AeroManageX

Database Management System

MSCS 542L-256

Aerotech Titans

Marist College

School of Computer Science and Mathematics

Submitted To:

Dr. Reza Sadeghi

September 13, 2023

Table of Contents

Project Report of AeroManageX

1. ## Team Name
2. ## Aerotech Titans

Team Members
1. Bashir Dahir   bashir.dahir1@marist.edu   (Team Head)
2. Nihar Lodaya   nihar.lodaya1@marist.edu   (Team Member)
3. Marguerite McGahay   marguerite.mcgahay@marist.edu   (Team Member)

Description of Team Members

1. Bashir Dahir

I'm a Computer Science student at Marist College, Beacon, New York, in my fifth year, with just two semesters to go before graduation. My academic journey has sharpened my skills in programming and data structures, but my true passion lies in database management systems. Currently, I'm eagerly gearing up for a project focused on airline management systems, where I plan to apply my expertise in database management to create efficient and robust solutions for the aviation industry. Beyond academics, I enjoy problem-solving and community engagement.

2. Nihar Lodaya

Hey there, I'm Nihar Lodaya, and I come from India. I've got a bachelor's degree in computer science from back home, and right now, I'm in the middle of my master's program in Computer Science at Marist College. I'm stoked about working with this bunch of awesome folks on our current project. What really got me excited about my teammates Bashir & Marguerite is how dedicated they are to making this project a success.

3. Marguerite McGahay

I grew up in Poughkeepsie and then attended the University of Delaware, where I graduated in 2021 with a BS in Mathematics and a Minor in Computer Science. While I was there, I was a TA for multiple computer science classes and was also on the Women's Rowing Team! I currently work at Marist as the Assistant Women's Rowing Coach. This is my first semester in the Computer Science – Software Development program, so I don't know many people, but I was excited when Bashir and Nihar extended an invitation for me to be a member of this group. We selected our team head, Bashir, since it was his initial idea to pursue this project, but I truly believe each member of this team has the capability to be able to step into that role if asked of us.

3. AeroManageX Objective

The primary objective of the AeroManageX project is to elevate your airline's existing database infrastructure to be able to better compete with the world's top airlines. Our company will help you condense the mass amount of information needed to be able to have thousands of planes arrive and depart each day. Our system operates to help your airline in multiple ways, including but not limited to managing flight bookings, optimizing the cost of flights, making sure planes are in the necessary airports based on different flights' place of departure and arrival, and efficiently assigning pilots to flights that make sense for your company and their schedules.

4. Review Related Works

There are many competitions for our company including Ramco Aviation, Airline Suite, and AvPro. Each of these companies has positive and negative aspects. To begin with, Ramco Aviation published on their website that they provide the best aviation software for the following three reasons: "Proven technology champions, usability focusses and in memory planning and optimization" (2). However, Ramco Aviation critics emphasize on their "Security and data control issues and difficulties of data migration" (*2*). Second, Airline Suite promotes that their system is easy to use, scalable and affordable. On the other hand, however, Airline Suite is notorious for having "no history of previous or editions and no import options for Microsoft Excel or file" (1). Finally, AvPro is famous for their "reliability reporting, budget forecasting, and inventory management" (4). On the contrary, AvPro lacks the availability of "instruction manuals and system stability" as there are system glitches (3).

5. The Merits of Our Project

Our company will provide airlines with more features than any other competitive company. We provide airlines with the ability to see and organize information which can be categorized into three principal operations:

Our Land Operations:

- Checked baggage: Each passenger has a bag that must be on their flight.
- Staff management: Such as gate-workers, baggage handlers, custodial staff, etc.
- Times of arrival and departure of flights
- Fleet inventory

Air Operations:

- Flight plan: Including non-stop flights and layovers.
- Assigning pilots and airline staff.

Billing Operations:

- Bookings: Secure database for credit card numbers.
- Cost of seats
- Checked luggage (including oversized): If a passenger checks a bag that is over 50lbs, they must pay an extra fee on top of the base cost.

An airline should choose AeroManageX to manage their data because we are security focused and can benefit any size airline, from companies with just a small fleet of planes to global "Aero-Titans"!

6. GitHub Repository Address

Here is our GitHub repository's URL.

https://github.com/bashirad/MSCS-542L-256_AeroManageX_Aerotech-Titans.git

7. External ER Diagrams

Manager Description

In the External Entity Relationship Diagram for the Manager Model in our AeroManageX, the primary entities include Flight, Origin Airport, Destination Airport, Pilot, Flight Attendant, Plane, and Airline. Flight is the central entity, while Origin Airport and Destination Airport are directly connected to Flight, representing the departure and arrival locations. Pilots are associated with specific flights as they operate them, and Flight

Attendants work on board these flights. The Plane entity represents the aircraft used for the flights, and Airline serves as a parent entity encompassing various flights operated by the airline. This diagram illustrates how these entities are interrelated, providing a clear overview of the airline's operations.

Manager Model
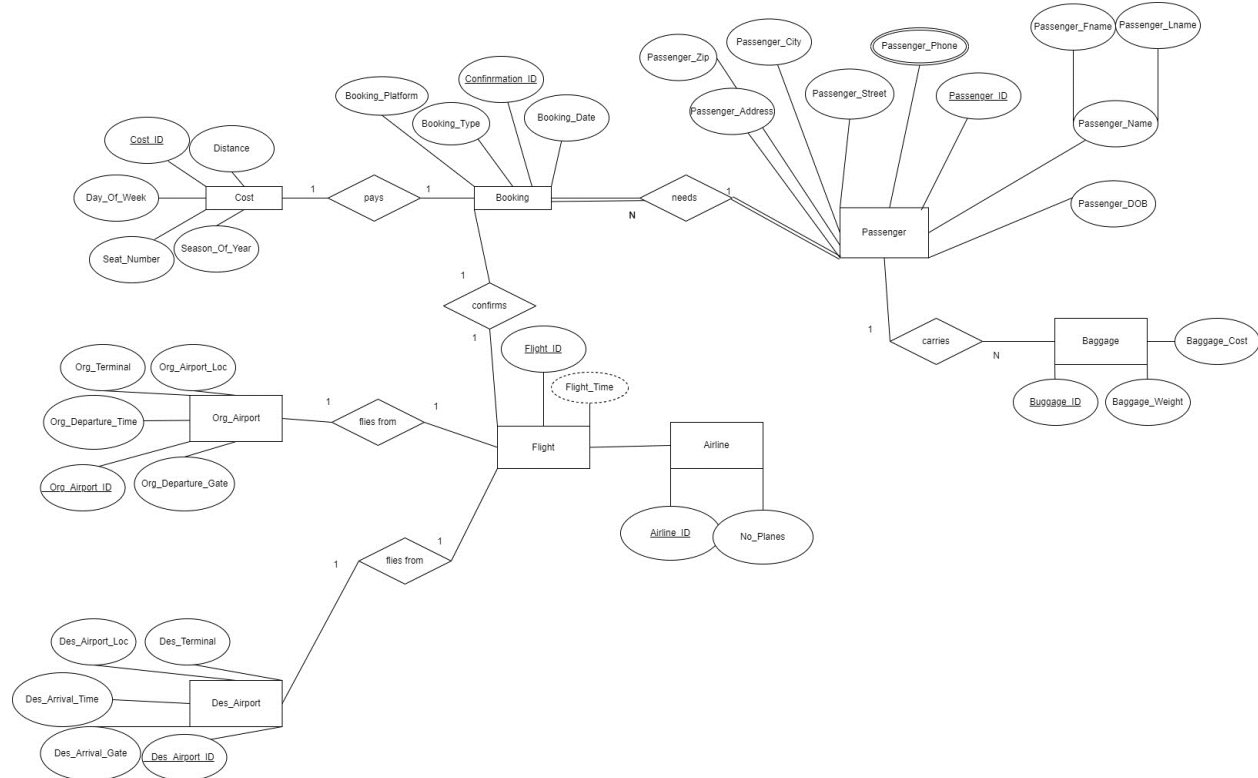
Figure 1

Booking Agent Description

In the External Entity Relationship Diagram for the Booking Agent Model in the AeroManageX, a web of interconnected entities and relationships emerges. Cost is linked to Booking, reflecting the financial transaction between booking agents and reservations. Booking necessitates Passenger, showing the connection between bookings and the passengers they are made for, while Passenger also carries Baggage, signifying the belongings associated with passengers. Bookings confirm Flights, illustrating the reservation process, and Flights are tied to both Origin and Destination Airports, representing the departure and arrival points of the journeys.

Booking Agent Model
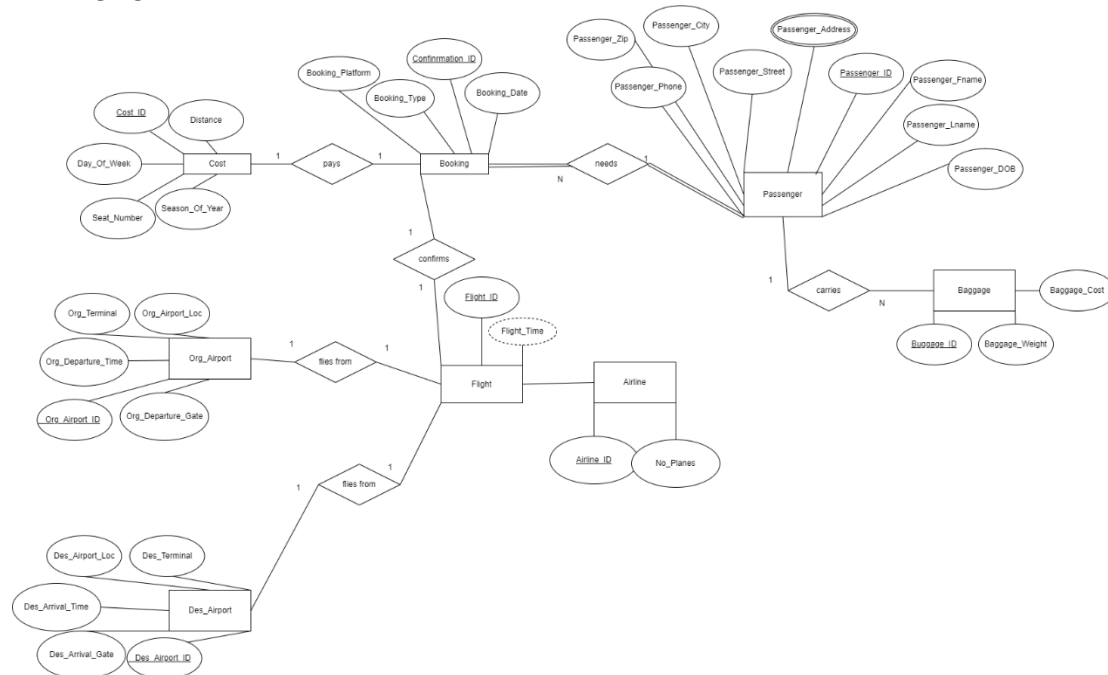
Figure 2

8. Entity Relationship Diagram (ER Diagram)

Description

      Creating this project involves a thoughtful selection of entities, attributes, relationships, participations, and cardinalities to represent a simplified domain. In this case, the chosen entities include "Cost," "Baggage," "Booking," "Plane," "Flight," "Flight_Attendant," "Org_Airport," "Des_Airport," "Pilot," "Passenger," and "Airline." Each of these entities represents key elements in the domain of Airline travel. Attributes are then identified for each entity to capture relevant information; for instance, "Passenger" have attributes like Passenger_Fname, Passenger_Lname, Passenger_Address, etc. While "Cost" has attributes like Confirmation_ID, Distance, Season_Of_Year, etc. Relationships are established between entities to represent how they are connected, such as the relationship between "Booking" and "Passenger" to show that a passenger can make a booking.

