PROJECT: AIRLINE RESERVATION SYSTEM

DANIEL JEUN
JAYSON ZELAYA

SECTION 1

The Reservation System for a Single Airline is designed to manage passenger services exclusively. The core functionalities and requirements of the system are as follows:

1. Ticket Booking:

- Passengers can book tickets for flights that are available in the system.
- Each ticket corresponds to a specific flight.

2. Flight Association:

- Each flight is linked to a single aircraft, which determines the seating capacity.
- Flights are scheduled for specific departure and arrival times at designated airports.

3. Airport Details:

- Each airport is uniquely identified by a code.
- Airports can host multiple flights arriving and departing at different times.

4. **Aircraft Capacity:**

-Each aircraft has a predefined maximum seating capacity.
- The system ensures that ticket booking do not exceed the available seats on any flight.

5. Passenger Details:

- Passengers must provide personal information (e.g., name) when booking tickets.

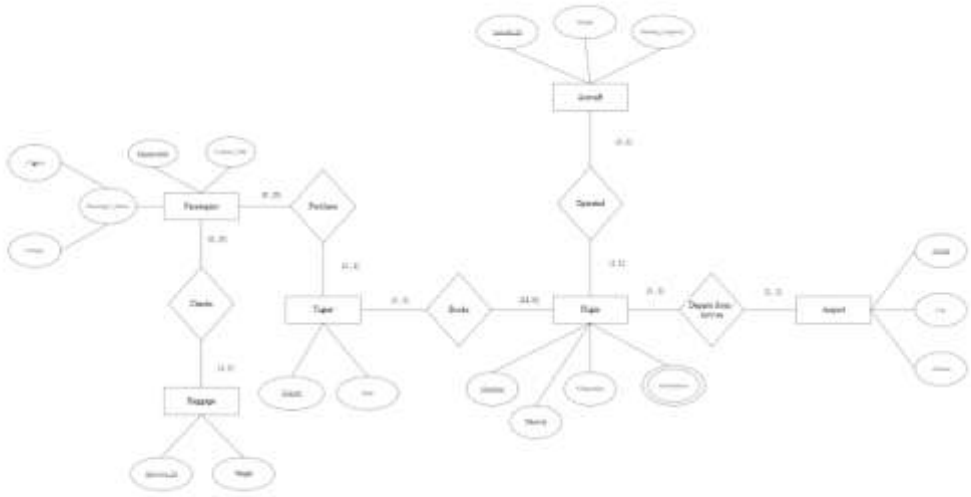
6. Flight Nature:

- All flights are direct and non-stop, with no layovers or connections.

This system is designed to maintain accurate records of passengers, flights, and seating availability while ensuring smooth booking and scheduling operations.

THE ENTITY RELATIONSHIP (ER) DESIGN

The Entity-Relationship (ER) Design for the Airline Reservation System serves as a blueprint for organizing and structuring the database to represent the system's core operations. It identifies key entities such as *Passengers*, *Flights*, *Airports*, and *Aircraft*, along with their attributes and the relationships that connect them. The design captures critical interactions, such as passengers booking tickets, flights being scheduled between airports, and aircraft being assigned to flights. By defining these relationships and applying constraints like seating capacity limits, the ER design ensures data integrity and supports the system's operational requirements. This structured approach lays the groundwork for an efficient, scalable, and normalized database, facilitating seamless data management and retrieval.

