ITCS413: Database Design

Project 4: Implementing a Database

Airline: Star Airline

(March 2025)

Prepared by

6588070 Nakarin Phoorahong

6588096 Panipak Sittiprasert

6588183 Achiraya Mankham

Presented To

Asst.Prof. Dr. Charnyote Pluempitiwiriyawej

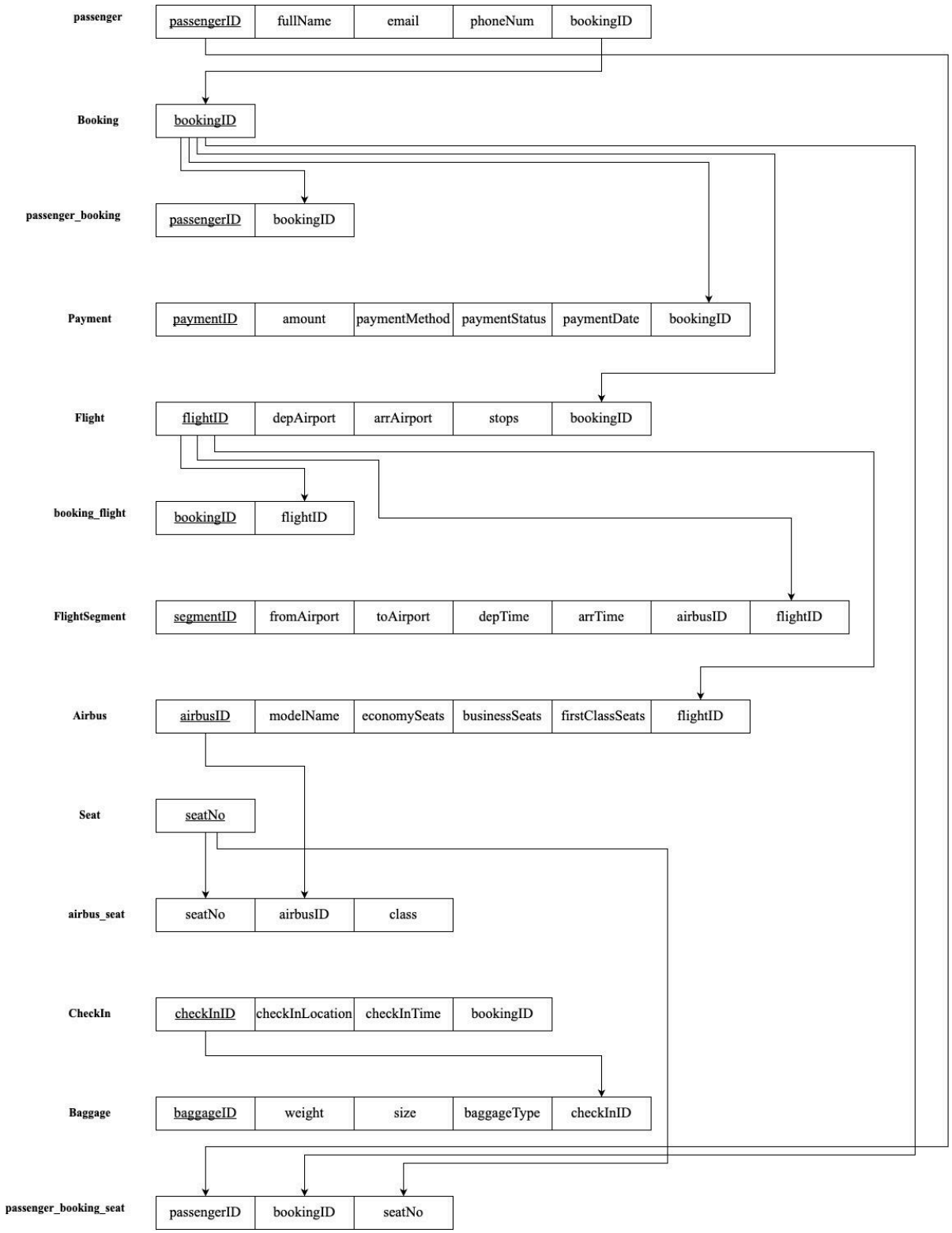
Faculty of Information and Communication Technology

