CSDS341 Project - Airline Querying System - Initial Report

Quynh Nguyen, Jiamu Zhang, Luke Zhang

June 17, 2022

1 Introduction

In recent decades, the growing demand for leisure and business travel leads to the prosperity of the airline market. An increasing number of people have been choosing to take flights to travel domestically or internationally. Therefore, an organized and comprehensive database that stores the airline system is critical for both travelers and crew to obtain plenty and simultaneous information.

Although there do exist several flight databases or applications for commercial airlines, it is rare to find comprehensive information - including weather at the departure airport and destination, aircraft type, the total flight hour of pilots, and the number of luggage allowed - in just one database. This information offers travelers a chance to be better prepared for traveling.

Since our airline querying system contains a relatively extensive data set, the crew members who choose to use our database are able to access the basic information about the travelers who will be on their flight and provides updates about the airline information.

2 Entity-Relationship Model

2.1 Assumptions

Before performing the high-level design of the airline querying system, our group lists the following assumptions that need to be considered in our database systems.

1. Assume that there have and only have two types of users of the airline querying system: travelers and crew.
2. Assume that plane ticket information is stored in the database system and each ticket is only valid for one traveler. However, a traveler may own zero or more plane tickets. This matches the real situation in which travelers need to transfer their flights.
3. Assume that each ticket contains a specific seat location for exactly one flight. However, a flight may have multiple tickets being sold to travelers since a flight has obviously more than one seat.
4. Assume that a crew member can be either an air attendant or a pilot. Therefore, a crew member can serve zero or more flights. Additionally, a flight must be served by at least one crew. There does have a slight chance that a small propeller airplane only needs one crew member (i.e. the pilot).
5. Assume that a flight is operated by exactly one areoplane. For example, figure 1 shows that the aircraft with registration number B-6075 is operating a specific flight (flight number: CA862) from Beijing(PEK) to Geneva(GVA). However, it is likely that one aeroplane can fly multiple flights. It is worth noticing that the registration number is unique for each aeroplane.
6. Assume that an aeroplane can only belong to one company. This database system does not consider private aeroplanes that do not belong to any company. For instance, the aeroplane B-6075 belongs to Air China. Additionally, a airline company can have multiple planes.

7. Assume that each airline company must have at least one airport as its hub, a place where the headquarter of the company locates and where the aeroplanes get maintenance and repaired. However, some large airports can provide services for multiple airline companies. For example, Los Angeles International Airport (LAX) is a hub for both United and Delta Airlines, and Delta Airlines has another hub : Detroit Metropolitan Airport (DTW).

8. Assume that each flight can have multiple schedules, and a schedule can be mapped to multiple flights. It is common for most domestic flights to have the same flight flying the same route on two successive days. There is also a slight chance that two flights have the exact same schedule.

9. Assume that each flight only departures from exactly one airport and only arrives at exactly one airport. However, an airport can have many flights. If a plane can arrive at two destinations simultaneously, then the pilot will win the next Nobel Physics Price.

Figure 1: Aircraft with Registration Number B-6075 (©Pascal Simon)

