jebakumar

AssociateProfessor,DepartmentofComputingTechnologies

in partial fulfillment of the requirements for the degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

DEPARTMENT OF COMPUTING TECHNOLOGIES

COLLEGE OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR– 603 203

MAY 2024

SRM

DEPARTMENT OF COMPUTING TECHNOLOGIES
SCHOOL OF COMPUTING
COLLEGE OF ENGINEERING AND TECHNOLOGY
SRM INSTITUTE OF SCIENCE AND TECHNOLOGY
KATTANKULATHUR–603 203

BONAFIDE CERTIFICATE

Certified that this project report "**AIRLINE RESERVATION SYSTEM**" is the bonafide work of **G.MIDHUN CHAKRAVARTHY YADAV [Reg. No. RA2211003011289],T. SRI RANGASUTHAN [Reg. No. RA2211003011319] and M.VISHNUPRASANTH [Reg. No. RA2211003011326]** who carried out the project work under my supervision in the project course **Database Management Systems [21CSC205P]** forthe academic year 2023-2024 [II year / IV semester].

Date: 6/5/2024

6/5/24

Faculty in Charge
Dr. R. Jebakumar
Assistant Professor
Department of Computing Technologies
SRMSIT -KTR

Department of Computing Technology
Kattankulathur
603 203
SRM IST

M. Pushpalatha

HEAD OF THE DEPARTMENT
Dr. PUSHPALATHA M
Professor & Head
Department of Computing Technologies
SRMIST - KTR

ACKNOWLEDGEMENTS

We express our heartfelt thanks to our honorable **Vice Chancellor Dr. C. MUTHAMIZHCHELVAN**, for being the beacon in all our endeavors.

We would like to express my warmth of gratitude to our **Registrar Dr. S. Ponnusamy,** for his encouragement

We express our profound gratitude to our **Dean (College of Engineering and Technology) Dr. T. V.Gopal,** for bringing out novelty in all executions.

We would like to express my heartfelt thanks to Chairperson, School of Computing **Dr. Revathi Venkataraman,** for imparting confidence to complete my course project

We wish to express my sincere thanks to **Course Audit Professor Dr. C. Lakshmi, Professor, Department of Computational Intelligence** and **Course Coordinators** fortheir constant encouragement and support.

We are highly thankful to our my Course Faculty **Dr. R. Jebakumar, Associate Professor, Department of Computing Technologies,** for hisassistance, timely suggestion and guidance throughout the duration of this course project.

We extend my gratitude to our **HoD Dr. M. Pushpalatha, Professor, Department of Computing Technologies** and my Departmental colleagues for their Support.

Finally, we thank our parents and friends near and dear ones who directly and indirectly contributed to the successful completion of our project. Above all, I thank the almighty for showering his blessings on me to complete my Course project.

G.MIDHUN CHAKRAVARTHY YADAV [Reg. No. RA2211003011289]
T. SRI RANGASUTHAN [Reg. No. RA2211003011319]
M.VISHNUPRASANTH [Reg. No. RA2211003011326]

ABSTRACT

The Airline Reservation System (ARS) is a crucial component in the aviation industry, streamlining the process of flight booking, management, and passenger services. This project focuses on designing and implementing a comprehensive database management system for an ARS, encompassing the storage, retrieval, and manipulation of critical data related to airlines, flights, airports, passengers, bookings, seats, and payments.

The database design begins with the identification of key entities and their attributes, establishing relationships, and defining constraints to ensure data integrity. Implementation involves selecting a suitable database management system, creating tables, and populating them with sample data. The system facilitates CRUD operations, allowing users to add, retrieve, update, and delete information about airlines, flights, airports, passengers, bookings, seats, and payments.

A user-friendly interface is developed to enable seamless interaction with the database, providing functionalities such as flight search, booking, cancellation, and payment processing. Authentication and authorization mechanisms ensure secure access to sensitive data, while encryption safeguards privacy and confidentiality.

Testing and validation procedures validate system functionality, performance, and reliability, with emphasis on handling edge cases and stress testing. Deployment on a suitable platform and ongoing maintenance ensure the system's availability, scalability, and security.

TABLE OF CONTENTS

LIST OF FIGURES

Chapter-1

PROBLEM STATEMENT

Design and develop a robust airline reservation system that efficiently manages flight bookings, passenger information, and seat availability to streamline the process of booking flights for travelers. The system should provide a user-friendly interface for both passengers and administrators, ensuring secure transactions, accurate information retrieval, and real-time updates on flight schedules and availability. Additionally, the system should incorporate features such as seat selection, payment processing, booking modifications, and cancellations while maintaining data integrity and confidentiality. The goal is to create a seamless and reliable platform that enhances the overall experience for both passengers and airline staff, improving efficiency and customer satisfaction in the airline reservation process.

Chapter-2

Problem understanding, Identification of Entity and Relationships, Construction of DB using ER Model for the project

2.1 Problem understanding

The project involves developing an airline reservation system to facilitate booking flights for passengers. The system needs to manage various aspects such as flights, airports, bookings, passengers, payments, and seating arrangements.

Airline Management: The system should allow the management of different airlines, including their names and contact information.

Airport Management: It should handle information about airports, including their codes, names, and locations.

Booking Management: The system needs to facilitate the booking of flights by passengers. This includes tracking booking IDs, passenger IDs, flight numbers, booking dates, and payment statuses.

Flight Management: It should manage information related to flights, such as flight numbers, departure times, arrival times, aircraft types, and airline associations.

Flight Route Management: This involves defining routes for flights, including departure and arrival airports for each flight.

Passenger Management: The system should maintain passenger details like names and contact information.

Payment Management: It needs to handle payments made for bookings, including payment IDs, booking IDs, payment amounts, and payment methods.

Seating Management: This involves managing seat arrangements for flights, including seat numbers, flight numbers, class types, and availability statuses.

Ticket Management: The system should generate tickets for bookings, including ticket IDs, booking IDs, seat numbers, and flight numbers.

2.2 Identification of Entity and Relationships

Entities:

- Airline
- Airport
- Flight
- Booking
- Passenger
- Payment
- Seat
- Ticket

Relationships:

- Airline has flights (One-to-Many)
- Airport serves flights (Many-to-Many)
- Flight is booked by passengers (Many-to-Many)
- Booking includes tickets (One-to-Many)
- Passenger makes bookings (One-to-Many)
- Payment is associated with bookings (One-to-One)
- Seat belongs to a flight (One-to-Many)
- Ticket corresponds to a booking (One-to-One)

Attributes:

- Airline: AirlineID, Name, ContactInfo
- Airport: AirportCode, Name, Location
- Flight: FlightNumber, DepartureTime, ArrivalTime, AircraftType
- Booking: BookingID, PassengerID, FlightNumber, BookingDate, PaymentStatus
- Passenger: PassengerID, Name, ContactInfo
- Payment: PaymentID, BookingID, Amount, PaymentMethod
- Seat: SeatNumber, FlightNumber, Class, Availability
- Ticket: TicketID, BookingID, SeatNumber, FlightNumber

2.3 Construction of DB using ER Model for the airline reservation system

1. **Airline Entity:**
    - The Airline entity represents different airlines operating flights.
    - It contains attributes such as AirlineID (a unique identifier for each airline), Name (the name of the airline), and ContactInfo (contact information for the airline).
    - Each airline can have multiple flights, establishing a one-to-many relationship between the Airline entity and the Flight entity.
2. **Airport Entity:**
    - The Airport entity represents various airports from where flights depart and arrive.
    - It includes attributes like AirportCode (a unique code for each airport), Name (the name of the airport), and Location (the location of the airport).
    - Airports can serve as departure or arrival points for multiple flights, leading to a many-to-many relationship between the Airport entity and the Flight entity.

3. **Flight Entity:**
    - The Flight entity represents individual flights operated by airlines.
    - It includes attributes such as FlightNumber (a unique identifier for each flight), DepartureTime (the time when the flight departs), ArrivalTime (the time when the flight arrives), AircraftType (the type of aircraft used for the flight), and AirlineID (a foreign key referencing the Airline entity).

4. **Booking Entity:**
    - The Booking entity represents bookings made by passengers for flights.
    - It contains attributes like BookingID (a unique identifier for each booking), PassengerID (a foreign key referencing the Passenger entity), FlightNumber (a foreign key referencing the Flight entity), BookingDate (the date when the booking was made), and PaymentStatus (the status of payment for the booking).
    - Passengers can make multiple bookings, and each booking is for a single flight, establishing a many-to-many relationship between the Passenger entity and the Flight entity.
5. **Passenger Entity:**
    - The Passenger entity represents individuals who book flights.
    - It includes attributes such as PassengerID (a unique identifier for each passenger), Name (the name of the passenger), and ContactInfo (contact information for the passenger).
    - Passengers can make multiple bookings, establishing a one-to-many relationship between the Passenger entity and the Booking entity.

6. **Payment Entity:**
    - The Payment entity represents payments made for bookings.
    - It contains attributes like PaymentID (a unique identifier for each payment), BookingID (a foreign key referencing the Booking entity), Amount (the amount paid), and PaymentMethod (the method used for payment).
    - Each booking can have a single payment, establishing a one-to-one relationship between the Payment entity and the Booking entity.
7. **Seat Entity:**
    - The Seat entity represents individual seats on flights.
    - It includes attributes such as SeatNumber (a unique identifier for each seat), FlightNumber (a primary key and foreign key referencing the Flight entity), Class (the class of the seat), and Availability (the availability status of the seat).
    - Seats belong to specific flights, establishing a one-to-many relationship between the Seat entity and the Flight entity.
8. **Ticket Entity:**
    - The Ticket entity represents tickets issued for bookings.
    - It contains attributes like TicketID (a unique identifier for each ticket), BookingID (a foreign key referencing the Booking entity), SeatNumber (a foreign key referencing the Seat entity), and FlightNumber (a foreign key referencing the Flight entity).
    - Each booking can have a single ticket, establishing a one-to-one relationship between the Ticket entity and the Booking entity.

Entity-Relationship Diagram

The central entities are "Airline," with attributes like name and fleet size, "Airport," with details like name and location coordinates, "Flight," with flight number, departure, and arrival times, "Booking," linking passengers and seats, "Payment," managing transactions, and "Ticket," consolidating booking and payment data. This ERD simplifies airline operations, from booking to departure.

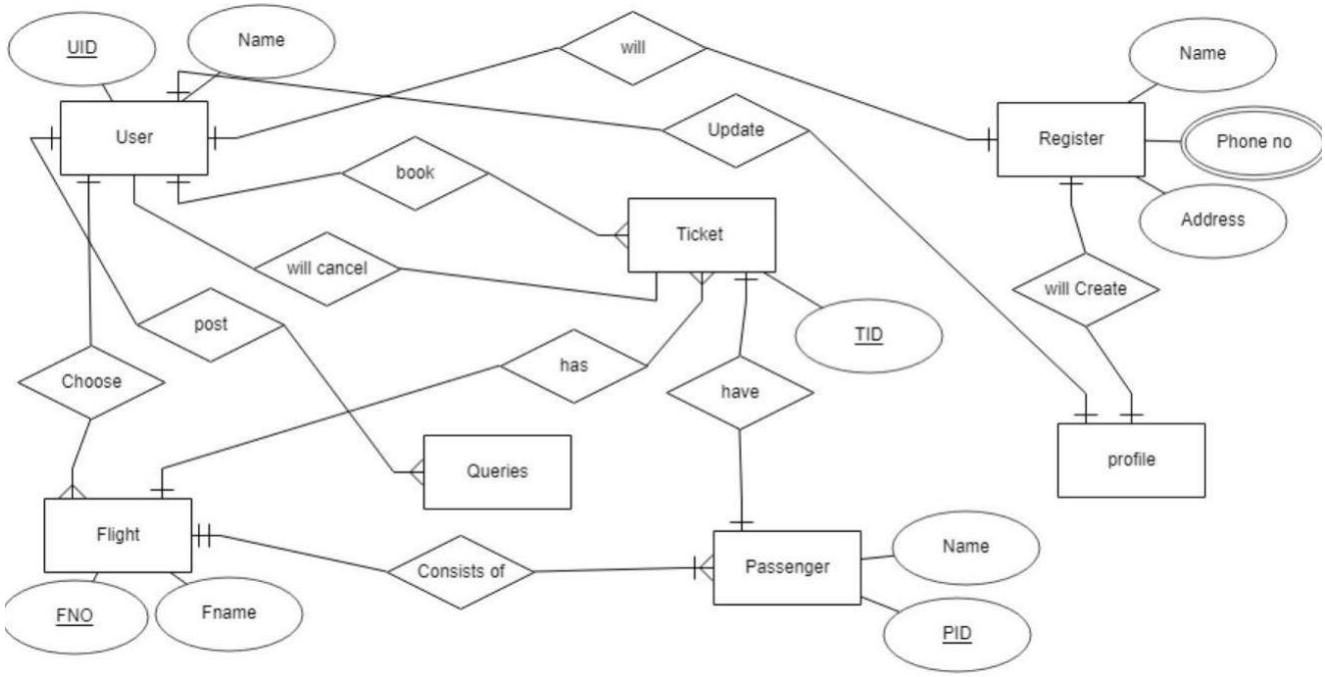


Fig 2.1

Chapter-3

Design of Relational Schemas

3.1 Design of Relational Schemas

Designing the relational schemas for the airline reservation system involves organizing the entities and their attributes into tables, identifying primary and foreign keys, and establishing relationships between tables. Here's how the relational schemas can be designed:

1. **Airline Table:**
    - This table stores information about different airlines.
    - Attributes: AirlineID (Primary Key), Name, ContactInfo.
2. **Airport Table:**
    - This table holds data about various airports.
    - Attributes: AirportCode (Primary Key), Name, Location.
3. **Flight Table:**
    - This table contains details of individual flights.
    - Attributes: FlightNumber (Primary Key), DepartureTime, ArrivalTime, AircraftType, AirlineID (Foreign Key).
4. **Booking Table:**
    - This table records bookings made by passengers.
    - Attributes: BookingID (Primary Key), PassengerID (Foreign Key), FlightNumber (Foreign Key), BookingDate, PaymentStatus.
5. **Passenger Table:**
    - This table stores information about passengers.
    - Attributes: PassengerID (Primary Key), Name, ContactInfo.
6. **Payment Table:**
    - This table stores payment details for bookings.
    - Attributes: PaymentID (Primary Key), BookingID (Foreign Key), Amount, PaymentMethod.
7. **Seat Table:**
    - This table holds data about seats on flights.
    - Attributes: SeatNumber, FlightNumber (Composite Primary Key), Class Availability.

8. **Ticket Table:**
    - This table contains information about tickets issued for bookings.
    - Attributes: TicketID (Primary Key), BookingID (Foreign Key), SeatNumber (Foreign Key), FlightNumber (Foreign Key).