Diagram
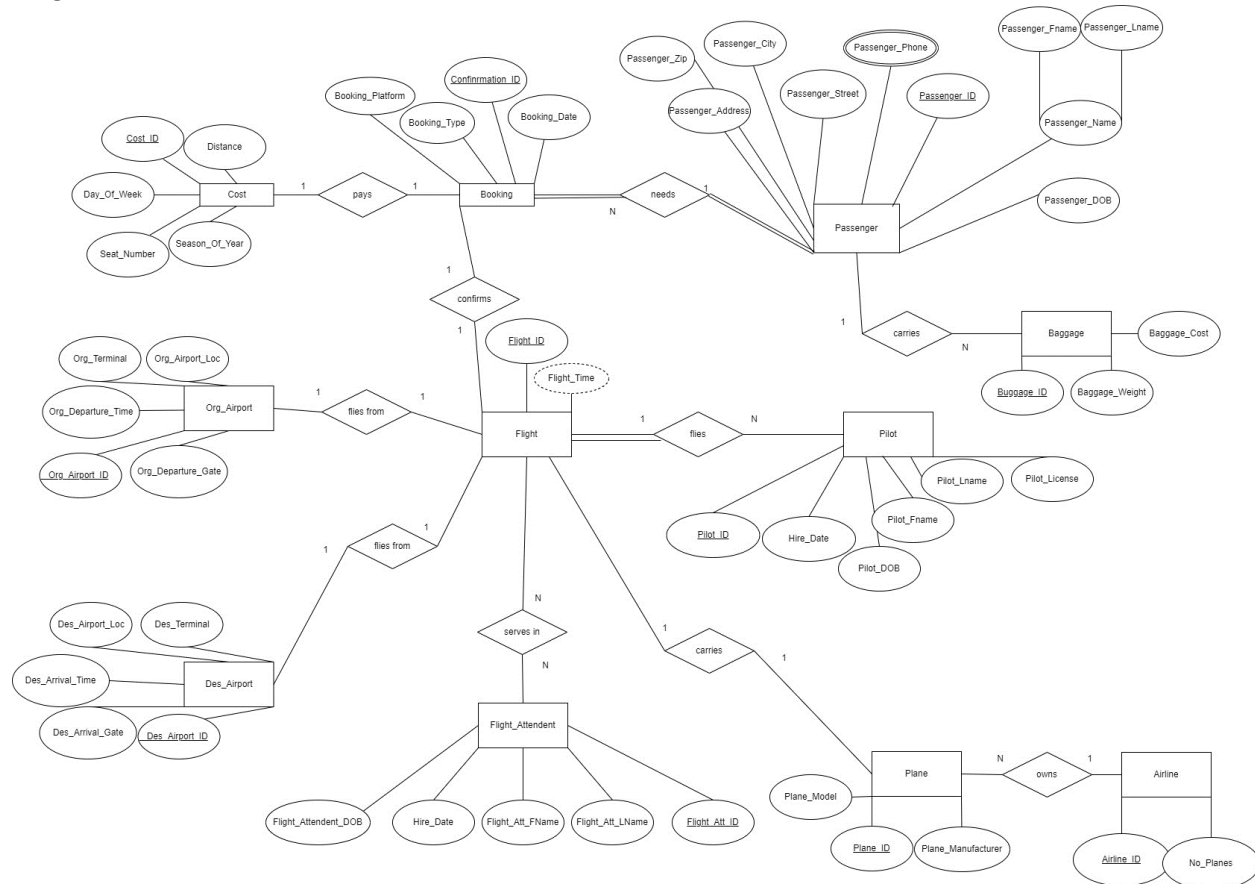
Figure 3

9. Enhanced Entity Relationship Diagram (EER)

Description

#	Entity	Key	Relationships
1	Flight	Flight_ID	Each flight has one origin airport, one destination airport, multiple bookings for their passengers, as well as a one plane, one pilot, and multiple flight attendants
2	Plane	Plane_ID	Each flight must have a plane, which owned by an airline
3	Airline	Airline_ID	Each airline has multiple planes in its fleet
4	Cost	Cost_ID	Cost is related to Booking via a 1-1 relationship.
5	Booking	Confirmation_ID	Each booking is related to a passenger, which results in a cost of

			booking. When a passenger books, they are then connected to a flight
6	Passenger	Passenger_ID	Each passenger can have one or more bags
7	Baggage	Baggage_ID	Each bag is related to one passenger
8	Org_Airport	Org_Airport_ID	Each flight must depart from one origin airport
9	Des_Airport	Des_Airport_ID	Each flight must arrive at one destination airport.
10	Pilot	Pilot_ID	Each flight must have one pilot. A pilot can have multiple flights in one day
11	Flight Attendant	Flight_Att_ID	Each flight can have multiple flight attendants and a flight attendant can have multiple flights in one day

Figure 4

Diagram

Cost
- Cost_ID INT
- Confirmation_ID INT
- Distance VARCHAR(45)
- Season_Of_Year VARCHAR(45)
- Day_Of_Week VARCHAR(45)
- Seat_Number VARCHAR(45)
- Booking_Confirmation_ID INT
- Booking_Flight_Flight_ID INT
- Booking_Cost_Cost_ID INT
- Booking_Passenger_Passenger_ID INT
- Booking_Flight_Flight_ID1 INT
  5 more...
- Indexes

Booking
- Confirmation_ID INT
- Passenger_ID INT
- Booking_Date DATETIME
- Booking_Platform VARCHAR(45)
- Booking_Type VARCHAR(45)
- Flight_Flight_ID INT
- Cost_Cost_ID INT
- Passenger_Passenger_ID INT
- Flight_Flight_ID1 INT
- Flight_Org_Airport_Org_Airport_ID1 INT
- Flight_Des_Airport_Des_Airport_ID1 INT
- Flight_Flight Attendent_Flight_Att_ID1 INT
- Flight_Pilot_Pilot_ID1 INT
- Flight_Plane_Plane_ID1 INT
- Indexes

Passenger
- Passenger_ID INT
- Passenger_FName VARCHAR(45)
- Passenger_LName VARCHAR(45)
- Passenger_DOB DATETIME
- Passenger_Address VARCHAR(45)
- Passenger_Street VARCHAR(45)
- Passenger_City VARCHAR(45)
- Passenger_Zip INT
- Passenger_Phone VARCHAR(45)
- Indexes

Phone
- Phone_ID INT
- Phone_Num VARCHAR(45)
- Passenger_Passenger_ID INT
- Indexes

Baggage
- Baggage_ID VARCHAR(45)
- Passenger_ID INT
- Baggage_Weight DECIMAL(2)
- Baggage_Cost DECIMAL(2)
- Org_Airpot_Loc DATETIME
- Org_Departure_Time VARCHAR(45)
- Flight_ID INT
- Des_Airport_Loc VARCHAR(45)
- Des_Depature_Time DATETIME
- Des_Depature_Gate VARCHAR(45)
- Passenger_ID INT
- Passenger_Passenger_ID INT
- Passenger_Passenger_ID1 INT
- Indexes

Org_Airport
- Org_Airport_ID INT
- Org_Airport_Loc VARCHAR(45)
- Org_Depature_Time DATETIME
- Org_Depature_Gate VARCHAR(45)
- Org_Terminal VARCHAR(45)
- Indexes

Flight
- Flight_ID INT
- Confirmation_ID INT
- Org_Airport_ID INT
- Des_Airport_ID INT
- Plane_ID INT
- Passenger_ID INT
- Pilot_ID INT
- Org_Depature_Time DATETIME
- Des_Arrival_Time DATETIME
- Flight_Time DECIMAL(2)
- Org_Airport_Org_Airport_ID1 INT
- Des_Airport_Des_Airport_ID1 INT
- Flight Attendent_Flight_Att_ID1 INT
- Pilot_Pilot_ID1 INT
- Plane_Plane_ID1 INT
- Indexes

Pilot
- Pilot_ID INT
- Pilot_FName VARCHAR(45)
- Pilot_LName VARCHAR(45)
- Pilot_License VARCHAR(45)
- Hire_Date DATETIME
- Pilot_DOB DATETIME
- Indexes

Des_Airport
- Des_Airport_ID INT
- Des_Airport_Loc VARCHAR(45)
- Des_Arrival_Time DATETIME
- Des_Arrival_Gate VARCHAR(45)
- Des_Terminal VARCHAR(45)
- Indexes

Plane
- Plane_ID INT
- Plane_Model VARCHAR(45)
- Plane_Manufacturer VARCHAR(45)
- Flight_Flight_ID INT
- Flight_Booking_Confirmation_ID INT
- Flight_Booking_Flight_Flight_ID INT
- Flight_Org_Airport_Org_Airport_ID INT
- Flight_Des_Airport_Des_Airport_ID INT
- Indexes

Airline
- Airline_ID INT
- No_Planes INT
- Plane_ID INT
- Plane_Plane_ID1 INT
- Airport_Gate VARCHAR(45)
- Indexes