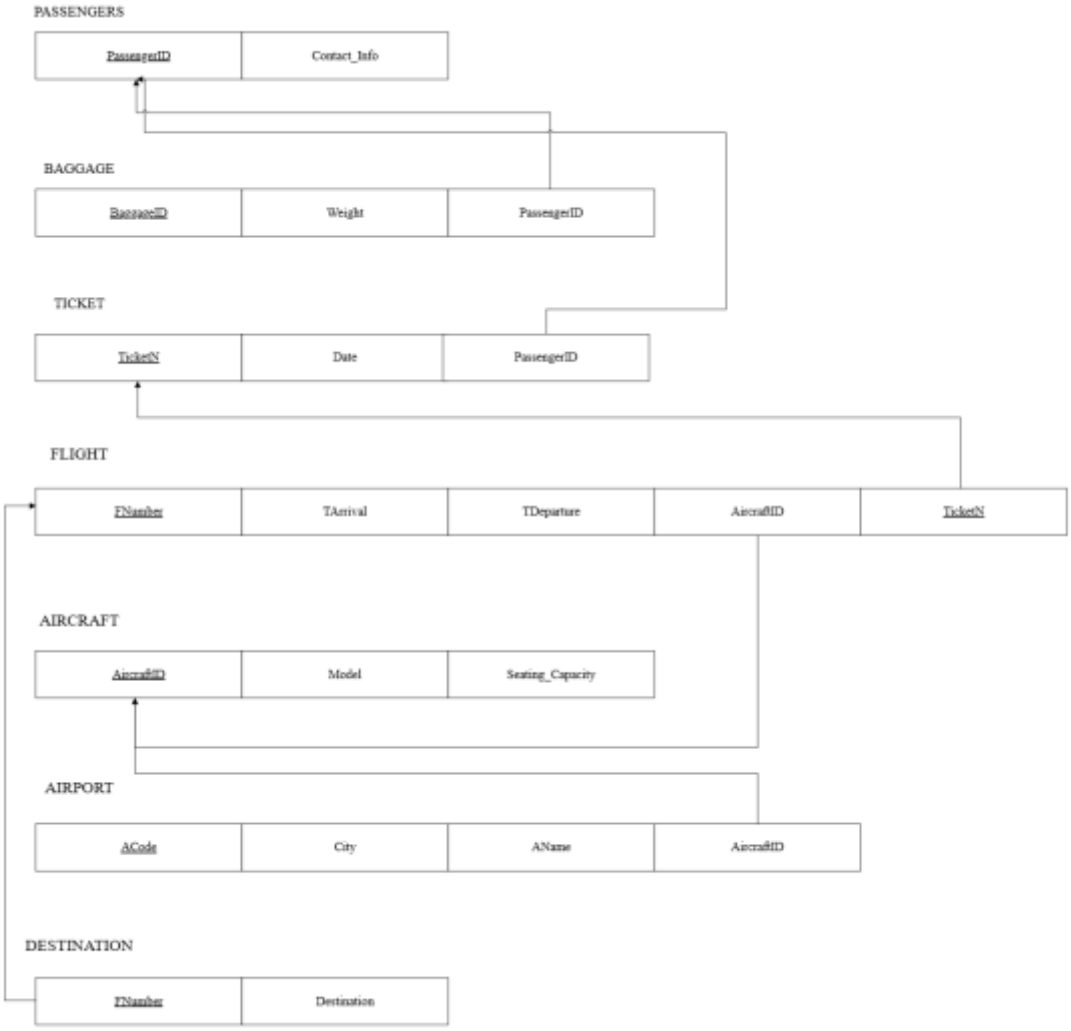


In designing the Entity-Relationship model, key decisions were made to ensure the system captures the airline reservation process comprehensively and aligns with operational requirements. The primary entities include *Passengers*, *Flights*, *Airports*, *Aircraft*, *Tickets*, and *Baggage*, which are essential components of the reservation system. Relationships like *Passengers Purchase Tickets*, *Tickets Book Flights*, *Flights Operate Aircraft*, and *Flights Depart From/Arrive At Airports* were established to reflect real-world processes. Attributes were assigned to each entity, such as *Passenger*

Name, *Flight Number*, *Airport Code*, and *Aircraft Seating Capacity*, ensuring critical information is stored while avoiding unnecessary complexity. Primary keys, including *Passenger ID* and *Airport Code*, were used to uniquely identify entities, while foreign keys maintain referential integrity in relationships. Constraints, such as ensuring a flight's seating capacity is not exceeded and requiring passengers to check baggage, were included to enforce real-world limitations. Finally, a one-to-one relationship was established between *Aircraft* and *Flight* to reflect the operational reality that each flight is assigned a single aircraft, while *Passengers* and *Tickets* have a one-to-many relationship, allowing passengers to book multiple flights. These decisions aim to balance simplicity, usability, and scalability for future system enhancements.

THE (RELATIONAL) LOGICAL DATABASE DESIGN

The (Relational) Logical Database Design translates the conceptual Entity-Relationship model into a structured relational schema optimized for efficient data storage and management. This step involves defining tables, attributes, primary keys, and foreign keys to represent entities and their relationships accurately. For the Airline Reservation System, key entities such as *Passengers*, *Flights*, *Airports*, and *Aircraft* are mapped into relations, with clear identification of constraints like seating capacity and unique airport codes. By ensuring normalization, the design minimizes redundancy and avoids anomalies in data operations. The relational schema serves as a foundation for implementing the database, enabling reliable and consistent handling of core functionalities such as ticket booking, flight scheduling, and passenger check-in.