3.2 Schema:

1. Airline:
    - AirlineID (Primary Key)
    - Name
    - ContactInfo
2. Airport:
    - AirportCode (Primary Key)
    - Name
    - Location
3. Flight:
    - FlightNumber (Primary Key)
    - DepartureTime
    - ArrivalTime
    - AircraftType
    - AirlineID (Foreign Key)
    -
4. Booking:
    - BookingID (Primary Key)
    - PassengerID (Foreign Key)
    - FlightNumber (Foreign Key)
    - BookingDate
    - PaymentStatus

5. Passenger:
    - PassengerID (Primary Key)
    - Name
    - ContactInfo
6. Payment:
    - PaymentID (Primary Key)
    - BookingID (Foreign Key)
    - Amount
    - PaymentMethod
7. Seat:
    - SeatNumber
    - FlightNumber (Composite Primary Key)
    - Class
    - Availability
8. Ticket:
    - TicketID (Primary Key)
    - BookingID (Foreign Key)
    - SeatNumber (Foreign Key)
    - FlightNumber (Foreign Key)

Schema diagram:

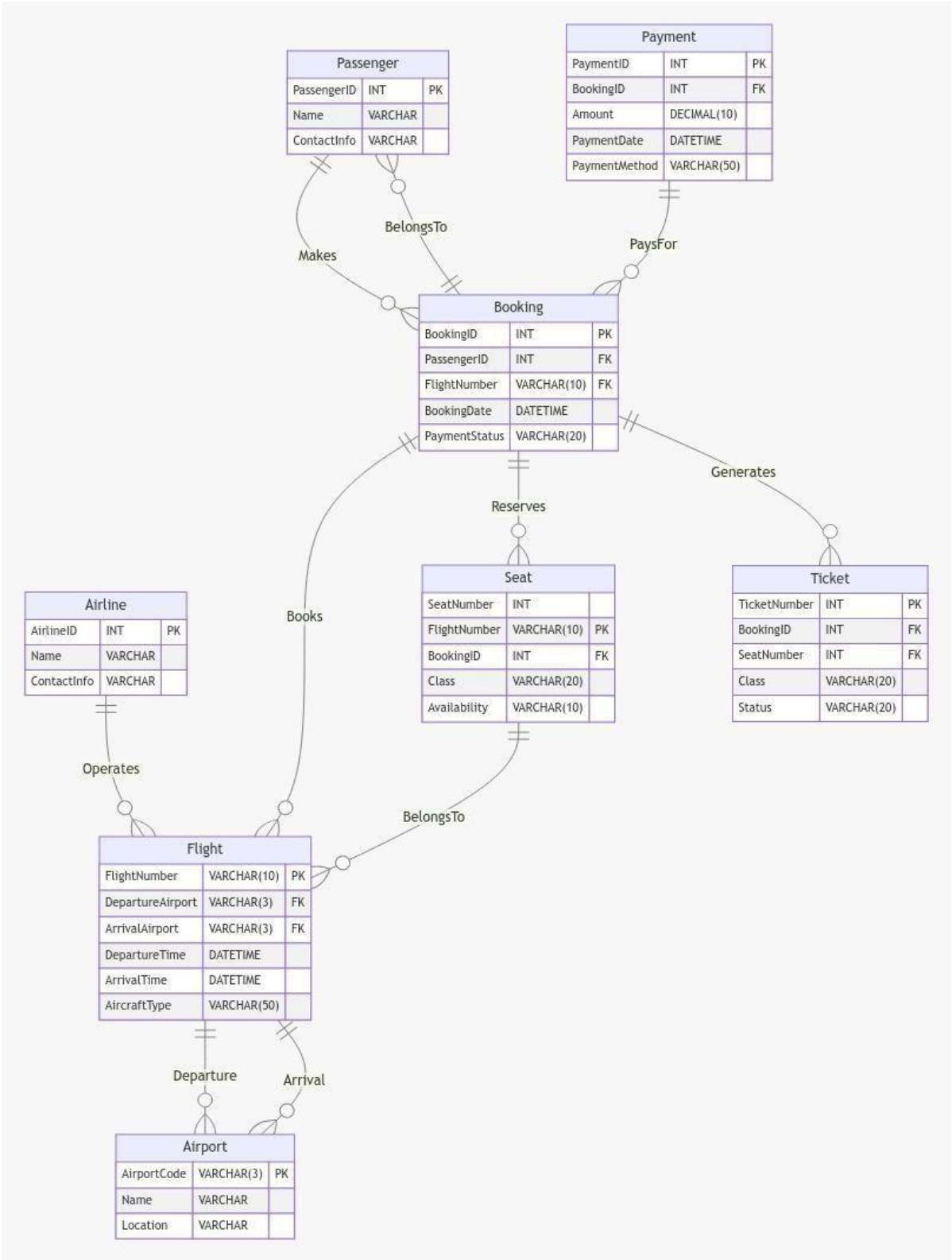


Fig 3.1

Chapter-4

Creation of Database Tables for airline reservation system

4.1 Creation of Database Tables for the project

Creating the User table:

The CREATE query in MySQL is used to create a new database, table, view, index, or trigger based on the specified parameters and structure.

```
-- Create Airline table
CREATE TABLE Airline (
AirlineID INT PRIMARY KEY,
   Name VARCHAR(255),
ContactInfoVARCHAR(255)
);
-- Create Airport table
CREATE TABLE Airport (
AirportCodeVARCHAR(3) PRIMARY KEY,
   Name VARCHAR(255),
   Location VARCHAR(255)
);
-- Create Flight table
CREATE TABLE Flight (
FlightNumberVARCHAR(10) PRIMARY KEY,
DepartureAirportVARCHAR(3),
ArrivalAirportVARCHAR(3),
DepartureTime DATETIME,
ArrivalTime DATETIME,
AircraftTypeVARCHAR(50),
   FOREIGN KEY (DepartureAirport) REFERENCES Airport(AirportCode),
   FOREIGN KEY (ArrivalAirport) REFERENCES Airport(AirportCode)
);
```

```
-- Create Passenger table
CREATE TABLE Passenger (
PassengerID INT PRIMARY KEY,
    Name VARCHAR(255),
ContactInfoVARCHAR(255)
);
-- Create Booking table
CREATE TABLE Booking (
BookingID INT PRIMARY KEY,
PassengerID INT,
FlightNumberVARCHAR(10),
BookingDate DATETIME,
PaymentStatusVARCHAR(20),
    FOREIGN KEY (PassengerID) REFERENCES Passenger(PassengerID),
    FOREIGN KEY (FlightNumber) REFERENCES Flight(FlightNumber)
);
-- Create Seat table
CREATE TABLE Seat (
SeatNumber INT,
FlightNumberVARCHAR(10),
BookingID INT,
    Class VARCHAR(20),
    Availability VARCHAR(10),
    PRIMARY KEY (SeatNumber, FlightNumber),
    FOREIGN KEY (FlightNumber) REFERENCES Flight(FlightNumber),
    FOREIGN KEY (BookingID) REFERENCES Booking(BookingID)
);
-- Create Payment table
CREATE TABLE Payment (
PaymentID INT PRIMARY KEY,
BookingID INT,
```

```
    Amount DECIMAL(10, 2),
PaymentDate DATETIME,
PaymentMethodVARCHAR(50),
    FOREIGN KEY (BookingID) REFERENCES Booking(BookingID)
);
-- Create Ticket table
CREATE TABLE Ticket (
TicketNumber INT PRIMARY KEY,
BookingID INT,
SeatNumber INT,
    Class VARCHAR(20),
    Status VARCHAR(20),
    FOREIGN KEY (BookingID) REFERENCES Booking(BookingID),
    FOREIGN KEY (SeatNumber) REFERENCES Seat(SeatNumber)
);
CREATE TABLE FlightRoute (
RouteID INT PRIMARY KEY AUTO_INCREMENT,
DepartureAirportCodeVARCHAR(3),
ArrivalAirportCodeVARCHAR(3),
    CONSTRAINT fk_departure_airport_flight_route
        FOREIGN KEY (DepartureAirportCode) REFERENCES Airport(AirportCode),
    CONSTRAINT fk_arrival_airport_flight_route
        FOREIGN KEY (ArrivalAirportCode) REFERENCES Airport(AirportCode),
    CONSTRAINT chk_departure_arrival_diff
        CHECK (DepartureAirportCode != ArrivalAirportCode)
);
```

```
mysql> DESCRIBE Airline;
+-------------+--------------+------+-----+---------+-------+
| Field       | Type         | Null | Key | Default | Extra |
+-------------+--------------+------+-----+---------+-------+
| AirlineID   | int          | NO   | PRI | NULL    |       |
| Name        | varchar(255) | YES  |     | NULL    |       |
| ContactInfo | varchar(255) | YES  |     | NULL    |       |
+-------------+--------------+------+-----+---------+-------+
3 rows in set (0.01 sec)

mysql> DESCRIBE Airport;
+-------------+--------------+------+-----+---------+-------+
| Field       | Type         | Null | Key | Default | Extra |
+-------------+--------------+------+-----+---------+-------+
| AirportCode | varchar(3)   | NO   | PRI | NULL    |       |
| Name        | varchar(255) | YES  |     | NULL    |       |
| Location    | varchar(255) | YES  |     | NULL    |       |
+-------------+--------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

```
mysql> DESCRIBE Booking;
+---------------+-------------+------+-----+---------+-------+
| Field         | Type        | Null | Key | Default | Extra |
+---------------+-------------+------+-----+---------+-------+
| BookingID     | int         | NO   | PRI | NULL    |       |
| PassengerID   | int         | YES  | MUL | NULL    |       |
| FlightNumber  | varchar(10) | YES  | MUL | NULL    |       |
| BookingDate   | datetime    | YES  |     | NULL    |       |
| PaymentStatus | varchar(20) | YES  |     | NULL    |       |
+---------------+-------------+------+-----+---------+-------+
5 rows in set (0.00 sec)

mysql> DESCRIBE Flight;
+-------------------+-------------+------+-----+---------+-------+
| Field             | Type        | Null | Key | Default | Extra |
+-------------------+-------------+------+-----+---------+-------+
| FlightNumber      | varchar(10) | NO   | PRI | NULL    |       |
| DepartureAirport  | varchar(3)  | YES  | MUL | NULL    |       |
| ArrivalAirport    | varchar(3)  | YES  | MUL | NULL    |       |
| DepartureTime     | datetime    | YES  |     | NULL    |       |
| ArrivalTime       | datetime    | YES  |     | NULL    |       |
| AircraftType      | varchar(50) | YES  |     | NULL    |       |
+-------------------+-------------+------+-----+---------+-------+
```

```
mysql> DESCRIBE FlightRoute;
+----------------------+------------+------+-----+---------+----------------+
| Field                | Type       | Null | Key | Default | Extra          |
+----------------------+------------+------+-----+---------+----------------+
| RouteID              | int        | NO   | PRI | NULL    | auto_increment |
| DepartureAirportCode | varchar(3) | YES  | MUL | NULL    |                |
| ArrivalAirportCode   | varchar(3) | YES  | MUL | NULL    |                |
+----------------------+------------+------+-----+---------+----------------+
3 rows in set (0.00 sec)

mysql> DESCRIBE Login;
+-------------+--------------+------+-----+-------------------+-------------------+
| Field       | Type         | Null | Key | Default           | Extra             |
+-------------+--------------+------+-----+-------------------+-------------------+
| LoginID     | int          | NO   | PRI | NULL              | auto_increment    |
| PassengerID | int          | YES  | MUL | NULL              |                   |
| UserName    | varchar(100) | YES  |     | NULL              |                   |
| OTP         | varchar(10)  | YES  |     | NULL              |                   |
| LoginTime   | timestamp    | YES  |     | CURRENT_TIMESTAMP | DEFAULT_GENERATED |
+-------------+--------------+------+-----+-------------------+-------------------+
5 rows in set (0.00 sec)

mysql> DESCRIBE Passenger;
+-------------+--------------+------+-----+---------+-------+
| Field       | Type         | Null | Key | Default | Extra |
+-------------+--------------+------+-----+---------+-------+
| PassengerID | int          | NO   | PRI | NULL    |       |
| Name        | varchar(255) | YES  |     | NULL    |       |
| ContactInfo | varchar(255) | YES  |     | NULL    |       |
+-------------+--------------+------+-----+---------+-------+
3 rows in set (0.00 sec)

mysql> DESCRIBE Payment;
+---------------+---------------+------+-----+---------+-------+
| Field         | Type          | Null | Key | Default | Extra |
+---------------+---------------+------+-----+---------+-------+
| PaymentID     | int           | NO   | PRI | NULL    |       |
| BookingID     | int           | YES  | MUL | NULL    |       |
| Amount        | decimal(10,2) | YES  |     | NULL    |       |
| PaymentDate   | datetime      | YES  |     | NULL    |       |
| PaymentMethod | varchar(50)   | YES  |     | NULL    |       |
+---------------+---------------+------+-----+---------+-------+
5 rows in set (0.00 sec)
```

```
mysql> DESCRIBE Seat;
+--------------+-------------+------+-----+---------+-------+
| Field        | Type        | Null | Key | Default | Extra |
+--------------+-------------+------+-----+---------+-------+
| SeatNumber   | int         | NO   | PRI | NULL    |       |
| FlightNumber | varchar(10) | NO   | PRI | NULL    |       |
| BookingID    | int         | YES  | MUL | NULL    |       |
| Class        | varchar(20) | YES  |     | NULL    |       |
| Availability | varchar(10) | YES  |     | NULL    |       |
+--------------+-------------+------+-----+---------+-------+
5 rows in set (0.00 sec)

mysql> DESCRIBE Ticket;
+--------------+-------------+------+-----+---------+-------+
| Field        | Type        | Null | Key | Default | Extra |
+--------------+-------------+------+-----+---------+-------+
| TicketNumber | int         | NO   | PRI | NULL    |       |
| BookingID    | int         | YES  | MUL | NULL    |       |
| SeatNumber   | int         | YES  | MUL | NULL    |       |
| Class        | varchar(20) | YES  |     | NULL    |       |
| Status       | varchar(20) | YES  |     | NULL    |       |
+--------------+-------------+------+-----+---------+-------+
5 rows in set (0.00 sec)
```

4.2 Inserting values for the table:

The INSERT query in MySQL is used to add new rows of data into a table.

```
-- Insert data into Airline table
INSERT INTO Airline (Name, ContactInfo) VALUES
('Delta Airlines', 'contact@delta.com'),
('United Airlines', 'contact@united.com'),
('American Airlines', 'contact@aa.com');

-- Insert data into Airport table
INSERT INTO Airport (AirportCode, Name, Location) VALUES
('ATL', 'Hartsfield-Jackson Atlanta International Airport', 'Atlanta, Georgia'),
('ORD', 'O"Hare International Airport', 'Chicago, Illinois'),
('LAX', 'Los Angeles International Airport', 'Los Angeles, California');

-- Insert data into Flight table
INSERT INTO Flight (FlightNumber, DepartureTime, ArrivalTime, AircraftType) VALUES
('DL100', '2024-03-20 08:00:00', '2024-03-20 10:30:00', 'Boeing 737'),
('UA200', '2024-03-21 10:00:00', '2024-03-21 12:30:00', 'Airbus A320');

-- Insert data into FlightRoute table
INSERT INTO FlightRoute (FlightNumber, DepartureAirportCode, ArrivalAirportCode) VALUES
('DL100', 'ATL', 'ORD'),
('UA200', 'ORD', 'LAX');

-- Insert data into Login table
INSERT INTO Login (PassengerID, UserName, OTP, LoginTime) VALUES

(1, 'user1', '123456', NOW()),
(2, 'user2', '789012', NOW());

-- Insert data into Passenger table
INSERT INTO Passenger (Name, ContactInfo) VALUES
('John Doe', 'john@example.com'),
('Jane Smith', 'jane@example.com');

-- Insert data into Payment table
INSERT INTO Payment (BookingID, Amount, PaymentMethod) VALUES
(1, 200.00, 'Credit Card'),
(2, 300.00, 'PayPal');

-- Insert data into Seat table
```

```
INSERT INTO Seat (SeatNumber, FlightNumber, Class, Availability) VALUES
(1, 'DL100', 'Economy', 'Available'),
(2, 'UA200', 'Business', 'Available');

-- Insert data into Ticket table
INSERT INTO Ticket (TicketNumber, BookingID, SeatNumber, Class, Status) VALUES
(1, 1, 1, 'Economy', 'Available'),
(2, 2, 2, 'Business', 'Available');

-- Insert data into Booking table
INSERT INTO Booking (PassengerID, FlightNumber, BookingDate, PaymentStatus) VALUES
(1, 'DL100', '2024-03-19', 'Pending'),
(2, 'UA200', '2024-03-20', 'Paid');
```

