Flight Booking Management System

Project submitted to the

SRM University – AP, Andhra Pradesh

for the partial fulfilment of the requirements to award the degree of

Bachelor of Technology/Master of Technology

In

Computer Science and Engineering

School of Engineering and Sciences

Submitted by

RAVI PUJITH
(AP23110010645)

Under the Guidance of

Prof. Subhankar Ghatak

SRM University–AP

Neerukonda, Mangalagiri, Guntur

Andhra Pradesh – 522 240

April-2025

CERTIFICATE

Date: 15-April-2025

This is to certify that the work present in this Project entitled “**Flight booking Management System**” has been carried out by **RAVI PUJITH(AP23110010645)** under my/our supervision. The work is genuine, original, and suitable for submission to the SRM University – AP for the award of Bachelor of Technology/Master of Technology in **School of Engineering and Sciences**.

Supervisor

(Signature)

Prof. Subhankar Ghatak

Acknowledgements

I would like to express my sincere gratitude to all those who helped me throughout the development of this project titled "**Flight Booking Management System**."

First and foremost, I would like to thank my faculty Prof. **Subhankar Ghatak** for their continuous support, valuable guidance, and constructive feedback throughout this project. Their expertise and encouragement were instrumental in shaping this project to completion.

I would also like to extend my heartfelt thanks to my friends and peers who provided insights and motivation during the various phases of this work. Special appreciation goes to the Department of Computer Science and Engineering, SRM University, for providing the learning platform and resources necessary for this academic endeavor.

Finally, I thank my family for their unwavering support and encouragement, which kept me focused and motivated.

This project has deepened my understanding of real-world database systems, SQL optimization, and how technology can simplify complex airline operations. It stands as a reflection of my academic learning and personal growth during this phase of my education.

TABLE OF CONTENTS

ABSTRACT

The Flight Booking Management System (FBMS) is a comprehensive database project designed to streamline and manage the core functions of an airline booking process. It is developed to support real-time tracking, reservation management, and operational efficiency by integrating various key entities such as Flights, Passengers, Bookings, Routes, Aircraft, Payments, Tickets, and Fare Classes.

This system aims to automate the entire booking cycle—from selecting flight routes to generating tickets and managing check-ins—thus reducing human error, saving time, and improving the customer experience. The project leverages SQL-based database design and advanced query techniques including joins, subqueries, aggregations, and constraints to ensure data accuracy, consistency, and integrity.

The database schema is carefully normalized and supports complex operations like checking seat availability, calculating ticket sales by class, analyzing revenue, identifying high-value customers, and generating insights for active and upcoming flights. It also includes well-structured ER diagrams, sample data insertion, and optimized queries for business intelligence reporting.

In summary, the Flight Booking Management System exemplifies the effective use of DBMS principles in a real-world application, offering both scalability and reliability in airline operations.

1.Identification of Project Related to Database Management Systems:

The **Flight Booking Management System** is an integrated software solution developed to streamline and optimize the process of booking, managing, and tracking flight-related data. Serving as a real-world application of Database Management System (DBMS) concepts, this project emphasizes the structured manipulation, organization, and retrieval of critical aviation data in an efficient and secure manner. At the core of this system lies a robust relational database architecture that manages various essential components such as passengers, aircraft, routes, flights, bookings, tickets, fare classes, and payments. Each of these entities is carefully modelled and normalized to ensure data integrity, eliminate redundancy, and support scalability. Through effective use of primary and foreign keys, the system maintains strong referential integrity and ensures consistent data relationships across the database. This student-designed project highlights the practical significance of DBMS in managing large volumes of real-time transactional data. It integrates relational database principles to enable dynamic data retrieval, seamless ticket booking workflows, secure payment processing, and tracking of passenger and flight information with high accuracy. From assigning aircraft to specific routes, processing flight bookings, and issuing tickets to handling customer payments and retrieving high-value customer reports, the **Flight Booking Management System** is a real-life demonstration of how DBMS plays a crucial role in the aviation and travel industry. The use of advanced SQL queries further enhances the ability to derive meaningful insights and generate useful reports for decision-making.In conclusion, this project stands as a solid representation of how database management systems can be effectively applied to address complex real-world challenges in airline reservation systems. It not only reinforces theoretical DBMS concepts but also showcases their real-time utility in a mission-critical domain, making it a valuable contribution to both academic learning and industry relevance.

2. Project Background:

A system designed to simplify and modernize the flight booking process, the **Flight Booking Management System** has been developed to showcase the real-world power and importance of a Database Management System (DBMS). This system efficiently stores and manages data related to passengers, flights, aircraft, routes, fare classes, bookings, tickets, and payments. Created by a student, the project emphasizes key DBMS principles such as data normalization, redundancy elimination, data integrity, and fast information retrieval.

Managing flight operations manually or through outdated systems often results in issues like booking errors, double ticketing, missing passenger information, and slow response times. Such problems are common in both small-scale and large-scale airline operations and inspired the development of this system. The primary goal is to keep flight-related data centralized and well-organized, improving user experience and administrative efficiency. Through the application of DBMS principles, the system handles complex relationships between various entities such as linking passengers to tickets, flights to routes, and payments to bookings. This structured approach ensures that all relevant information is accessible, accurate, and up to date. The system not only minimizes the chances of operational delays but also improves transparency and customer satisfaction. Developed as a student project, the **Flight Booking Management System** is more than just a demonstration of DBMS fundamentals—it also presents a practical, scalable solution to real-world airline booking challenges. It reflects how database systems can transform complex processes into efficient, user-friendly solutions, making it a valuable contribution to both academic understanding and real-world applicability.

3.Project Description

The **Flight Booking Management System** is a robust database-driven application designed to streamline and automate the process of booking and managing flights. Built using a relational Database Management System (DBMS), it organizes flight-related data into interrelated tables, ensuring clarity, consistency, and efficiency. The system effectively manages various components such as passengers, flights, aircraft, routes, bookings, tickets, and payments—making it a comprehensive solution for airline operations.

