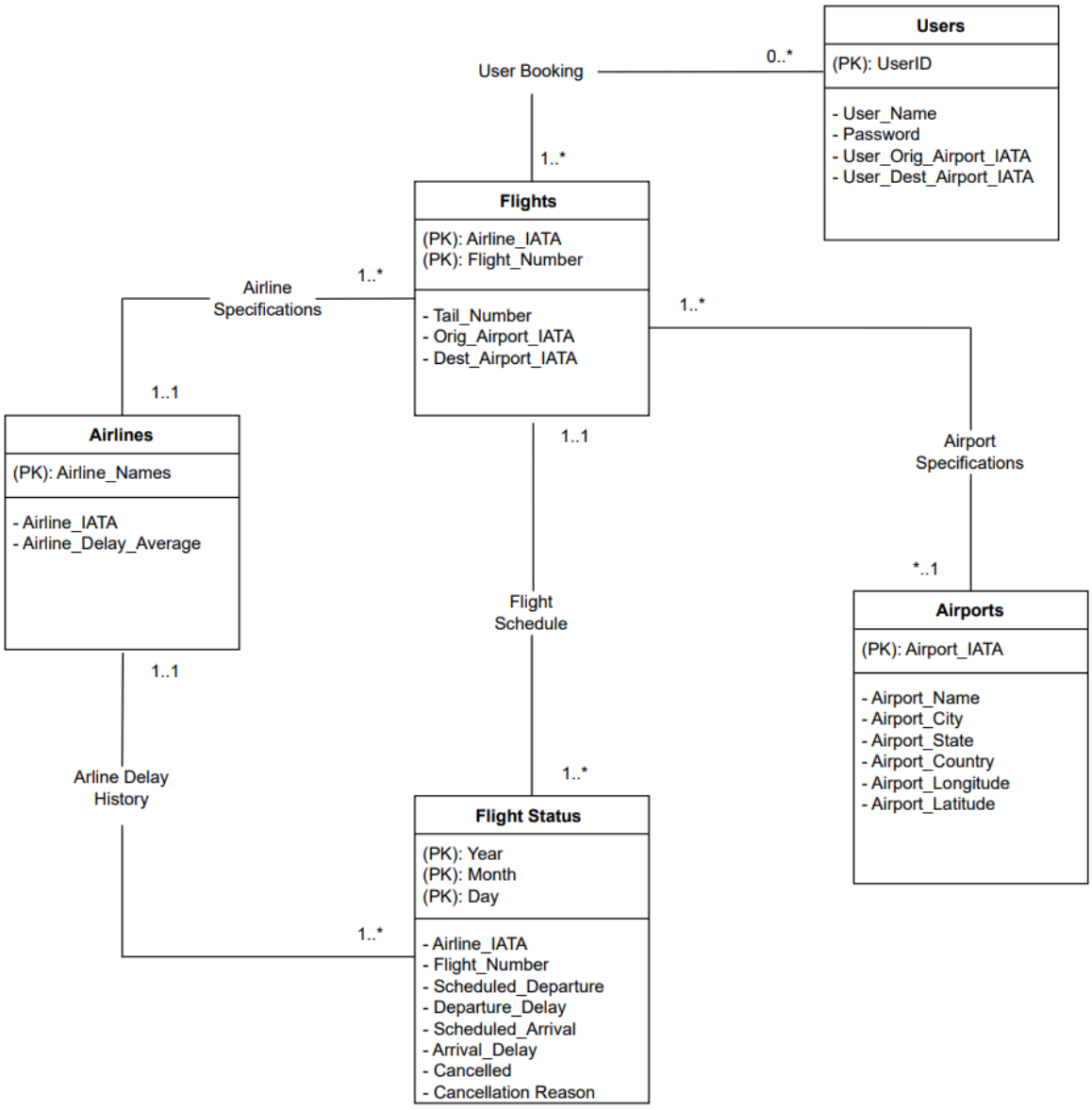
Flight Insight: Database Design

Authors: Abhee Jani, Jash Parekh, Advaith Yeluru, Arul Viswanathan

Course: CS 411
Instructor: Adbusalam Alawini
Submission Date: Oct. 2, 2023

UML Diagram Post Normalization



Dependencies:

UserID -> UserName, Password, UserOrigAirportIATA, UserDestAirportIATA
AirlineIATA, FlightNumber -> TailNumber, OrigAirportIATA, DestAirportIATA
AirportIATA -> AirportName, AirportCity, AirportState, AirportCountry, AirportLatitude, AirportLongitude
AirlineNames -> AirlineIATA, AirlineDelayAverage
AirlineIATA, FlightNumber, Year, Month, Day -> ScheduledDeparture, DepartureDelay, ScheduledArrival, ArrivalDelay, Canceled, CancellationReason

Normalization of Database:

Previously, we had 3 tables where the data was not fully normalized. A description of our previous tables is below:

1. UserFlights:
    - Contains user-related and flight booking information
    - Attributes: UserID, UserName, UserEmail, FlightNumber, DepartureTime, ArrivalTime, BookingDate, DepartureAirport, ArrivalAirport, AirlineName
2. FlightDetails:
    - Contains detailed flight information, airport, and airline specifications.
    - Attributes: FlightNumber, DepartureTime, ArrivalTime, DepartureAirportName, DepartureAirportLocation, ArrivalAirportName, ArrivalAirportLocation, AirlineName, AirplaneModel, FlightCapacity
3. AirlineDelays:
    - Contains delay history and real-time flight status information.
    - Attributes: AirlineName, FlightNumber, ScheduledDepartureTime, ScheduledArrivalTime, ActualDepartureTime, ActualArrivalTime, AverageDelayDuration, DelayFrequency

Why did we normalize these tables using 3NF to get our current schema?

1. Redundancy:
    - In the UserFlights table, every time a user books a flight, the flight details, airport names, and airline name would be repeated.
    - In the FlightDetails table, for each flight, the airline and airport details would be repeated.
    - The AirlineDelays table would repeatedly store the airline name for every delay entry.
2. Anomalies:

- Insert Anomaly: We couldn't add an airline's delay history unless there was a corresponding flight in the AirlineDelays table.
- Update Anomaly: If the name of an airline changed, it would have to be updated in multiple places.
- Delete Anomaly: Deleting a flight from FlightDetails might unintentionally remove useful information about an airport or airline if it's not present in any other row.

3. Data Integrity:
    - There's a risk of entering inconsistent data. For instance, a typo in the airline name in one of the bookings in UserFlights would make it hard to relate that booking to the correct airline.
4. Efficiency:
    - Having denormalized tables could be inefficient in terms of storage because of repeated data. Also, queries that require data aggregation or distinct values could become slower.

Why did we not normalize the table using BCNF?

1. Simplicity:
    - 3NF strikes a good balance between minimizing redundancy and maintaining simplicity. From our understanding, it’s easier to understand and work with.
    - BCNF can sometimes lead to a decomposition that, while reducing redundancy and dependency, can make the schema more complex and harder to understand and work with.
2. Query Performance:
    - From our research, BCNF can sometimes lead to more tables than 3NF, and querying across multiple tables can be computationally expensive due to the requirement of multiple JOIN operations.
    - Ensuring efficient query execution might sometimes be prioritized, especially in scenarios where real-time data retrieval is crucial, like in flight booking systems.

How did we normalize the database?

- Step 1: Ensuring that all entries in a column are of the same kind and uniquely identifying each row with a primary key.
- Step 2: Ensuring that all non-key attributes are fully functionally dependent on the primary key.
- Step 3: Ensuring that all the columns are functionally dependent only on the primary key.

By moving to a 3NF normalized schema like the one we provided:

- Each piece of information is stored once and only once, reducing redundancy.

- The data integrity is maintained.
- Relationships are clearly defined, making the database more flexible and scalable.
- Anomalies are minimized, making CRUD operations (Create, Read, Update, Delete) more straightforward and safer.

Assumptions of the UML diagram:

- Geographic Data Accuracy: Assuming that the latitude and longitude information associated with airports and users is accurate and up-to-date, as inaccuracies could affect flight recommendations and location-based services.
- Accuracy of Data: We are assuming that the application of the 2015 data has direct relevance to flights today and that the delay information is applicable to flights today.
- Assuming that each individual can find their latitude and longitude. Also this is assuming that each individual would have their respective longitude and latitude
- Each flight only takes off once a day (By flight we are referring to a unique origin and destination for a specific plane)
- Each user is limited to one flight per booking. The users themselves are able to have multiple bookings but each booking needs to have a unique flight associated with it.
- Flight Data Accuracy: We assume that the 2015 flight delay dataset is accurate and reliable, and it includes all relevant information about flights, such as departure and arrival times, airline details, origin and destination airports, and delays.
- User Authentication: Assuming that there is a secure authentication mechanism in place to ensure that each user can only access their own booking and flight information. This is also assuming that it wouldn't be possible for multiple users to access/modify each others' booking information
- Booking System: We assume that there is a booking system that allows users to search for flights, select flights, and make bookings.
- Flight Availability: Each flight can be identified by a unique identifier (e.g., flight number or combination of airline and flight number), and it is assumed that this information is available for users to search and book flights.
- Unique Booking Limit: Each user is limited to one booking at a time. This assumption ensures fairness and availability of flights to all users.
- Proximity of User Location: We assume that the user's latitude and longitude information is indeed in proximity to the departure airport they are interested in. This proximity can be used to provide relevant flight recommendations and calculate travel times to the airport.
- No Same-Time Conflict: Assuming that the system enforces the rule that no two flights from the same airline can have the same origin and destination at the same time to avoid scheduling conflicts.
- Flight Scheduling: We assume that flights in the dataset are scheduled to take off once a day, and there are no multiple departures for the same flight on the same day.