```
mysql> select * from airline;
+-----------+-----------------+--------------------+
| AirlineID | Name            | ContactInfo        |
+-----------+-----------------+--------------------+
|         1 | Delta Airlines  | contact@delta.com  |
|         2 | United Airlines | contact@united.com |
|         3 | American Airlines | contact@aa.com   |
+-----------+-----------------+--------------------+
3 rows in set (0.00 sec)

mysql> select * from airport;
+-------------+-----------------------------------------------+-------------------------+
| AirportCode | Name                                          | Location                |
+-------------+-----------------------------------------------+-------------------------+
| ATL         | Hartsfield-Jackson Atlanta International Airport | Atlanta, Georgia        |
| LAX         | Los Angeles International Airport             | Los Angeles, California |
| ORD         | O'Hare International Airport                  | Chicago, Illinois       |
+-------------+-----------------------------------------------+-------------------------+
3 rows in set (0.00 sec)

mysql> select * from booking;
+-----------+-------------+--------------+---------------------+---------------+
| BookingID | PassengerID | FlightNumber | BookingDate         | PaymentStatus |
+-----------+-------------+--------------+---------------------+---------------+
|         1 |           1 | DL100        | 2024-03-19 00:00:00 | Pending       |
|         2 |           2 | UA200        | 2024-03-20 00:00:00 | Paid          |
+-----------+-------------+--------------+---------------------+---------------+
2 rows in set (0.00 sec)

mysql> select * from flight;
+--------------+------------------+----------------+---------------------+---------------------+--------------+-------------------+
| FlightNumber | DepartureAirport | ArrivalAirport | DepartureTime       | ArrivalTime         | AircraftType | DepartureTerminal |
+--------------+------------------+----------------+---------------------+---------------------+--------------+-------------------+
| DL100        | NULL             | NULL           | 2024-03-20 08:00:00 | 2024-03-20 10:30:00 | Boeing 737   | NULL              |
| UA200        | NULL             | NULL           | 2024-03-21 10:00:00 | 2024-03-21 12:30:00 | Airbus A320  | NULL              |
+--------------+------------------+----------------+---------------------+---------------------+--------------+-------------------+
2 rows in set (0.00 sec)
```

```
mysql> select * from flightroute;
+---------+--------------+----------------------+--------------------+
| RouteID | FlightNumber | DepartureAirportCode | ArrivalAirportCode |
+---------+--------------+----------------------+--------------------+
|       1 | DL100        | ATL                  | ORD                |
|       2 | UA200        | ORD                  | LAX                |
+---------+--------------+----------------------+--------------------+
2 rows in set (0.00 sec)

mysql> select * from login;
Empty set (0.00 sec)

mysql> select * from passenger;
+-------------+------------+-------------------+
| PassengerID | Name       | ContactInfo       |
+-------------+------------+-------------------+
|           1 | John Doe   | john@example.com  |
|           2 | Jane Smith | jane@example.com  |
+-------------+------------+-------------------+
2 rows in set (0.00 sec)

mysql> select * from payment;
Empty set (0.00 sec)

mysql> select * from seat;
+------------+--------------+-----------+----------+--------------+
| SeatNumber | FlightNumber | BookingID | Class    | Availability |
+------------+--------------+-----------+----------+--------------+
|          1 | DL100        |      NULL | Economy  | Available    |
|          2 | UA200        |      NULL | Business | Available    |
+------------+--------------+-----------+----------+--------------+
2 rows in set (0.00 sec)

mysql> select * from Ticket;
+--------------+-----------+------------+----------+-----------+
| TicketNumber | BookingID | SeatNumber | Class    | Status    |
+--------------+-----------+------------+----------+-----------+
|            1 |         1 |          1 | Economy  | Available |
|            2 |         2 |          2 | Business | Available |
+--------------+-----------+------------+----------+-----------+
2 rows in set (0.00 sec)
```

Chapter-5

Complex queries based on the concepts of constraints, sets, joins, views, Triggers and Cursors

5.1 Constraints:

CONSTRAINT_NAME	TABLE_NAME	COLUMN_NAME	REFERENCED_TABLE_NAME	REFERENCED_COLUMN_NAME
PRIMARY	airline	AirlineID	NULL	NULL
PRIMARY	airport	AirportCode	NULL	NULL
PRIMARY	login	LoginID	NULL	NULL
PRIMARY	seat	SeatNumber	NULL	NULL
PRIMARY	seat	FlightNumber	NULL	NULL
PRIMARY	ticket	TicketNumber	NULL	NULL
PRIMARY	flightroute	RouteID	NULL	NULL
PRIMARY	passenger	PassengerID	NULL	NULL
PRIMARY	booking	BookingID	NULL	NULL
PRIMARY	flight	FlightNumber	NULL	NULL
PRIMARY	payment	PaymentID	NULL	NULL
fk_passenger	login	PassengerID	passenger	PassengerID
seat_ibfk_1	seat	FlightNumber	flight	FlightNumber
seat_ibfk_2	seat	BookingID	booking	BookingID
fk_flight	seat	FlightNumber	flight	FlightNumber
ticket_ibfk_1	ticket	BookingID	booking	BookingID
ticket_ibfk_2	ticket	SeatNumber	seat	SeatNumber
fk_booking_ticket	ticket	BookingID	booking	BookingID
fk_flight_route_flight	flightroute	FlightNumber	flight	FlightNumber
fk_flight_route_departure_airport	flightroute	DepartureAirportCode	airport	AirportCode
fk_flight_route_arrival_airport	flightroute	ArrivalAirportCode	airport	AirportCode
booking_ibfk_1	booking	PassengerID	passenger	PassengerID
booking_ibfk_2	booking	FlightNumber	flight	FlightNumber
fk_airline	flight	AirlineID	airline	AirlineID
flight_ibfk_1	flight	DepartureAirport	airport	AirportCode
flight_ibfk_2	flight	ArrivalAirport	airport	AirportCode
fk_departure_airport	flight	DepartureAirport	airport	AirportCode
fk_arrival_airport	flight	ArrivalAirport	airport	AirportCode
fk_booking	payment	BookingID	booking	BookingID
payment_ibfk_1	payment	BookingID	booking	BookingID
fk_booking_payment	payment	BookingID	booking	BookingID

5.2 Sets:

5.2.1 UNION:

-- Retrieve all booking IDs from the Booking table

SELECT BookingID FROM Booking

UNION

-- Retrieve all passenger IDs from the Passenger table

SELECT PassengerID FROM Passenger;

BookingID
1
2
10

3 rows in set (0.00 sec)

5.2.2 INTERSECTION (using INNER JOIN):

SELECT Passenger.PassengerID

FROM Passenger

INNER JOIN Booking ON Passenger.PassengerID = Booking.PassengerID;

```
+-------------+
| PassengerID |
+-------------+
|           1 |
|           2 |
+-------------+
2 rows in set (0.00 sec)
```

3.2.3 MINUS (using LEFT JOIN with WHERE condition):

SELECT Passenger.PassengerID

FROM Passenger

LEFT JOIN Booking ON Passenger.PassengerID = Booking.PassengerID

WHERE Booking.PassengerID IS NULL;

```
+-------------+
| PassengerID |
+-------------+
|          10 |
+-------------+
1 row in set (0.00 sec)
```

5.3 Joins:

5.3.1 Inner Join:

An INNER JOIN in SQL returns only the rows from both tables that have matching values in the specified columns.

SELECT Booking.BookingID, Booking.FlightNumber, Passenger.Name, Passenger.ContactInfo

FROM Booking

INNER JOIN Passenger ON Booking.PassengerID = Passenger.PassengerID;

BookingID	FlightNumber	Name	ContactInfo
1	DL100	John Doe	john.doe@example.com
2	UA200	Jane Smith	jane@example.com

2 rows in set (0.00 sec)

5.3.2 Outer Join:

Left Join: Returns all rows from the left table (the table specified before the JOIN keyword), and the matching rows from the right table (the table specified after the JOIN keyword). If there are no matching rows in the right table, NULL values are returned for the columns from the right table.

```
SELECT Booking.BookingID, Booking.FlightNumber, Passenger.Name, Passenger.ContactInfo
FROM Booking
LEFT JOIN Passenger ON Booking.PassengerID = Passenger.PassengerID;
```

Right Join: Returns all rows from the right table, and the matching rows from the left table. If there are no matching rows in the left table, NULL values are returned for the columns from the left table.

```
SELECT Booking.BookingID, Booking.FlightNumber, Passenger.Name, Passenger.ContactInfo
FROM Passenger RIGHT JOIN Booking ON Passenger.PassengerID = Booking.PassengerID;
```

Full Join: Returns all rows when there is a match in either the left or right table. If there are no matching rows, NULL values are returned for the columns from the table that lacks a matching row.

```
SELECT Booking.BookingID, Booking.FlightNumber, Passenger.Name, Passenger.ContactInfo
FROM Booking
LEFT JOIN Passenger ON Booking.PassengerID = Passenger.PassengerID
UNION
SELECT Booking.BookingID, Booking.FlightNumber, Passenger.Name, Passenger.ContactInfo
FROM Passenger
RIGHT JOIN Booking ON Passenger.PassengerID = Booking.PassengerID
WHERE Booking.BookingID IS NULL;
```

BookingID	FlightNumber	Name	ContactInfo
1	DL100	John Doe	john.doe@example.com
2	UA200	Jane Smith	jane@example.com

2 rows in set (0.00 sec)

5.4 Views:

It combines information from the Booking, Passenger, and Flight tables to provide a summary of bookings, including passenger names and flight details.

```
mysql> -- Create the view
mysql> CREATE VIEW BookingSummary AS
   -> SELECT
   ->b.BookingID,
   ->p.Name AS PassengerName,
   ->f.FlightNumber,
   ->f.DepartureTime,
   ->f.ArrivalTime
   -> FROM
   ->    Booking b
   -> JOIN
   ->    Passenger p ON b.PassengerID = p.PassengerID
   -> JOIN
   ->    Flight f ON b.FlightNumber = f.FlightNumber;
Query OK, 0 rows affected (0.01 sec)

mysql> -- Call the view
mysql> SELECT * FROM BookingSummary;
```

BookingID	PassengerName	FlightNumber	DepartureTime	ArrivalTime
1	John Doe	DL100	2024-03-20 09:00:00	2024-03-20 10:30:00
2	Jane Smith	UA200	2024-03-21 10:00:00	2024-03-21 12:30:00

2 rows in set (0.00 sec)

5.5 Triggers and Cursors:

5.5.1Triggers:

Triggers in MySQL are database objects that are automatically executed in response to certain events on a particular table.

- a trigger that automatically updates the PaymentStatus in the Booking table when a new payment is inserted.

```
DELIMITER //
CREATE TRIGGER UpdatePaymentStatus
AFTER INSERT ON Payment
FOR EACH ROW
BEGIN
   UPDATE Booking
   SET PaymentStatus = 'Paid'
   WHERE BookingID = NEW.BookingID;
END //
DELIMITER ;
INSERT INTO Payment (BookingID, Amount, PaymentDate)
VALUES (1, 100.00, '2024-03-25');
```

```
mysql> select * from payment;
+-----------+-----------+--------+---------------------+---------------+
| PaymentID | BookingID | Amount | PaymentDate         | PaymentMethod |
+-----------+-----------+--------+---------------------+---------------+
|         1 |         1 | 200.00 | 2024-03-20 19:00:53 | Debit Card    |
|         2 |         2 | 300.00 | 2024-03-20 19:00:53 | PayPal        |
|         3 |         1 | 100.00 | 2024-03-25 00:00:00 | NULL          |
+-----------+-----------+--------+---------------------+---------------+
3 rows in set (0.00 sec)
```

5.3Cursors:

A cursor in SQL is a database object that enables traversal over the rows of a result set.

```
-- Create the stored procedure with a parameterized cursor
DELIMITER //
CREATE PROCEDURE GetPassengerDetailsByName(IN searchName VARCHAR(100))
BEGIN
   DECLARE done INT DEFAULT FALSE;
   DECLARE passengerIDINT;
   DECLARE passengerNameVARCHAR(100);
   DECLARE contactInfoVARCHAR(255);
```

```
    -- Declare a cursor with parameters
    DECLARE cur_passenger CURSOR FOR
        SELECT PassengerID, Name, ContactInfo
        FROM Passenger
        WHERE Name = searchName;
    -- Declare continue handler to exit loop when no more rows
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
    -- Open the cursor
    OPEN cur_passenger;
    -- Start fetching rows
    FETCH cur_passenger INTO passengerID, passengerName, contactInfo;
    -- Loop through the cursor results
    WHILE NOT done DO
        -- Process the fetched row (you can perform any desired operations here)
        SELECT passengerID, passengerName, contactInfo;
        -- Fetch the next row
        FETCH cur_passenger INTO passengerID, passengerName, contactInfo;
    END WHILE;
    -- Close the cursor
    CLOSE cur_passenger;
END //
DELIMITER ;
```

```
+-------------+---------------+-------------+
| passengerID | passengerName | contactInfo |
+-------------+---------------+-------------+
|        NULL | John Doe      | NULL        |
+-------------+---------------+-------------+
1 row in set (0.04 sec)
```

Chapter-6

Analyzing the pitfalls, identifying the dependencies, and applying normalizations

6.1 Analyzing the pitfalls:

6.1.1 Redundancy:

Redundancy in database design leads to data inconsistencies, increased storage requirements, and risks of anomalies during updates, insertions, and deletions. It complicates data maintenance and management, requiring extra effort to ensure consistency across redundant copies. To mitigate these issues, database designers should aim for proper normalization to minimize redundancy and dependency, and enforce referential integrity constraints to maintain data consistency.

6.1.2 Inconsistency:

Inconsistency in database design arises from redundant data, leading to discrepancies between copies. It results in challenges in data management, with updates made to one copy not reflecting in others, risking data integrity. Addressing redundancy through normalization and enforcing constraints helps mitigate inconsistency issues, ensuring data accuracy and reliability.

6.1.3 Inefficiency:

Inefficiency in database design stems from poor indexing, over-normalization, and inadequate query optimization. It leads to slow query performance, resource wastage, and scalability limitations. Prioritizing indexing on frequently queried columns, balancing normalization with performance, and optimizing queries can mitigate inefficiency.

6.1.4 Complexity:

Complexity in database design results from over-normalization, inconsistent naming conventions, and inadequate documentation. It leads to difficulties in understanding and maintaining the database schema, increasing the likelihood of errors and inefficiencies. Simplifying normalization, establishing clear naming conventions, and thorough documentation can alleviate complexity, facilitating easier database management and development.

6.2 Identifying the dependencies

6.2.1 Airline Table:

AirlineID -> Name, ContactInfo

Name -> AirlineID, ContactInfo

ContactInfo -> AirlineID, Name

6.2.2 Airport Table:

AirportCode -> Name, Location

Name -> AirportCode, Location

Location -> AirportCode, Name

6.2.3 Flight Table:

FlightNumber -> DepartureTime, ArrivalTime, AircraftType, AirlineID

DepartureTime -> FlightNumber, ArrivalTime, AircraftType, AirlineID

ArrivalTime -> FlightNumber, DepartureTime, AircraftType, AirlineID

AircraftType -> FlightNumber, DepartureTime, ArrivalTime, AirlineID

AirlineID -> FlightNumber, DepartureTime, ArrivalTime, AircraftType

6.2.4 Passenger Table:

PassengerID -> Name, ContactInfo

Name -> PassengerID, ContactInfo

ContactInfo -> PassengerID, Name

6.2.5 Payment Table:

PaymentID -> BookingID, Amount, PaymentMethod

BookingID -> PaymentID, Amount, PaymentMethod

Amount -> PaymentID, BookingID, PaymentMethod

PaymentMethod -> PaymentID, BookingID, Amount

6.2.6 Seat Table:

SeatNumber, FlightNumber -> Class, Availability

Class, Availability -> SeatNumber, FlightNumber

6.2.7 Ticket Table:

TicketID -> BookingID, SeatNumber, FlightNumber

BookingID -> TicketID, SeatNumber, FlightNumber

SeatNumber -> TicketID, BookingID, FlightNumber

FlightNumber -> TicketID, BookingID, SeatNumber

6.2.8 Booking Table:

BookingID -> PassengerID, FlightNumber, BookingDate, PaymentStatus

PassengerID -> BookingID, FlightNumber, BookingDate, PaymentStatus

FlightNumber -> BookingID, PassengerID, BookingDate, PaymentStatus

6.3 Applying Normalizations:

6.3.1 Denormalization:

To intentionally fail normalization conditions in the denormalized Booking table, we can introduce redundancy and dependencies that violate normalization rules:

```
CREATE TABLE DenormalizedBooking (
BookingID INT,
PassengerID INT,
PassengerNameVARCHAR(255),
FlightNumberVARCHAR(10),
DepartureAirportVARCHAR(3),
ArrivalAirportVARCHAR(3),
DepartureTime DATETIME,
ArrivalTime DATETIME,
BookingDate DATETIME,
PaymentStatusVARCHAR(20),
    -- Introduce a multi-valued dependency by repeating PassengerName for each PassengerID
    PassengerName2 VARCHAR(255),
    PassengerName3 VARCHAR(255),
    -- Introduce a partial dependency by having FlightNumber determine ArrivalAirport
FlightNumberArrivalAirportVARCHAR(225), -- Adjusted length to accommodate both FlightNumber and ArrivalAirport
    -- Introduce a transitive dependency by having DepartureTime determine FlightNumber
DepartureTimeFlightNumberVARCHAR(225), -- Adjusted length to accommodate both DepartureTime and
FlightNumber
    -- Introduce a join dependency by combining multiple attributes into a single column
    CombinedInfo VARCHAR(255)
);


-- Populate the DenormalizedBooking table with some data (example)
```

```
INSERT INTO DenormalizedBooking (BookingID, PassengerID, PassengerName, FlightNumber, DepartureAirport,
ArrivalAirport, DepartureTime, ArrivalTime, BookingDate, PaymentStatus, PassengerName2, PassengerName3,
FlightNumberArrivalAirport, DepartureTimeFlightNumber, CombinedInfo)

VALUES

(1, 1, 'John Doe', 'DL100', 'ATL', 'ORD', '2024-03-20 08:00:00', '2024-03-20 10:30:00', '2024-03-19 00:00:00', 'Paid',
'John Doe', 'John Doe', 'DL100-ORD', '2024-03-20 08:00:00-DL100', 'John Doe-DL100-ATL-ORD');
```

BookingID	PassengerID	PassengerName	FlightNumber	DepartureAirport	ArrivalAirport	DepartureTime	ArrivalTime	BookingDate
1	1	John Doe	DL100	ATL	ORD	2024-03-20 08:00:00	2024-03-20 10:30:00	2024-03-19 00:00:00

PaymentStatus	PassengerName2	PassengerName3	FlightNumberArrivalAirport	DepartureTimeFlightNumber	CombinedInfo
Paid	John Doe	John Doe	DL100-ORD	2024-03-20 08:00:00-DL100	John Doe-DL100-ATL-ORD

6.3.2 Normalization:

Normalization in databases is the process of organizing data in a relational database to reduce redundancy and dependency. It involves breaking down large tables into smaller, more manageable tables and defining relationships between them. Normalization typically involves several normal forms (1NF, 2NF, 3NF, BCNF, etc.) to ensure data integrity and optimize database performance. The goal of normalization is to minimize data redundancy, improve data integrity, and make the database structure more flexible and adaptable to changes.

6.3.3 First Normal Form (1NF):

- Each cell in a table should hold a single, indivisible value.
- Each column in a table must have a unique name, and the order of columns should not matter.
- Avoid storing multiple values in a single field. Each field should represent a single piece of data.
- The order in which data is stored should not impact its interpretation or retrieval

```
ALTER TABLE DenormalizedBooking

DROP COLUMN PassengerName2,
```

DROP COLUMN PassengerName3;

BookingID	PassengerID	PassengerName	FlightNumber	DepartureAirport	ArrivalAirport	DepartureTime	ArrivalTime	BookingDate	PaymentStatus	FlightNumberArrivalAirport
1	1	John Doe	DL100	ATL	ORD	2024-03-20 08:00:00	2024-03-20 10:30:00	2024-03-19 00:00:00	Paid	DL100-ORD
2	2	Jane Smith	UA200	ORD	LAX	2024-03-21 10:00:00	2024-03-21 12:30:00	2024-03-20 00:00:00	Paid	UA200-LAX
3	3	Alice Johnson	DL100	ATL	ORD	2024-03-22 08:00:00	2024-03-22 10:30:00	2024-03-21 00:00:00	Pending	DL100-ORD

DepartureTime	ArrivalTime	BookingDate	PaymentStatus	FlightNumberArrivalAirport	DepartureTimeFlightNumber	CombinedInfo
2024-03-20 08:00:00	2024-03-20 10:30:00	2024-03-19 00:00:00	Paid	DL100-ORD	2024-03-20 08:00:00-DL100	John Doe-DL100-ATL-ORD
2024-03-21 10:00:00	2024-03-21 12:30:00	2024-03-20 00:00:00	Paid	UA200-LAX	2024-03-21 10:00:00-UA200	Jane Smith-UA200-ORD-LAX
2024-03-22 08:00:00	2024-03-22 10:30:00	2024-03-21 00:00:00	Pending	DL100-ORD	2024-03-22 08:00:00-DL100	Alice Johnson-DL100-ATL-ORD

6.3.4 Second Normal Form (2NF):

- The table must already be in First Normal Form (1NF), meaning it satisfies the criteria of atomic values in each cell.
- Each non-prime attribute (attribute not part of any candidate key) must be fully functionally dependent on the entire primary key, ensuring that no attribute is dependent on only a subset of the primary key. This eliminates partial dependencies, where part of the primary key determines some attributes' values independently.

Created a separate Passenger table to store passenger details.

```
-- Create a Customer table to store customer details
CREATE TABLE Customer (
CustomerID INT PRIMARY KEY,
CustomerNameVARCHAR(255)
);

-- Insert data into Customer table
INSERT INTO Customer (CustomerID, CustomerName) VALUES
(1, 'John Doe'),
(2, 'Jane Smith');

-- Rename PassengerID column to CustomerID in DenormalizedBooking table
ALTER TABLE DenormalizedBooking
CHANGE COLUMN PassengerID CustomerIDINT;

-- Add foreign key constraint to CustomerID in DenormalizedBooking table
ALTER TABLE DenormalizedBooking
ADD FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID);
```

CustomerID	CustomerName
1	John Doe
2	Jane Smith

6.3.5 Third Normal Form (3NF):

- The table must already satisfy the criteria of Second Normal Form (2NF).
- There should be no transitive dependencies, meaning that no non-prime attribute should depend on another non-prime attribute. All non-prime attributes must depend only on the primary key.

Created a separate FlightDetails table to store flight information.

```
-- Create a FlightDetails table with adjusted column name
CREATE TABLE FlightDetails (
FlightCodeVARCHAR(10) PRIMARY KEY,
DepartureAirportVARCHAR(3),
ArrivalAirportVARCHAR(3),
   DepartureTime DATETIME,
ArrivalTime DATETIME
);

-- Update DenormalizedBooking table to reference the adjusted column name
ALTER TABLE DenormalizedBooking
DROP COLUMN FlightNumberArrivalAirport,
DROP COLUMN DepartureTimeFlightNumber,
ADD COLUMN FlightCodeVARCHAR(10),
ADD FOREIGN KEY (FlightCode) REFERENCES FlightDetails(FlightCode);
```

6.3.6 Boyce Codd Normal Form (BCNF):

- The table must already satisfy the criteria of Third Normal Form (3NF).
- Every non-trivial functional dependency must be a dependency on a superkey, meaning that if A determines B, then A must be a superkey. This ensures that there are no non-trivial dependencies on attributes that are not part of any candidate key.
- BookingID → PassengerID, FlightNumber, BookingDate, PaymentStatus
- PassengerID →PassengerName
- FlightNumber →DepartureAirport, ArrivalAirport, DepartureTime, ArrivalTime
- Now, let's determine the candidate keys:
- Candidate Keys:
- {BookingID}
- {PassengerID}
- {FlightNumber}
- Since each determinant is a candidate key, the DenormalizedBooking table already satisfies BCNF. No further decomposition is required.
- If any non-trivial functional dependencies were present on non-candidate keys, we would decompose the table accordingly to ensure BCNF compliance. However, in this case, the table is already in BCNF.

6.3.7 Fourth Normal Form (4NF):

- The table must already satisfy the criteria of Boyce-Codd Normal Form (BCNF).
- There should be no non-trivial multivalued dependencies between attributes. This means that the values in one set of attributes should not determine the values in another set of attributes independently of the primary key.

The DenormalizedBooking table has the following attributes:

- BookingID (Primary Key)
- PassengerID
- FlightNumber
- DepartureAirport
- ArrivalAirport
- DepartureTime
- ArrivalTime
- BookingDate

To identify multi-valued dependencies, we need to check if any non-key attributes are functionally dependent on a subset of the primary key attributes.

Upon analysis, it appears that there are no multi-valued dependencies present in the DenormalizedBooking table. Each attribute seems to be functionally dependent on the entire primary key (BookingID). Therefore, the table already satisfies Fourth Normal Form (4NF) requirements.

6.3.8 Fifth Normal Form (5NF):

- The table must already satisfy the criteria of Fourth Normal Form (4NF).
- All join dependencies are satisfied, meaning that every decomposition into smaller tables preserves certain implied relationships, ensuring that no redundancies or anomalies occur when joining these tables back together.

Given the current state of the DenormalizedBooking table, let's first identify the functional dependencies:

Functional Dependencies:

BookingID → PassengerID, FlightNumber, BookingDate, PaymentStatus

PassengerID →PassengerName

FlightNumber →DepartureAirport, ArrivalAirport, DepartureTime, ArrivalTime

Now, let's examine if there are any join dependencies that are not implied by the candidate keys. If any are found, we need to decompose the table accordingly.

Upon analyzing the table, it appears that all join dependencies are already implied by the candidate keys (BookingID, PassengerID, FlightNumber). Therefore, the DenormalizedBooking table is already in Fifth Normal Form (5NF), and no further decomposition is required. In summary, the DenormalizedBooking table satisfies Fifth Normal Form (5NF) requirements, ensuring that every join dependency is implied by a candidate key.

Chapter-7

Implementation of concurrency control and recovery mechanisms

Concurrency control and recovery mechanisms are fundamental aspects of database management systems (DBMS). Concurrency control ensures that multiple transactions can operate concurrently without interfering with each other, preventing issues like lost updates and inconsistent data. Techniques such as locking, timestamp ordering, and optimistic concurrency control are employed to manage concurrency. On the other hand, recovery mechanisms ensure that the database can recover to a consistent state after a failure, such as a system crash or power outage. Techniques like logging and checkpoints are used to maintain a record of transactions and ensure that changes are either fully committed or rolled back, preserving data integrity. These mechanisms are crucial for ensuring the reliability, consistency, and availability of data in database systems, especially in environments with high transaction volumes and critical business operations.

The four important aspects of database management are:

1. Atomicity: Ensures that transactions are all or nothing, meaning either all operations within the transaction are successfully completed, or none of them are.

2. Consistency: Guarantees that the database remains in a consistent state before and after the execution of a transaction.

3. Isolation: Ensures that concurrent transactions do not interfere with each other, maintaining data integrity and preventing anomalies.

4. Durability: Ensures that once a transaction is committed, its effects are permanently saved and persisted, even in the event of a system failure.

7.1 Commit:

7.1.1 Committing a Simple Transaction:

```
START TRANSACTION;

-- Selecting data from the Airline table

SELECT * FROM Airline;
COMMIT;
```

7.1.2Committing Multiple Transactions:

```
START TRANSACTION;

-- Selecting data from the Airport table before update
SELECT * FROM Airport;

-- Updating a record in the Airport table
UPDATE Airport SET Location = 'New Location' WHERE AirportCode = 'ATL';

COMMIT;

-- Starting a new transaction to select the updated record
START TRANSACTION;

-- Selecting the updated record from the Airport table
SELECT * FROM Airport WHERE AirportCode = 'ATL';

COMMIT;
```

7.1.3 Committing after Savepoint:

```
START TRANSACTION;

-- Inserting a new record into the Booking table
INSERT INTO Booking (PassengerID, FlightNumber, BookingDate, PaymentStatus)
VALUES (3, 'DL101', '2024-03-21', 'Pending');

-- Setting a savepoint
SAVEPOINT before_update;

-- Updating the record
UPDATE Booking SET PaymentStatus = 'Paid' WHERE BookingID = 3;

-- Committing after the savepoint
COMMIT;

-- Selecting the updated record
SELECT * FROM Booking WHERE BookingID = 3;
```

7.2 Rollback:

7.2.1 Rolling Back a Transaction:

```
START TRANSACTION;

-- Selecting data from the Seat table before update
SELECT * FROM Seat WHERE SeatNumber = 1;

-- Updating the record in the Seat table
UPDATE Seat SET Availability = 'Booked' WHERE SeatNumber = 1;

-- Rolling back the transaction to revert the changes
ROLLBACK;

-- Selecting the record again to verify that the changes were reverted
SELECT * FROM Seat WHERE SeatNumber = 1;
```

7.2.2 Rolling Back to a Savepoint:

```
START TRANSACTION;

-- Inserting a new record into the Ticket table
INSERT INTO Ticket (BookingID, SeatNumber, FlightNumber) VALUES (4, 3, 'DL100');

-- Setting a savepoint
SAVEPOINT before_update;

-- Updating the record
UPDATE Ticket SET SeatNumber = 4 WHERE BookingID = 4;
-- Rolling back to the savepoint to undo the update
ROLLBACK TO before_update;
-- Selecting the record to verify that the changes were undone
SELECT * FROM Ticket WHERE BookingID = 4;
```

7.2.3 Rolling Back Multiple Transactions:

```
START TRANSACTION;

-- Inserting a new record into the Payment table
INSERT INTO Payment (BookingID, Amount, PaymentMethod) VALUES (3, 400.00, 'Credit Card');

-- Updating the record in the Payment table
UPDATE Payment SET Amount = 450.00 WHERE BookingID = 3;

-- Rolling back the entire transaction to undo both insert and update operations
ROLLBACK;

-- Selecting the record to verify that no changes were made
SELECT * FROM Payment WHERE BookingID = 3;
```

7.3 Savepoint:

7.3.1 Creating a Savepoint:

```
START TRANSACTION;

-- Inserting a new record into the Passenger table
INSERT INTO Passenger (Name, ContactInfo) VALUES ('Alice', 'alice@example.com');

-- Creating a savepoint named 'before_update'
SAVEPOINT before_update;
-- Updating the record in the Passenger table
UPDATE Passenger SET ContactInfo = 'alice@gmail.com' WHERE Name = 'Alice';
-- Committing the transaction
COMMIT;
-- Selecting the record to verify changes
SELECT * FROM Passenger WHERE Name = 'Alice';
```

7.3.2Rolling Back to a Savepoint:

```
START TRANSACTION;

-- Inserting a new record into the Passenger table
INSERT INTO Passenger (Name, ContactInfo) VALUES ('Bob', 'bob@example.com');

-- Creating a savepoint named 'before_update'
SAVEPOINT before_update;

-- Updating the record in the Passenger table
UPDATE Passenger SET ContactInfo = 'bob@gmail.com' WHERE Name = 'Bob';

-- Rolling back to the savepoint 'before_update' to undo the update operation
ROLLBACK TO before_update;

-- Selecting the record to verify that changes were undone
SELECT * FROM Passenger WHERE Name = 'Bob';
```

7.3.3Releasing a Savepoint:

```
START TRANSACTION;
-- Inserting a new record into the Passenger table
INSERT INTO Passenger (Name, ContactInfo) VALUES ('Charlie', 'charlie@example.com');

-- Creating a savepoint named 'before_update'
SAVEPOINT before_update;

-- Updating the record in the Passenger table
UPDATE Passenger SET ContactInfo = 'charlie@gmail.com' WHERE Name = 'Charlie';

-- Releasing the savepoint 'before_update'
RELEASE SAVEPOINT before_update;

-- Committing the transaction
COMMIT;
```

Chapter-8

Code for the project

8.1 Login Module:

```
import MySQLdb
from configure import user, password
from tkinter import Button, Checkbutton, Entry, IntVar, Label, StringVar, Tk, Toplevel, messagebox
from registration import Register
from pastBookings import PastBookings


class Login:
    def__init_(self):
        root = Toplevel()
        #root = Tk()
root.title("Login System - Udan Khatola")
root.geometry("400x300")

self.email = StringVar()
self.passwor = StringVar()
self.checkButton = IntVar()
self.check_value = 0
self.pass_entry = None

self.buttons(root)
root.mainloop()

    def callback(self):
Register()

    def checkbox(self, root):
self.check_value = self.checkButton.get()
```

```
        if self.check_value == 0:
self.pass_entry.configure(show="*")
        else:
self.pass_entry.configure(show="")


    def database(self,root):
        if self.email.get()=="":
messagebox.showerror(master=root,title="Form Empty", message="Please enter email id!")
root.lift()
            return
        try:
            con = MySQLdb.connect(host='localhost', user=user,
                                  password=password, database="airline_reservation")
            cursor = con.cursor()
            command = 'SELECT emailid,password FROM registration'
cursor.execute(command)
registered_email = cursor.fetchall()
            for each in registered_email:
                if each[0]==self.email.get():
                    if each[1]==self.passwor.get():
messagebox.showinfo(master=root,title="Welcome",message="You have been logged in!")
root.destroy()
PastBookings(self.email.get())
                        return
                    else:
messagebox.showerror(master=root,title="",message="Incorrect password")
root.lift()
                        return
messagebox.showwarning(master=root,title="Please register",message="Email id not registered. Register now!")
root.lift()
```

```
        except Exception as e:
            print(e)
messagebox.showerror(master=root,title="Error",message= "Error\nUnable to login.")
root.lift()

    def buttons(self, root):
Label(root, text="Login", font=("Times New Roman", 25,'bold')).place(x=150, y=20)

Label(root, text="Email:", font=("Times New Roman", 15)).place(x=50,y=100)
Entry(root, width=30, textvar=self.email).place(x=150, y=105)

        button = Checkbutton(root, text = "Show password",
                    variable = self.checkButton,
onvalue = 1,
offvalue = 0,
                    height = 2,
                    width = 10,
                    command = lambda: self.checkbox(root)).place(x=150,y=175)
Label(root, text="Password:", font=("Times New Roman", 15)).place(x=50, y=150)
self.pass_entry = Entry(root, width=30,textvar=self.passwor,show="*")
self.pass_entry.place(x=150, y=155)

Button(root, text='Submit', width=15, height=1, bg='brown', fg='white', command=lambda:
self.database(root)).place(x=140, y=230)

Label(root, text="Not a registered user?", font=("Times New Roman", 10)).place(x=90,y=260)

        link = Label(root, text="Register Now!", fg="blue", cursor="hand2")
link.place(x=215,y=260)
link.bind("<Button-1>", lambda e: self.callback())
```

8.2 Airline TypeModule:

```
airline_type = {
    "Vistara":{
        "name" : "Vistara",
        "rate": 5000,
        "logo" : "src/vistara.svg.png"
    },
    "AirAsia" : {
        "name" : "AirAsia",
        "rate": 5200,
        "logo" : "src/airAsia.svg.png"
    },
    "Indigo" : {
        "name" : "Indigo",
        "rate": 4500,
        "logo" : "src/indigo.svg.png"
    },
    "Air India" : {
        "name" : "Air India",
        "rate": 4800,
        "logo" : "src/airindia.png"
    },
}
```

8.3 BaseModule:

```
import tkinter as tk
from tkinter import *
from fullScreen import FullScreenApp
from bookTicket import  Booking
from webcheckin import WebCheckin
from login import Login
import datetime



class Main:
    def__init_(self):
        root = Tk()
root.title("Udan Khatola")
        app = FullScreenApp(root)
background_image = PhotoImage(file="src/bkg.png")
background_label = Label(root, image=background_image)
background_label.place(x=0, y=0, relwidth=1, relheight=1)
self.buttons(root)
root.mainloop()


    def userLogin(self):
Login()


    def bookTicket(self):
Booking()


    def checkin(self):
WebCheckin()


    def buttons(self, root):
```

```
        but1 = Button(
            root,
            bd=0,
            relief="groove",
            compound=tk.CENTER,
bg="white",
fg="black",
activeforeground="pink",
activebackground="white",
            font="arial 12",
            text="Login/Register",
pady=10,
borderwidth=10,
            height=1,
            width=30,
            command=self.userLogin
        )

        but1.place(x=100, y=100)
        but2 = Button(
            root,
            bd=0,
            relief="groove",
            compound=tk.CENTER,
bg="white",
fg="black",
activeforeground="pink",
activebackground="white",
            font="arial 12",
            text="Book a ticket",
pady=10,
```

```
borderwidth=10,
            height=1,
            width=30,
            command=self.bookTicket
        )

        but2.place(x=100, y=200)

        but3 = Button(
            root,
            bd=0,
            relief="groove",
            compound=tk.CENTER,
bg="white",
fg="black",
activeforeground="pink",
activebackground="white",
            font="arial 12",
            text="Web Checkin",
pady=10,
borderwidth=10,
            height=1,
            width=30,
            command = self.checkin
        )

        but3.place(x=100, y=300)



Main()
```

8.4 Boarding PassModule:

```
from tkinter import Frame, Label, PhotoImage, Tk, Toplevel, messagebox
from PIL import Image, ImageTk
import MySQLdb
from configure import user,password
from airlineTypes import airline_type


class BoardingPass:
    def__init_(self,pnr):
        window = Toplevel()
        # window = Tk()
window.title("Invoice - Udan Khatola")
window.geometry("600x300")

self.pnr = pnr
self.data = self.database(window)
self.widgets(window)
window.mainloop()


    def database(self,top):
        try:
            con = MySQLdb.connect(host='localhost', user=user,
                                  password=password, database="airline_reservation")
            cursor = con.cursor()
cursor.execute("SELECT * FROM booking")
pnr_data = cursor.fetchall()
            for data in pnr_data:
                if self.pnr == data[7]:
                    return data
        except Exception as e:
            print(e)
```

```
messagebox.showerror(master=top,title="Error",message= "Error\nUnable to generate Boarding pass.")
top.lift()

    def widgets(self,top):
        one = Frame(top, bg = '#ED1B24', width = 60, height = 450)
one.pack(side = 'left')

        two = Frame(top, bg = '#1B1464', width = 700, height = 40)
two.pack(side = 'top')

        load = Image.open(airline_type[self.data[-3]]["logo"])
        load = load.resize((80, 60), Image.ANTIALIAS)
        render = ImageTk.PhotoImage(load)
img = Label(top, image=render)
img.image = render
img.place(x=60, y=40)

        lbl1 = Label(two, text='Boarding Pass', font=('Times New Roman', 20), fg='#fff', bg='#1B1464')
        lbl1.place(x=305, y=6, anchor='ne')

        lbl2 = Label(top, text='Passenger Name:', font=('Times New Roman', 10))
        lbl2.place(x=300, y=45, anchor='ne')
        lbl3 = Label(top, text=self.data[0]+" "+self.data[1], font=('Times New Roman', 15))
        lbl3.place(x=350, y=70, width = 200, anchor='ne')

        lbl4 = Label(top, text='From', font=('Times New Roman', 10))
        lbl4.place(x=120, y=110, anchor='ne')
        lbl5 = Label(top, text=self.data[8], font=('Times New Roman', 15))
        lbl5.place(x=205, y=130, width = 100, anchor='ne')

        lbl6 = Label(top, text='To', font=('Times New Roman', 10))
```

```
        lbl6.place(x=108, y=160, anchor='ne')
        lbl7 = Label(top, text=self.data[9], font=('Times New Roman', 15))
        lbl7.place(x=205, y=180, width = 100, anchor='ne')

        lbl8 = Label(top, text='Flight', font=('Times New Roman', 9))
        lbl8.place(x=290, y=110, anchor='ne')
        lbl9 = Label(top, text=self.data[-3], font=('Times New Roman', 13))
        lbl9.place(x=315, y=130, width = 80, anchor='ne')

        lbl10 = Label(top, text='Date', font=('Times New Roman', 9))
        lbl10.place(x=370, y=110, anchor='ne')
        lbl11 = Label(top, text=self.data[10], font=('Times New Roman', 15))
        lbl11.place(x=430, y=130, width = 100, anchor='ne')

        lbl12 = Label(top, text='Time', font=('Times New Roman', 9))
        lbl12.place(x=370, y=160, anchor='ne')
        lbl13 = Label(top, text=self.data[11], font=('Times New Roman', 15))
        lbl13.place(x=417, y=180, width = 100, anchor='ne')

Label(top, text='Seat No:',bg='yellow', font=('Times New Roman', 20)).place(x=470, y=220, anchor='ne')
Label(top, text=self.data[-1],bg='yellow', font=('Times New Roman', 20)).place(x=515, y=220,
anchor='ne',width=50)


# BoardingPass('0EIqFy')
```

8.5 BookTicketModule:

```
import json
from tkinter import Button, Entry, Label, Listbox, Scrollbar, StringVar, Toplevel, messagebox, ttk
from tkcalendar import Calendar, DateEntry
import datetime
from selectAirline import Airline
from tkinter.messagebox import askokcancel



class Booking:
    def__init (self):
        top = Toplevel()
top.title("Book a Ticket - Udan Khatola")
top.geometry("800x350")

self.airport_names = []
self.airport_code = []
self.source = StringVar()
self.destination = StringVar()
self.calendar = StringVar()
self.time = StringVar()
self.travel_class = StringVar()

self.loadJson()
self.buttons(top)
top.mainloop()
        def loadJson(self):
         data = open('json_data/airports.json',)
json_data = json.load(data)
airport_data = json_data["airports"]
         for airports in airport_data:
```

```
self.airport_names.append(airports["airport_name"])
data.close()

    def isEmpty(self, booking_dict, top):
        if booking_dict["source"] == '':
messagebox.showerror(master=top, title="Form Empty",
                             message="Please Enter Source Location!")
top.lift()
            return True

elifbooking_dict["destination"] == '':
messagebox.showerror(
                master=top, title="Form Empty", message="Please Enter Destination Location!")
top.lift()
            return True
elifbooking_dict["time"] == '':
messagebox.showerror(master=top, title="Form Empty",
                             message="Please Enter Time of Travel!")
top.lift()
            return True
elifbooking_dict["travel_class"] == '':
messagebox.showerror(master=top, title="Form Empty",
                             message="Please Enter Class of Travel!")
top.lift()
            return True

elifbooking_dict["source"] == booking_dict["destination"]:
messagebox.showerror(
                master=top, title="Form Empty", message="Source and destination cannot be same!")
top.lift()
            return True
```

```
        return False


    def onSubmit(self, top):
booking_dict = {
            "source": self.source.get(),
            "destination": self.destination.get(),
            "date": self.calendar.get(),
            "time": self.time.get(),
            "travel_class": self.travel_class.get()
        }
if(self.isEmpty(booking_dict, top) == True):
            return
        else:
top.destroy()
            Airline(booking_dict)


    def buttons(self, top):
        label = Label(top, text="Flight Booking", padx="3",
pady="3", font=("Times New Roman", 25)).pack()

Label(top, text="Source Location: ", font=(
            "Times New Roman", 15)).place(x=180, y=60)
source_chosen = ttk.Combobox(top, width=38, textvariable=self.source)
source_chosen['values'] = self.airport_names
source_chosen.place(x=375, y=65)
source_chosen.current()

Label(top, text="Destination Location: ", font=(
            "Times New Roman", 15)).place(x=180, y=100)
destination_chosen = ttk.Combobox(
            top, width=38, textvariable=self.destination)
```

```
destination_chosen['values'] = self.airport_names
destination_chosen.place(x=375, y=105)
destination_chosen.current()

    today = datetime.date.today()
Label(top, text="Date of Journey: ", font=(
        "Times New Roman", 15)).place(x=180, y=140)
cal = DateEntry(top, selectmode="day", width=15, year=today.year, month=today.month, day=today.day,
                background='darkblue', foreground='white', borderwidth=2, textvariable=self.calendar)
cal.place(x=375, y=145)

Label(top, text="Time of Journey: ", font=(
        "Times New Roman", 15)).place(x=180, y=180)
time_chosen = ttk.Combobox(top, width=10, textvariable=self.time)
time_chosen['values'] = ['7:30', '9:30', '11:30',
                         '13:30', '15:30', '17:30', '19:30', '21:30', '23:30']
time_chosen.place(x=375, y=185)
time_chosen.current()

Label(top, text="Class of Travel: ", font=(
        "Times New Roman", 15)).place(x=180, y=220)
class_chosen = ttk.Combobox(
        top, width=20, textvariable=self.travel_class)
class_chosen['values'] = ['First Class',
                          'Business Class', 'Economy Class']
class_chosen.place(x=375, y=225)
class_chosen.current()

Button(top, text='Submit', width=20, bg='brown',
fg='white', command=lambda: self.onSubmit(top)).place(x=300, y=280)
```

8.6 Booking DetailsModule:

```
from tkinter import Button, Canvas, Entry, Frame, IntVar, Label, Radiobutton, StringVar, Text, Tk, Toplevel,
messagebox
# from PIL.Image import Image
from PIL import ImageTk, Image
import json
from configure import user, password
import MySQLdb
import string
import random
from invoice import Invoice



class Details:
    def__init_(self, details, travel_details):
        window = Toplevel()
        # window = Tk()
window.title("Confirm booking - Udan Khatola")
window.geometry("800x600")

self.airline_details = details
self.travel_details = travel_details
self.source_city = ''
self.destination_city = ''
self.first_name = StringVar()
self.last_name = StringVar()
self.email = StringVar()
self.address = StringVar()
self.phone = StringVar()
self.Gender = IntVar()
self.age = StringVar()
```

```
self.loadJson()
self.widgets(window)
self.inputDetails(window)
window.mainloop()


    def loadJson(self):
        data = open('json_data/airports.json',)
json_data = json.load(data)
airport_data = json_data["airports"]
        for airports in airport_data:
            if airports["airport_name"] == self.travel_details["source"]:
self.source_city = airports["city_name"]
            if airports["airport_name"] == self.travel_details["destination"]:
self.destination_city = airports["city_name"]
data.close()



    def isFormEmpty(self):
        if(self.first_name.get() == '' or self.last_name.get() == '' or self.email.get() == '' or self.phone.get() == '' or
self.Gender.get() == 0 or self.age.get() == ''):
            return True


    def generatePNR(self):
        lower = list(string.ascii_lowercase)
        upper = list(string.ascii_uppercase)
        number = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']
final_list = lower+upper+number
pnr = ''
        for _ in range(6):
pnr += random.choice(final_list)
        return pnr
```

```
 def onSubmit(self, top):
        check = True
firstName = self.first_name.get()
lastName = self.last_name.get()
emailid = self.email.get()
        num = self.phone.get()
        gender = self.Gender.get()
        age =  self.age.get()
        address = self.address.get()
pnr = self.generatePNR()

        if self.isFormEmpty() == True:
messagebox.showerror(master=top, title="Form Empty",
                             message="Please enter all details!")
top.lift()
            return
        if '@' not in emailid:
messagebox.showerror(master=top, title="Error",
                             message="Invalid Email")
top.lift()
            return
        try:
            con = MySQLdb.connect(host='localhost', user=user,
                              password=password, database="airline_reservation")
            cursor = con.cursor()
cursor.execute("INSERT INTO booking(firstname,lastname,phonenumber, emailid, address,gender,
age,pnr,source,destination,date,time,airline_name,class) VALUES
(%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s)",
                         (firstName, lastName, num, emailid, address, gender, age, pnr, self.source_city,
self.destination_city, self.travel_details["date"], self.travel_details["time"], self.airline_details["name"],
self.travel_details["travel_class"]))
```