Flight Attendent
- Flight_Att_ID INT
- Flight_Att_FName VARCHAR(45)
- Flight_Att_LName VARCHAR(45)
- Hire_Date DATETIME
- Flight_Attendent_DOB DATETIME
- Indexes

Flight Attendent_has_Flight
- Flight Attendent_Flight_Att_ID INT
- Flight_Flight_ID INT
- Flight_Org_Airport_Org_Airport_ID1 INT
- Flight_Des_Airport_Des_Airport_ID1 INT
- Flight_Flight Attendent_Flight_Att_ID1 INT
- Flight_Pilot_Pilot_ID1 INT
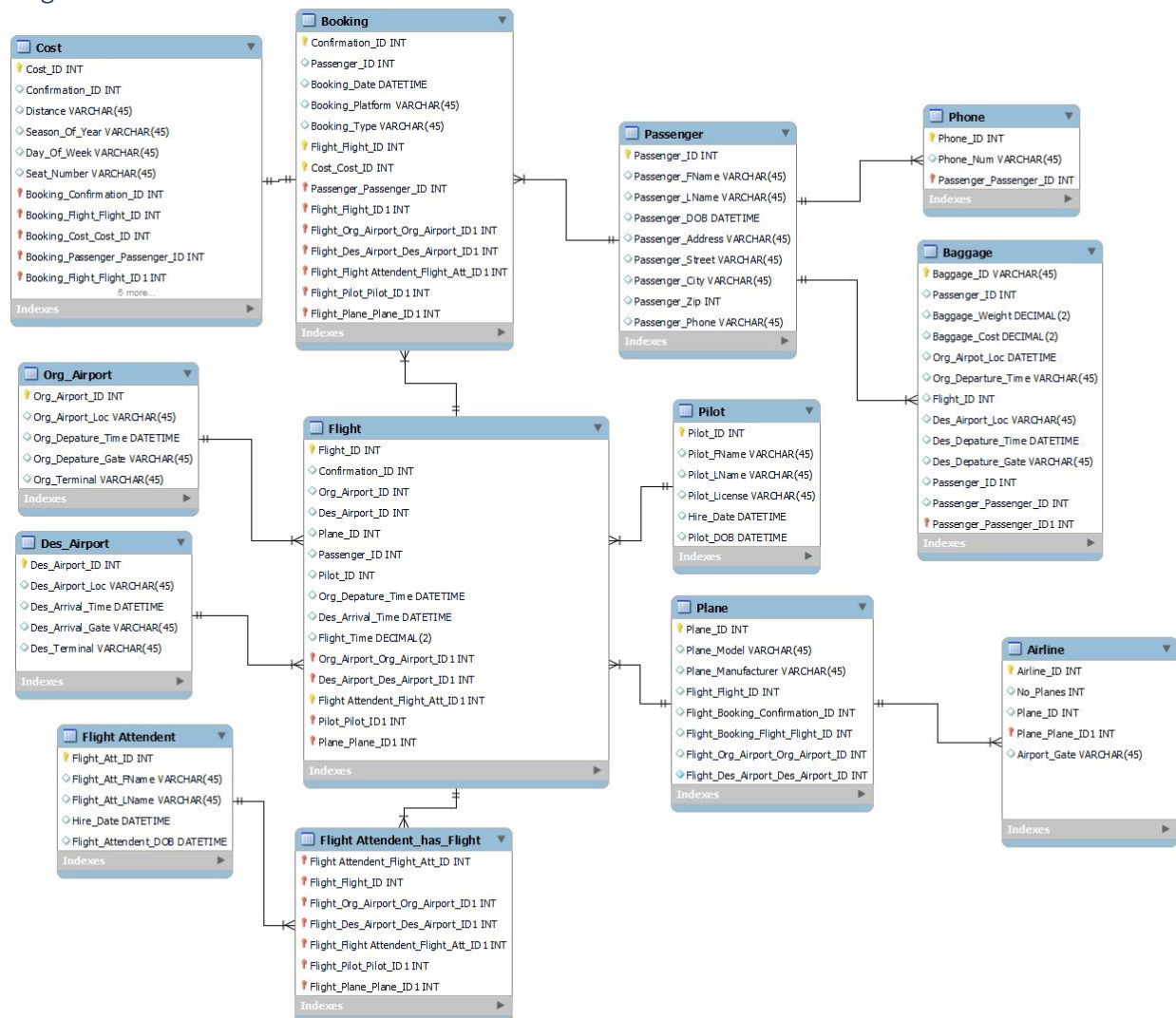- Flight_Plane_Plane_ID1 INT
- Indexes

Figure 5

10. Database Development

Create database and tables.

```
DROP DATABASE IF EXISTS mydb;

CREATE DATABASE IF NOT EXISTS mydb DEFAULT CHARACTER SET utf8;

USE mydb;


-- GET RID OF ALL FOREIGN KEYS EXCEPT BOOKING'S CONFIRMATION_ID

-- Table mydb.Cost

DROP TABLE IF EXISTS mydb.Cost;

CREATE TABLE IF NOT EXISTS mydb.Cost (

  Cost_ID INT NOT NULL,

  Amount INT NOT NULL,

  Distance VARCHAR(45) NULL,

  Season_Of_Year VARCHAR(45) NULL,

  Day_Of_Week VARCHAR(45) NULL,

  Seat_Number VARCHAR(45) NULL,

  PRIMARY KEY (Cost_ID)

);
```

Cost

Attribute	Data Type	Description
Cost_ID	INT	This attribute serves as the primary key for the "Cost" table, uniquely identifying each cost record. It is an integer (INT) data type.
Amount	INT	The "Amount" attribute represents the cost or price associated with a specific item or service. It is of integer (INT) data type and is required, meaning it cannot be null.

Distance	VARCHAR	This attribute stores textual information, likely related to the distance associated with the cost. It is of VARCHAR(45) data type, allowing up to 45 characters, and it is not marked as required (nullable).
Season_Of_Year	VARCHAR	The "Season_Of_Year" attribute stores information related to the season of the year associated with the cost. It is of VARCHAR(45) data type, allowing up to 45 characters, and it is not marked as required (nullable).
Day_Of_Week	VARCHAR	This attribute represents information related to the day of the week associated with the cost. It is of VARCHAR(45) data type, allowing up to 45 characters, and it is not marked as required (nullable).
Seat_Number	VARCHAR	The "Seat_Number" attribute stores information related to seat numbers associated with the cost. It is of VARCHAR(45) data type, allowing up to 45 characters, and it is not marked as required (nullable).

```
-- Table mydb.Airline
DROP TABLE IF EXISTS mydb.Airline;
CREATE TABLE IF NOT EXISTS mydb.Airline (
  Airline_ID INT NOT NULL,
  No_Planes INT NULL,
  Plane_ID INT NULL,
  Airport_Gate VARCHAR(45) NULL,
  Num_Employees INT,
```

```
    PRIMARY KEY (Airline_ID),

);
```

Airline

Attribute	Data Type	Description
Airline_ID	INT	This attribute serves as the primary key for the "Airline" table, uniquely identifying each airline record. It is an integer (INT) data type and is required, meaning it cannot be null.
No_Planes	INT	The "No_Planes" attribute represents the number of planes associated with the airline. It is of integer (INT) data type and is not marked as required (nullable).
Plane_ID	INT	This attribute is used to store the ID of a specific plane associated with the airline. It is of integer (INT) data type and is not marked as required (nullable).
Airport_Gate	VARCHAR	The "Airport_Gate" attribute stores textual information related to the airport gate used by the airline. It is of VARCHAR(45) data type, allowing up to 45 characters, and is not marked as required (nullable).
Num_Employees	INT	This attribute represents the number of employees working for the airline. It is of integer (INT) data type and is required, meaning it cannot be null.

```
    -- Table mydb.Passenger

DROP TABLE IF EXISTS mydb.Passenger;
```

```
CREATE TABLE IF NOT EXISTS mydb.Passenger (

  Passenger_ID INT NOT NULL,

  Passenger_FName VARCHAR(45) NULL,

  Passenger_LName VARCHAR(45) NULL,

  Passenger_DOB DATETIME NULL,

  Passenger_Address VARCHAR(45) NULL,

  Passenger_Street VARCHAR(45) NULL,

  Passenger_City VARCHAR(45) NULL,

  Passenger_Zip INT NULL,

  Passenger_Phone VARCHAR(45) NULL,

  PRIMARY KEY (Passenger_ID)

);
```

Passenger

Attribute	Data Type	Description
Passenger_ID	INT	This attribute serves as the primary key for the "Passenger" table, uniquely identifying each passenger record. It is an integer (INT) data type and is required, meaning it cannot be null.
Passenger_FName	VARCHAR	The "Passenger_FName" attribute represents the first name of the passenger. It is of VARCHAR(45) data type, allowing up to 45 characters, and is not marked as required (nullable).
Passenger_LName	VARCHAR	This attribute is used to store the last name of the passenger. It is of VARCHAR(45) data type, allowing up to 45 characters, and is not marked as required (nullable).
Passenger_DOB	DATETIME	The "Passenger_DOB" attribute is used to store the date of birth of the passenger. It is of

		DATETIME data type and is not marked as required (nullable).
Passenger_Address	VARCHAR	This attribute stores the primary address of the passenger. It is of VARCHAR(45) data type, allowing up to 45 characters, and is not marked as required (nullable).
Passenger_Street	VARCHAR	The "Passenger_Street" attribute represents the street information associated with the passenger's address. It is of VARCHAR(45) data type, allowing up to 45 characters, and is not marked as required (nullable).
Passenger_City	VARCHAR	This attribute is used to store the city information associated with the passenger's address. It is of VARCHAR(45) data type, allowing up to 45 characters, and is not marked as required (nullable).
Passenger_Zip	INT	The "Passenger_Zip" attribute is used to store the zip code associated with the passenger's address. It is of integer (INT) data type and is not marked as required (nullable).
Passenger_Phone	VARCHAR	This attribute represents the phone number of the passenger. It is of VARCHAR(45) data type, allowing up to 45 characters, and is not marked as required (nullable).

```
-- Table mydb.Pilot
DROP TABLE IF EXISTS mydb.Pilot;
```

```
CREATE TABLE IF NOT EXISTS mydb.Pilot (

  Pilot_ID INT NOT NULL,

  Pilot_FName VARCHAR(45) NULL,

  Pilot_LName VARCHAR(45) NULL,

  Pilot_License VARCHAR(45) NULL,

  Hire_Date DATETIME NULL,

  Pilot_DOB DATETIME NULL,

  PRIMARY KEY (Pilot_ID)

);
```

Pilot

Attribute	Data Type	Description
Pilot_ID	INT	This attribute serves as the primary key for the "Pilot" table, uniquely identifying each pilot record. It is an integer (INT) data type and is required, meaning it cannot be null.
Pilot_FName	VARCHAR	The "Pilot_FName" attribute represents the first name of the pilot. It is of VARCHAR(45) data type, allowing up to 45 characters, and is not marked as required (nullable).
Pilot_LName	VARCHAR	This attribute is used to store the last name of the pilot. It is of VARCHAR(45) data type, allowing up to 45 characters, and is not marked as required (nullable).
Pilot_License	VARCHAR	The "Pilot_License" attribute represents the license information associated with the pilot. It is of VARCHAR(45) data type, allowing up to 45 characters, and is not marked as required (nullable).

