UML Diagram

Assumptions:

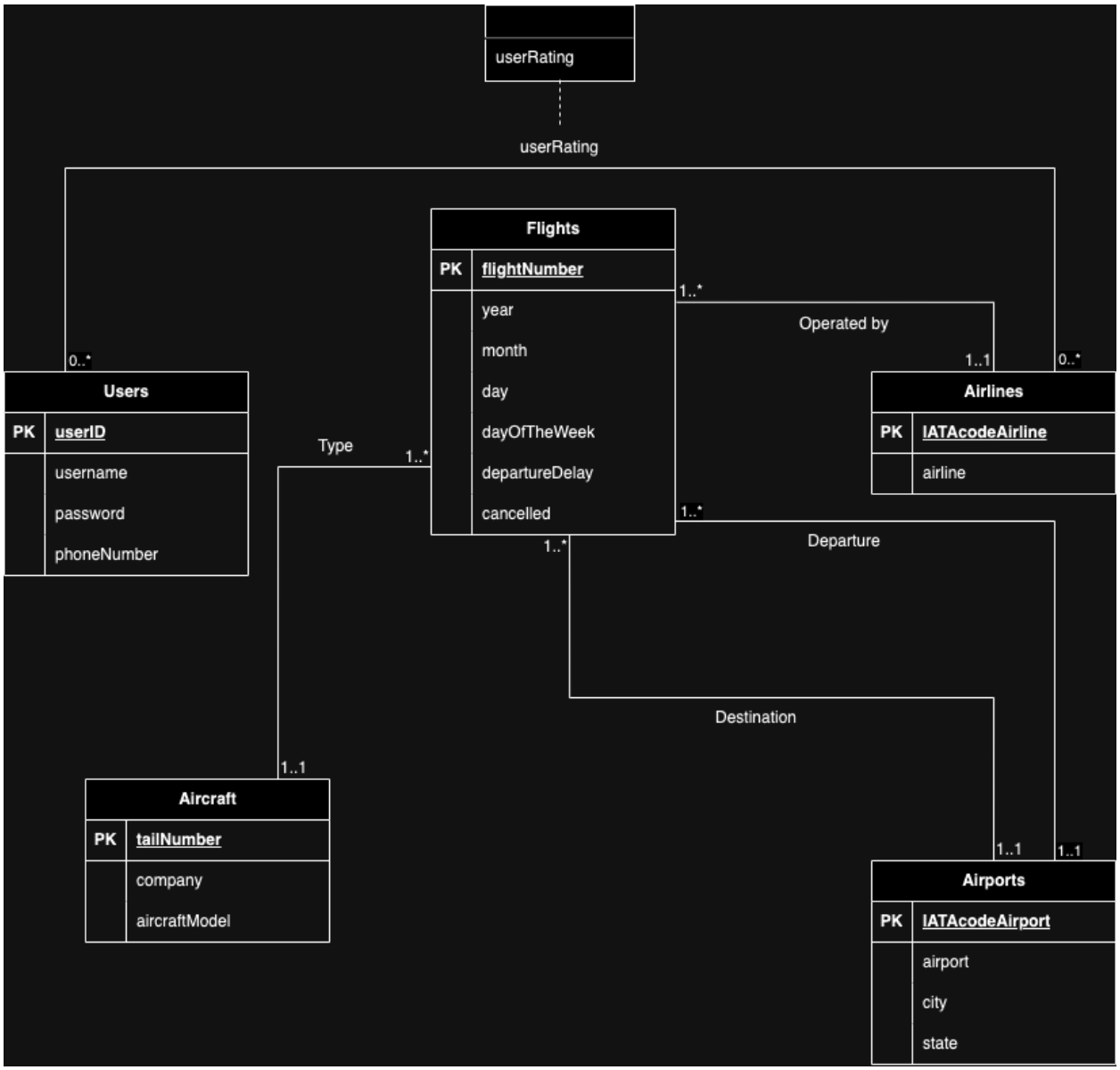
- We are assuming every user in our system has 0 or more airline ratings
- We are assuming every airline has 0 or more users from our system
- Each user rating will be out of 5 points
- We are assuming every flight belongs to only 1 airline
- We are assuming every airline has at least 1 airplane/flight
- We are assuming every airport is working and has at least 1 flight (for departing and arriving flights)
- We are assuming every flight, if not in path, has at least 1 departure airport
- We are assuming every flight, if not in path, has at least 1 destination airport
- We are assuming every flight has 1 aircraft
- We are assuming every aircraft can belong to more than 1 flight
- Our dataset will only contain flights in the United States
- Our dataset will not contain any diverted flights
- We will not take into account layover flights. Flights from one location to another to another will be considered 2 different flight paths, rather than 1 flight path with a layover
- Our dataset will not contain reasons for flight delays