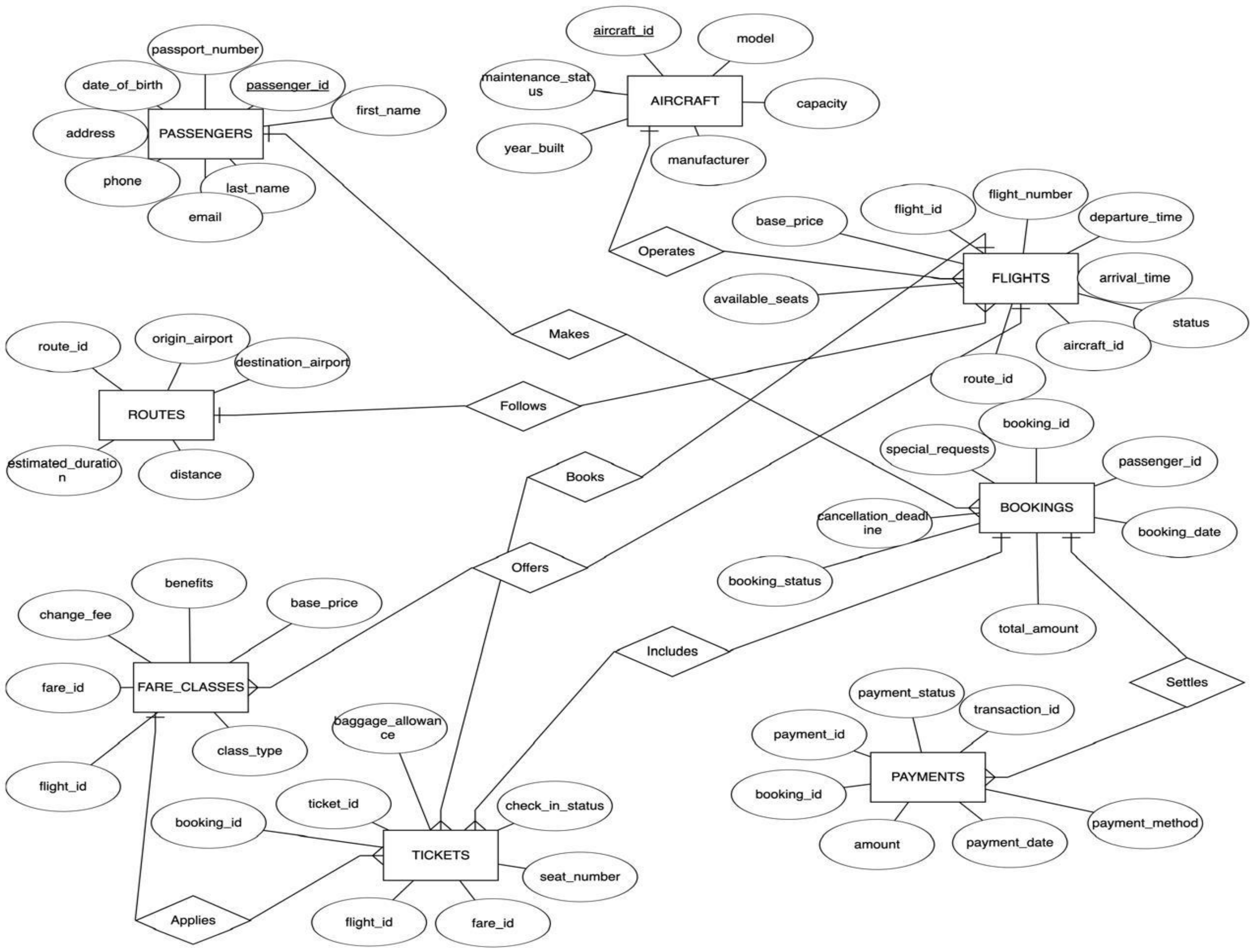
➢ **Main Elements:**

- **Passengers**: Stores details of passengers including full names, contact information, and unique identification numbers.
- **Flights**: Contains data about scheduled flights such as flight numbers, departure and arrival times, associated routes, and aircraft assigned.
- **Aircraft**: Maintains specifications of aircraft like aircraft ID, model, capacity, and airline association.
- **Routes**: Stores information about the origin and destination of flights along with distance and route codes.
- **Fare Classes**: Represents different pricing categories like Economy, Business, and First Class, including class-wise seat allocation and pricing.
- **Bookings**: Manages all booking records, associating passengers with flights and storing details like booking date, class selected, and booking status.
- **Tickets:** Each ticket is linked to a booking and includes a unique ticket

  number, seat assignment, and passenger details. Payments: Stores transaction-related information such as payment ID, method of payment, amount paid and confirmation status.All tables are logically linked using primary and foreign keys, allowing for efficient data retrieval and management.For example, one can easily fetch all flights scheduled between two cities on a specific date or view the payment details associated with a particular ticket.This system was developed by a student to highlight how DBMS concepts like normalization, entity relationships, and referential integrity can be applied to solve real-world problems in the airline industry. It showcases how complex datasets can be handled in a structured, error-free, and user-friendly manner, improving both backend administration and customer service experiences.

- **Payments:** Stores transaction-related information such as payment ID,method of payment,amount paid and confirmation status.ll tables are logically linked using primary and foreign keys, allowing for efficient data retrieval and management. For example, one can easily fetch all flights scheduled between two cities on a specific date or view the payment details associated with a particular ticket.This system was developed by a student to highlight how DBMS concepts like normalization, entity relationships, and referential integrity can be applied to solve real-world problems in the airline industry. It showcases how complex datasets can be handled in a structured, error-free, and user-friendly manner, improving both backend administration and customer service experiences.

4. ER Diagram:


aircraft_id
model
maintenance_status
AIRCRAFT
capacity
year_built
manufacturer
passport_number
date_of_birth
passenger_id
first_name
address
PASSENGERS
phone
last_name
email
flight_number
flight_id
base_price
departure_time
Operates
FLIGHTS
arrival_time
available_seats
status
aircraft_id
route_id
origin_airport
Makes
destination_airport
ROUTES
Follows
estimated_duration
distance
booking_id
special_requests
passenger_id
Books
BOOKINGS
cancellation_deadline
booking_date
Offers
booking_status
benefits
base_price
change_fee
total_amount
Includes
fare_id
FARE_CLASSES
Settles
payment_status
transaction_id
class_type
payment_id
baggage_allowance
flight_id
PAYMENTS
booking_id
ticket_id
check_in_status
booking_id
payment_method
amount
payment_date
TICKETS
seat_number
Applies
flight_id
fare_id

5. Description of ER Diagram:

The ER (Entity-Relationship) diagram of the Flight Booking Management System serves as a clear visual representation of how data is organized and interconnected within the system. Designed with clarity in mind, it captures the core entities, their attributes, and the relationships between them to provide a simplified view of a real-world airline database.