Mahidol University

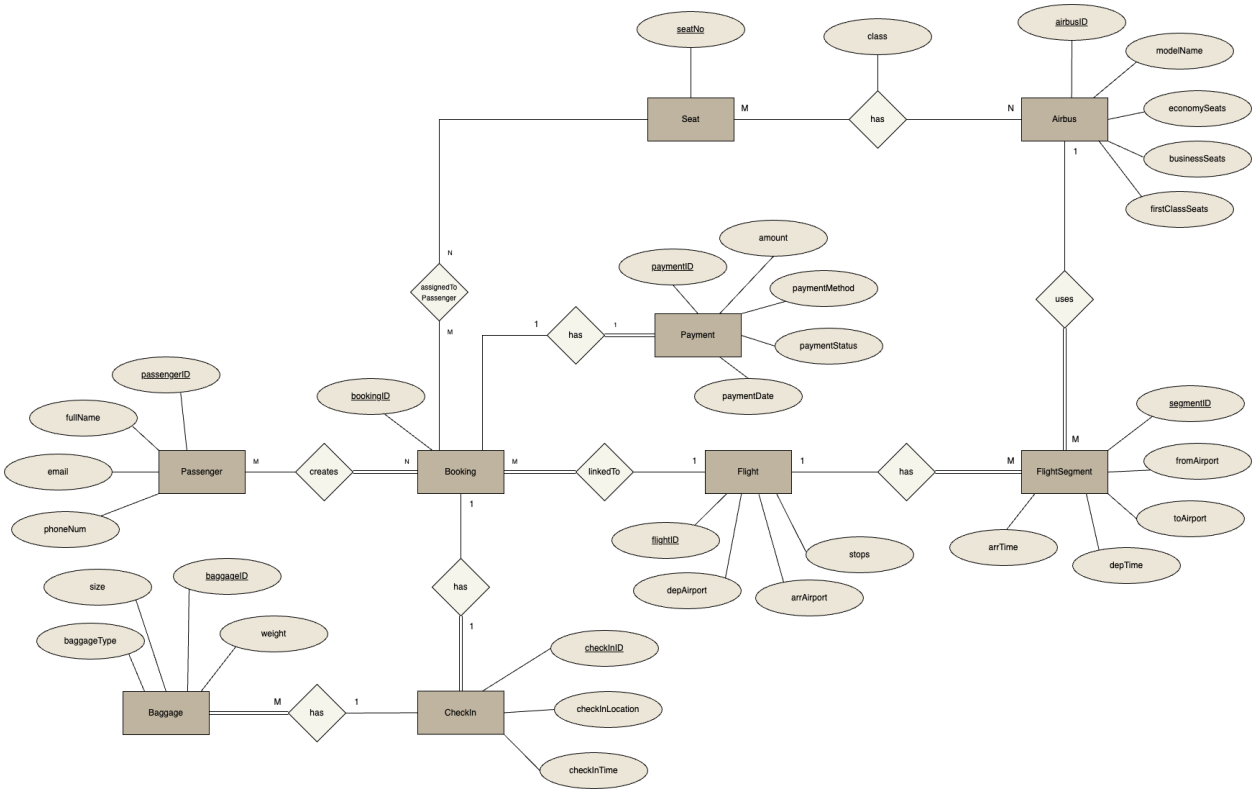
Table of contents

Relational Database Schema and ER Diagram

Relational Database Schema



ER Diagram


seatNo
class
airbusID
modelName
Seat
M
has
N
Airbus
economySeats
businessSeats
1
firstClassSeats
amount
paymentID
paymentMethod
N
assignedTo Passenger
M
1
has
1
Payment
paymentStatus
uses
paymentDate
passengerID
bookingID
segmentID
fullName
M
email
Passenger
M
creates
N
Booking
M
linkedTo
1
Flight
1
has
M
FlightSegment
fromAirport
phoneNum
1
flightID
stops
toAirport
arrTime
depTime
depAirport
arrAirport
size
baggageID
has
1
baggageType
weight
checkInID
Baggage
M
has
1
CheckIn
checkInLocation
checkInTime

Normalized Relational Database Schema and ER Diagram

In this project, we normalized to the Third Normal Form (3NF)
To do that we have to normalize through the UNF then 1NF then 2 NF and finally to 3NF.
In doing that, we will go through each table in each step.

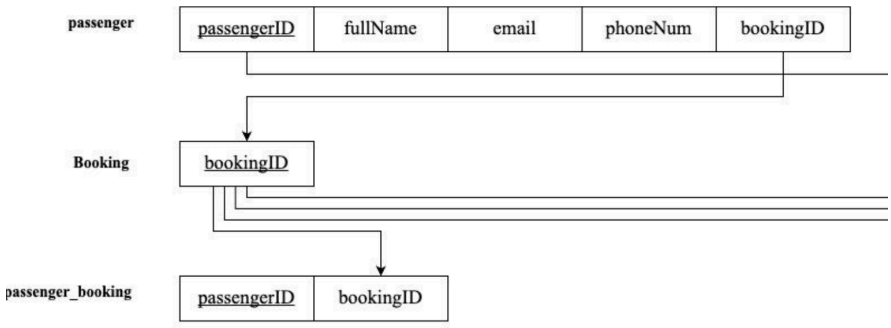
The Unnormalized form (UNF)

Unnormalized form are the tables that has one or more repeating group
There are no tables that need to be fixed for the UNF.