```
con.commit()
messagebox.showinfo(master=top, title="Booking successful",
                    message="Your ticket has been booked successfully!")
top.destroy()
        Invoice(pnr)

    except Exception as e:
        print(e)
messagebox.showerror(master=top, title="Error",
                     message="Error\nUnable to book.")
top.lift()

  def widgets(self, window):
Label(window, text="Confirm Booking", font=(
        "Times New Roman", 30)).place(x=250, y=10)
    frame = Frame(window, height=120, width=700, background='WHITE',
borderwidth=1, relief='sunken').place(x=50, y=70)

    load = Image.open(self.airline_details['logo'])
    load = load.resize((120, 80), Image.ANTIALIAS)
    render = ImageTk.PhotoImage(load)
img = Label(window, image=render)
img.image = render
img.place(x=70, y=90)

Label(window, text="DEPARTURE", font=(
        "Times New Roman", 10)).place(x=270, y=90)
Label(window, text=self.source_city, bg='WHITE', font=(
        "Times New Roman", 25)).place(x=230, y=110, width=150)
Label(window, text=self.travel_details["source"], bg='WHITE', fg='GRAY', font=(
        "Times New Roman", 10)).place(x=210, y=150, width=220)
```

```
load = Image.open("src/arrow.png")
load = load.resize((50, 40), Image.ANTIALIAS)
        render = ImageTk.PhotoImage(load)
img = Label(window, image=render, bg='WHITE')
img.image = render
img.place(x=385, y=110)

Label(window, text="Time: "+self.travel_details["time"], bg="WHITE", font=(
            "Times New Roman", 8)).place(x=385, y=168)

Label(window, text="ARRIVAL", font=(
            "Times New Roman", 10)).place(x=500, y=90)
Label(window, text=self.destination_city, bg='WHITE', font=(
            "Times New Roman", 25)).place(x=460, y=110, width=150)
Label(window, text=self.travel_details["destination"], bg='WHITE', fg='GRAY', font=(
            "Times New Roman", 10)).place(x=440, y=150, width=220)

Label(window, text="RATE", font=(
            "Times New Roman", 10)).place(x=660, y=90)
Label(window, text="₹"+str(self.airline_details["rate"]), bg='WHITE', font=(
            "Times New Roman", 15)).place(x=625, y=115, width=100)
Label(window, text="DATE", font=(
            "Times New Roman", 10)).place(x=660, y=140)
Label(window, text=str(self.travel_details["date"]), bg='WHITE', font=(
            "Times New Roman", 10)).place(x=625, y=165, width=100)

    def inputDetails(self, window):
Label(window, text="First Name:", font=(
            "Times New Roman", 15)).place(x=60, y=220)
Entry(window, width=30, textvar=self.first_name).place(x=200, y=225)
```

```
Label(window, text="Last Name:", font=(
        "Times New Roman", 15)).place(x=410, y=220)
Entry(window, width=30, textvar=self.last_name).place(x=550, y=225)

Label(window, text="Billing Address:",  font=(
        "Times New Roman", 15)).place(x=60, y=280)
Entry(window, width=90, textvar=self.address).place(x=200, y=285)

Label(window, text="Email Address:", font=(
        "Times New Roman", 15)).place(x=60, y=340)
Entry(window, width=30, textvar=self.email).place(x=200, y=345)

Label(window, text="Phone Number:", font=(
        "Times New Roman", 15)).place(x=410, y=340)
Entry(window, width=30, textvar=self.phone).place(x=550, y=345)

Label(window, text="Gender:", font=(
        "Times New Roman", 15)).place(x=60, y=400)
Radiobutton(window, text="Male", padx=5, variable=self.Gender,
            value=1).place(x=205, y=405)
Radiobutton(window, text="Female", padx=5,
            variable=self.Gender, value=2).place(x=305, y=405)
Radiobutton(window, text="Other", padx=5,
            variable=self.Gender, value=3).place(x=400, y=405)

Label(window, text="Age:", font=(
        "Times New Roman", 15)).place(x=60, y=460)
Entry(window, width=10, textvar=self.age).place(x=200, y=465)

Button(window, text='Submit', width=15, height=2, bg='brown',
fg='white', command=lambda: self.onSubmit(window)).place(x=345, y=510)
```

8.7 FullscreenModule:

```
class FullScreenApp(object):
    def__init_(self, master, **kwargs):
self.master = master
        pad = 3
self._geom = '200x200+0+0'
master.geometry("{0}x{1}+0+0".format(
master.winfo_screenwidth()-pad, master.winfo_screenheight()-pad))
master.bind('<Escape>', self.toggle_geom)

    def toggle_geom(self,  event):
geom = self.master.winfo_geometry()
self.master.geometry(self._geom)
self._geom = geom
```

8.8 InvoiceModule:

```
from tkinter import Frame, Label, Tk, Toplevel, messagebox
import MySQLdb
from configure import user,password
import json
from PIL import ImageTk, Image
from airlineTypes import airline_type

class Invoice:
    def__init_(self,pnr):
        window = Toplevel()
        # window = Tk()
window.title("Invoice - Udan Khatola")
window.geometry("800x600")

self.data = []
```

```
self.pnr = pnr


self.database(window)
self.loadJson()
self.widgets(window)
window.mainloop()


    def database(self,top):
        try:
            con = MySQLdb.connect(host='localhost', user=user,
                                  password=password, database="airline_reservation")
            cursor = con.cursor()
            command = "SELECT * FROM booking"
cursor.execute(command)
            data = cursor.fetchall()
            for current in data:
                if self.pnr==current[7]:
self.data = current
        except Exception as e:
            print(e)
top.lift()
messagebox.showerror("Error","Could not connect")
        return


    def loadJson(self):
        data = open('json_data/airports.json',)
json_data = json.load(data)
airport_data = json_data["airports"]
        for airports in airport_data:
            if airports["city_name"] == self.data[8]:
self.source_airport = airports["airport_name"]
```

```
            if airports["city_name"] == self.data[9]:
self.destination_airport = airports["airport_name"]
data.close()

    def widgets(self,top):
        frame = Frame(top,height = 80,width = 800,background = 'WHITE', borderwidth = 1, relief =
'sunken').pack()
Label(top, text="INVOICE",bg='WHITE', font=("Times New Roman", 30)).place(x=300, y=10)

Label(top, text=self.data[8], font=("Times New Roman", 25)).place(x=100, y=100,width=200)
Label(top, text="("+self.source_airport+")",fg='grey', font=("Times New Roman", 15)).place(x=50,
y=150,width=350)

        load = Image.open("src/arrow.png")
        load = load.resize((70, 40), Image.ANTIALIAS)
        render = ImageTk.PhotoImage(load)
img = Label(top, image=render)
img.image = render
img.place(x=360, y=105)

Label(top, text=self.data[9], font=("Times New Roman", 25)).place(x=500, y=100,width=200)
Label(top, text="("+self.destination_airport+")",fg='grey', font=("Times New Roman", 15)).place(x=450,
y=150,width=350)

        logo = airline_type[self.data[12]]["logo"]
        load = Image.open(logo)
        load = load.resize((150, 100), Image.ANTIALIAS)
        render = ImageTk.PhotoImage(load)
img = Label(top, image=render)
img.image = render
img.place(x=150, y=200)
```

```
Label(top, text="Date of departure: ", font=("Times New Roman", 15)).place(x=350, y=200)
Label(top, text="Time of departure: ", font=("Times New Roman", 15)).place(x=350, y=240)
Label(top, text="Class: ", font=("Times New Roman", 15)).place(x=350, y=280)

Label(top, text=self.data[10], font=("Times New Roman", 15)).place(x=510, y=200)
Label(top, text=self.data[11], font=("Times New Roman", 15)).place(x=510, y=240)
Label(top, text=self.data[-2], font=("Times New Roman", 15)).place(x=410, y=280)

Label(top, text="Name: ", font=("Times New Roman", 15)).place(x=200, y=350)
Label(top, text="Email ID: ", font=("Times New Roman", 15)).place(x=200, y=390)
Label(top, text="PNR: ", font=("Times New Roman", 15)).place(x=200, y=430)

Label(top, text=self.data[0] +" "+ self.data[1], font=("Times New Roman", 15)).place(x=300, y=350)
Label(top, text=self.data[3], font=("Times New Roman", 15)).place(x=300, y=390)
Label(top, text=self.data[7], font=("Times New Roman", 15)).place(x=300, y=430)

    rate = airline_type[self.data[-3]]["rate"]
Label(top, text="Amount Paid: ", font=("Times New Roman", 15)).place(x=450, y=500)
Label(top, text="₹ "+str(rate), font=("Times New Roman", 15)).place(x=575, y=500)

# Invoice('Ohvgs9')
```

8.9 Past BookingsModule:

```
from tkinter import Button, Frame, Label, Tk, Toplevel
import MySQLdb
from configure import user,password


class PastBookings:
   def__init_(self,user_id):
      top = Toplevel()
```

```
        # top = Tk()
top.title("Past bookings - Udan Khatola")
top.geometry("800x400")

self.user_email = user_id

self.widgets(top)
top.mainloop()

    def widgets(self,top):
        frame = Frame(top,height = 80,width = 800,background = 'WHITE', borderwidth = 1, relief =
'sunken').pack()
Label(top, text="Past Bookings",bg='WHITE', font=("Times New Roman", 30)).place(x=275, y=10)
        try:
            con = MySQLdb.connect(host='localhost', user=user,
                                  password=password, database="airline_reservation")
            cursor = con.cursor()
cursor.execute("SELECT * FROM booking")
            data = cursor.fetchall()
            frame = Frame(top,height = 24,width = 800,bg='GRAY', borderwidth = 1, relief = 'sunken').place(y = 79)
Label(top, text="Airline",bg='GRAY', font=("Times New Roman", 10)).place(x=30, y=80)
Label(top, text="Departure",bg='GRAY', font=("Times New Roman", 10)).place(x=140, y=80)
Label(top, text="Arrival",bg='GRAY', font=("Times New Roman", 10)).place(x=270, y=80)
Label(top, text="Date",bg='GRAY', font=("Times New Roman", 10)).place(x=430, y=80)
Label(top, text="Time",bg='GRAY', font=("Times New Roman", 10)).place(x=365, y=80)
Label(top, text="PNR",bg='GRAY', font=("Times New Roman", 10)).place(x=510, y=80)
Label(top, text="Seat No.",bg='GRAY', font=("Times New Roman", 10)).place(x=580, y=80)
Label(top, text="Class",bg='GRAY', font=("Times New Roman", 10)).place(x=700, y=80)
y_height = 100
            flag = 0
            for users in data:
```

```
            if users[3]==self.user_email:
                flag += 1
                frame = Frame(top,height = 30,width = 800, borderwidth = 1, relief = 'sunken').place(y = y_height)
rel = 'ridge'
btn = Label(top, text=users[12],relief=rel,font=('Times New Roman',12),padx=2,pady=3,width=10)
btn.place(y=y_height)
btn = Label(top, text=users[8],relief=rel,font=('Times New Roman',12),padx=2,pady=3,width=15)
btn.place(x=97,y=y_height)
btn = Label(top, text=users[9],relief=rel,font=('Times New Roman',12),padx=2,pady=3,width=15)
btn.place(x=220,y=y_height)
btn = Label(top, text=users[11],relief=rel,font=('Times New Roman',12),padx=1,pady=3,width=6)
btn.place(x=351,y=y_height)
btn = Label(top, text=users[10],relief=rel,font=('Times New Roman',12),padx=1,pady=3,width=8)
btn.place(x=410,y=y_height)
btn = Label(top, text=users[7],relief=rel,font=('Times New Roman',12),padx=1,pady=3,width=8)
btn.place(x=486,y=y_height)
                if users[14]==None:
btn = Label(top, text='-',relief=rel,font=('Times New Roman',12),padx=1,pady=3,width=8)
btn.place(x=563,y=y_height)
                else:
btn = Label(top, text=users[-1],relief=rel,font=('Times New Roman',12),padx=1,pady=3,width=8)
btn.place(x=563,y=y_height)
btn = Label(top, text=users[13],relief=rel,font=('Times New Roman',12),padx=1,pady=3,width=17)
btn.place(x=640,y=y_height)
y_height += 30
            else:
                pass
        if flag==0:
Label(top, text="No Bookings Found!", font=("Times New Roman", 30)).place(x=230, y=150
    except Exception as e:
        print(e)
```

8.10 Registration Module:

```
import tkinter as tk
from tkinter import Tk, Label, messagebox, StringVar, IntVar, Entry, Radiobutton, Button, Listbox, Toplevel,
Checkbutton
from configure import user, password
import MySQLdb



class BLabel(object):
    b = "•"

    def__init_(self, master):
        import tkinter as tk
self.l = tk.Label(master)

    def add_option(self, text):
        if self.l.cget("text") == "":
self.l.config(text=self.b+" "+text)
        else:
self.l.config(text=self.l.cget("text") + "\n" + self.b + " "+text)



class Register:
    def__init (self):
        top = Toplevel()
top.title("Registaration - Udan Khatola")
top.geometry("500x650+400+25")

self.first_name = StringVar()
self.last_name =  StringVar()
self.email = StringVar()
```

```
self.number = StringVar()
self.Gender = IntVar()
self.age = StringVar()
self.passwor = StringVar()
self.checkButton = IntVar()

self.buttons(top)
top.mainloop()

    def checkbox(self, root):
self.check_value = self.checkButton.get()
        if self.check_value == 0:
self.entry_6.configure(show="*")
        else:
self.entry_6.configure(show="")

    def isFormEmpty(self):
        if(self.first_name.get() == '' or self.last_name.get() == '' or self.email.get() == '' or self.number.get() == '' or
self.Gender.get() == 0 or self.age.get() == '' or self.passwor == ''):
            return True

    def database(self, top):
        check = True
firstName = self.first_name.get()
lastName = self.last_name.get()
emailid = self.email.get()
        num = self.number.get()
        g = self.Gender.get()
        Age = self.age.get()
passw = self.passwor.get()
        if self.isFormEmpty() == True:
```

```
messagebox.showerror(master=top,title="Form Empty",message= "Please enter all details!")
top.lift()
            return
        if '@' not in emailid:
messagebox.showerror(master=top,title="Error",message= "Invalid Email")
top.lift()
            return


        try:
            con = MySQLdb.connect(host='localhost', user=user,
                                  password=password, database="airline_reservation")
            cursor = con.cursor()
            command = 'SELECT emailid FROM registration'
cursor.execute(command)
registered_email = cursor.fetchall()
            for ids in registered_email:
if(emailid == ids[0]):
                check = False
            if(check):
               if(len(passw) < 8):
messagebox.showwarning(
                    master=top,title="Change password",message= "Password too short!")
top.lift()
                return
elif(len(passw) > 20):
messagebox.showwarning(
                    master=top,title="Change password",message= "Password too long!")
top.lift()
                return
elif(len(passw) >= 8 and len(passw) <= 20):
                a = b = c = d = 0
```

```
            for p in passw:
                if (p.islower()):
                    a = 1
                if (p.isdigit()):
                    b = 1
                if (p.isupper()):
                    c = 1
                if (p == "$" or p == "#" or p == "@"):
                    d = 1
if(a != 1):
messagebox.showwarning(
                    master=top,title="Change password",message= "Password should contain lower characters!")
top.lift()
                return
if(c != 1):
messagebox.showwarning(
                    master=top,title="Change password",message= "Password should contain upper characters!")
top.lift()
                return
if(b != 1):
messagebox.showwarning(
                    master=top,title="Change password",message= "Password should contain a digit!")
top.lift()
                return
if(d != 1):
messagebox.showwarning(
                    master=top,title="Change password",message= "Password should contain $ or # or @!")
top.lift()
                return


cursor.execute("INSERT INTO registration VALUES(%s,%s,%s,%s,%s,%s,%s)",
```

```
                    (firstName, lastName, num, emailid, passw, g, Age,))
con.commit()
messagebox.showinfo(master=top,title="Registration successful",
                        message="You have been registered successfully!")
top.destroy()
        else:
messagebox.showwarning(
            master=top,title="Oops",message= "Email id already registered!")
top.destroy()
Register()

    except Exception as e:
        print(e)
messagebox.showerror(master=top,title="Error",message= "Error\nUnable to register.")
top.lift()

  def buttons(self, top):
    label_0 = Label(top, text="Registration Form",
                width=20, font=("bold", 20))
    label_0.place(x=100, y=23)

    label_1 = Label(top, text="First Name",
                width=20, font=("bold", 10))
    label_1.place(x=75, y=100)
    entry_1 = Entry(top, width=30, textvar=self.first_name)
    entry_1.place(x=210, y=100)

    label_2 = Label(top, text="Last Name", width=20, font=("bold", 10))
    label_2.place(x=75, y=150)
    entry_2 = Entry(top, width=30, textvar=self.last_name)
    entry_2.place(x=210, y=150)
```