Entities and Attributes

1. PASSENGERS

- **Primary Key:** passenger_id
- **Attributes:** first_name, last_name, email, phone, address, date_of_birth, passport_number

2. AIRCRAFT

- **Primary Key:** aircraft_id
- **Attributes:** model, capacity, manufacturer, year_built, maintenance_status

3. ROUTES

- **Primary Key:** route_id
- **Attributes:** origin_airport, destination_airport, distance, estimated_duration

4. FLIGHTS

- **Primary Key:** flight_id
- **Attributes:** flight_number, departure_time, arrival_time, status, available_seats, base_price
- **Foreign Keys:** aircraft_id, route_id

5. FARE_CLASSES

- **Primary Key:** fare_id
- **Attributes:** class_type, base_price, benefits, change_fee
- **Foreign Key:** flight_id

6. BOOKINGS

- **Primary Key:** booking_id

- **Attributes:** booking_date, total_amount, booking_status, cancellation_deadline, special_requests
- **Foreign Key:** passenger_id

7. TICKETS

- **Primary Key:** ticket_id
- **Attributes:** seat_number, check_in_status, baggage_allowance
- **Foreign Keys:** booking_id, flight_id, fare_id

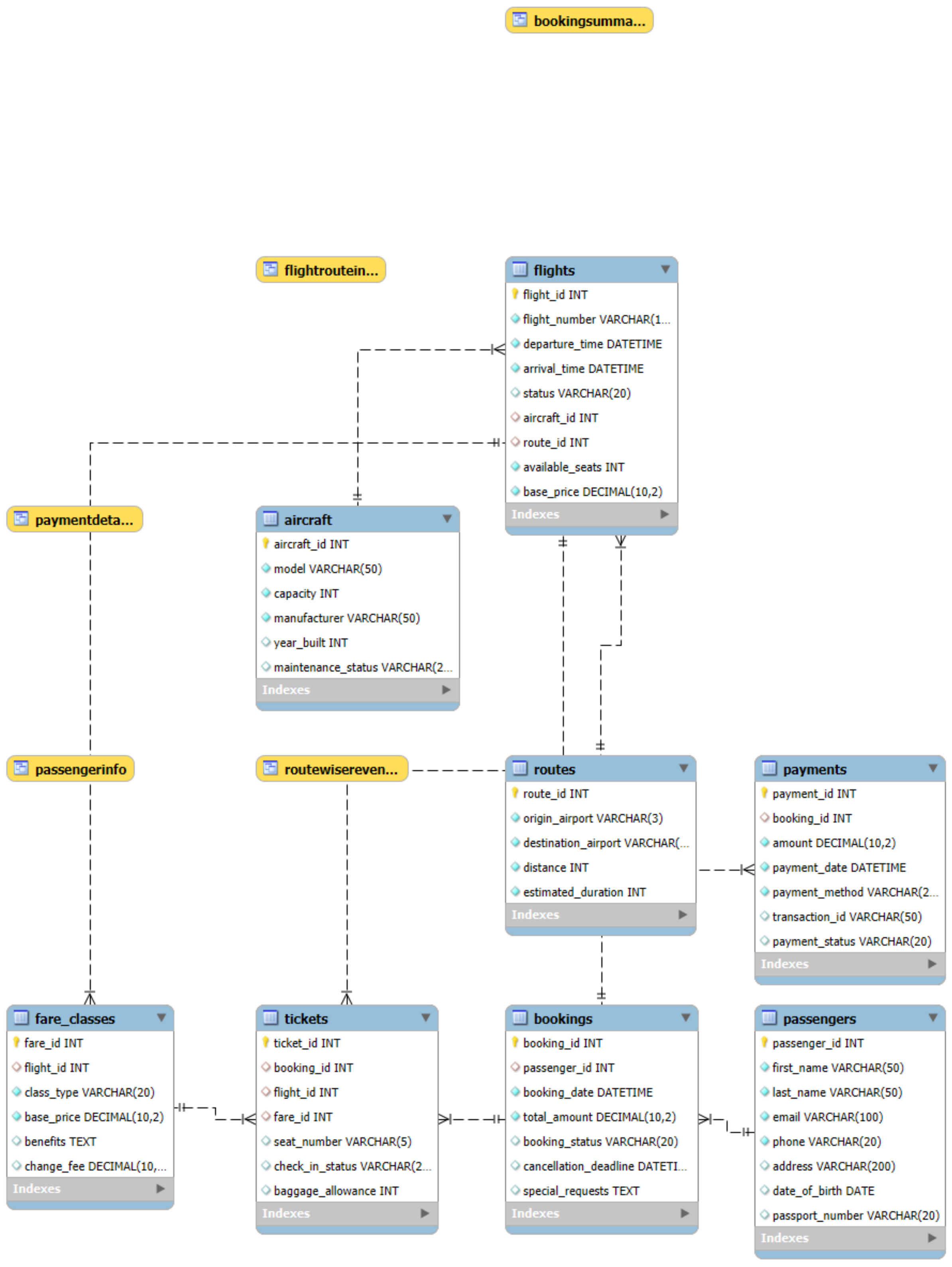
8. PAYMENTS

- **Primary Key:** payment_id
- **Attributes:** amount, payment_date, payment_method, transaction_id, payment_status
- **Foreign Key:** booking_id

Relationships Between Entities

- **PASSENGERS – BOOKS – BOOKINGS:** A passenger can book many bookings (1:M).
- **BOOKINGS – INCLUDES – TICKETS:** A booking includes multiple tickets (1:M).
- **BOOKINGS – SETTLES – PAYMENTS:** Each booking has one payment (1:1).
- **BOOKINGS – BELONGS TO – PASSENGERS**: Many bookings belong to one passenger (M:1).
- **TICKETS – APPLIES – FARE_CLASSES**: Each ticket is linked to a fare class (M:1).
- **TICKETS – FOR – FLIGHTS:** Each ticket is for a flight (M:1).
- **FARE_CLASSES – OFFERS – FLIGHTS:** A flight can have multiple fare classes (1:M).
- **FLIGHTS – OPERATES – AIRCRAFT:** A flight is operated by one aircraft (M:1).
- **FLIGHTS – FOLLOWS – ROUTES:** A flight follows one route (M:1).
- **AIRCRAFT – MAKES – FLIGHTS:** One aircraft can be assigned to many flights (1:M).

6. Conversion of ER diagram into Tables:



7. Description of tables:

The **Flight Management System** database consists of five core tables, each serving a specific purpose. These tables manage passengers, flight scheduling, ticket booking, and communication effectively. Here’s a detailed breakdown of their roles, fields, and contributions:

1. PASSENGERS

Purpose: Stores detailed information about each passenger who registers or books a flight.