Hire_Date	DATETIME	This attribute is used to store the hire date of the pilot. It is of DATETIME data type and is not marked as required (nullable).	
Pilot_DOB	DATETIME	The "Pilot_DOB" attribute is used to store the date of birth of the pilot. It is of DATETIME data type and is not marked as required (nullable).	

```sql
-- Table mydb.Des_Airport
DROP TABLE IF EXISTS mydb.Des_Airport;
CREATE TABLE IF NOT EXISTS mydb.Des_Airport (
  Des_Airport_ID INT NOT NULL,
  Des_Airport_Loc VARCHAR(45) NULL,
  Des_Arrival_Time DATETIME NULL,
  Des_Arrival_Gate VARCHAR(45) NULL,
  Des_Terminal VARCHAR(45) NULL,
  PRIMARY KEY (Des_Airport_ID)
);
```

Des_Airport

Attribute	Data Type	Description
Des_Airport_ID	INT	This attribute serves as the primary key for the "Des_Airport" table, uniquely identifying each destination airport record. It is an integer (INT) data type and is required, meaning it cannot be null.
Des_Airport_Loc	VARCHAR	The "Des_Airport_Loc" attribute represents the location of the destination airport. It is of VARCHAR(45) data type, allowing up to 45 characters,

		and is not marked as required (nullable).
Des_Arrival_Time	DATETIME	This attribute is used to store the arrival time at the destination airport. It is of DATETIME data type and is not marked as required (nullable).
Des_Arrival_Gate	VARCHAR	The "Des_Arrival_Gate" attribute represents the arrival gate at the destination airport. It is of VARCHAR(45) data type, allowing up to 45 characters, and is not marked as required (nullable).
Des_Terminal	VARCHAR	This attribute is used to store the terminal information at the destination airport. It is of VARCHAR(45) data type, allowing up to 45 characters, and is not marked as required (nullable).

```
-- Table mydb.Org_Airport
DROP TABLE IF EXISTS mydb.Org_Airport;
CREATE TABLE IF NOT EXISTS mydb.Org_Airport (
  Org_Airport_ID INT NOT NULL,
  Org_Airport_Loc VARCHAR(45) NULL,
  Org_Departure_Time DATETIME NULL,
  Org_Departure_Gate VARCHAR(45) NULL,
  Org_Terminal VARCHAR(45) NULL,
  PRIMARY KEY (Org_Airport_ID)
);
```

Org_Airport

Attribute	Data Type	Description

Org_Airport_ID	INT	This attribute serves as the primary key for the "Org_Airport" table, uniquely identifying each originating airport record. It is an integer (INT) data type and is required, meaning it cannot be null.
Org_Airport_Loc	VARCHAR	The "Org_Airport_Loc" attribute represents the location of the originating airport. It is of VARCHAR(45) data type, allowing up to 45 characters, and is not marked as required (nullable).
Org_Departure_Time	DATETIME	This attribute is used to store the departure time from the originating airport. It is of DATETIME data type and is not marked as required (nullable).
Org_Departure_Gate	VARCHAR	The "Org_Departure_Gate" attribute represents the departure gate at the originating airport. It is of VARCHAR(45) data type, allowing up to 45 characters, and is not marked as required (nullable).
Org_Terminal	VARCHAR	This attribute is used to store the terminal information at the originating airport. It is of VARCHAR(45) data type, allowing up to 45 characters, and is not marked as required (nullable).

```
-- Table mydb.Flight Attendent

DROP TABLE IF EXISTS mydb.Flight_Attendent;

CREATE TABLE IF NOT EXISTS mydb.Flight_Attendent (

  Flight_Att_ID INT NOT NULL,

  Flight_Att_FName VARCHAR(45) NULL,
```

```
        Flight_Att_LName VARCHAR(45) NULL,

        Hire_Date DATETIME NULL,

        Flight_Attendent_DOB DATETIME NULL,

        PRIMARY KEY (Flight_Att_ID)

    );
```

Flight_Attendent

Attribute	Data Type	Description
Flight_Att_ID	INT	This attribute serves as the primary key for the "Flight_Attendent" table, uniquely identifying each flight attendant's record. It is an integer (INT) data type and is required, meaning it cannot be null.
Flight_Att_FName	VARCHAR	The "Flight_Att_FName" attribute represents the first name of the flight attendant. It is of VARCHAR(45) data type, allowing up to 45 characters, and is not marked as required (nullable).
Flight_Att_LName	VARCHAR	This attribute represents the last name of the flight attendant. It is of VARCHAR(45) data type, allowing up to 45 characters, and is not marked as required (nullable).
Hire_Date	DATETIME	The "Hire_Date" attribute is used to store the date when the flight attendant was hired. It is of DATETIME data type and is not marked as required (nullable).
Flight_Attendant_DOB	DATETIME	This attribute represents the date of birth of the flight attendant. It is of DATETIME

		data type and is not marked as required (nullable).

```
-- Table mydb.Plane
DROP TABLE IF EXISTS mydb.Plane;
CREATE TABLE IF NOT EXISTS mydb.Plane (
  Plane_ID INT NOT NULL,
  Plane_Model VARCHAR(45) NULL,
  Plane_Manufacturer VARCHAR(45) NULL,
  Plane_Size VARCHAR(20),
  Plane_Engine_Power VARCHAR(20),
  PRIMARY KEY (Plane_ID),
);
```

Plane

Attribute	Data Type	Description
Plane_ID	INT	This attribute serves as the primary key for the "Plane" table, uniquely identifying each plane's record. It is an integer (INT) data type and is required, meaning it cannot be null.
Plane_Model	VARCHAR	The "Plane_Model" attribute represents the model of the plane. It is of VARCHAR(45) data type, allowing up to 45 characters, and is not marked as required (nullable).
Plane_Manufacturer	VARCHAR	This attribute represents the manufacturer of the plane. It is of VARCHAR(45) data type, allowing up to 45 characters, and is not marked as required (nullable).
Plane_Size	VARCHAR	The "Plane_Size" attribute describes the size or category of

		the plane. It is of VARCHAR(20) data type and is not marked as required (nullable).
Plane_Engine_Power	VARCHAR	This attribute represents the engine power of the plane. It is of VARCHAR(20) data type and is not marked as required (nullable).

```
-- Table mydb.Booking

DROP TABLE IF EXISTS mydb.Booking;

CREATE TABLE IF NOT EXISTS mydb.Booking (

  Confirmation_ID INT NOT NULL,

  Booking_Date DATETIME NULL,

  Booking_Platform VARCHAR(45) NULL,

  Booking_Type VARCHAR(45) NULL,

  PRIMARY KEY (Confirmation_ID),

  FOREIGN KEY (Passenger_ID) REFERENCES Passenger (Passenger_ID),

  );
```

Booking

Attribute	Data Type	Description
Confirmation_ID	INT	This attribute serves as the primary key for the "Booking" table, uniquely identifying each booking's record. It is an integer (INT) data type and is required, meaning it cannot be null.
Booking_Date	DATETIME	The "Booking_Date" attribute represents the date and time when the booking was made. It is of DATETIME data type and is not marked as required (nullable).

Booking_Platform	VARCHAR	This attribute describes the platform or source through which the booking was made. It is of VARCHAR(45) data type, allowing up to 45 characters, and is not marked as required (nullable).
Booking_Type	VARCHAR	The "Booking_Type" attribute specifies the type of booking made, such as "economy," "business," etc. It is of VARCHAR(45) data type and is not marked as required (nullable).
Passenger_ID	INT	This foreign key is used to establish a relationship with the "Passenger" table, specifically referencing the "Passenger_ID" attribute in the "Booking" table. It represents the passenger associated with the booking.

```sql
-- Table mydb.Baggage

DROP TABLE IF EXISTS mydb.Baggage;

CREATE TABLE IF NOT EXISTS mydb.Baggage (

  Baggage_ID VARCHAR(45) NOT NULL,

  Passenger_ID INT NULL,

  Baggage_Weight DECIMAL(2) NULL,

  Baggage_Cost DECIMAL(2) NULL,

  Org_Airport_Loc DATETIME NULL,

  Org_Departure_Time VARCHAR(45) NULL,

  Flight_ID INT NULL,

  Des_Airport_Loc VARCHAR(45) NULL,

  Des_Departure_Time DATETIME NULL,

  Des_Departure_Gate VARCHAR(45) NULL,
```

```
        PRIMARY KEY (Baggage_ID),

        FOREIGN KEY (Passenger_ID) REFERENCES Passenger (Passenger_ID)

    );
```