The relational logical database design translates the conceptual ER model into a schema, where key decisions were made to ensure data integrity, normalization, and system efficiency. Each entity, such as *Passengers*, *Baggage*, *Ticket*, *Flight*, *Aircraft*, *Airport*, and *Destination*, was mapped into individual tables with attributes defined as columns. Primary keys, such as *PassengerID*, *BaggageID*, *TicketN*, and *FNumber*, were assigned to uniquely identify records, ensuring referential integrity. Foreign keys, such as *TicketN* in the *Passengers* and *Flight* tables, maintain relationships and enforce data consistency. Redundancy was minimized by adhering to normalization principles; for instance, passenger information is stored separately, with tickets and baggage linked by IDs to avoid duplication. Constraints, such as assigning *AircraftID* to flights and linking destinations through *FNumber*, reflect real-world relationships and operational rules. The design decision to separate *Airport* and *Destination* into distinct tables ensures flexibility for flights operating between multiple locations. Additionally, attributes like *Seating_Capacity* for aircraft and *Weight* for baggage were included to

capture operational constraints. These decisions ensure the design supports scalability, simplifies queries, and upholds data consistency across the system.

SECTION 2

NORMALIZATION

- Relation: Passengers
  Passengers (PasssengerID, Contact_Info)
  Key: PassengerID

PassengerID	Contact_Info
P001	daniel.jeun@example.com
P002	jayson.zelaya@example.com
P003	jane.smith@example.com
P004	sam.jones@example.com
P005	alice.wilson@example.com

PassengerID → Contact_Info
This relation is 3NF because:
- The key (PassengerID) determines all attributes
- There are no transitive dependencies or partial-key dependencies.

- Relation: Baggage
  Baggage (BaggageID, Weight, PassengerID)
  Key: BaggageID
  Foreign Keys: PassengerID

BaggageID	Weight	PassengerID
B001	20 kg	P001
B002	15 kg	P001
B003	25 kg	P002
B004	18 kg	P002
B005	22 kg	P004

BaggageID → Weight, PassengerID
This relation is in 3NF because:

- The key (BaggageID) determines all attributes.
- No transitive dependencies or partial-key dependencies exist.

- Relation: Ticket
  Ticket (TicketN, Date, FNumber)
  Key: TicketN
  Foreign Keys: Fnumber (references Flight). PassengerID

TicektN	Date	Fnumber	PassengerID
T001	2024-12-10	F001	P001
T002	2024-12-11	F002	P001
T003	2024-12-12	F003	P002
T004	2024-12-13	F004	P003
T005	2024-12-14	F005	P004

TicketN → Date, FNumber, PassengerID
This relation is in 3NF because:

-The key (TicketN) determines all attributes.
- No transitive dependencies or partial-key dependencies exist.

- Relation: Flight
  Flight (FNumber, TDeparture, TArrival, AircraftID, TicketID)
  Key: FNumber
  Foreign Keys: AircraftID (references Aircraft)

FNumber	TDeparture	TArrival	AircraftID	TicketID
F001	08:00 AM	12:00 PM	A001	101
F002	09:00 AM	01:00 PM	A002	102
F003	10:00 AM	02:00 PM	A003	103
F004	11:00 AM	03:00 PM	A004	104
F005	12:00 PM	04:00 PM	A005	105

FNumber → TDeparture, TArrival, AircraftID, Destination

This relation is in 3NF because:

- The key (FNumber) determines all attributes.
- No transitive dependecies or partial-key dependencies exist.

- Relation: Aircraft
  Aircraft (AircraftID, Model, Seating_Capacity)
  Key: AircraftID

Aircraft	Model	Seating_Capacity
A001	Boeing 737	180
A002	Airbus A320	200
A003	Boeing 747	300
A004	Embraer 190	100
A005	Bombardier CRJ	90

AircraftID → Model, Seating_Capacity
This relation is 3NF because:

- The key (AircraftID) determines all attributes.
- No transitive dependencies or partial-key dependencies exist.

- Relation: Airport
  Airport (ACode, City, AName, AircraftID)
  Key: ACode
  Foreign Keys: AircraftID (references Aircraft)

ACode	City	AName	AircraftID
JFK	New York	JFK International	A001
LAX	Los Angeles	LAX Airport	A002
ORD	Chicago	O'Hare Airport	A003
IAH	Houston	George Bush Inter	A004
SFO	San Francisco	SFO Airport	A005

ACode → City, AName, AircraftID
This relation is in 3NF because:

- The key (ACode) determines all attributes.
- No transitive dependencies or partial-key dependencies exist.