Fields:

- passenger_id: Unique identifier for each passenger.
- first_name, last_name: Passenger’s full name.
- email: Unique email address (used for contact and login).
- phone: Contact number.
- address: Optional field for residential address.
- date_of_birth: Used for age verification or travel regulations.
- passport_number: Useful for international flights and identification.

2. AIRCRAFT

Purpose: Maintains the inventory of all aircraft used by the airline.

Fields:

- aircraft_id: Unique identifier for each aircraft.
- model: Model name/number of the aircraft.
- capacity: Total seating capacity.
- manufacturer: Company that made the aircraft.
- year_built: Manufacturing year of the aircraft.
- maintenance_status: Status such as "Operational", "Under Maintenance", etc.

3. ROUTES

Purpose: Defines all flight routes available in the system.

Fields:

- route_id: Unique identifier for the route.
- origin_airport, destination_airport: 3-letter airport codes (IATA standard).
- distance: Distance in kilometers or miles between origin and destination.
- estimated_duration: Estimated travel time (in minutes/hours).

4. FLIGHTS

Purpose: Contains details of scheduled or past flights.

Fields:

- flight_id: Unique identifier for each flight instance.
- flight_number: Airline flight number (e.g., AI202).
- departure_time, arrival_time: Scheduled departure and arrival times.
- status: Current status like "Scheduled", "Cancelled", or "Delayed".
- aircraft_id: Links to the **AIRCRAFT** table to assign an aircraft.
- route_id: Links to the **ROUTES** table to define the flight path.
- available_seats: Number of seats currently available for booking.
- base_price: Basic cost before fare class adjustments.

5. FARE_CLASSES

Purpose: Details different fare classes available for each flight (e.g., Economy, Business).

Fields:

- fare_id: Unique identifier for the fare class.
- flight_id: Connects to a specific flight.

- class_type: Type of class (e.g., Economy, Premium, Business).
- base_price: Fare for the class (may be different from flight base price).
- benefits: Description of perks (e.g., lounge access, extra baggage).
- change_fee: Fee charged to change the ticket.

6. BOOKINGS

Purpose: Stores records of all bookings made by passengers.

Fields:

- booking_id: Unique ID for each booking.
- passenger_id: Links the booking to a passenger.
- booking_date: Date and time when the booking was made.
- total_amount: Total cost of the booking.
- booking_status: Indicates if booking is "Confirmed", "Cancelled", etc.
- cancellation_deadline: Last time to cancel without penalty.
- special_requests: Optional requests like vegetarian meals or wheelchair assistance.

7. TICKETS

Purpose: Keeps track of individual flight tickets linked to bookings.

Fields:

- ticket_id: Unique ticket identifier.
- booking_id: Associates ticket with a booking.
- flight_id: Specifies which flight this ticket is for.
- fare_id: Fare class used for this ticket.
- seat_number: Allocated seat (e.g., 12A).
- check_in_status: Check-in progress ("Checked In", "Not Checked In").

- baggage_allowance: Weight limit allowed (in kg or lbs).

8. PAYMENTS

Purpose: Stores payment details for bookings.

Fields:

- payment_id: Unique identifier for each payment.
- booking_id: Links payment to the corresponding booking (one-to-one).
- amount: Amount paid.
- payment_date: When the payment was processed.
- payment_method: Credit card, UPI, net banking, etc.
- transaction_id: Unique reference for the transaction.
- payment_status: Current status like "Completed", "Pending", or "Failed".

8. Normalization of tables up to 3-NF:

The normalization process ensures that the database is free from redundancy and follows best practices for efficient data storage and management. Let's go through each step and table in detail, starting from the Unnormalized Form (UNF) and progressing to the Third Normal Form (3NF).

1. Unnormalized Form (UNF)

The UNF represents a raw, unstructured form of data with repeating and grouped fields. In this case, the FlightBooking_UNF table contains all the data in a single table, leading to redundancy and potential anomalies.

Issues in UNF:

- **Repeating and grouped fields**: A single row contains details of a booking, passenger, flight, and payment.

- **Redundancy**: Repeated information such as passenger details, flight details, and payment info for each booking.
- **Anomalies**: Update, insert, and delete anomalies due to repeating data.

```
CREATE TABLE FlightBooking_UNF (
  booking_id INT,
  passenger_name VARCHAR(100),
  email VARCHAR(100),
  phone VARCHAR(20),
  address VARCHAR(200),
  date_of_birth DATE,
  passport_number VARCHAR(20),
  flight_number VARCHAR(10),
  departure_time DATETIME,
  arrival_time DATETIME,
  origin_airport VARCHAR(3),
  destination_airport VARCHAR(3),
  distance INT,
  estimated_duration INT,
  aircraft_model VARCHAR(50),
  capacity INT,
  manufacturer VARCHAR(50),
  year_built INT,
  maintenance_status VARCHAR(20),
  class_type VARCHAR(20),
  base_price DECIMAL(10, 2),
  benefits TEXT,
  change_fee DECIMAL(10, 2),
  seat_number VARCHAR(5),
  check_in_status VARCHAR(20),
  baggage_allowance INT,
  booking_date DATETIME,
  total_amount DECIMAL(10, 2),
  booking_status VARCHAR(20),
  cancellation_deadline DATETIME,
  special_requests TEXT,
  payment_amount DECIMAL(10, 2),
  payment_date DATETIME,
  payment_method VARCHAR(20),
  transaction_id VARCHAR(50),
  payment_status VARCHAR(20)
);
```

2. First Normal Form (1NF)

To achieve 1NF, we must ensure that all fields contain atomic values (no multi-valued fields). In this step, the data is structured so that there are no repeating groups.

1NF Fixes:

- **Atomic columns**: Each column contains indivisible values.
- **Unique rows**: Each row represents a unique booking with no repeating groups.