```
        label_3 = Label(top, text="Email", width=20, font=("bold", 10))
        label_3.place(x=75, y=200)
        entry_3 = Entry(top, width=30, textvar=self.email)
        entry_3.place(x=210, y=200)

        label_4 = Label(top, text="Phone Number",
                        width=20, font=("bold", 10))
        label_4.place(x=75, y=250)
        entry_4 = Entry(top, width=20, textvar=self.number)
        entry_4.place(x=210, y=250)

        label_5 = Label(top, text="Gender", width=20, font=("bold", 10))
        label_5.place(x=75, y=300)
Radiobutton(top, text="Male", padx=5, variable=self.Gender,
                    value=1).place(x=205, y=300)
Radiobutton(top, text="Female", padx=20,
                    variable=self.Gender, value=2).place(x=260, y=300)

        label = Label(top, text="Age", width=20, font=("bold", 10))
label.place(x=75, y=350)
        entry = Entry(top, textvar=self.age)
entry.place(x=210, y=350)

        label_6 = Label(top, text="Password",
                        width=20, font=("bold", 10))
        label_6.place(x=75, y=400)
self.entry_6 =Entry(top, textvar=self.passwor, show="*")
self.entry_6.place(x=210, y=400)
Checkbutton(top, text = "Show password",
                    variable = self.checkButton,
onvalue = 1,
```

```
offvalue = 0,
                height = 2,
                width = 10,
                command = lambda: self.checkbox(top)).place(x=210,y=420)


        Lb1 = BLabel(master=top)
        Lb1.add_option("At least 1 letter between [a-z]")
        Lb1.add_option("At least 1 number between [0-9]")
        Lb1.add_option("At least 1 letter between [A-Z]")
        Lb1.add_option("At least 1 character from [$#@]")
        Lb1.add_option("Minimum length of password: 8")
        Lb1.add_option("Maximum length of password: 15")
Lb1.l.place(x=155, y=460)
Button(top, text='Submit', width=20, height=2, bg='brown',
fg='white', command=lambda: self.database(top)).place(x=180, y=560)


# Register()
```

8.11 Seat selectionModule:

```
from tkinter import Button, Canvas, Frame, Label, Tk, Toplevel, messagebox
from configure import user,password
from boardingPass import BoardingPass
import MySQLdb


class SelectSeat:
    def__init_(self,pnr):
        # top = Toplevel()
        top = Tk()
top.title("Book a Ticket - Udan Khatola")
top.geometry("400x600")
```

```
self.pnr = pnr
self.prev = None
self.widgets(top)
top.mainloop()


    def color_change(self,i,top):
        if self.prev!=None:
self.prev["bg"]='grey'
i["bg"] = "blue"
button_value = str(i)
        index = button_value[8:]
        if index==":
seat_num = self.seat_number(0)
        else:
            index = int(index)
seat_num = self.seat_number(index-1)
self.button.config(text=seat_num)
self.prev = i
Button(top, text='Submit', width=10, bg='brown',
fg='white', command=lambda:self.database(seat_num,top)).place(x=270, y=550)


    def database(self,seat_num,top):
        try:
            con = MySQLdb.connect(host='localhost', user=user,
                                  password=password, database="airline_reservation")
            cursor = con.cursor()
cursor.execute("UPDATE booking SET seat=(%s) WHERE pnr=(%s)",(seat_num,self.pnr,))
con.commit()
messagebox.showinfo(master=top, title="Success",
                                message="Your seat has been selected successfully!")
top.destroy()
```

```
BoardingPass(self.pnr)
        except Exception as e:
            print(e)
messagebox.showerror(master=top,title="Error",message="Could not select seat!")


    def seat_number(self,index):
lst = []
        for i in range(1,16):
            for j in ['A','B','C','D','E','F']:
lst.append(str(i)+j)
        return lst[index]


    def widgets(self,top):
        frame = Frame(top, height=510, width=250, background='WHITE',
borderwidth=1, relief='sunken').place(x=75)
Label(top, text="A",bg="WHITE", font=(
            "Times New Roman", 15)).place(x=95, y=10)
Label(top, text="B",bg="WHITE", font=(
            "Times New Roman", 15)).place(x=125, y=10)
Label(top, text="C",bg="WHITE", font=(
            "Times New Roman", 15)).place(x=155, y=10)
Label(top, text="D",bg="WHITE", font=(
            "Times New Roman", 15)).place(x=225, y=10)
Label(top, text="E",bg="WHITE", font=(
            "Times New Roman", 15)).place(x=255, y=10)
Label(top, text="F",bg="WHITE", font=(
            "Times New Roman", 15)).place(x=285, y=10)


y_height = 0
        for i in range (1,10):
Label(top, text=i,bg="WHITE", font=(
```

```
        "Times New Roman", 12)).place(x=195, y=35+y_height)
y_height += 30
    for i in range (10,16):
Label(top, text=i,bg="WHITE", font=(
        "Times New Roman", 12)).place(x=190, y=50+y_height)
y_height += 30


    h=0
    for i in range(1,16):
        w =  0
        if i<10:
            for j in range(6):
                if(j<3):
btn = Button(top, width=2,bg='grey')
btn.place(x=92+w,y=35+h)
btn["command"]=lambda btn = btn: self.color_change(btn,top)
                    w += 32
                else:
btn = Button(top, width=2,bg='grey')
btn.place(x=125+w,y=35+h)
btn["command"] = lambda btn = btn:self.color_change(btn,top)
                    w += 32
        else:
            for j in range(6):
                if(j<3):
btn = Button(top, width=2,bg='grey')
btn.place(x=92+w,y=50+h)
btn["command"] = lambda btn = btn:self.color_change(btn,top)
                    w += 32
                else:
btn = Button(top, width=2,bg='grey')
```

```
btn.place(x=125+w,y=50+h)
btn["command"] = lambda btn = btn:self.color_change(btn,top)
                    w += 32
            h += 30

self.button = Label(top, text="", width=5,height=1,bg="white", font=("Times New Roman",20,"bold"))
self.button.place(x=100,y=530)

# SelectSeat()
```

8.12 SelectAirline:

```
from tkinter import Button, Canvas, Frame, Label, PhotoImage, Tk, Toplevel
from airlineTypes import airline_type
import json
from PIL import Image, ImageTk
from bookingDetails import Details


class Airline:
    def__init_(self,booking_ticket):
        window = Toplevel()
        # window = Tk()
window.title("Book a Ticket")
window.geometry("800x600")

self.booking_ticket = booking_ticket
self.source_city = ''
self.destination_city = ''

self.loadJson()
self.button(window)
```

```
window.mainloop()


    def loadJson(self):
        data = open('json_data/airports.json',)
json_data = json.load(data)
airport_data = json_data["airports"]
        for airports in airport_data:
            if airports["airport_name"]==self.booking_ticket["source"]:
self.source_city = airports["city_name"]
            if airports["airport_name"]==self.booking_ticket["destination"]:
self.destination_city = airports["city_name"]
data.close()


    def back(self,top):
top.destroy()
        from bookTicket import Booking
Booking()


    def confirmBooking(self,details,top):
top.destroy()
        Details(details,self.booking_ticket)


    def button(self,window):
        heading = "Flights from "+self.source_city+" to "+self.destination_city
        label = Label(window, text=heading, padx="3",
pady="3", font=("Times New Roman", 25)).pack()


y_height=50
        for a in airline_type:
            frame = Frame(window,height = 120,width = 800,background = 'WHITE', borderwidth = 1, relief =
'sunken').place(y=y_height)
```

```
        load = Image.open(airline_type[a]['logo'])
        load = load.resize((120, 80), Image.ANTIALIAS)
        render = ImageTk.PhotoImage(load)
img = Label(window,image=render)
img.image =  render
img.place(x=40, y=20+y_height)

Label(window, text="Departure:", font=("Times New Roman", 15)).place(x=300, y=15+y_height)
Label(window, text=self.booking_ticket["time"],bg='WHITE', font=("Times New Roman",
15,'bold')).place(x=320, y=55+y_height)
        text = self.booking_ticket["source"] + " (" + self.source_city+")"
Label(window, text=text,bg='WHITE',fg='GRAY', font=("Times New Roman", 10)).place(x=280,
y=85+y_height)

Label(window, text="Price:", font=("Times New Roman", 15)).place(x=500, y=15+y_height)
Label(window, text='₹'+str(airline_type[a]['rate']),bg='WHITE', font=("Times New Roman",
15,'bold')).place(x=500, y=55+y_height)

y_height += 120

Button(window, text='Book', width=10, height=2, bg='brown',
            fg='white',command=lambda:self.confirmBooking(airline_type["Vistara"],window)).place(x=650,
y=90)
Button(window, text='Book', width=10, height=2, bg='brown',
            fg='white',command=lambda:self.confirmBooking(airline_type["AirAsia"],window)).place(x=650,
y=210)
Button(window, text='Book', width=10, height=2, bg='brown',
            fg='white',command=lambda:self.confirmBooking(airline_type["Indigo"],window)).place(x=650,
y=330)
Button(window, text='Book', width=10, height=2, bg='brown',
fg='white',command=lambda:self.confirmBooking(airline_type["Air India"],window)).place(x=650, y=450)
```

```
Button(window, text='Change details', width=15, height=2, bg='brown',
fg='white',command=lambda:self.back(window)).place(x=345, y=545)

# Airline({'source': 'Amausi Airport', 'destination': 'Bhavnagar Airport', 'calendar': '4/30/21', 'time': '19:30',
'travel_class': 'Economy Class'})
```

8.13 Webcheckin:

```
import tkinter as tk
from tkinter import Button, Entry, IntVar, Label, Listbox, Radiobutton, StringVar, Tk, Toplevel, messagebox
from configure import user, password
from seatSelection import SelectSeat
from boardingPass import BoardingPass
import MySQLdb

class WebCheckin:
    def__init_(self):
        top = Toplevel()
        # top = Tk()
top.title("Udan Khatola")
top.geometry("500x300")

self.pnr_value = StringVar()
self.buttons(top)
top.mainloop()

    def checkPNR(self,top):
pnr = self.pnr_value.get()
        try:
            con = MySQLdb.connect(host='localhost', user=user,
                                  password=password, database="airline_reservation")
            cursor = con.cursor()
```

```
cursor.execute("SELECT pnr,seat FROM booking")
pnr_data = cursor.fetchall()
            for data in pnr_data:
                if pnr == data[0]:
                    if data[1]==None:
top.destroy()
SelectSeat(pnr)
                        return
                    else:
top.destroy()
BoardingPass(pnr)
messagebox.showerror(
                master=top, title="Error", message="Invalid PNR")


top.lift()


  except Exception as e:


    print(e)


    def buttons(self,top):
        label_1 = Label(top, text="Enter PNR number:", width=20, font=("bold", 10))
        label_1.place(x=50, y=100)
        entry_1 = Entry(top, width=30,textvar=self.pnr_value)
        entry_1.place(x=210, y=100)


Button(top, text='Submit', width=20, bg='brown',
fg='white', command=lambda:self.checkPNR(top)).place(x=180, y=150)