Baggage

Attribute	Data Type	Description
Baggage_ID	VARCHAR	This attribute serves as the primary key for the "Baggage" table, uniquely identifying each piece of baggage. It is of VARCHAR(45) data type, allowing up to 45 characters, and it is required, meaning it cannot be null.
Passenger_ID	INT	The "Passenger_ID" attribute represents the passenger associated with the baggage. It is an integer (INT) data type and is not marked as required (nullable). This attribute is linked to the "Passenger" table through a foreign key relationship.
Baggage_Weight	DECIMAL	This attribute indicates the weight of the baggage, represented as a decimal value with two decimal places. It is not marked as required (nullable).
Baggage_Cost	DECIMAL	The "Baggage_Cost" attribute specifies the cost associated with the baggage. It is represented as a decimal value with two decimal places and is not marked as required (nullable).
Org_Airport_Loc	DATETIME	This attribute represents the location of the originating airport. It is of DATETIME data type and is not marked as required (nullable). The data type choice may be inappropriate for this attribute, and it might need to be adjusted

		based on the actual data it represents.
Org_Departure_Time	VARCHAR	The "Org_Departure_Time" attribute denotes the departure time of the flight from the originating airport. It is of VARCHAR(45) data type, allowing up to 45 characters, and is not marked as required (nullable).
Flight_ID	INT	This attribute represents the ID of the flight associated with the baggage. It is an integer (INT) data type and is not marked as required (nullable).
Des_Airport_Loc	VARCHAR	The "Des_Airport_Loc" attribute specifies the location of the destination airport. It is of VARCHAR(45) data type, allowing up to 45 characters, and is not marked as required (nullable).
Des_Departure_Time	DATETIME	This attribute represents the departure time of the flight from the destination airport. It is of DATETIME data type and is not marked as required (nullable).
Des_Departure_Gate	VARCHAR	The "Des_Departure_Gate" attribute indicates the departure gate at the destination airport. It is of VARCHAR(45) data type, allowing up to 45 characters, and is not marked as required (nullable).

```
-- Table mydb.Flight

DROP TABLE IF EXISTS mydb.Flight;

CREATE TABLE IF NOT EXISTS mydb.Flight (
```

```
    Flight_ID INT NOT NULL,

    Org_Departure_Time DATETIME NULL,

    Des_Arrival_Time DATETIME NULL,

    Flight_Time DECIMAL(2) NULL,

    PRIMARY KEY (Flight_ID),

    FOREIGN KEY (Passenger_ID) REFERENCES Passenger(Passenger_ID),

    FOREIGN        KEY        (Confirmation_ID)        REFERENCES
Booking(Confirmation_ID),

    FOREIGN    KEY    (Org_Airport_ID)    REFERENCES    Org_Airport
(Org_Airport_ID),

    FOREIGN    KEY    (Des_Airport_ID)    REFERENCES    Des_Airport
(Des_Airport_ID),

    FOREIGN KEY (Plane_ID) REFERENCES Plane (Plane_ID),

    FOREIGN KEY (Pilot_ID) REFERENCES Pilot (Pilot_ID),

    FOREIGN    KEY    (Flight_Att_ID)    REFERENCES    Flight_Attendent
(Flight_Att_ID)

    );
```

Flight

Attribute	Data Type	Description
Flight_ID	INT	This attribute serves as the primary key for the "Flight" table, uniquely identifying each flight. It is of INT data type, and it is required, meaning it cannot be null.
Org_Departure_Time	DATETIME	The "Org_Departure_Time" attribute represents the departure time of the flight from the originating airport. It is of DATETIME data type and is not marked as required (nullable).
Des_Arrival_Time	DATETIME	This attribute denotes the arrival time of the flight at the destination airport. It is of DATETIME data type and is not marked as required (nullable).

Flight_Time	DECIMAL	The "Flight_Time" attribute specifies the duration of the flight in decimal format with two decimal places. It is not marked as required (nullable).
Passenger_ID	INT	This foreign key establishes a relationship with the "Passenger" table, specifically referencing the "Passenger_ID" attribute in the "Flight" table. It associates flights with the corresponding passenger.
Confirmation_ID	INT	This foreign key relates to the "Confirmation_ID" attribute in the "Booking" table, allowing flights to be associated with specific bookings.
Org_Airport_ID	INT	This foreign key links to the "Org_Airport_ID" attribute in the "Org_Airport" table, indicating the originating airport of the flight.
Des_Airport_ID	INT	This foreign key references the "Des_Airport_ID" attribute in the "Des_Airport" table, specifying the destination airport of the flight.
Plane_ID	INT	This foreign key is associated with the "Plane_ID" attribute in the "Plane" table, allowing flights to be linked to specific planes.
Pilot_ID	INT	This foreign key relates to the "Pilot_ID" attribute in the "Pilot" table, indicating the pilot responsible for the flight.
Flight_Att_ID	INT	This foreign key establishes a relationship with the "Flight_Attendent" table, specifically referencing the "Flight_Att_ID" attribute in the "Flight" table. It associates flight attendants with specific flights.

```
-- Table mydb.Flight_Attendent_has_Flight

DROP TABLE IF EXISTS mydb.Flight_Attendent_has_Flight;

CREATE TABLE IF NOT EXISTS mydb.Flight_Attendent_has_Flight (

  PRIMARY    KEY   (Flight_Att_ID,    Flight_ID,    Org_Airport_ID,
Des_Airport_ID, Pilot_ID, Plane_ID),

  FOREIGN    KEY   (Flight_Att_ID)   REFERENCES   Flight_Attendent
(Flight_Att_ID),

  FOREIGN KEY (Flight_ID) REFERENCES Flight (Flight_ID),

  FOREIGN    KEY    (Org_Airport_ID)    REFERENCES    Org_Airport
(Org_Airport_ID),

  FOREIGN    KEY    (Des_Airport_ID)    REFERENCES    Des_Airport
(Des_Airport_ID),

  FOREIGN KEY (Pilot_ID) REFERENCES Pilot (Pilot_ID),

  FOREIGN KEY (Plane_ID) REFERENCES Plane (Plane_ID)

);
```

Flight_Attendent_has_Flight

Attribute	Data Type	Description
Flight_Att_ID	INT	This attribute is part of the primary key and establishes a relationship with the "Flight_Attendent" table. It uniquely identifies the flight attendant involved in the flight.
Flight_ID	INT	This attribute is part of the primary key and references the "Flight" table. It uniquely identifies the flight in which the flight attendant is assigned.
Org_Airport_ID	INT	This foreign key is associated with the "Org_Airport" table, indicating the originating airport of the flight.
Des_Airport_ID	INT	This foreign key references the "Des_Airport" table, specifying the destination airport of the flight.

Pilot_ID	INT	This foreign key establishes a relationship with the "Pilot" table, indicating the pilot responsible for the flight.
Plane_ID	INT	This foreign key is associated with the "Plane" table, allowing the association of flight attendants with specific planes used for flights.

11. Loading data and performance enhancements

Handling foreign key constraints

We used the following steps:

```
SET FOREIGN_KEY_CHECKS=0;

-- insert into Cost

INSERT INTO mydb.Cost (Cost_ID, Amount, Distance, Season_Of_Year,
Day_Of_Week, Seat_Number)

VALUES

        (1, 50, 100, 'Summer', 'Monday', 'A1'),

        (2, 60, 150, 'Spring', 'Wednesday', 'B3'),

        (3, 45, 180, 'Fall', 'Friday', 'C7'),

        (4, 55, 75, 'Winter', 'Tuesday', 'D2'),

        (5, 70, 750, 'Summer', 'Thursday', 'E5'),

        (6, 65, 1000, 'Spring', 'Monday', 'F9'),

        (7, 40, 325, 'Fall', 'Sunday', 'G4'),

        (8, 75, 650, 'Winter', 'Saturday', 'H6'),

        (9, 58, 3400, 'Summer', 'Tuesday', 'I8'),

        (10, 68, 140, 'Spring', 'Thursday', 'J10');

SET FOREIGN_KEY_CHECKS=1;
```

We set the foreign key checks to zero temporarily, so we could perform the insert operation without MySQL raising errors about the foreign key violations. Then we set it back to one to minimize the chance of data inconsistency in the future.

Importing data

Here is sample of data insertion:

```
-- Booking
```

```sql
INSERT INTO mydb.Booking (Confirmation_ID, Booking_Date,
Booking_Platform, Booking_Type, Passenger_ID)
      VALUES
      (1, '2023-10-31 10:00:00', 'Website', 'One-way', 101),
      (2, '2023-11-01 14:30:00', 'Mobile App', 'Round-trip', 102),
      (3, '2023-11-02 12:45:00', 'Website', 'One-way', 103),
      (4, '2023-11-03 08:15:00', 'Mobile App', 'Round-trip', 104),
      (5, '2023-11-04 16:20:00', 'Website', 'One-way', 105),
      (6, '2023-11-05 11:30:00', 'Mobile App', 'Round-trip', 106),
      (7, '2023-11-06 09:10:00', 'Website', 'One-way', 107),
      (8, '2023-11-07 15:55:00', 'Mobile App', 'Round-trip', 108),
      (9, '2023-11-08 13:25:00', 'Website', 'One-way', 109),
      (10, '2023-11-09 17:40:00', 'Mobile App', 'Round-trip', 110);
```

Insertion optimization

Query Statistics	Query Statistics
Timing (as measured at client side): Execution time: 0:00:0.00000000	Timing (as measured at client side): Execution time: 0:00:0.00000000
Timing (as measured by the server): Execution time: 0:00:0.00032850 Table lock wait time: 0:00:0.00000200	Timing (as measured by the server): Execution time: 0:00:0.00028310 Table lock wait time: 0:00:0.00000300
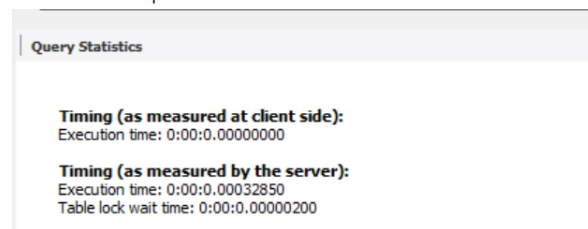
Figure 6. Insertion Optimization   Figure 7.
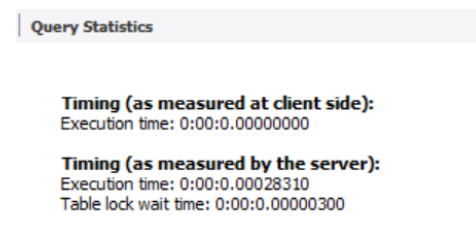
Normalization Check

All tables are in 1NF, 2NF and 3NF

12. Application Development

Graphical User Interface Design

Here's a brief outline of the capabilities provided in each page and the connections between them:

Start:

This is the initial page, likely serving as the entry point for the application. It doesn't have specific functionality but serves as a starting point for users.

Login Page:

Users can input their username and password to access the application. It serves as an authentication barrier for the application, ensuring only authorized users can proceed.

Main Menu:

After successful login, users are directed to the main menu. It likely provides options for various actions, including adding or removing flights, making bookings, and more.

Add Flight:

This page allows authorized users to input information for adding a new flight to the system. This is a feature for administrators or flight operators to update flight details.

Administration Set Password:

This page might be used by administrators to set or change passwords for authorized users or other administrators. It's a security and user management feature.