However, even in 1NF, redundancy still exists as the passenger, flight, and payment data are still repeated for each booking.

```
CREATE TABLE FlightBooking_1NF (
    booking_id INT,
    passenger_name VARCHAR(100),
    email VARCHAR(100),
    phone VARCHAR(20),
    address VARCHAR(200),
    date_of_birth DATE,
    passport_number VARCHAR(20),
    flight_number VARCHAR(10),
    departure_time DATETIME,
    arrival_time DATETIME,
    origin_airport VARCHAR(3),
    destination_airport VARCHAR(3),
    distance INT,
    estimated_duration INT,
    aircraft_model VARCHAR(50),
    capacity INT,
    manufacturer VARCHAR(50),
    year_built INT,
    maintenance_status VARCHAR(20),
    class_type VARCHAR(20),
    base_price DECIMAL(10, 2),
    benefits TEXT,
    change_fee DECIMAL(10, 2),
    seat_number VARCHAR(5),
    check_in_status VARCHAR(20),

    baggage_allowance INT,

    booking_date DATETIME,

    total_amount DECIMAL(10, 2),

    booking_status VARCHAR(20),

    cancellation_deadline DATETIME,

    special_requests TEXT,

    payment_amount DECIMAL(10, 2),

    payment_date DATETIME,

    payment_method VARCHAR(20),

    transaction_id VARCHAR(50),

    payment_status VARCHAR(20)

);
```

3. Second Normal Form (2NF)

To achieve 2NF, we remove partial dependencies (dependencies where a non-key attribute depends on part of a composite primary key). This requires splitting the data into multiple related tables, each based on a primary key.

2NF Fixes:

- **Removed partial dependencies**: We separate passenger, aircraft, flight, and payment data into their own tables.
- **Foreign keys**: We link these tables using foreign keys to maintain relationships.

```
-- PASSENGERS

CREATE TABLE PASSENGERS_2NF (

    passenger_id INT PRIMARY KEY,

    name VARCHAR(100),

    email VARCHAR(100),
    phone VARCHAR(20),
    address VARCHAR(200),
    date_of_birth DATE,
    passport_number VARCHAR(20)
);
-- AIRCRAFT
CREATE TABLE AIRCRAFT_2NF (
    aircraft_id INT PRIMARY KEY,
    model VARCHAR(50),
    capacity INT,
    manufacturer VARCHAR(50),
    year_built INT,
    maintenance_status VARCHAR(20)
);
-- ROUTES
CREATE TABLE ROUTES_2NF (
    route_id INT PRIMARY KEY,
    origin_airport VARCHAR(3),
    destination_airport VARCHAR(3),
    distance INT,
    estimated_duration INT
);
-- FLIGHTS
CREATE TABLE FLIGHTS_2NF (
    flight_id INT PRIMARY KEY,
    flight_number VARCHAR(10),
    departure_time DATETIME,
    arrival_time DATETIME,
    aircraft_id INT,
    route_id INT,
    base_price DECIMAL(10, 2),
    available_seats INT,
    FOREIGN KEY (aircraft_id) REFERENCES AIRCRAFT(aircraft_id),
    FOREIGN KEY (route_id) REFERENCES ROUTES(route_id)
);
-- FARE_CLASSES
CREATE TABLE FARE_CLASSES_2NF (
    fare_id INT PRIMARY KEY,
    flight_id INT,
    class_type VARCHAR(20),
    base_price DECIMAL(10, 2),
    benefits TEXT,
    change_fee DECIMAL(10, 2),
    FOREIGN KEY (flight_id) REFERENCES FLIGHTS(flight_id)
);
-- BOOKINGS
CREATE TABLE BOOKINGS_2NF (
    booking_id INT PRIMARY KEY,
    passenger_id INT,
    booking_date DATETIME,
    total_amount DECIMAL(10, 2),
    booking_status VARCHAR(20),
    cancellation_deadline DATETIME,
    special_requests TEXT,
    FOREIGN KEY (passenger_id) REFERENCES PASSENGERS(passenger_id)
);
-- TICKETS
CREATE TABLE TICKETS_2NF (
    ticket_id INT PRIMARY KEY,
    booking_id INT,
    flight_id INT,
    fare_id INT,
    seat_number VARCHAR(5),
    check_in_status VARCHAR(20),
    baggage_allowance INT,
    FOREIGN KEY (booking_id) REFERENCES BOOKINGS(booking_id),
    FOREIGN KEY (flight_id) REFERENCES FLIGHTS(flight_id),
    FOREIGN KEY (fare_id) REFERENCES FARE_CLASSES(fare_id)
);
-- PAYMENTS
CREATE TABLE PAYMENTS_2NF (
    payment_id INT PRIMARY KEY,
    booking_id INT,
    amount DECIMAL(10, 2),
    payment_date DATETIME,
    payment_method VARCHAR(20),

    transaction_id VARCHAR(50),

    payment_status VARCHAR(20),

    FOREIGN KEY (booking_id) REFERENCES BOOKINGS(booking_id)

);
```

4. Third Normal Form (3NF)

To achieve 3NF, we remove transitive dependencies (where non-key attributes depend on other non-key attributes). In 3NF, all attributes are functionally dependent only on the primary key.

3NF Fixes:

- **Removed transitive dependencies**: We ensure that no non-key attributes depend on other non-key attributes.
- **Final Structure**: Tables are now fully normalized, and the data is logically organized, making it easier to maintain.

```
CREATE TABLE PASSENGERS (

    passenger_id INT PRIMARY KEY,

    first_name VARCHAR(50) NOT NULL,

    last_name VARCHAR(50) NOT NULL,

    email VARCHAR(100) UNIQUE NOT NULL,

    phone VARCHAR(20) NOT NULL,

    address VARCHAR(200),

    date_of_birth DATE,

    passport_number VARCHAR(20)

);

CREATE TABLE AIRCRAFT (

    aircraft_id INT PRIMARY KEY,

    model VARCHAR(50) NOT NULL,
    capacity INT NOT NULL,

    manufacturer VARCHAR(50) NOT NULL,

    year_built INT,

    maintenance_status VARCHAR(20)

);

CREATE TABLE ROUTES (

    route_id INT PRIMARY KEY,

    origin_airport VARCHAR(3) NOT NULL,

    destination_airport VARCHAR(3) NOT NULL,

    distance INT NOT NULL,

    estimated_duration INT NOT NULL

);

CREATE TABLE FLIGHTS (

    flight_id INT PRIMARY KEY,

    flight_number VARCHAR(10) NOT NULL,

    departure_time DATETIME NOT NULL,

    arrival_time DATETIME NOT NULL,

    status VARCHAR(20) DEFAULT 'Scheduled',

    aircraft_id INT,

    route_id INT,

    available_seats INT NOT NULL,

    base_price DECIMAL(10, 2) NOT NULL,

    FOREIGN KEY (aircraft_id) REFERENCES AIRCRAFT(aircraft_id),

    FOREIGN KEY (route_id) REFERENCES ROUTES(route_id)

);
```

```
CREATE TABLE FARE_CLASSES (
    fare_id INT PRIMARY KEY,
    flight_id INT,
    class_type VARCHAR(20) NOT NULL,
    base_price DECIMAL(10, 2) NOT NULL,
    benefits TEXT,
    change_fee DECIMAL(10, 2),
    FOREIGN KEY (flight_id) REFERENCES FLIGHTS(flight_id)
);
CREATE TABLE BOOKINGS (
    booking_id INT PRIMARY KEY,
    passenger_id INT,
    booking_date DATETIME NOT NULL,
    total_amount DECIMAL(10, 2) NOT NULL,
    booking_status VARCHAR(20) DEFAULT 'Confirmed',
    cancellation_deadline DATETIME,
    special_requests TEXT,
    FOREIGN KEY (passenger_id) REFERENCES PASSENGERS(passenger_id)
);
CREATE TABLE TICKETS (
    ticket_id INT PRIMARY KEY,
    booking_id INT,
    flight_id INT,
    fare_id INT,
    seat_number VARCHAR(5),
    check_in_status VARCHAR(20) DEFAULT 'Not Checked In',
    baggage_allowance INT DEFAULT 20,
    FOREIGN KEY (booking_id) REFERENCES BOOKINGS(booking_id),
    FOREIGN KEY (flight_id) REFERENCES FLIGHTS(flight_id),
    FOREIGN KEY (fare_id) REFERENCES FARE_CLASSES(fare_id)
);
CREATE TABLE PAYMENTS (
    payment_id INT PRIMARY KEY,
    booking_id INT UNIQUE,
    amount DECIMAL(10, 2) NOT NULL,
    payment_date DATETIME NOT NULL,
    payment_method VARCHAR(20) NOT NULL,
    transaction_id VARCHAR(50) UNIQUE,
    payment_status VARCHAR(20) DEFAULT 'Completed',
    FOREIGN KEY (booking_id) REFERENCES BOOKINGS(booking_id)
);
```