2.2 ER Diagram

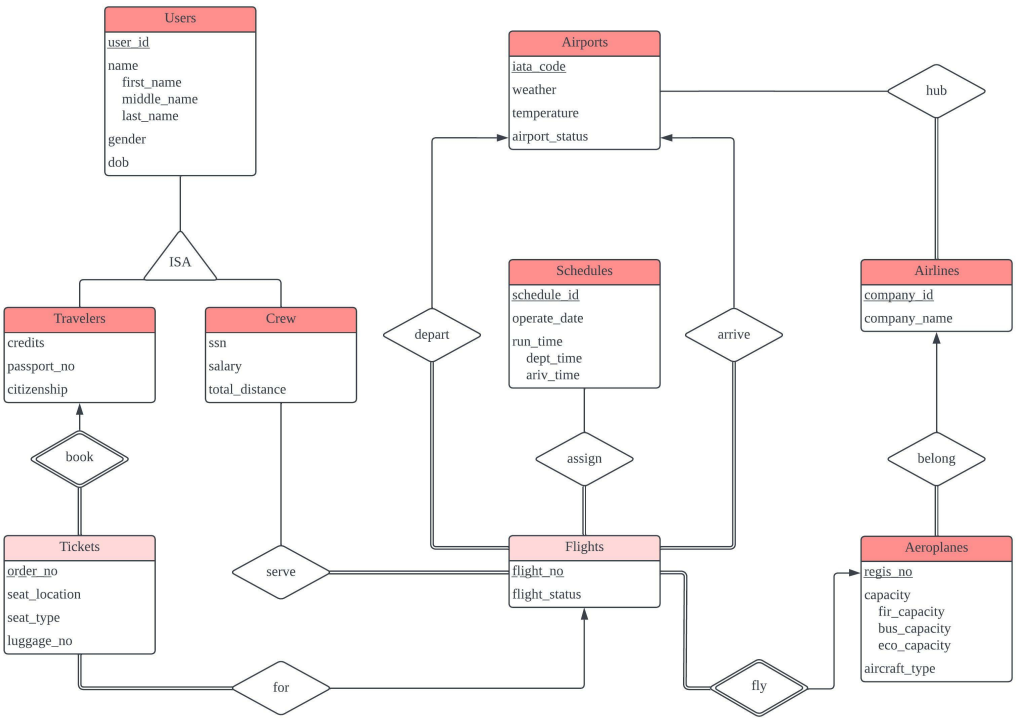

Figure 2: ER Diagram for Airline Querying System

3 Schemas

3.1 Strong Entities

```
Travelers(user_id: int,
          first_name: CHAR(50),
          middle_name: CHAR(50),
          last_name: CHAR(50),
          gender: CHAR(1),
          dob: date,
          credits: int,
          passport_no: CHAR(20),
          citizenship: CHAR(30)
          )
```

The entity `Travelers` is a type of `Users` in this Airline Querying Systems. The `user_id` attribute is the primary key of this entity.

```
Crew(user_id: int,
     first_name: CHAR(50),
     middle_name: CHAR(50),
     last_name: CHAR(50),
     gender: CHAR(1),
     dob: date,
     ssn: int,
     salary: double,
     total_distance: int
     )
```

The entity `Crew` is the other type of `Users` in this Airline Querying Systems. The `user_id` attribute is the primary key of this entity.

```
Airports(iata_code: CHAR(3),
         weather: CHAR: CHAR(15),
         temperature: int,
         airport_status: CHAR(10)
         )
```

The entity `Airports` stores the information of airports, with a primary key `iata_code`. IATA Code stands for International Air Transport Association Code. It is a three-character code that is unique for each airport. For example, the IATA Code for Los Angeles International Airport is LAX.

```
Airlines(company_id: int,
         company_name: CHAR(30)
         )
```

The entity `Airports` stores the information of airports, with a primary key `iata_code`. IATA Code stands for International Air Transport Association Code. It is a three-character code that is unique for each airport. For example, the IATA Code for Los Angeles International Airport is LAX.

```
Aeroplanes(regis_no: CHAR(10),
           fir_capacity: int,
           bus_capacity: int,
           eco_capacity: int,
           aircraft_type: CHAR(10)
           )
```

The entity `Aeroplanes` stores the information of each aircraft, with a primary key `regis_no` (registration number). Before each plane starts operating, it will be assigned a unique registration number.

```
Schedules(schedule_id: int,
          operate_date: date,
          dept_time: time,
          ariv_time: time
          )
```

The entity `Schedules` stores the flight schedules. The primary key is `schedule_id`.

3.2 Weak Entities

```
Tickets(traveler_id: int,
        order_no: int,
        seat_location: CHAR(3),
        seat_type: CHAR(1),
        luggage_no: int
        )
        Foreign Key (traveler_id) references (Travelers.user_id)
```

The weak entity `Tickets` stores the ticket information of each traveler. The primary key of the weak entity is `traveler_id` and `order_no`.

```
Flights(regis_no: int,
        flight_no: CHAR(7),
        flight_status: CHAR(10)
        )
        Foreign Key (regis_no) references (Aeroplanes.regis_no)
```

The weak entity `Flights` stores the information of each flight operated by each aeroplane. The primary key of the weak entity is `regis_no` and `flight_no`.