Remove Flight:

Allows administrators to remove flights from the system. It's a feature for managing flight data.

Make Booking:

Provides a form or interface for users to make flight bookings. Likely used by passengers or customers.

Search Flight:

Users can search for available flights based on specific criteria, like destination, date, or flight number. It assists users in finding suitable flights.

Delete Booking:

Allows users to cancel or delete their existing flight bookings. Provides a means for passengers to manage their bookings.

Change Password:

Enables users to change their own passwords for security reasons. Part of user account management.

Display Flight Details:

Likely shows detailed information about a specific flight, such as departure and arrival times, seat availability, and pricing. Helps users get more information about a particular flight.

Authorized?

It appears to be a validation step, where the application checks if the user is authorized. If authorization is successful, the user is directed to the "Access Granted" page. Otherwise, the user might encounter further login attempts.

Enter Username & Password:

This is part of the login process where users input their credentials for authentication. If successful, they are directed to the "Authorized?" page.

Access Granted:

This page signifies that the user has been successfully authenticated and authorized. It serves as a gateway to the Main Menu or other sections of the application.

Yes / No Attempted:

These options might be presented to users when authentication fails. "Yes" could indicate a retry, while "No" might lead to exiting the application or other appropriate actions.
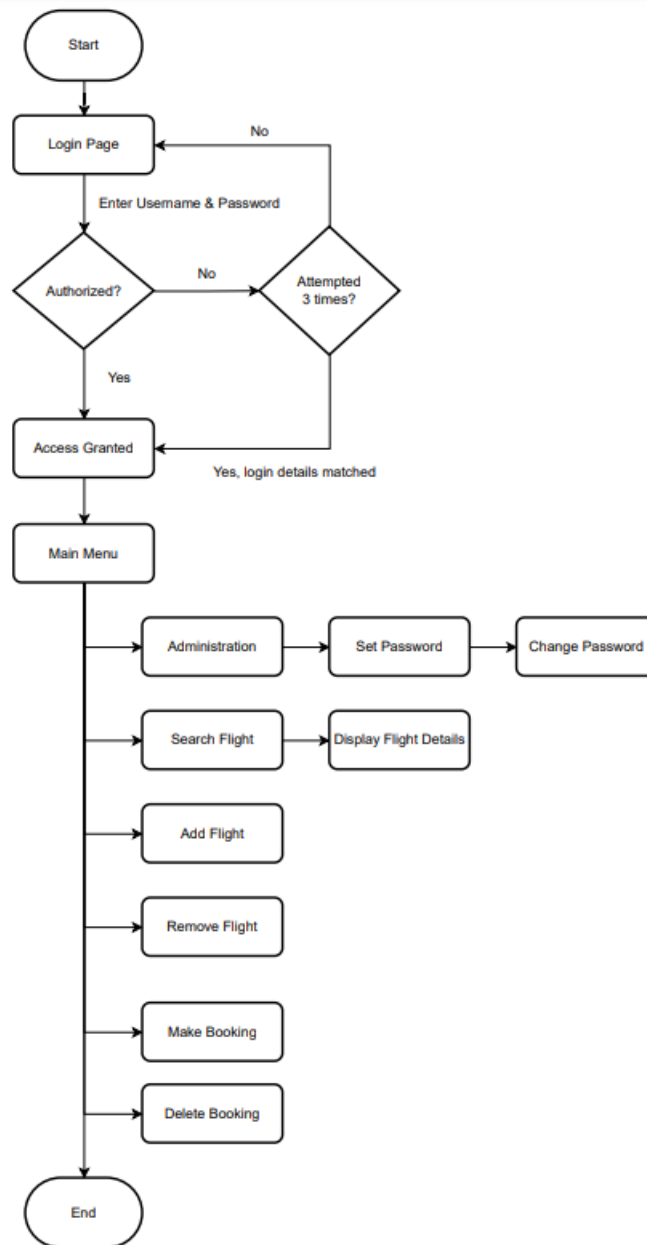
Flowchart

Figure 8.

View's implementation

Administrator View (Admin_View):

This view provides information relevant to administrators, such as passenger details and flight times for bookings. It selects data from the Booking, Passenger, and Flight tables.

```
-- Administrator view
```

```sql
CREATE VIEW Admin_View AS

SELECT

    Booking.Confirmation_ID,

    Passenger.Passenger_FName,

    Passenger.Passenger_LName,

    Flight.Org_Departure_Time,

    Flight.Des_Arrival_Time

FROM

    Booking

JOIN

    Passenger ON Booking.Passenger_ID = Passenger.Passenger_ID

JOIN

    Flight ON Booking.Confirmation_ID = Flight.Confirmation_ID;


select * from Admin_View;
```

Add Booking View (Add_Booking_View):
This view is likely used for adding bookings, providing information about the booking, passenger, flight, and associated costs. It selects data from the Booking, Passenger, Flight, and Cost tables.

```sql
-- Add Booking view

CREATE VIEW Add_Booking_View AS

SELECT

    Booking.Confirmation_ID,

    Passenger.Passenger_FName,

    Passenger.Passenger_LName,

    Flight.Org_Departure_Time AS Departure_Time,

    Flight.Des_Arrival_Time AS Arrival_Time,

    Cost.Amount AS Booking_Cost,

    Cost.Distance,

    Cost.Season_Of_Year,

    Cost.Day_Of_Week
```

```
FROM

    Booking

JOIN

    Passenger ON Booking.Passenger_ID = Passenger.Passenger_ID

JOIN

    Flight ON Booking.Confirmation_ID = Flight.Confirmation_ID

JOIN

    Cost ON Booking.Confirmation_ID = Cost.Cost_ID;
```

Delete Booking View (Delete_Booking_View):

This view is designed for deleting bookings and displays details about the booking, passenger, flight, and booking cost. It selects data from the Booking, Passenger, Flight, and Cost tables.

```
-- Delete Booking view

CREATE VIEW Delete_Booking_View AS

SELECT

    Booking.Confirmation_ID,

    Booking.Booking_Date,

    Booking.Booking_Platform,

    Booking.Booking_Type,

    Passenger.Passenger_ID,

    Passenger.Passenger_FName,

    Passenger.Passenger_LName,

    Flight.Flight_ID,

    Flight.Org_Departure_Time,

    Flight.Des_Arrival_Time,

    Cost.Amount AS Booking_Cost

FROM

    Booking

JOIN

    Passenger ON Booking.Passenger_ID = Passenger.Passenger_ID

JOIN
```

```
    Flight ON Booking.Confirmation_ID = Flight.Confirmation_ID
JOIN

    Cost ON Booking.Confirmation_ID = Cost.Cost_ID;
```

Add Flight View (Add_Flight_View):

This view offers information about flights, including details about the flight, pilot, plane, airports, and flight attendants. It selects data from the Flight table, as well as several other related tables via LEFT JOIN operations.

```
    -- Add Flight view
CREATE VIEW Add_Flight_View AS
SELECT
    Flight.Flight_ID,
    Flight.Org_Departure_Time,
    Flight.Des_Arrival_Time,
    Flight.Flight_Time,
    Pilot.Pilot_ID,
    Pilot.Pilot_FName,
    Pilot.Pilot_LName,
    Plane.Plane_ID,
    Plane.Plane_Model,
    Plane.Plane_Manufacturer,
    Org_Airport.Org_Airport_ID AS Org_Airport_ID,
    Org_Airport.Org_Airport_Loc AS Org_Airport_Location,
    Des_Airport.Des_Airport_ID AS Des_Airport_ID,
    Des_Airport.Des_Airport_Loc AS Des_Airport_Location,
    Flight_Attendent.Flight_Att_ID AS Flight_Attendant_ID,
    Flight_Attendent.Flight_Att_FName AS Flight_Attendant_FName,
    Flight_Attendent.Flight_Att_LName AS Flight_Attendant_LName
FROM
    Flight
LEFT JOIN
```

```
    Pilot ON Flight.Pilot_ID = Pilot.Pilot_ID
LEFT JOIN
    Plane ON Flight.Plane_ID = Plane.Plane_ID
LEFT JOIN
    Org_Airport ON Flight.Org_Airport_ID = Org_Airport.Org_Airport_ID
LEFT JOIN
    Des_Airport ON Flight.Des_Airport_ID = Des_Airport.Des_Airport_ID
LEFT JOIN
    Flight_Attendent ON Flight.Flight_Att_ID =
Flight_Attendent.Flight_Att_ID;
```

Remove Flight View (Remove_Flight_View):
This view is likely used to remove flights and provides information about flights, pilots, planes, and airport locations. It selects data from the Flight table, as well as other related tables via LEFT JOIN operations.

```
-- Remove Flight view
CREATE VIEW Remove_Flight_View AS
SELECT
    Flight.Flight_ID,
    Flight.Org_Departure_Time,
    Flight.Des_Arrival_Time,
    Pilot.Pilot_ID,
    Pilot.Pilot_FName,
    Pilot.Pilot_LName,
    Plane.Plane_ID,
    Plane.Plane_Model,
    Org_Airport.Org_Airport_ID AS Org_Airport_ID,
    Org_Airport.Org_Airport_Loc AS Org_Airport_Location,
    Des_Airport.Des_Airport_ID AS Des_Airport_ID,
    Des_Airport.Des_Airport_Loc AS Des_Airport_Location
FROM
    Flight
```

```
LEFT JOIN

    Pilot ON Flight.Pilot_ID = Pilot.Pilot_ID

LEFT JOIN

    Plane ON Flight.Plane_ID = Plane.Plane_ID

LEFT JOIN

    Org_Airport ON Flight.Org_Airport_ID = Org_Airport.Org_Airport_ID

LEFT JOIN

    Des_Airport ON Flight.Des_Airport_ID = Des_Airport.Des_Airport_ID;
```

Graphical User Interface Design

Connection to Database

```
= RESTART: C:\Users\Bashir\OneDrive\1. Marist Col
st Semester\3. Database Mgt Sys\Project\Phase06\I
ions\project_6.1_sql_connect.py
Database connection established successfully!
```

Figure 9. Successful Database Connection

Login Page

Description:

The Login Page is the initial interface where users provide their credentials to access the application. It consists of input fields for username and password, along with a Client Login and Admin Login button. Clients who wish to search through flights can login and find flights based of origin and departure airports. Admins who wish to add, delete, and print users, as well as add or remove flights can log in by pressing the Admin Login button. If invalid credentials are inputted, an error window is displayed.
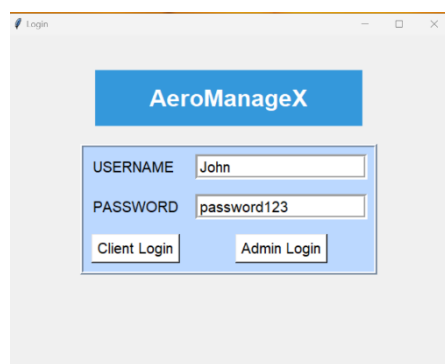
Visualization:

Figure 10. Login View

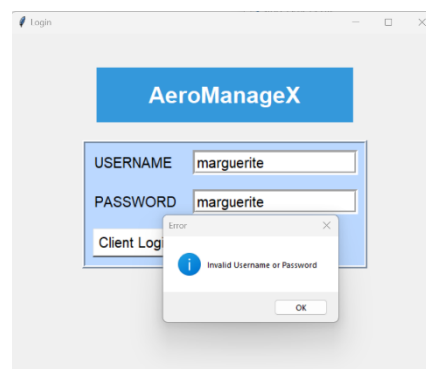Figure 11. Login Error Pop-up

Implementation of AeroManageX_main.py

```python
from tkinter import *

from tkinter import messagebox

from AeroManageX_UserPage import *

from AeroManageX_Admin import create_search_flights_gui

import mysql.connector


mypass = "Savfish4$"

mydatabase = "mydb"

con = mysql.connector.connect(

    host="localhost",

    user="root",

    password=mypass,

    database=mydatabase

)

cur = con.cursor()

root = Tk()

root.title("Login")

root.geometry("600x480")

headingFrame1 = Frame(root, bg="#3498DB", bd=5)

headingFrame1.place(relx=0.2, rely=0.1, relwidth=0.6, relheight=0.16)


headingLabel = Label(headingFrame1, text="AeroManageX", bg='#3498DB',
fg='white', font=('Arial', 24, 'bold'))

headingLabel.place(relx=0, rely=0, relwidth=1, relheight=1)



def callClient():

    usern = username_box.get()

    passw = password_box.get()
```

```python
    if usern == "" or passw == "":
        messagebox.showinfo("Error", "All fields are required")
    else:
        try:
            cur.execute("select * from user where user_name=%s and
user_password=%s", (usern, passw))
            op = cur.fetchone()
            print(op[1])
            if op is None:
                messagebox.showinfo("Error", "Invalid Username or
Password")
            else:
                root.destroy()
                create_search_flights_gui()
        except Exception as es:
            messagebox.showerror("Error", f"Error Due to: {str(es)}")


def callUser():
    usern = username_box.get()
    passw = password_box.get()

    if usern == "" or passw == "":
        messagebox.showinfo("Error", "All fields are required")
    else:
        try:
            cur.execute("select * from user where user_name=%s and
user_password=%s", (usern, passw))
            op = cur.fetchone()
            if op == None:
                messagebox.showinfo("Error", "Invalid Username or
Password")
            else:
```

```python
                root.destroy()

                mainmenu1(op[0])

        except Exception as es:

            messagebox.showerror("Error", f"Error Due to: {str(es)}")


del_frame = Frame(root, bd=4, relief=RIDGE, bg="#BBD8FF")

del_frame.place(x=100, y=150, width=400, height=180)  # Adjusted y
coordinate


username = Label(del_frame, text="USERNAME", bg="#BBD8FF", fg="black",
font=15)

username.grid(row=12, column=0, sticky=W, padx=10, pady=10)

password = Label(del_frame, text="PASSWORD", bg="#BBD8FF", fg="black",
font=15)

password.grid(row=14, column=0, sticky=W, padx=10, pady=10)


global username_box

username_box = Entry(del_frame, font=15, bd=5, relief=GROOVE)

username_box.grid(row=12, column=1, pady=10, padx=10, sticky="w")

global password_box

password_box = Entry(del_frame, font=15, bd=5, relief=GROOVE)

password_box.grid(row=14, column=1, pady=10, padx=10, sticky="w")


client_login = Button(del_frame, text="Client Login", bg="white",
fg="black", font=15, command=callClient)

client_login.grid(row=20, column=0, pady=10, padx=10)


admin_login = Button(del_frame, text="Admin Login", bg="white",
fg="black", font=15, command=callUser)

admin_login.grid(row=20, column=1, pady=10, padx=10)


root.mainloop()
```

Admin Main Menu Page

Description:

The Admin Main Menu page allows users to find a variety of options. There are options like insertion, deletion, and print users, as well as the addition or deletion of flights. Users use this page to perform specific actions on the database.
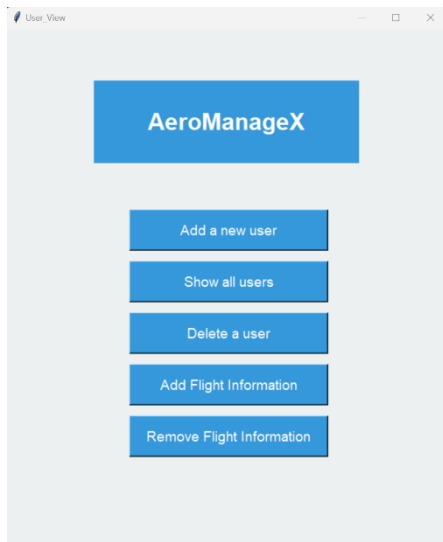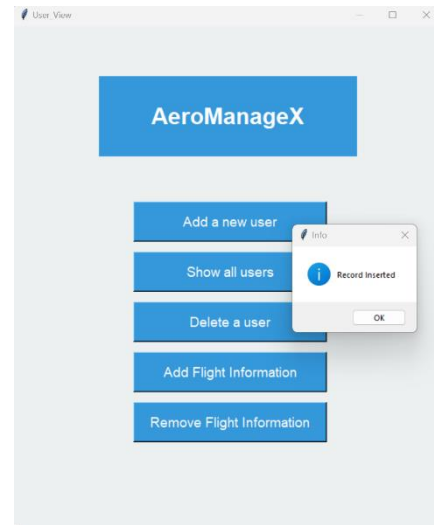
Visualization:

| Figure 12. Remove User Page | Figure 13. Add User Success Pop-up |

Implementation:

```
from tkinter import *

import tkinter as tk

from tkinter import messagebox

from AeroManageX_AddUser import *

from AeroManageX_DeleteUser import *

from AeroManageX_showAllRecord import showAll

import mysql.connector


# Add your own database name and password here to reflect in the code

mypass = "S@nde780yepuri"

mydatabase = "CMS_db"
```

```python
mypass = "Savfish4$"

mydatabase = "mydb"

con = mysql.connector.connect(

    host="localhost",

    user="root",

    password=mypass,

    database=mydatabase

)

cur = con.cursor()


def add_flight_to_database(org_departure_time, des_arrival_time,
flight_time, passenger_id, confirmation_id,

                          org_airport_id, des_airport_id, plane_id,
pilot_id, flight_attendant_id):
    try:

        # Your SQL query to insert flight data into the database

        sql = ("INSERT INTO Flight "

               "(Org_Departure_Time, Des_Arrival_Time, Flight_Time,
Passenger_ID, Confirmation_ID, "

               "Org_Airport_ID, Des_Airport_ID, Plane_ID, Pilot_ID,
Flight_Att_ID) "

               "VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s)")


        flight_data = (org_departure_time, des_arrival_time,
flight_time, passenger_id, confirmation_id,

                       org_airport_id, des_airport_id, plane_id,
pilot_id, flight_attendant_id)


        cur.execute(sql, flight_data)

        con.commit()


        messagebox.showinfo("Success", "Flight added successfully.")
```

```python
        except mysql.connector.Error as err:

            messagebox.showerror("Error", f"Error: {err}")


def remove_flight_from_database(flight_id):

    try:

        # Your SQL query to delete flight data from the database

        sql = "DELETE FROM Flight WHERE Flight_ID = %s"

        cur.execute(sql, (flight_id,))

        con.commit()


        messagebox.showinfo("Success", "Flight removed successfully.")

    except mysql.connector.Error as err:

        messagebox.showerror("Error", f"Error: {err}")


def add_flight_gui(userId):

    add_flight_window = Toplevel()

    add_flight_window.title("Add Flight Information")

    add_flight_window.geometry("400x400")


    # Labels

    tk.Label(add_flight_window, text="Org Departure
Time:").grid(row=0, column=0, padx=10, pady=5)

    tk.Label(add_flight_window, text="Des Arrival Time:").grid(row=1,
column=0, padx=10, pady=5)

    tk.Label(add_flight_window, text="Flight Time:").grid(row=2,
column=0, padx=10, pady=5)

    tk.Label(add_flight_window, text="Passenger ID:").grid(row=3,
column=0, padx=10, pady=5)

    tk.Label(add_flight_window, text="Confirmation ID:").grid(row=4,
column=0, padx=10, pady=5)

    tk.Label(add_flight_window, text="Org Airport ID:").grid(row=5,
column=0, padx=10, pady=5)
```

```python
    tk.Label(add_flight_window, text="Des Airport ID:").grid(row=6,
column=0, padx=10, pady=5)

    tk.Label(add_flight_window, text="Plane ID:").grid(row=7,
column=0, padx=10, pady=5)

    tk.Label(add_flight_window, text="Pilot ID:").grid(row=8,
column=0, padx=10, pady=5)

    tk.Label(add_flight_window, text="Flight Attendant
ID:").grid(row=9, column=0, padx=10, pady=5)


    # Entry fields
    org_departure_time_entry = tk.Entry(add_flight_window)

    org_departure_time_entry.grid(row=0, column=1, padx=10, pady=5)


    des_arrival_time_entry = tk.Entry(add_flight_window)

    des_arrival_time_entry.grid(row=1, column=1, padx=10, pady=5)


    flight_time_entry = tk.Entry(add_flight_window)

    flight_time_entry.grid(row=2, column=1, padx=10, pady=5)


    passenger_id_entry = tk.Entry(add_flight_window)

    passenger_id_entry.grid(row=3, column=1, padx=10, pady=5)


    confirmation_id_entry = tk.Entry(add_flight_window)

    confirmation_id_entry.grid(row=4, column=1, padx=10, pady=5)


    org_airport_id_entry = tk.Entry(add_flight_window)

    org_airport_id_entry.grid(row=5, column=1, padx=10, pady=5)


    des_airport_id_entry = tk.Entry(add_flight_window)

    des_airport_id_entry.grid(row=6, column=1, padx=10, pady=5)
```

```python
    plane_id_entry = tk.Entry(add_flight_window)
    plane_id_entry.grid(row=7, column=1, padx=10, pady=5)


    pilot_id_entry = tk.Entry(add_flight_window)
    pilot_id_entry.grid(row=8, column=1, padx=10, pady=5)


    flight_attendant_id_entry = tk.Entry(add_flight_window)
    flight_attendant_id_entry.grid(row=9, column=1, padx=10, pady=5)


    add_flight_button = tk.Button(add_flight_window, text="Add
Flight",
                                  command=lambda:
add_flight_to_database(org_departure_time_entry.get(),

des_arrival_time_entry.get(),

flight_time_entry.get(),

passenger_id_entry.get(),

confirmation_id_entry.get(),

org_airport_id_entry.get(),

des_airport_id_entry.get(),

plane_id_entry.get(),

pilot_id_entry.get(),

flight_attendant_id_entry.get()))
    add_flight_button.grid(row=10, column=1, pady=10)
```

```python
def remove_flight_gui(userId):

    remove_flight_window = Toplevel()

    remove_flight_window.title("Remove Flight Information")

    remove_flight_window.geometry("300x200")


    tk.Label(remove_flight_window, text="Enter Flight ID:").pack()


    flight_id_entry = Entry(remove_flight_window)

    flight_id_entry.pack()


    remove_flight_button = Button(remove_flight_window, text="Remove
Flight",

                                    command=lambda:
remove_flight_from_database(flight_id_entry.get()))

    remove_flight_button.pack()



# Rest of the code remains unchanged


def mainmenu1(userId):

    root = Tk()

    root.title("User_View")

    root.minsize(width=400, height=700)

    root.geometry("600x500")


    # Set background color

    root.configure(bg="#ECF0F1")


    headingFrame1 = Frame(root, bg="#3498DB", bd=5)

    headingFrame1.place(relx=0.2, rely=0.1, relwidth=0.6,
relheight=0.16)
```

```
    headingLabel = Label(headingFrame1, text="AeroManageX",
bg='#3498DB', fg='white', font=('Arial', 24, 'bold'))

    headingLabel.place(relx=0, rely=0, relwidth=1, relheight=1)


    btn1 = Button(root, text="Add a new user", bg='#3498DB',
fg='white', command=lambda: addUser(userId),

                    font=('Arial', 14))

    btn1.place(relx=0.28, rely=0.35, relwidth=0.45, relheight=0.08)  #
Reduced relheight and rely


    btn2 = Button(root, text="Show all users", bg='#3498DB',
fg='white', command=lambda: showAll(userId),

                    font=('Arial', 14))

    btn2.place(relx=0.28, rely=0.45, relwidth=0.45, relheight=0.08)  #
Adjusted rely


    btn3 = Button(root, text="Delete a user", bg='#3498DB',
fg='white', command=delete, font=('Arial', 14))

    btn3.place(relx=0.28, rely=0.55, relwidth=0.45, relheight=0.08)  #
Adjusted rely


    btn4 = Button(root, text="Add Flight Information", bg='#3498DB',
fg='white', command=lambda: add_flight_gui(userId),

                    font=('Arial', 14))

    btn4.place(relx=0.28, rely=0.65, relwidth=0.45, relheight=0.08)  #
Adjusted rely


    btn5 = Button(root, text="Remove Flight Information",
bg='#3498DB', fg='white',

                    command=lambda: remove_flight_gui(userId),
font=('Arial', 14))

    btn5.place(relx=0.28, rely=0.75, relwidth=0.45, relheight=0.08)  #
Adjusted rely
```

```
    root.mainloop()
```