The First Normalized forms (1NF)

First normalized forms remove repeating groups, remove multivalued forms.

Passenger and Passenger_booking already normalized from the design phase.
As you can see, the passenger_booking is already there



Payment and Booking tables have a 1:1 relationship so it does not involve this step.

Flight table has to be normalized to add one separate table called booking_flight which has attributes of bookingID and flightID because 1 flight can have multiple bookings which violate the first normal form rules that it can not have multivalued.
So, the updated database of these table will look like this:

The flight table

flightID	depAirport	arrAirport	stops
901	John F. Kennedy International Airport (JFK)	Berlin Brandenburg Airport (BER)	1
902	Tokyo Haneda Airport (HND)	Los Angeles International Airport (LAX)	1

The booking table

bookingID
101
102

The booking_flight table

bookingID	flightID
101	901
102	902

Flight_Segment, Airbus, Check and Baggage tables are already in the 1NF because there is no repeating group among those tables.

The Second Normal Form (2NF)

Second Normal Form is when the primary key contains multiple attributes, removing partial dependencies; every non-primary solely depends on the primary key.

Passenger table will be passengerID, fullName, email, phoneNum. Cut bookingID out because it already has “passenger_booking” which contains passengerID and bookingID that follow the 1NF and it does not comply with the second normal form.

Analysis of dependencies

{passengerID} -> {fullName, email, phoneNum}; fullName, email, phoneNumber depend on passengerID.

{passengerID} -> {bookingID} bookingID does not depend on passengerID. Therefore the passenger_booking is needed to comply with the second normal form rules which will be solved by creating passenger_booking(passengerID, bookingID).

Payment Table already 2NF because its attributes depend on paymentID.

Analysis of dependencies

{paymentID} -> {amount, paymentMethod, paymentStatus, paymentDate,bookingID}
Without booking, there can not be payment.

The flight table same as the passenger table separate booking from the table

Analysis of dependencies

Because {flightID} -> {depAirport, arrAirport, stops} departureAirport, arrivalAirport, stops are depend on flight(PK), but bookingID does not depend on flight, so therefore the table booking_flight was created. booking_flight(bookingID, flightID).

flightSegment table already in second normal form because all non-primary key attributes depend on the primary key.

Airbus_seat table is already in second normal form because all non-primary key attributes depend on the composite key (<u>seatNo, airbusID</u>).

In the airbus table, we removed flightID because it does not depend on airbusID because one airbus can fly multiple flights and let flight_segment to describe which airbus is used in a specific flight.

Analysis of dependencies

So, the updated airbus table will be airbus(airbusID, modelName, economySeats, businessSeats, firstClassSeats).

{airbusID} -> {modelName, economySeats, businessSeats, firstClassSeats, flightID}

CheckIn table is already in 2NF because all non-primary key attributes depend on the primary key checkInID. For example, in order to check in, you have to book first and that is where bookingID is generated.

Baggage table is already in second normal form because all non-primary key attributes depend on the primary key baggageID. In the case of checkInID, there is a simple explanation: passengers must checkIn before they can have BaggageID.

Passenger_booking_seat table is already in second normal form because all non-primary key attributes depend on the primary key passengerID.

The Third Normal Form (3NF)

Removed Transitive dependencies; non-key attributes depend on another non-key attribute.

Passenger table is already in the third normal form because there is no transitive dependency.

Payment table is already in the third normal form because there is no transitive dependency.

Flight table after update made in second normal form, flight table already in third normal form because there is no transitive dependency.

Flight Segment table is already in the third normal form because there is no transitive dependency.