- Relation: Destination
  Key: FNumber

FNumber	Destination
F001	New York
F002	Los Angeles
F003	Chicago
F004	Houston
F005	San Francisco

FNumber → Destination
This relation is in 3NF because:
- The key (FNumber) determines all attributes.
- No transitive dependencies or partial-key dependencies exist.

SECTION 3

SQL STATEMENTS

1. ENHANCE RELATIONAL SCHEMA

```
CREATE TABLE Passengers (
        PassengerID INT PRIMARY KEY,
        Contact_Info VARCHAR(255),
        TicketID INT,
        BaggageID INT,
        FOREIGN KEY (TicketID) REFERENCES Ticket(TicketID),
        FOREIGN KEY (BaggageID) REFERENCES Baggage(BaggageID)
);

CREATE TABLE Baggage (
        BaggageID INT PRIMARY KEY,
        Weight DECIMAL(5, 2)
        FOREIGN KEY (PassengerID) REFERENCES Passengers(PassengerID)
);
```

```
CREATE TABLE Ticket (
        TicketID INT PRIMARY KEY,
        Date DATE,
        FlightNumber INT,
        FOREIGN KEY (FlightNumber) REFERENCES Flight(FlightNumber)
        FOREIGN KEY (PassengerID) REFERENCES Passengers(PassengerID)
);

CREATE TABLE Flight (
        FlightNumber INT PRIMARY KEY,
        TArrival DATETIME,
        TDeparture DATETIME,
        AircraftID INT,
        TicketID INT,
        FOREIGN KEY (AtributeID) REFERENCES Aircraft(AircraftID),
        FOREIGN KEY (TicketID) REFERENCES Ticket(TicketID)
);

CREATE TABLE Aircraft (
        AircraftID INT PRIMARY KEY,
        Model VARCHAR(50),
        Seating_Capacity INT
);

CREATE TABLE Airport (
        ACode CHAR(3) PRIMARY KEY,
        City VARCHAR(100),
        AName VARCHAR(100),
        AircraftID INT,
        FOREIGN KEY (AircraftID) REFERENCES Aircraft(AircraftID)
);

CREATE TABLE Destination (
        FlightNumber INT PRIMARY KEY,
        Destination VARCHAR(255),
        FOREIGN KEY (FlightNumber) REFERENCES Flight(FlightNumber)
);
```

2. INSERT AT LEAST 5 ROWS INTO EACH TABLE

```
INSERT INTO Passengers (PassengerID, Contact_Info)
 VALUES (1, ‘daniel.jeun@example.com’),
        (2, ‘jayson.zelaya@example.com’),
        (3, ‘jane.smith@example.com’),
        (4, ‘alice.brown@example.com’),
        (5, ‘mary.green@example.com’);

INSERT INTO Baggage (BaggageID, Weight, PassengerID)
 VALUES (201, 23.5, 1),
        (202, 18.2, 1),
        (203, 25.0, 2),
        (204, 15.8, 2),
        (205, 20.0, 3);

 INSERT INTO Ticket (TicketID, Date, FlightNumber, PassengerID)
 VALUES (101, ‘2024-11-20’, 301, 1),
        (102, ‘2024-11-21’, 302, 1),
        (103, ‘2024-11-22’, 303, 2),
        (104, ‘2024-11-23’, 304, 3),
        (105, ‘2024-11-24’, 305, 4);

INSERT INTO Flight (FlightNumber, TArrival, TDeparture, AircraftID, TicketID)
 VALUES (301, '2024-11-20 18:00:00', '2024-11-20 15:00:00', 401, 101),
        (302, '2024-11-21 20:00:00', '2024-11-21 16:00:00', 402, 102),
        (303, '2024-11-22 22:00:00', '2024-11-22 18:00:00', 403, 103),
        (304, '2024-11-23 23:00:00', '2024-11-23 19:00:00', 404, 104),
        (305, '2024-11-24 21:00:00', '2024-11-24 17:00:00', 405, 105);

INSERT INTO Aircraft (AircraftID, Model, Seating_Capacity)
 VALUES (401, 'Boeing 737', 150),
        (402, 'Airbus A320', 180),
        (403, 'Embraer E190', 120),
        (404, 'Boeing 787', 250),
        (405, 'Airbus A380', 800);

INSERT INTO Airport (ACode, City, AName, AircraftID)
VALUES ('JFK', 'New York', 'John F. Kennedy International', 401),
        ('LAX', 'Los Angeles', 'Los Angeles International', 402),
```

```
('ORD', 'Chicago', 'O\'Hare International', 403),
('DFW', 'Dallas', 'Dallas/Fort Worth International', 404),
('ATL', 'Atlanta', 'Hartsfield-Jackson Atlanta', 405);
```

```
INSERT INTO Destination (FlightNumber, Destination)
VALUES (301, 'London'),
       (302, 'Paris'),
       (303, 'Tokyo'),
       (304, 'Sydney'),
       (305, 'Dubai');
```

