CSDS341 Project - Airline Querying System - Initial Report

Quynh Nguyen (qtn2), Jiamu Zhang (jxz1217), Luke Zhang (rxz330)

June 24, 2022

1 Introduction

In recent decades, the growing demand for leisure and business travel has led 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 of 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 hours 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 sets, 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 member. It 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 aeroplane. 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. Additionally, each schedule should have unique `schedule_id`.

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.

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

2.2 ER Diagram

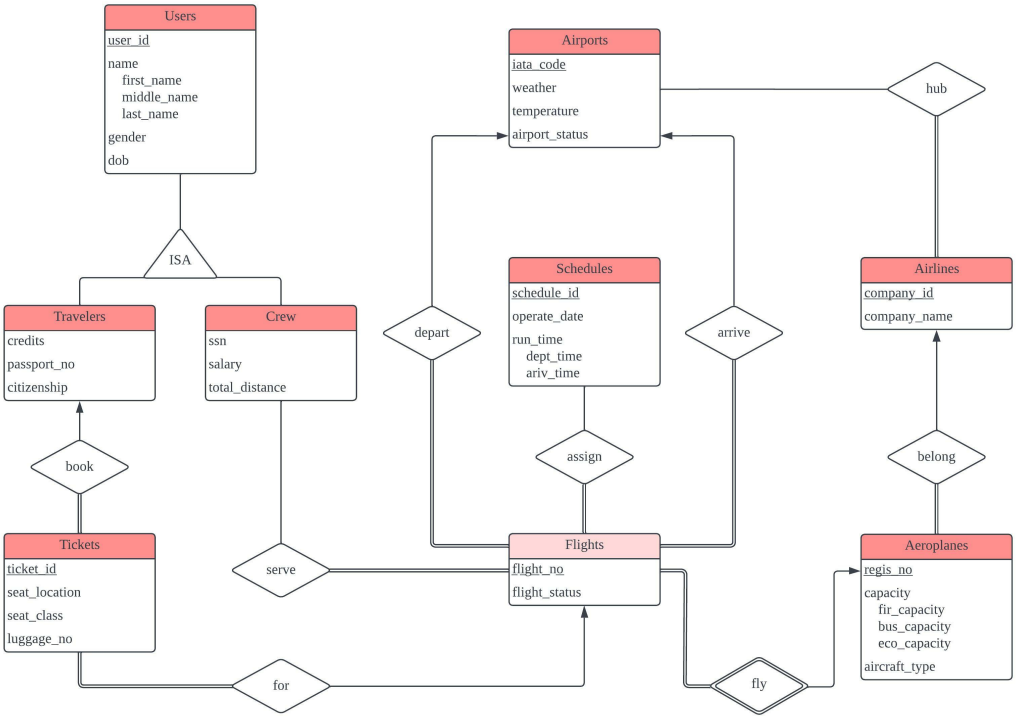

Figure 2: ER Diagram for Airline Querying System

3 Schemas

3.1 Strong Entities

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

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

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.

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

The table `Tickets_book_for` stores the ticket information of each traveler. The primary key of the entity is `ticket_id`. Since both the `book` and `for` relationships are one-to-many relationships, they are merged with the entity on many side which is `Tickets`. Therefore, there are three different foreign keys in the table. `traveler_id` comes from the one side of the `book` relationship, while `regis_no` and `flight_no` comes from the one side of the `for` relationship.

```
Tickets_book_for(ticket_id: int,
                 seat_location: char(3),
                 seat_class: char(1),
                 luggage_no: int,
                 traveler_id: int,
                 regis_no: int,
                 flight_no: varchar(7)
                 )
                 Foreign Key (traveler_id) references (Travelers.user_id)
                 Foreign Key (regis_no) references (Aeroplanes_belong.regis_no)
                 Foreign Key (flight_no) references (Flights_ariv_dept.flight_no)
```

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.

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

```
          )
```

The entity `Airlines` stores the information of airline companies, with a primary key `company_id`.