9.Creation of Data in the Tables:

Sample data was inserted into all the core tables of the Flight Booking Management System to simulate realistic operations. These datasets support comprehensive testing of queries, validation of design, and demonstration of system functionalities across booking, scheduling, passenger management, and payment tracking.

1. PASSENGERS Table:

- **Records**:
    - (1, 'Rajesh', 'Sharma', 'rajesh.sharma@gmail.com', '+91-9876543210', '42 Rajaji Nagar, Bengaluru, Karnataka', '1985-03-15', 'Z4567891'),

  - (2, 'Priya', 'Patel', 'priya.patel@yahoo.com', '+91-8765432109', '15 Bandra West, Mumbai, Maharashtra', '1990-07-22', 'M1234567'),
  - (3, 'Vikram', 'Singh', 'vikram.singh@hotmail.com', '+91-7654321098', '78 Vasant Vihar, New Delhi, Delhi', '1978-11-30', 'P7891234'),
  - (4, 'Anjali', 'Gupta', 'anjali.gupta@gmail.com', '+91-6543210987', '23 Salt Lake, Kolkata, West Bengal', '1995-05-08', 'J5678912'),
  - (5, 'Suresh', 'Kumar', 'suresh.kumar@gmail.com', '+91-9876543211', '105 Anna Nagar, Chennai, Tamil Nadu', '1982-09-17', 'N3456789');
- **Purpose**: Diverse passenger profiles with different addresses, birthdates, and ID proofs were added to test identification, booking eligibility, and age-based fare logic.

2. AIRCRAFT Table:

- **Records**:
  - (1, 'Boeing 737-800', 180, 'Boeing', 2015, 'Operational'),
  - (2, 'Airbus A320', 150, 'Airbus', 2018, 'Operational'),
  - (3, 'Boeing 787 Dreamliner', 280, 'Boeing', 2019, 'Operational'),
  - (4, 'Airbus A350', 300, 'Airbus', 2020, 'Operational'),
  - (5, 'ATR 72-600', 78, 'ATR', 2017, 'Maintenance');
- **Purpose**: The data simulates a mix of aircraft types from different manufacturers with varying capacities and operational statuses, supporting capacity planning and maintenance module testing.

3. ROUTES Table:

- **Records**:
  - (1, 'DEL', 'BOM', 1148, 130),
  - (2, 'BLR', 'CCU', 1560, 150),
  - (3, 'HYD', 'MAA', 520, 70),
  - (4, 'PNQ', 'JAI', 980, 110),
  - (5, 'AMD', 'GOI', 650, 85);
- **Purpose**: Routes between major Indian cities with associated distances and durations allow simulation of flight time calculations and route-based pricing strategies.

4. FLIGHTS Table:

- **Records**:

  - (1, 'AI204', '2025-04-10 08:00:00', '2025-04-10 10:10:00', 'Scheduled', 1, 1, 150, 5500.00),
  - (2, 'IX336', '2025-04-11 10:30:00', '2025-04-11 13:00:00', 'Scheduled', 2, 2, 120, 6200.00),
  - (3, 'SG123', '2025-04-12 12:15:00', '2025-04-12 13:25:00', 'Scheduled', 3, 3, 260, 3500.00),
  - (4, '6E417', '2025-04-13 16:45:00', '2025-04-13 18:35:00', 'Scheduled', 4, 4, 280, 4750.00),
  - (5, 'UK875', '2025-04-14 07:20:00', '2025-04-14 08:45:00', 'Scheduled', 2, 5, 130, 4200.00);
- **Purpose**: Scheduled flights on various dates and routes, linking aircraft and route data to represent real-time scheduling and seat availability features.

5. FARE_CLASSES Table:

- **Records**:
  - (1, 1, 'Economy', 5500.00, 'Standard baggage 15kg', 2000.00),
  - (2, 1, 'Business', 12500.00, 'Priority boarding, 30kg baggage, Meals', 1500.00),
  - (3, 2, 'Economy', 6200.00, 'Standard baggage 15kg', 2200.00),
  - (4, 2, 'Business', 14500.00, 'Priority boarding, 30kg baggage, Meals', 1700.00),
  - (5, 3, 'Economy', 3500.00, 'Standard baggage 15kg', 1500.00),
  - (6, 3, 'Business', 9000.00, 'Priority boarding, 30kg baggage, Meals', 1200.00),
  - (7, 4, 'Economy', 4750.00, 'Standard baggage 15kg', 1800.00),
  - (8, 5, 'Economy', 4200.00, 'Standard baggage 15kg', 1600.00);
- **Purpose**: Demonstrates tiered pricing and service differentiation, useful in testing dynamic pricing models and upgrade options.