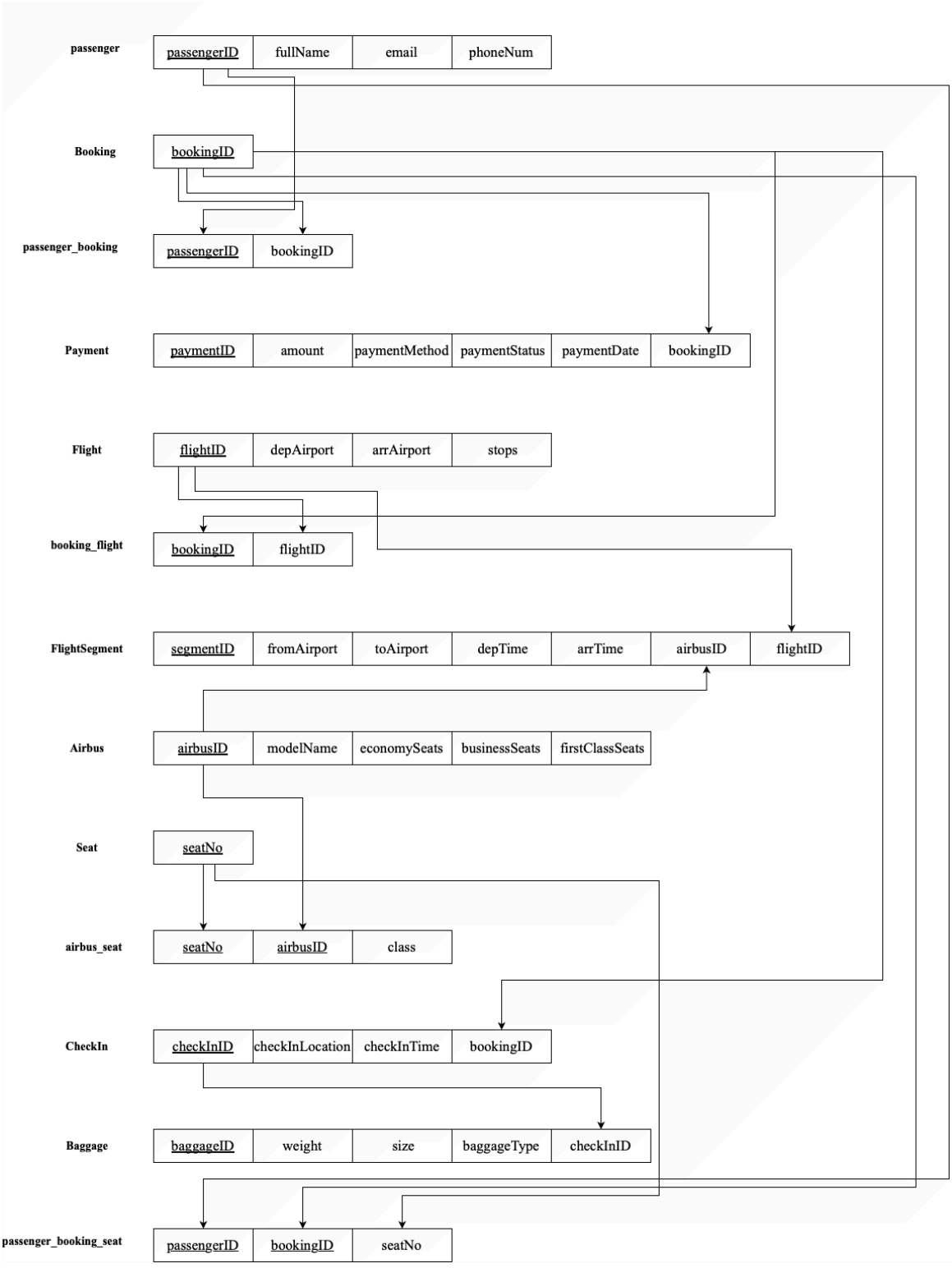
Airbus table is already in the third normal form because there is no transitive dependency with update made in second normal form to rule out flightID since it does not depend on.

Airbus_seat table is already in the third normal form because there is no transitive dependency.

CheckIn table is already in the third normal form because there is no transitive dependency.

Baggage table is already in the third normal form because there is no transitive dependency.

Normalized Relational Schema



Verification of Relational Database Schema

Verification Process

We transform from the unnormalized to first normal and then to second level and finally in third normal form successfully with help from ChatGPT and internet, to ensure that our relational database schema is correctly designed and normalized, we followed a structured verification process. The verification process included the following steps:

- **Schema Review:** We thoroughly reviewed the relational schema to confirm that all relationships, primary keys, and foreign keys were correctly defined. This ensured that the database structure accurately represented the business requirements of Star Airline.

- **Normalization Validation:** We checked whether each table adhered to the rules of normalization up to the Third Normal Form (3NF). This process included:

    - Ensuring there were no repeating groups (1NF).

    - Eliminating partial dependencies (2NF).

    - Removing transitive dependencies (3NF).

- **Data Integrity Checks:** We verified that the integrity constraints were properly enforced to maintain consistency within the database. This included:

    - Ensuring referential integrity between related tables.

    - Checking for redundant data or unnecessary duplication.

- **Functional Dependency Analysis:** Each table's attributes were analyzed to ensure that every non-key attribute was fully functionally dependent on the primary key.