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

The entity `Aeroplanes_belong` stores the information of each aeroplane and the airline company they belong to, with a primary key `regis_no` (registration number). Before each plane starts operating, it will be assigned a unique registration number.

```
Aeroplanes_belong(regis_no: varchar(10),
                  fir_capacity: int,
                  bus_capacity: int,
                  eco_capacity: int,
                  aircraft_type: varchar(10),
                  company_id: int
                  )
                  Foreign Key (company_id) references (Airlines.company_id)
```

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

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

3.2 Weak Entities

The weak entity `Flights_ariv_dept` stores the information of each flight operated by each aeroplane and the `depart` & `arrive` information. The primary key of the weak entity is `regis_no` and `flight_no`. It contains foreign keys obtains from both `Aeroplanes` and `Airports` Entities. This table merges the weak entity `Flights`, the identifying relationship `fly` and two many to one relationships (`depart` and `arrive`).

```
Flights_ariv_dept(regis_no: int,
                  flight_no: varchar(7),
                  flight_status: varchar(10),
                  dept_iata_code: char(3),
                  ariv_iata_code: char(3)
                  )
                  Foreign Key (regis_no) references (Aeroplanes_belong.regis_no)
                  Foreign Key (dept_iata_code) references (Airports.iata_code)
                  Foreign Key (ariv_iata_code) references (Airports.iata_code)
```

3.3 Relationships

The `book` relationship is merged with the entity `Tickets` in the table `Tickets_book_for`.

The `for` relationship is merged with the entity `Tickets` in the table `Tickets_book_for`.

The `dept` relationship is merged with the entity `Flights` in the table `Flights_ariv_dept`.

The `ariv` relationship is merged with the entity `Flights` in the table `Flights_ariv_dept`.

The `belong` relationship is merged with the entity `Aeroplanes` in the table `Aeroplanes_belong`.

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 keys from both sides should be set as the primary keys of this relationship.

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

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

```
assign(regis_no: varchar(10),
       flight_no: varchar(7),
       schedule_id: int
       )
       Foreign Key (regis_no) references (Aeroplanes_belong.regis_no)
       Foreign Key (flight_no) references (Flights_ariv_dept.flight_no)
       Foreign Key (schedule_id) references (Schedules.schedule_id)
```

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

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.

4 Example Queries

Find the passport number of all Chinese Traveler who has already booked at least one Plane Ticket.

```
SELECT    passport_no
FROM      Travelers
WHERE     (Citizenship = "Chinese")
          AND
          (user_id IN (SELECT    traveler_id, count(ticket_id)
                       FROM      Tickets_book_for
                       GROUP BY  traveler_id
                       HAVING    count(ticket_id) >= 1));
```

Find the flight number of all flights that belongs to *Emirates* and has over 30-seat capacity in first class.

```
SELECT    flight_no
FROM      Flights_ariv_dept NATURAL JOIN Aeroplanes_belongs
WHERE     (Aeroplanes_belongs.fir_capacity > 30)
          AND
          (regis_no IN (SELECT    regis_no
                        FROM      Aeroplanes_belongs NATURAL JOIN Airlines
                        WHERE     Airlines.company_name = "Emirates"))
```

Find the flight number of all flights that departs from Los Angeles International Airport on both 2021-06-22 and 2021-06-23.

```
SELECT flight_no
FROM   Flights_ariv_dept
WHERE  (dept_iata_code = "LAX")
       AND
       (flight_no IN (SELECT flight_no
                      FROM   Flights_ariv_dept NATURAL JOIN assign NATURAL JOIN Schedules AS s22
                      WHERE  s22.operate_date = "2021-06-22"
                             AND
                             (exists(SELECT flight_no
                                     FROM   Flights_ariv_dept NATURAL JOIN assign
                                            NATURAL JOIN Schedules AS s23
                                     WHERE  s23.operate_date = "2021-06-23"
                                            AND
                                            s22.flight_no = s23.flight_no))))
```

5 Data Sources

Since there exists various sources about flight numbers and their departure or arrival information, we are able to use real data for most flights. However, it is hard to keep track on the real-time temperature or status of the airport. For this part of data, our group decide to use make-up data. For travelers and crew information, since the data are usually protected by the airline companies, we also decide to use make up data.