6. BOOKINGS Table:

- **Records**:
  - (1, 1, '2025-03-15 09:23:15', 25000.00, 'Confirmed', '2025-04-03 08:00:00', NULL),
  - (2, 2, '2025-03-17 14:45:32', 12400.00, 'Confirmed', '2025-04-04 10:30:00', 'Vegetarian Jain meal'),
  - (3, 3, '2025-03-20 11:12:54', 14500.00, 'Confirmed', '2025-04-05 12:15:00', 'Wheelchair assistance'),
  - (4, 4, '2025-03-22 16:33:21', 9500.00, 'Confirmed', '2025-04-06 16:45:00', 'Infant traveling'),

   - (5, 5, '2025-03-25 08:55:47', 8400.00, 'Confirmed', '2025-04-07 07:20:00', 'Aisle seat preference');
- **Purpose**: Simulates confirmed reservations, enabling payment status tracking and special service requests.

7. TICKETS Table:

- **Records**:
   - (1, 1, 1, 2, '12A', 'Not Checked In', 30),
   - (2, 1, 1, 2, '12B', 'Not Checked In', 30),
   - (3, 2, 2, 3, '15F', 'Not Checked In', 15),
   - (4, 2, 2, 3, '15G', 'Not Checked In', 15),
   - (5, 3, 3, 4, '8C', 'Not Checked In', 30),
   - (6, 4, 4, 7, '22D', 'Not Checked In', 15),
   - (7, 4, 4, 7, '22E', 'Not Checked In', 15),
   - (8, 5, 5, 8, '10A', 'Not Checked In', 15),
   - (9, 5, 5, 8, '10B', 'Not Checked In', 15);
- **Purpose**: Represents passenger manifests, supports check-in system testing, and visualizes seat allocation.

8. PAYMENTS Table:

- **Records**:
   - (1, 1, 25000.00, '2025-03-15 09:25:10', 'Credit Card', 'HDFC123456789', 'Completed'),
   - (2, 2, 12400.00, '2025-03-17 14:47:22', 'UPI', 'UPIREF234567890', 'Completed'),
   - (3, 3, 14500.00, '2025-03-20 11:15:33', 'Net Banking', 'ICICI345678901', 'Completed'),
   - (4, 4, 9500.00, '2025-03-22 16:35:44', 'Debit Card', 'SBI456789012', 'Completed'),
   - (5, 5, 8400.00, '2025-03-25 08:57:55', 'Credit Card', 'AXIS567890123', 'Completed');
- **Purpose**: Ensures that diverse payment methods and gateways are represented for end-to-end booking and financial transaction tracking.

10.SQL Queries on the Created Tables:

A sophisticated SQL query was developed to extract project insights, incorporating subqueries, aggregates, and joins.

1. Retrieve All Passenger Records

```
SELECT * FROM PASSENGERS;
```

Functionality:

Retrieves complete information of all passengers in the system.

Concepts Used:

- Simple SELECT statement from a single table.

2. View Seat Numbers with Passenger Names

```
SELECT T.seat_number, P.first_name
FROM TICKETS T
JOIN BOOKINGS B ON T.booking_id = B.booking_id
JOIN PASSENGERS P ON B.passenger_id = P.passenger_id;
```

Functionality:

Displays which seat is booked by which passenger by tracing through tickets to bookings to passengers.

Concepts Used:

- Multiple INNER JOINs
- Table aliasing

3. Get Flight Numbers with Aircraft Model

```
SELECT FL.flight_number, AC.model
FROM FLIGHTS FL
JOIN AIRCRAFT AC ON FL.aircraft_id = AC.aircraft_id;
```

Functionality:

Displays the aircraft model used for each flight number.

Concepts Used:

- INNER JOIN
- Table aliasing

4. Find High-Value Customers

```
SELECT first_name
FROM PASSENGERS
WHERE passenger_id IN (
   SELECT passenger_id FROM BOOKINGS WHERE total_amount > 10000
);
```

Functionality:

Finds passengers who made bookings above 10,000 units of currency.

Concepts Used:

- WHERE with IN
- Subquery (Nested SELECT)

5. List All Routes with Flights

```
SELECT R.origin_airport, R.destination_airport, FL.flight_number
FROM ROUTES R
JOIN FLIGHTS FL ON R.route_id = FL.route_id;
```

Functionality:

Displays flight numbers operating on each route between origin and destination.

Concepts Used:

- INNER JOIN

6. Payments Above Average

```
SELECT * FROM PAYMENTS
WHERE amount > (
   SELECT AVG(amount) FROM PAYMENTS
);
```

Functionality:

Fetches all payments that are above the average payment amount.

Concepts Used:

- Aggregate function AVG()
- Subquery
- WHERE clause

7. Count of Flights per Aircraft Model

```
SELECT ac.model, COUNT(f.flight_id) AS flight_count
FROM AIRCRAFT ac
LEFT JOIN FLIGHTS f ON ac.aircraft_id = f.aircraft_id
GROUP BY ac.model;
```

Functionality:

Shows how many flights are assigned to each aircraft model.

Concepts Used:

- LEFT JOIN
- GROUP BY
- Aggregate function COUNT()

8. Flight Details with Ticket and Seat Insights

```
SELECT
   f.flight_id,
   f.flight_number,
   r.origin_airport,
   r.destination_airport,
   COUNT(t.ticket_id) AS total_tickets,
   f.available_seats,
   f.available_seats - COUNT(t.ticket_id) AS remaining_seats
FROM
   FLIGHTS f
INNER JOIN
   ROUTES r ON f.route_id = r.route_id
LEFT JOIN
   TICKETS t ON f.flight_id = t.flight_id
GROUP BY
   f.flight_id, f.flight_number, r.origin_airport, r.destination_airport, f.available_seats;
```

Functionality:

Provides a summary of each flight, including the number of tickets booked and remaining seats.

Concepts Used:

- INNER JOIN, LEFT JOIN
- Aggregate function COUNT()

- GROUP BY
- Calculated columns

9. Booking and Payment Summar

```
SELECT
   b.booking_id,
   p.first_name,
   p.last_name,
   b.booking_date,
   b.total_amount,
   b.booking_status,
   pay.payment_method,
   pay.payment_date,
   pay.payment_status
FROM
   PASSENGERS p
INNER JOIN
   BOOKINGS b ON p.passenger_id = b.passenger_id
RIGHT JOIN
   PAYMENTS pay ON b.booking_id = pay.booking_id;
```

Functionality:

Combines booking and payment details along with passenger names.