Findings and Results

Through our verification process, we obtained the following results:

- **Normalization Compliance:** All tables were successfully normalized to 3NF. This ensured an optimized structure with minimal redundancy and better data integrity.
- **Relationship Validation:** The entity relationships were correctly established, with many-to-many relationships properly managed through junction tables such as passenger_booking and booking_flight.
- **Foreign Key Enforcement:** All foreign keys were correctly implemented, ensuring referential integrity. This prevented orphaned records and maintained consistency across tables.
- **Efficient Query Performance:** The refined database structure allowed for efficient querying and data retrieval, reducing unnecessary data redundancy and improving performance.
- **Logical Consistency:** No anomalies were found in data insertion, updates, or deletions, confirming the integrity of the database design.

Database Implementation Discussion

Challenges

- The mapping from conceptual to logical and then to physical design is very challenging because it is hard to look and get the whole picture from the starting point. Changes have to be made throughout the design process. Much research has to be performed to get the whole picture. The first plan is never the final plan. During development, many changes were needed such as adjusting table structures, adding extra relationships, and modifying data fields. Every change also affected other parts of the database, so updates had to be done carefully and in the correct order. Some parts of the database needed many-to-many relationships, such as bookings and passengers (one booking can have many passengers, and one passenger can have many bookings). Setting up these relationships correctly required using bridge tables and foreign keys.

- Seat assignments were also a challenge. Each plane had different numbers of seats for Economy, Business, and First Class. When assigning seats to passengers, the system needed to group people in the same booking close together and make sure they were in the correct class. This required thoughtful logic.

- Flights have departure and arrival times, check-in times, and more. Ensuring that these times were consistent (for example, check-in happening before the flight leaves) was important. Mistakes in time data could cause confusion or incorrect results in queries.

- The database required a lot of data to simulate a real-world airline. This included hundreds of flights, passengers, and seat entries. Managing this data, making sure IDs matched correctly, and avoiding duplication took a lot of attention.

Solutions

- Take time to research all related information to the selected business domain, which is Airline, and consult information that has gathered with experts or people who have experienced.
- Normalization and Refinements, applying normalization rules ensured that data was effectively organized while ensuring integrity and relationships.
- Since many tables depended on each other through foreign keys, insert statements were planned carefully in order. For example, seat entries had to be added before airbus_seat, and booking had to exist before adding passengers to it.
- Instead of designing new seat layouts for every flight, one standard set of seat labels (like '1A', '1B', etc.) was created and reused. This saved a lot of time and kept things consistent.
- Tables like payment, checkIn, and baggage used IDENTITY to automatically generate primary keys. This saved time and reduced the chance of duplicate values.

Limitations

- Complex Queries, some queries involving multiple joins could be computationally expensive, potentially impacting performance under heavy load.
- If two users try to book the same seat at the same time, the current setup doesn't prevent conflicts. In a real system, more checks or locking mechanisms would be needed to avoid these problems.
- Some processes, like automatically assigning seats or creating baggage entries after check-in, must be done manually or using SQL scripts. These could be improved by adding stored procedures or triggers.
- While the system works for testing and demonstration, it may not handle very large amounts of data (millions of passengers or bookings) without slowdowns. Features like indexing or partitioning would help in that case.
- Many rules (like seat class matching or time checking) are handled by the SQL logic or manually, but not all validations are enforced at the database level. A real application would need to perform these checks too.

Number of Records in Each Table

By using the object catalog views script, it provides an overall look at the structure of the database.

```
2383    SELECT
2384        t.NAME AS TableName,
2385        s.Name AS SchemaName,
2386        p.rows AS RowCounts
2387    FROM
2388        sys.tables t
2389    INNER JOIN
2390        sys.schemas s ON t.schema_id = s.schema_id
2391    INNER JOIN
2392        sys.partitions p ON t.object_id = p.object_id
2393    WHERE
2394        p.index_id IN (0, 1)  -- 0 = heap, 1 = clustered index
2395    GROUP BY
2396        t.Name, s.Name, p.rows
2397    ORDER BY
2398        p.rows DESC;
```

Results | Messages

	TableName	SchemaName	RowCounts
1	airbus_seat	dbo	1244
2	passenger	dbo	800
3	flight_segment	dbo	317
4	flight	dbo	251
5	seat	dbo	232
6	baggage	dbo	157
7	passenger_booking	dbo	155
8	passenger_booking_seat	dbo	155
9	booking	dbo	60
10	booking_flight	dbo	60
11	checkIn	dbo	60
12	payment	dbo	60
13	airbus	dbo	7