Project: Flight Reservation System

Course: CS 301 Fundamentals of Database Systems

Semester: Fall 2025

1. Goal

The purpose of this project is to give students hands-on experience in the **conceptual design**, **logical design**, **implementation**, and **maintenance** of a relational database system for a **Flight Reservation System**.

This project emphasizes individual understanding of full-stack database development rather than specialization in isolated tasks. Extensions and creative improvements (e.g., loyalty programs, route analytics, dynamic pricing) are strongly encouraged and may evolve into senior design or research projects.

2. Enterprise Description

The enterprise to be modeled is an **airline flight reservation platform** (for a single airline or multiple airlines) that supports searching flights, booking seats, issuing tickets, and handling payments.

Each student should choose a **specific scenario**, such as:

- A single airline operating domestic routes,
- A regional airline with multiple hubs,
- A small multi-airline booking platform aggregating flights from several carriers.

The system should capture at least the following aspects:

- **Airlines** (optional if you model a single carrier): code, name, and contact details.
- **Airports:** Code (e.g., 3-letter IATA), name, city, country, and optionally time zone.
- **Aircraft Types:** Model, manufacturer, seating capacity, and optionally cabin layout or classes.
- **Flights (Routes):** A scheduled service between two airports, with airline, flight number, departure airport, arrival airport, and scheduled times.
- **Flight Instances:** A specific occurrence of a flight on a date (e.g., Flight XY123 on 2025-10-15), with status (scheduled, delayed, cancelled, departed, arrived) and possibly actual times.

- **Cabin Classes / Fares:** Classes such as economy, business, first, with associated base fares and optional fare types (refundable, non-refundable, flexible).
- **Passengers:** Personal information (name, date of birth, contact details, ID/passport), and optionally frequent-flyer information.
- **Bookings (Reservations):** A booking may contain one or more passengers and one or more flight segments. It has a booking reference, creation date, and status (booked, ticketed, cancelled).
- **Tickets:** Tickets link bookings, passengers, and specific flight segments. Each ticket has a ticket number, issue date, fare, and status (valid, used, refunded, void).
- **Seats:** Seat assignments per flight instance with seat number and cabin class; seats must not be double-booked.
- **Payments:** Payments related to bookings or tickets with amount, currency, payment method (card, cash, online wallet), date, and status (successful, failed, refunded).

You may extend the model with optional features such as baggage tracking or loyalty points if time permits.

3. Data Generation

Realistic sample data is required to support meaningful queries and testing. You may generate random data programmatically (e.g., using Python scripts or SQL procedures).

At a minimum, aim for:

- 10–20 airports,
- 3–5 airlines (if modeling multiple carriers),
- 10–20 aircraft types,
- 100–200 flights (routes) connecting different airport pairs,
- Several hundred flight instances across a range of dates,
- 300–500 passengers,
- Several hundred bookings, each with one or more passengers and segments,
- Tickets, seat assignments, and payment records consistent with the bookings.

Ensure that data is consistent (e.g., flight instances refer to existing flights, seats are not double-assigned, airports exist for each route, paid amounts match ticket prices).

4. Client Requests

1. E-R Model

Construct an **E-R diagram** that represents your conceptual design. Clearly indicate:

- Main entities (e.g., Airline, Airport, AircraftType, Flight, FlightInstance, Passenger, Booking, BookingSegment, Ticket, SeatAssignment, Payment),
- Relationships (e.g., airline *operates* flights, flight *departs from* and *arrives at* airports, booking *includes* passengers and flight instances, payment *pays for* booking),
- **Primary keys**, attributes, cardinalities, and participation constraints,
- Any specialization/generalization (e.g., Payment → CardPayment / CashPayment).

2. Relational Model

Convert your E-R model to a **relational schema**. Apply normalization up to **3NF or BCNF**. Define:

- Tables and attributes corresponding to entities and relationships,
- **Primary keys** and **foreign keys**,
- **Constraints** (NOT NULL, UNIQUE, CHECK) and appropriate **indexes** (e.g., on flight number + date, airport codes, booking reference).

Implement the schema in a relational DBMS such as Oracle, MySQL, or PostgreSQL.

3. Populate Relations

Populate all tables with enough data to test typical operations and queries:

- Ensure each flight has multiple flight instances on different dates,
- Ensure each flight instance has several bookings and seat assignments,
- Ensure each booking has one or more passengers, tickets, and at least one payment attempt,
- Use INSERT scripts or data-generation programs to create this data.

4. Queries

Implement at least **8–10 queries** that are useful for passengers, agents, and managers, such as:

1. Given origin, destination, and date, list available flight options with departure/arrival times, airline, and remaining seats.
2. For a selected flight instance, list all passengers and their seat assignments.
3. For a given passenger, list all upcoming and past flights with dates and booking references.
4. For each route (origin–destination pair), compute total passengers carried and total revenue over a given period.
5. Identify flight instances or routes with occupancy below a given threshold (e.g., load factor $< 60\%$).
6. List bookings with unpaid or partially paid status (outstanding balances).
7. Find all flight instances that were cancelled and list affected passengers per flight.
8. For each airline, summarize total revenue and average occupancy by month.
9. (Optional) Find one-stop itineraries between two cities within a specified time window (connections).
10. (Optional) For each passenger, compute loyalty metrics (e.g., number of flights or total distance flown).

5. Interfaces

You may implement one or more of the following:

- **Customer / Passenger Interface:**
  - Search for flights by origin, destination, and date,
  - View available options and basic details,
  - Create a booking for one or more passengers,
  - View existing bookings and tickets.
- **Agent / Staff Interface:**
  - Create, modify, or cancel bookings,
  - Assign or change seat allocations,

  - Record payments and issue tickets,
  - View passenger lists for flight instances.

- **Manager / Admin Interface:**
  - Manage flights, flight instances, and fares,
  - Run reports on occupancy, revenue, and cancellations.

Interfaces may be:

- Command-line based (menu-driven or prompt-based), or
- Simple web-based (optional) if you are comfortable with basic web development.

This is a database-focused project; a clean and functional command-line interface is sufficient. You may add triggers or views (e.g., to prevent overbooking beyond capacity or to simplify reporting) as optional enhancements.

5. What to Submit

1. **E-R Diagram** with explanatory notes.
2. **Relational Schema** consistent with your final E-R design.
3. **SQL Scripts** for table creation, constraints, and indexes.
4. **Data Loading Scripts** or programs used to populate the database.
5. **Sample Queries** (SQL files) with example outputs (you can demonstrate these during the lab).
6. **Interface Code** (Java, Python, or command-line scripts).
7. **README File** explaining the project structure, setup steps, and how to run your code.
8. **ZIP Archive** containing all files (.sql, .java, .py, .txt, etc.).

Avoid platform-specific or IDE-dependent submissions. Provide plain, compilable source files and clear instructions.

6. Grading Criteria

Component	Points
E-R Design	30
Relational Design (constraints, normalization)	30
Data Creation (realism, integrity)	10
User Interfaces (functionality, usability)	30
Creativity / Exceptional Work	+10 bonus
Total	**100 points**

7. Collaboration Policy

- Group integration work: each student should be able to defend and explain the project.
- A detailed explanation is required from each team member.
- All team conflicts should first be resolved internally, then brought to the instructor if unresolved.
- If any team member did not contribute during the project implementation, you may mention this when submitting the project files.

8. Notes

- Submit all deliverables electronically by the due date.
- Late submissions without prior request will incur penalties.
- Creativity is encouraged: features such as loyalty programs, route analytics, disruption handling, or seat-map visualizations can earn bonus credit.