Concepts Used:

- INNER JOIN, RIGHT JOIN
- Multiple table joins
- Displaying booking and payment data together

10. Route-wise Ticket Count

```
SELECT
   r.route_id,
   r.origin_airport,
   r.destination_airport,
   COUNT(t.ticket_id) AS ticket_count
FROM
```

```
    ROUTES r
LEFT JOIN
    FLIGHTS f ON r.route_id = f.route_id
LEFT JOIN
    TICKETS t ON f.flight_id = t.flight_id
GROUP BY
    r.route_id, r.origin_airport, r.destination_airport
ORDER BY
    ticket_count DESC;
```

Functionality:

Ranks routes based on the number of tickets booked.

Concepts Used:

- LEFT JOIN (to include routes even if there are no flights or tickets)
- GROUP BY
- Aggregate function COUNT()
- ORDER BY DESC

11. Creation of Views Using the Tables:

1.Booking Summary View

```
CREATE VIEW BookingSummary AS
SELECT B.booking_id, P.first_name, B.total_amount
FROM BOOKINGS B
JOIN PASSENGERS P ON B.passenger_id = P.passenger_id;
SELECT * FROM BookingSummary;
```

Functionality:

Creates a view that shows summarized booking information including:

- Booking ID
- Passenger's First Name
- Total Amount Paid for the Booking

This view simplifies access to commonly needed booking data for reporting or dashboards.

Concepts Used:

- CREATE VIEW: To define a virtual table
- JOIN: Combines BOOKINGS and PASSENGERS on passenger_id
- SELECT * FROM ViewName: Retrieves all rows from the created view

2. Payment Details View

```
CREATE VIEW PaymentDetails AS
SELECT B.booking_id, P.first_name, PM.amount, PM.payment_method
FROM BOOKINGS B
JOIN PASSENGERS P ON B.passenger_id = P.passenger_id
JOIN PAYMENTS PM ON B.booking_id = PM.booking_id;
SELECT * FROM PaymentDetails;
```

Functionality:

Creates a detailed view of payment information with:

- Booking ID
- Passenger's First Name
- Amount Paid
- Payment Method

It is useful for generating payment reports, finance audits, or billing modules.

Concepts Used:

- CREATE VIEW: To define a virtual table
- Multiple JOINs:
  - BOOKINGS with PASSENGERS (on passenger_id)
  - BOOKINGS with PAYMENTS (on booking_id)
- Selecting specific fields to form a meaningful report view

3. Route-Wise-Revenue View

```
CREATE VIEW RouteWiseRevenue AS
SELECT r.route_id, r.origin_airport, r.destination_airport, SUM(p.amount) AS total_revenue
FROM ROUTES r
```

```
JOIN FLIGHTS f ON r.route_id = f.route_id
JOIN TICKETS t ON f.flight_id = t.flight_id
JOIN BOOKINGS b ON t.booking_id = b.booking_id
JOIN PAYMENTS p ON b.booking_id = p.booking_id
GROUP BY r.route_id, r.origin_airport, r.destination_airport;
SELECT * FROM RouteWiseRevenue;
```

Functionality:

- Creates a view that shows total revenue generated from each route.
- Useful for financial analysis of different routes.

Concepts Used:

- **CREATE VIEW**: To define a virtual table for summarizing route revenue.
- **JOIN**: Combines **ROUTES**, **FLIGHTS**, **TICKETS**, and **PAYMENTS** on their respective keys.
- **GROUP BY**: Groups the result set by route details to calculate total revenue.
- **SUM()**: Aggregates the payment amounts to get total revenue per route.
- Selecting specific fields to form a meaningful report view.

12. Conclusion:

The **Flight Booking Management System** has been carefully designed to showcase the practical benefits and significance of a **Database Management System (DBMS)** in organizing, managing, and streamlining airline reservation data. This project highlights how a structured and normalized database approach can effectively reduce data redundancy, ensure consistency, and maintain high levels of data integrity in real-time booking scenarios.

To demonstrate these concepts, a fully normalized database schema was developed, representing real-world components such as **passengers, flights, routes, aircraft, bookings, tickets, payments**, and **fare classes**. Sample data was populated to simulate real-world airline operations, enabling meaningful testing and validation of the system.

Multiple **SQL queries** were written and executed to extract relevant insights, such as viewing flight availability, tracking passenger bookings, monitoring fare details, and analyzing payment records. A **database view** was also created to present key booking and ticketing information in an accessible and readable format for administrators and users.

Additionally, an **Entity-Relationship (ER) diagram** has been planned to visually represent the database structure. This will provide a clear overview of how entities interact, helping users and developers understand the relationships and data flow within the system.

Overall, this project serves as a solid example of applying DBMS principles to solve a real-world problem. It not only demonstrates technical knowledge in **database normalization, querying, and schema design**, but also reinforces how such systems play a vital role in enhancing the efficiency and organization of large-scale operations like airline ticket management.

FUTURE WORK

While the current version of the Flight Booking Management System effectively fulfills the core requirements of flight and passenger management, there are several areas for potential enhancement in future versions:

1.User Interface Integration

Development of a responsive and intuitive graphical user interface using web technologies (HTML, CSS, JavaScript, React.js) to enable seamless interaction for administrators, passengers, and airline staff.

2. Real-Time Flight Tracking

Integration of real-time flight tracking APIs to provide passengers with live updates on flight status, delays, gate information, and estimated arrival times.

3. Notification System

Implementation of automated email and SMS notifications to inform users about booking confirmations, flight delays, cancellations, and check-in reminders.

4. Role-Based Access Control

Enhancing system security with role-based authentication and authorization to restrict and customize access for Admins, Airline Staff, and Passengers.

5. Dynamic Seat Selection

Integration of a real-time interactive seat map for passengers to choose available seats during the booking process.

6. Mobile Application Development

Designing a cross-platform mobile application (Android & iOS) to allow users to book flights, check flight status, view tickets, and receive notifications on the go.

7. AI-Based Fare Prediction

Incorporating machine learning algorithms to analyze fare trends and provide intelligent suggestions for the best booking times and flight options.

8. Cloud-Based Deployment

Migrating the system to a cloud environment (such as AWS, Azure, or Google Cloud) to ensure scalability, data security, remote accessibility, and efficient backup management.

9. Frequent Flyer Program

Adding a loyalty program with reward points, member tiers, and special discounts to enhance user retention and customer satisfaction.

10. Advanced Analytics Dashboard

Development of analytics and reporting modules for administrators to monitor revenue, occupancy rates, customer behavior, and popular routes.

REFERENCES

- https://erdplus.com/ - For Creating ER Diagram
- Database System Concepts 6th edition - Henry F Korth Abraham Silberschatz, S Sudharshan