- Data Integration: The dataset and user information are assumed to be integrated into the system, allowing users to access real-time flight information and make bookings based on the available data.
- Each airline from the 2015 table is still active.

Relationship/Cardinality:

Relational Schema:

1) Users (

    UserID VARCHAR(100) [PK]
    Password VARCHAR(255)
    User_Orig_Airport VARCHAR(100) [FK to flights.orig_IATA]
    User_Dest_Airport VARCHAR(100) [FK to flights.dest_IATA])

2) Airports (Airport_IATA VARCHAR(100) [PK]
    Airport_City VARCHAR(255)
    Airport_State VARCHAR(255)
    Airport_Country VARCHAR(255)
    Airport_Longitude VARCHAR(100)
    Airport_Latitude VARCHAR(100))

3) Flight_Status (Year INT [PK]
    Month INT [PK]
    Day INT [PK]
    Scheduled_Departure VARCHAR(255)
    Departure_Delay INT
    Scheduled_Arrival VARCHAR(255)
    Arrival_Delay INT
    Canceled VARCHAR(100)
    Cancellation Reason VARCHAR(255)

    Airport_IATA VARCHAR(100) [FK to flights.Airline_IATA]
    Flight_Number VARCHAR(100) [FK to flights.Flight_Number])

4) Flights (Airport_IATA VARCHAR(100) [PK]
   Flight_Number VARCHAR(100) [PK]
   Tail_Number VARCHAR(100) [FK to Airlines.Airline_Names]
   Orig_Airport_IATA VARCHAR(30) [FK to Airports.Airport_IATA]
   Dest_Airport_IATA VARCHAR(30) [FK to Airports.Airport_IATA]
)

5) Airlines (Airline_Names VARCHAR(255) [PK]
   Airline_IATA VARCHAR(100)
   Airline_Delay_Average INT)

Description of Relational Cardinality

Users to Flights:

- This is a many-to-many relationship because each user can have multiple flight bookings (one user can book multiple flights), but each flight is associated with many users (as in a singular flight can have many passengers).

Flights to Users:

- This is a many-to-one relationship because many flights can be associated with one user (many flights can belong to the same user), but each flight booking is uniquely linked to one user.

Flights to Airports:

- This is a many-to-one relationship because many flights depart from and arrive at the same airports (multiple flights can share the same departure and arrival airports), but each flight is uniquely connected to one departure airport and one arrival airport.

Airports to Flights:

- This is a one-to-many relationship because each airport can have multiple flights departing from or arriving at it (one airport can serve as the departure or arrival point for multiple flights), but each flight is associated with one departure airport and one arrival airport.

Flights to Flight Status:

- This is a one-to-many relationship because each flight can have multiple status entries (one flight can have multiple records indicating its status on different days), but each status entry is associated with one flight.

Flight Status to Flight:

- This is a one-to-one relationship because each status entry corresponds to one specific flight on a particular day, and each flight has only one status entry for that specific day.

Flight Status to Airlines:

- This is a one-to-one relationship because each status entry corresponds to one specific airline's flight on a specific day, and each airline's flight has only one status entry for that specific day.

Airlines to Flight Status:

- This is a one-to-many relationship because each airline can have multiple status entries for its flights on different days (one airline can have multiple flight status records), but each status entry is associated with one airline.

Airlines to Flights:

- This is a one-to-many relationship because each airline can operate multiple flights (one airline can have multiple flights), but each flight is uniquely linked to one airline.

Flights to Airlines:

- This is a one-to-one relationship because each flight is operated by one specific airline, and each airline operates one specific flight.