Relationships & Cardinality:

- UserRating: users have a rating of none or more airlines and each airline will have none or more ratings
    - Users to airlines has a 0-many relationship because every user in our database can include (or choose to not include) a rating for 0 or more airlines
    - Airlines to users has a 0-many relationship because every airline in our database can have 0 or more ratings
- Operated by: airlines have at least 1 flight and each flight has only 1 airline
    - Airlines to flights has a 1-many relationship because every airline has at least 1 flight (airlines can have more than 1) since all airlines are assumed to be running
    - Flights to airlines has a 1-1 relationship because every flight has only 1 airline (flights cannot have more than 1) since flights cannot be used for multiple airlines
- Departure: airports have at least 1 flight and each flight has at only 1 departure airport
    - Airports to flights has a 1-many relationship because airports have at least 1 flight departing at all times (airports can have more than 1)
    - Flights to airports has a 1-1 relationship because every flight is linked to only 1 airport through origin airport
- Destination: airports have at least 1 flight and each flight has at only 1 destination airport
    - Airports to flights has a 1-many relationship because airports have at least 1 flight landing at all times (airports can have more than 1)

    - Flights to airports has a 1-1 relationship because every flight is linked to only 1 destination through destination airport
- Type: aircrafts have at least 1 flight and each flight has only 1 aircraft
    - Aircrafts to flight has a 1-many relationship because every aircraft has at least 1 flight (aircrafts can be used for more than 1 flight)
    - Flights to aircraft has a 1-1 relationship because every flight has only 1 aircrafts (flights cannot have more than 1) since aircrafts cannot be used for multiple flights in progress

Diagram:



Relationship Schema:

Users (
    userID: INT [PK],
    username: VARCHAR(100),
    password: VARCHAR(100),
    phoneNumber: VARCHAR(15)
);

Flights (
    flightNumber: INT [PK],
    airline: VARCHAR(100) [FK for Airports.airline]
    year: INT,
    month: VARCHAR(15),
    day: VARCHAR(2),
    dayOfTheWeek: VARCHAR(15),
    originAirport: VARCHAR(100) [FK for Airports.originAirport],
    destinationAirport: VARCHAR(100) [FK for Airports.destinationAirport],
    departureDelay: INT,
    cancelled: INT
);

Airlines (
    IATAcodeAirline: VARCHAR(3) [PK],
    airline: VARCHAR(100),
);

Aircraft (
    tailNumber: INT [PK],
    company: VARCHAR(100),
    aircraftModel: VARCHAR(100),
);

Airports (
    IATAcodeAirport: VARCHAR(3) [PK],
    airport: VARCHAR(100),
    city: VARCHAR(100),
    state: VARCHAR(100)
);

userRating (

userID: INT [PK] [FK for Users.userID]
IATACodeAirline: VARCHAR(3) [PK] [FK for Airlines.IATACodeAirline]
userRating: INT
);

Conclusion:

The functional dependencies in this case can be identified by the primary keys, which can uniquely identify the remaining attributes in the table. For example, in the Airports schema, the primary key is IATAcodeAirport, which can uniquely identify the remaining attributes in the table, which are airport, city, and state. IATAcodeAirport and airport can be a superkey, but only IATAcodeAirport is the key since not every superkey is a key. One example of a functional dependency is IATAcodeAirport -> airport, city, state, where the left hand side is a superkey, so there is no need to normalize the table. Another functional dependency is tailNumber -> company, aircraftModel. We also have IATAcodeAirline -> airline, userRating for the Airlines schema. For the Flights table, we have flightNumber -> airline, year, month, day, dayOfTheWeek, originAirport, destinationAirport. Lastly, in the Users table, we have userID -> username, password, phoneNumber. All of the functional dependencies do not need to be normalized into 3NF because every key implies the remaining attributes. Still, we analyzed 3NF because it improves the organization of our data and removes duplicate values, thus improving data accuracy and quality. This form of normalization also helps maintain data integrity by reducing the risk of errors when inserting, updating, and deleting data.