# WebCheckin()
```

Chapter-9

Result and Discussion (Screen shots of the implementation with front end)

9.1 GUI Output:

9.1.1 Home Page:

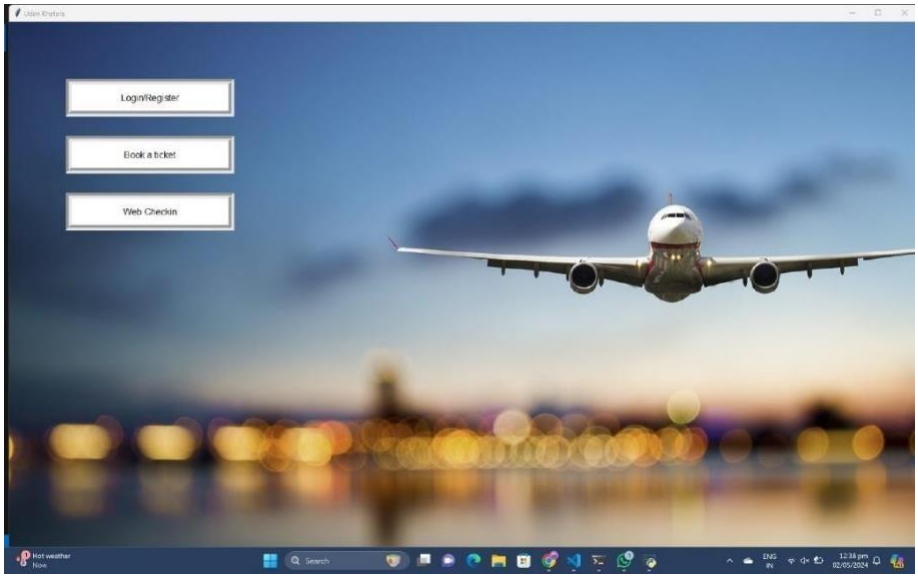


Fig 9.1

The project's home page serves as the gateway to essential functionalities, including login/registration, booking, and web check-in. Users are greeted with a user-friendly interface that seamlessly guides them through the booking process. The login/registration feature offers secure access for both new and returning users, ensuring personalized experiences. Booking functionality allows users to easily search for flights, select seats, and confirm reservations with convenience. Web check-in further enhances user experience by enabling hassle-free pre-flight procedures, ensuring a smooth journey from start to finish. With intuitive design and comprehensive features, the home page sets the stage for seamless travel planning and management.

9.1.2 Login/Register:

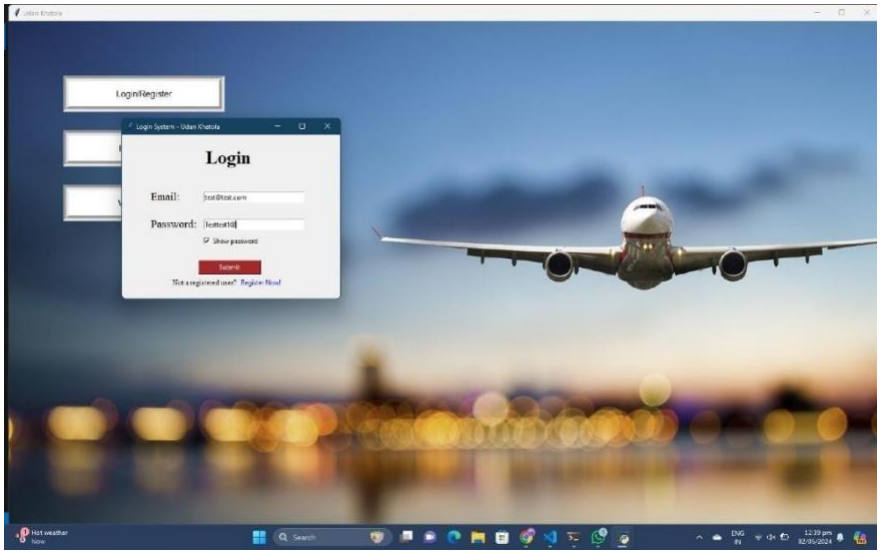


Fig 9.2

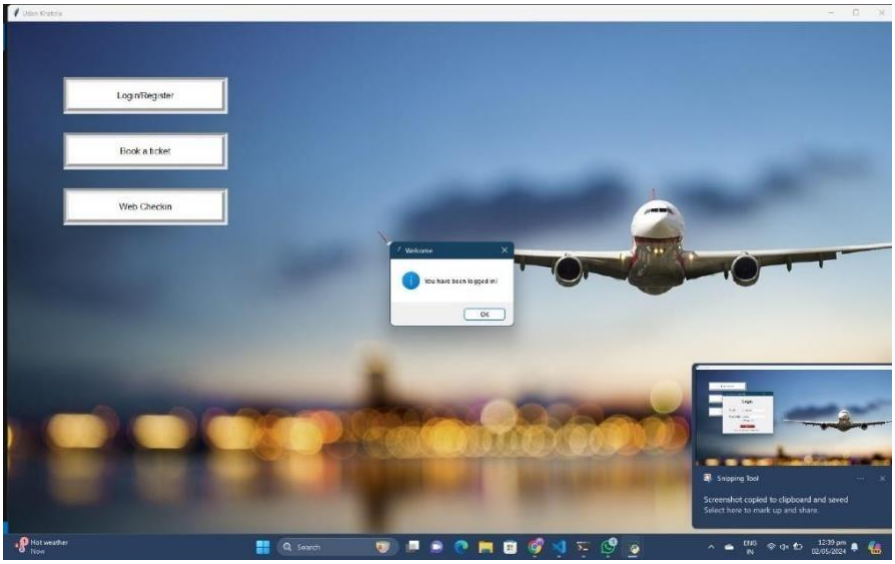


Fig 9.3

The login page of the project serves as the entry point for authenticated access, providing a secure environment for users to sign in to their accounts. With a clean and intuitive interface, users are prompted to enter their credentials, ensuring a seamless login experience. Error handling mechanisms are in place to notify users of any incorrect inputs, enhancing usability and reducing frustration. Additionally, the login page may include options for password recovery or account registration for new users, offering a comprehensive approach to user authentication. Overall, the login page prioritizes security, usability, and accessibility, laying the foundation for a reliable and user-centric platform.

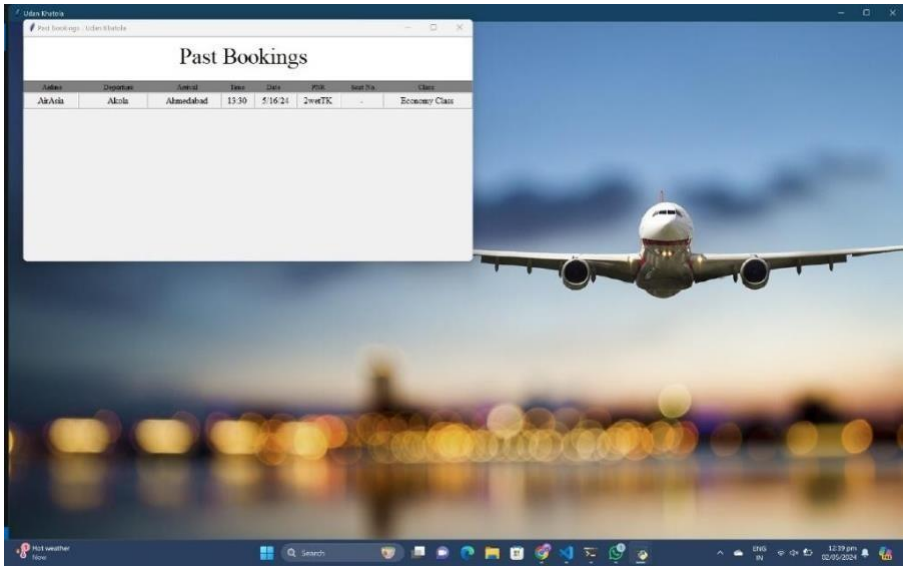


Fig 9.4

The past booking details page provides users with a comprehensive overview of their previous flight reservations, offering valuable insights and easy access to historical travel information. Users can conveniently review details such as flight numbers, departure/arrival times, seat assignments, and payment statuses for each booking. With a user-friendly interface, the page allows for quick navigation and retrieval of past booking records, facilitating efficient trip planning and management. Additionally, users may have the option to filter or search for specific bookings based on various criteria, enhancing the overall usability of the feature. Whether for reference, review, or future planning, the past booking details page serves as a valuable resource for users to track their travel history and make informed decisions.

9.1.3 Flight Booking:

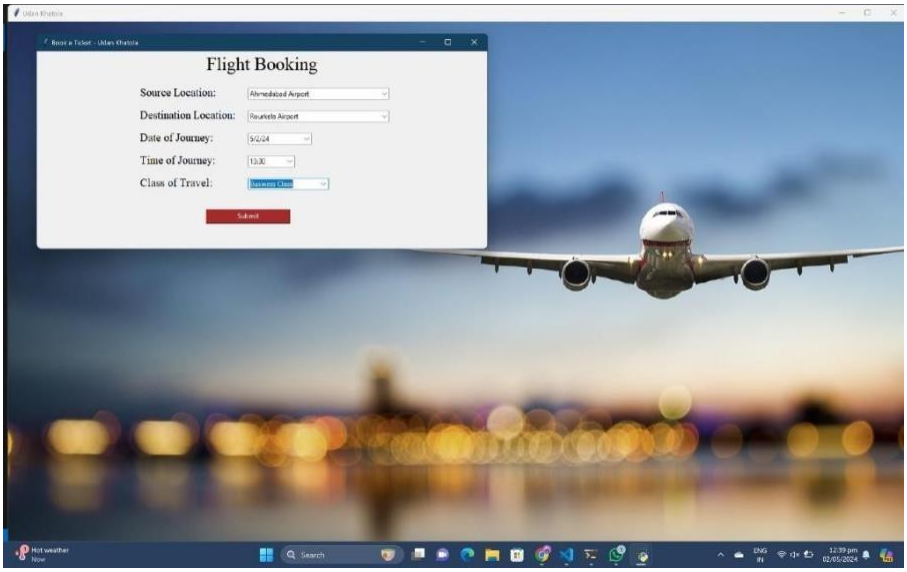


Fig 9.5

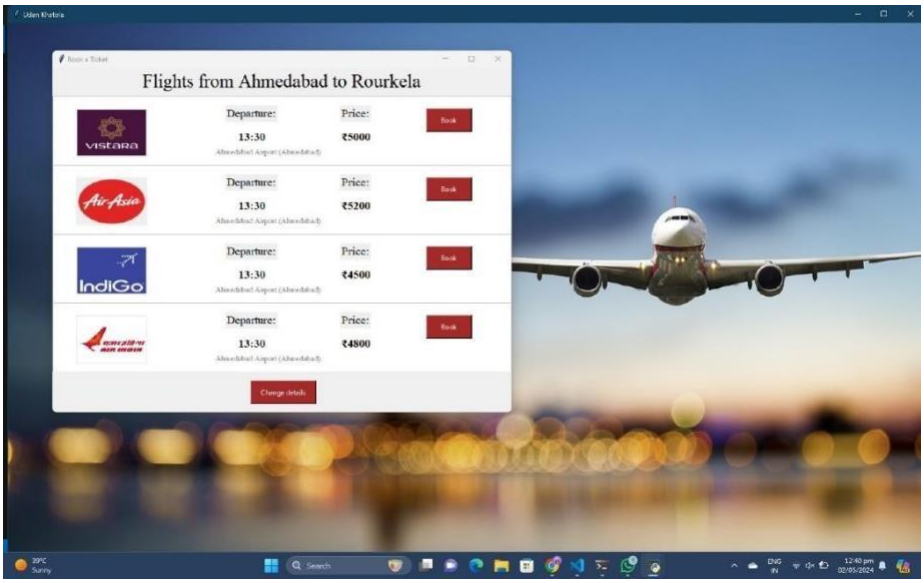


Fig 9.6

Flight booking functionality enables users to search for and reserve flights seamlessly. Users input their travel details like departure and destination cities, travel dates, and passenger information. The system then presents available flight options, allowing users to compare and select flights based on preferences such as price, schedule, and airline. Once a flight is chosen, users proceed to book their seats.Users can refine their search results using advanced filtering options to tailor their flight selection further. These filters may include criteria such as layover duration, departure/arrival times, preferred airlines,

and specific flight amenities. Additionally, interactive features such as seat maps enable users to preview seating arrangements and select preferred seats before finalizing their booking.

Once users have chosen their desired flight, they proceed to the booking stage, where they provide necessary passenger details and payment information for confirmation. The booking process is designed to be secure, efficient, and user-friendly, guiding users through each step with clear instructions and prompts.

9.1.4 Invoice:

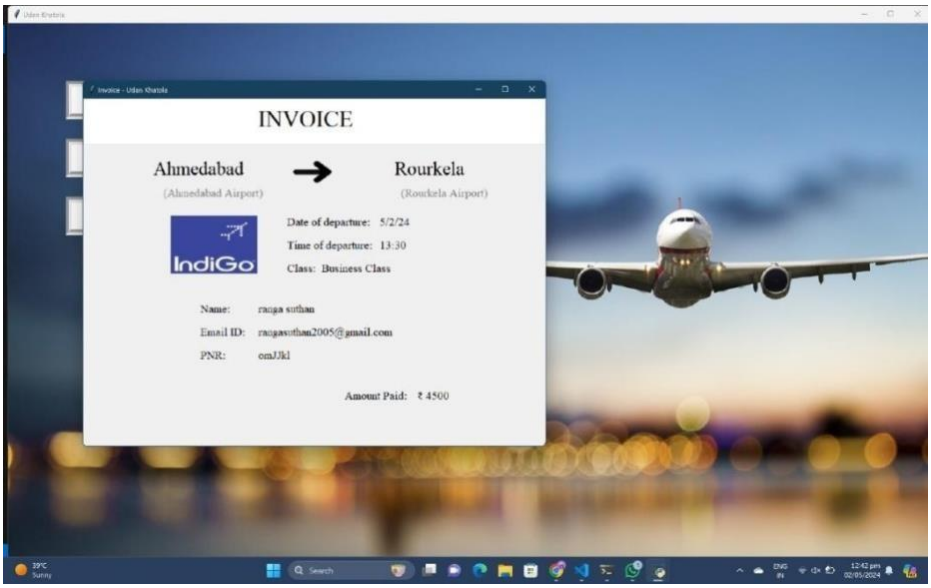


Fig 9.7

The invoice feature in the airline reservation system generates a detailed summary of the user's booking transaction. It compiles essential information related to the booked flight(s) and associated charges, providing users with a clear breakdown of their expenses. The invoice typically includes details such as flight itinerary (departure/arrival cities, dates, and times), passenger names, seat assignments, fare breakdown (base fare, taxes, fees), total amount paid, payment method, and booking reference number.

Chapter-10

Online course certificate

Name: T Sri Rangasuthan

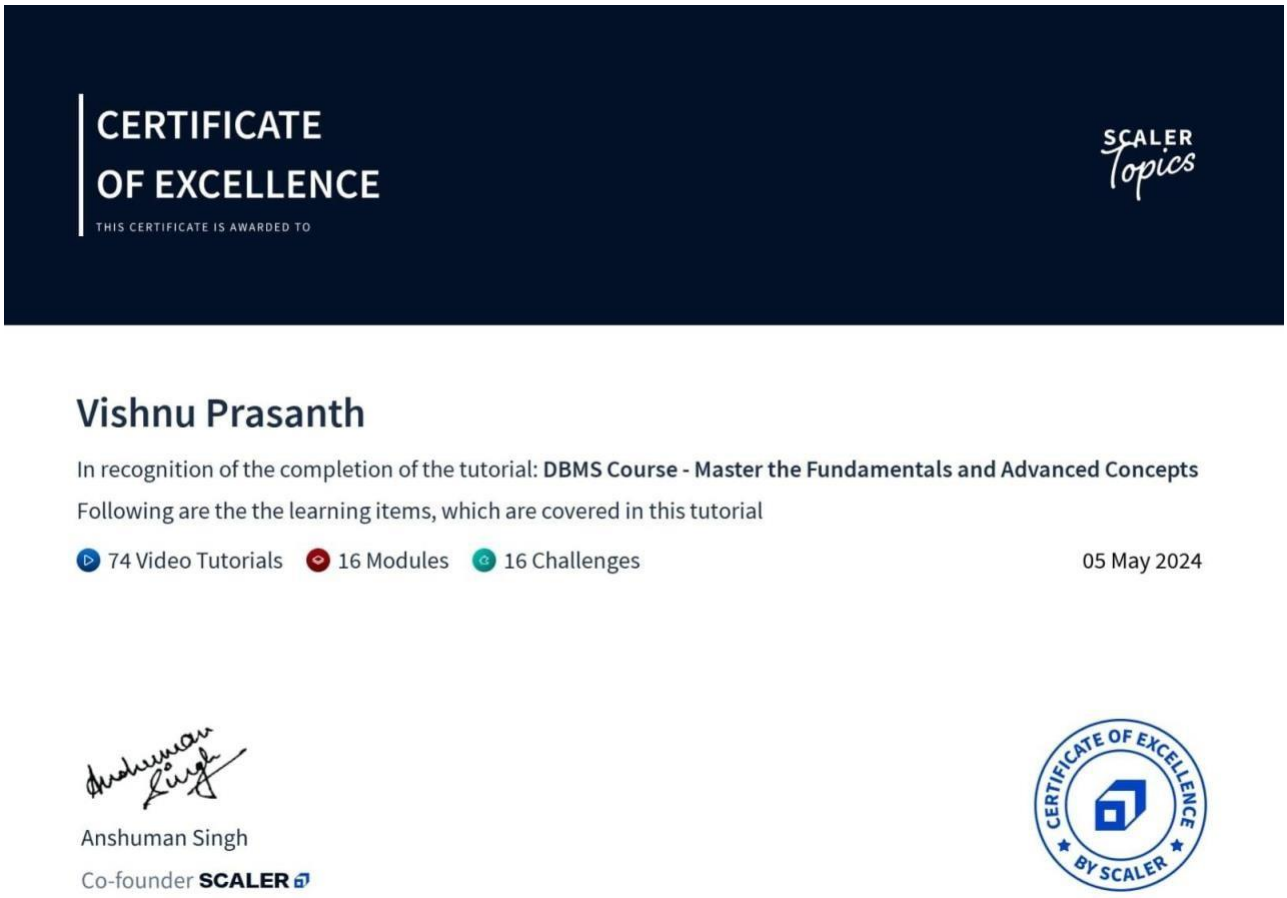
Register Number: RA2211003011319

Name: G.midhun chakravarthy yadav

Register Number: RA2211003011289

Name: M.Vishnu prasanth

Register Number: RA2211003011326



Chapter-11

Conclusion

The completion of the airline reservation system project marks a significant milestone in the realm of aviation technology. From its inception to execution, the project has traversed through meticulous planning, rigorous analysis, and diligent implementation phases. The initial stages involved comprehensive analysis to identify the core entities and functionalities essential for an efficient reservation system. This laid the groundwork for the subsequent ER modeling phase, which visually depicted the intricate relationships between entities and their attributes. With the ER model as a guide, the project transitioned into the database design phase, where the schema was meticulously crafted to ensure data integrity and adherence to normalization principles. Challenges emerged throughout the journey, ranging from ensuring referential integrity to optimizing transaction handling, but collaborative problem-solving and iterative development strategies proved effective in overcoming these hurdles. Looking forward, there are ample opportunities for further enhancement, such as integrating additional features like seat selection and itinerary planning, as well as continuous monitoring and optimization to ensure scalability and reliability. In conclusion, the successful completion of the airline reservation system project underscores the importance of thorough planning, effective communication, and adaptability in software development endeavors, poised to meet the evolving needs of passengers and airlines in the dynamic aviation industry.

REFERENCES

Research Papers:

1. https://www.researchgate.net/publication/374391896_Airline_Reservation_System
2. https://www.researchgate.net/publication/370050120_Flight_Reservation_System

Journal Articles:

1. https://www.linkedin.com/pulse/article-airline-reservation-system-lohitha-rongala?utm_source=share&utm_medium=member_android&utm_campaign=share_via

Books:

1. Smith, J. (2019). Database Systems: The Complete Book. Pearson.
2. Database Systems: Design, Implementation, and Management by Carlos Coronel, Steven Morris, and Peter Rob.