3. FOUR SQL QUERIES

QUERY 1: GROUP BY

Retrieve the aircraft model and the total seating capacity for each model. Group the results by aircraft model.

```
SELECT Model, SUM(Seating_Capacity) AS TotalSeatingCapacity
FROM Aircraft
GROUP BY Model;
```

QUERY 2: GROUP BY and HAVING

Retrieve the city of each airport and the total seating capacity of all aircraft associated with that airport, but only for cities where the total seating capacity exceeds 500. Group the results by city.

```
SELECT Airport.City, SUM(Aircraft.Seating_Capacity) AS TotalSeatingCapacity
FROM Airport
JOIN Aircraft ON Airport.AircraftID = Aircraft.AircraftID
GROUP BY Airport.City
HAVING SUM(Aircraft.Seating_Capacity) > 500;
```

QUERY 3: Nested Query with ALL
Retrieve the model of the aircraft that has the highest seating capacity among all aircraft, excluding the aircraft with AircraftID equal to 401.

```
SELECT Model
FROM Aircraft
WHERE Seating_Capacity >= ALL (SELECT Seating_Capacity FROM Aircraft WHERE
AircraftID <>  401);
```

QUERY 4: Nested Query with IN
Retrieve the PassengerID and Contact_Info of all passengers who have tickets for the flight with FlightNumber equal to 301.

```
SELECT PassenegrID, Contact_Info
FROM Passengers
WHERE TicketID IN (SELECT TicketID FROM Ticket WHERE FlightNumber = 301;)
```

SECTION 4

GROUP EXPERIENCE WITH THE PROJECT

Working on the airline reservation system project was both challenging and rewarding for our group. Each step provided unique learning opportunities and required a significant degree of collaboration. Here's an overview of our experience:

- **Most Difficult Steps**:
  The most challenging part of the project was normalizing the relations to 3NF. While identifying the functional dependencies and breaking down the relations to eliminate partial and transitive dependencies was conceptually clear, implementing the process in practice often led to unexpected complexities. Specifically, understanding how foreign keys would cascade through multiple tables while maintaining the integrity of the schema required multiple revisions and debates within the group. Additionally, designing the Entity-Relationship Diagram (ERD) was time-consuming as we had to ensure that every entity and relationship was appropriately defined to avoid redundancy or missing data relationships.

- **Easiest Steps**:
  The data entry and sample SQL statements were relatively straightforward once we finalized the normalized schema. Writing 'CREATE TABLE' and 'INSERT' statements allowed us to apply the theoretical database design practically, and seeing the schema come to life through SQL brought clarity and satisfaction. Additionally, defining the initial list of entities and attributes was an easier step since the problem statement provided clear guidelines.

- **Unexpected Learnings**:
  One of the most surprising insights from this project was the realization of how interconnected database components are. While we initially thought that splitting data into multiple relations (tables) might complicate queries, we learned how normalization simplifies data integrity and reduces redundancy in the long term. Another unexpected challenge was balancing normalization with practicality, over-normalization led to more complexity than we anticipated, and finding the balance was a valuable lesson.

- **What We Would Do Differently**:
  If we had to redo this project, we would focus more on the planning phase, dedicating additional time to refine the ERD before proceeding with normalization. A more detailed blueprint would have saved us time later in resolving ambiguities during schema design. Additionally, we would have incorporated test queries earlier in the process to verify our schema's functionality and identify potential issues before moving to the implementation phase.

FINAL COMMENTS AND CONCLUSIONS

This project gave us a comprehensive understanding of the practical aspects of database design, including normalization, schema creation, and SQL implementation. It bridged the gap between theoretical concepts and their application in a real-world scenario. Beyond technical skills, the project emphasized the importance of teamwork, communication, and iterative problem-solving.

As a group, we feel proud of the final product we created. The process helped us develop a solid foundation for designing relational databases, which will undoubtedly benefit us in future courses and professional work. While the project was challenging, the experience was ultimately rewarding, and it reinforced the importance of careful planning and logical design in database systems.

We appreciate the opportunity to work on this project, and we are confident that the lessons we learned will carry forward into future endeavors.