3.3 Relationships

```
for(traveler_id: int,
    order_no: int,
    regis_no: int,
    flight_no: CHAR(7)
    )
    Foreign Key (traveler_id) references (Travelers.user_id)
    Foreign Key (order_no) references (Tickets.order_no)
    Foreign Key (regis_no) references (Aeroplanes.regis_no)
    Foreign Key (flight_no) references (Flights.flight_no)
```

The `for` relationship stores the information regrading tickets for flights. Since each ticket is for exactly one flight and a flight can have many tickets, this is a many-to-one relationship. Therefore, the primary key from the many side (i.e. `Tickets.order_no` and `Travelers.user_id`) should be set as the primary key of this relationship.

```
serve(crew_id: int,
      regis_no: int,
      flight_no: CHAR(7)
      )
      Foreign Key (crew_id) references (Crew.user_id)
```

```
        Foreign Key (regis_no) references (Aeroplanes.regis_no)
        Foreign Key (flight_no) references (Flights.flight_no)
```

The `serve` relationship stores the information regarding how crew members serve for flights. Since each crew member can serve multiple flights and a flight may need multiple crews, this is a many-to-many relationship. Therefore, the primary key from both sides should be set as the primary key of this relationship.

```
depart(regis_no: CHAR(10),
       flight_no: CHAR(7),
       dept_iata_code: CHAR(3)
       )
       Foreign Key (regis_no) references (Aeroplanes.regis_no)
       Foreign Key (flight_no) references (Flights.flight_no)
       Foreign Key (dept_iata_code) references (Airports.iata_code)
```

The `depart` relationship stores the information regarding departure information. Since each flight can depart from only one airport while an airport can have many departing planes, this is a one-to-many relationship. Therefore, the primary key from many sides (i.e. `Aeroplanes.regis_no` and `Flights.flight_no`) should be set as the primary key of this relationship.

```
arrive(regis_no: CHAR(10),
       flight_no: CHAR(7),
       ariv_iata_code: CHAR(3),
       )
       Foreign Key (regis_no) references (Aeroplanes.regis_no)
       Foreign Key (flight_no) references (Flights.flight_no)
       Foreign Key (ariv_iata_code) references (Airports.iata_code)
```

The `arrive` relationship stores the information regarding arrival information. Since each flight can arrive only one airport while an airport can have many departing planes, this is a one-to-many relationship. Therefore, the primary key from many sides (i.e. `Aeroplanes.regis_no` and `Flights.flight_no`) should be set as the primary key of this relationship.

```
assign(regis_no: CHAR(10),
       flight_no: CHAR(7),
       schedule_id: int
       )
       Foreign Key (regis_no) references (Aeroplanes.regis_no)
       Foreign Key (flight_no) references (Flights.flight_no)
       Foreign Key (schedule_id) references (Schedules.schedule_id)
```

The `assign` relationship stores the information regarding the mapping between flights and schedules. Since this is a many-to-many relationship, the primary key from both sides should be set as the primary key of this relationship.

```
hub(company_id: int,
    iata_code: CHAR(3)
    )
    Foreign Key (company_id) references (Airlines.company_id)
    Foreign Key (iata_code) references (Airports.iata_code)
```

The `hub` relationship stores the information regarding airline companies and their hub airports. Since this is a many-to-many relationship, the primary key from both sides should be set as the primary key of this relationship.

```
belong(regis_no: CHAR(10),
       company_id: int
       )
       Foreign Key (regis_no) references (Aeroplanes.regis_no)
       Foreign Key (company_id) references (Airlines.company_id)
```

The `belong` relationship stores the information regarding airline companies and planes they owned. Since this is a one-to-many relationship, the primary key from the many side (i.e. `Aeroplanes.regis_no`) should be set as the primary key of this relationship.

3.4 Identifying Relationships

All the identifying relationships in the schema are merged with the weak entity. There is no need to create separate tables because that may cause redundancies.