Action Pages

Description:

The Search Flights page allows clients to find flights that match their criteria of airport of origin and airport of arrival, and displays the Flight ID, departure date and time, and arrival date and time.
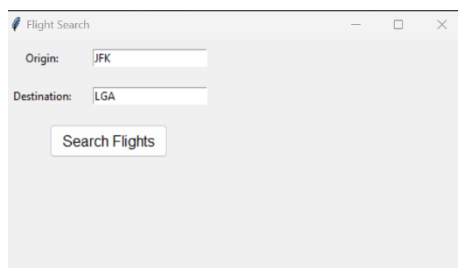
Visualization:

Figure 14. Search Flights Page

Figure 15. Search Flight Result Page

Search Page

AeroManageX_Admin.py

```python
from tkinter import *

from tkinter import ttk, messagebox

import mysql.connector


def connect_to_database():

    mypass = "Savfish4$"

    mydatabase = "mydb"


    try:

        connection = mysql.connector.connect(

            host="localhost",

            user="root",

            password=mypass,
```

```python
        database=mydatabase

        )

        return connection

    except mysql.connector.Error as err:

        print(f"Error: {err}")

        return None


def search_flights(origin, destination):

    connection = connect_to_database()


    if connection:

        try:

            cursor = connection.cursor()

            cursor.execute("SELECT * FROM flight WHERE Org_Airport_ID
= %s AND Des_Airport_ID = %s", (origin, destination))

            flights = cursor.fetchall()


            if flights:

                show_flight_info_popup(flights)

            else:

                messagebox.showinfo("No Flights", "No flights found
for the given criteria.")


        except mysql.connector.Error as err:

            print(f"Error: {err}")

        finally:

            cursor.close()

            connection.close()


def show_flight_info_popup(flights):

    popup = Toplevel()
```

```python
    popup.title("Flight Information")


    # Define columns
    columns = ("Flight ID", "Departure Date and Time", "Arrival Date
and Time")


    # Create Treeview
    tree = ttk.Treeview(popup, columns=columns, show="headings")


    # Set column headings
    for col in columns:
        tree.heading(col, text=col)
        tree.column(col, width=200, anchor="center")  # Adjust width
as needed


    # Insert data into the Treeview
    for flight in flights:
        tree.insert("", "end", values=flight)


    tree.pack(expand=YES, fill=BOTH)


def create_search_flights_gui():
    connection = connect_to_database()
    if connection:
        try:
            root = Tk()
            root.title("Flight Search")


            # Create a style for the buttons
            button_style = ttk.Style()
```

```python
            button_style.configure('TButton', font=('Arial', 12),
padding=(10, 5))


            # Labels and Entry widgets for origin and destination
            origin_label = Label(root, text="Origin:")
            origin_label.grid(row=1, column=0, pady=10, padx=10)
            origin_entry = Entry(root)
            origin_entry.grid(row=1, column=1, pady=10, padx=10)


            destination_label = Label(root, text="Destination:")
            destination_label.grid(row=2, column=0, pady=10, padx=10)
            destination_entry = Entry(root)
            destination_entry.grid(row=2, column=1, pady=10, padx=10)


            # Button to trigger the flight search
            search_button = ttk.Button(root, text="Search Flights",
command=lambda: search_button_click(origin_entry, destination_entry))
            search_button.grid(row=3, column=0, columnspan=2, pady=10)


            # Apply the ttk style to the search_button
            search_button["style"] = 'TButton'


            # Calculate center coordinates and set window size
            window_width = 500
            window_height = 250
            x_coordinate = (root.winfo_screenwidth() - window_width)
// 2
            y_coordinate = (root.winfo_screenheight() - window_height)
// 2

root.geometry(f"{window_width}x{window_height}+{x_coordinate}+{y_coord
inate}")
```

```python
        except mysql.connector.Error as err:

            print(f"Error: {err}")


def search_button_click(origin_entry, destination_entry):

    origin_value = origin_entry.get()

    destination_value = destination_entry.get()


    if origin_value and destination_value:

        search_flights(origin_value, destination_value)

    else:

        messagebox.showinfo("Error", "Both origin and destination
fields are required")


# Create the main window and run the Tkinter event loop

create_search_flights_gui()
```

Administration

Description:

As an administrator, there are options like insertion, deletion, and print users, as well as the addition or deletion of flights. Users use this page to perform specific actions on the database.

Visualization:

Figure 16. Add Flight Page

Figure 17. Remove Flight Page

Figure 18. Add User Page

Figure 19. Delete User Page

AeroManageX_AddUser.py

```python
from tkinter import *
from tkinter import messagebox
from tkinter import Button
import mysql.connector
import hashlib  # For password hashing


def userRegister(user_id):
    try:
        user_name = userName.get()
```

```python
        user_password = userPassword.get()


        if user_name != "" and user_password != "":
            hashed_password =
hashlib.sha256(user_password.encode()).hexdigest()  # Hash the
password

            role_id = 1 if user_id > 0 else 2


            insert_user_query = "INSERT INTO user (user_name,
user_password, role_id) VALUES (%s, %s, %s)"

            user_values = (user_name, hashed_password, role_id)


            # Execute the insert query
            with con.cursor() as cur:
                cur.execute(insert_user_query, user_values)
                con.commit()


                if user_id > 0:
                    cur.execute("SELECT user_id FROM user WHERE
user_name=%s", (user_name,))
                    contact_user_id = cur.fetchone()[0]


                    insert_contacts_query = "INSERT INTO userContacts
(user_id, name) VALUES (%s, %s)"
                    contacts_values = (user_id, user_name)


                    cur.execute(insert_contacts_query,
contacts_values)
                    con.commit()


        messagebox.showinfo("Info", "Record Inserted")
        root.destroy()
```

```python
        else:

            messagebox.showinfo("Info", "Enter Valid Records")


    except mysql.connector.Error as err:

        print(f"Error: {err}")


def addUser(args):

    global userName, userPassword, con, root


    root = Tk()

    root.title("Library")

    root.minsize(width=400, height=400)

    root.geometry("600x500")


    mypass = "Savfish4$"

    mydatabase = "mydb"


    con = mysql.connector.connect(

        host="localhost",

        user="root",

        password=mypass,

        database=mydatabase

    )


    headingFrame1 = Frame(root, bg="#F8F9F9", bd=5)

    headingFrame1.place(relx=0.25, rely=0.1, relwidth=0.5,
relheight=0.13)


    headingLabel = Label(headingFrame1, text="ADD A USER",
bg='#F8F9F9', fg='black', font=15)

    headingLabel.place(relx=0, rely=0, relwidth=1, relheight=1)
```

```python
    labelFrame = Frame(root, bg='#F8F9F9')

    labelFrame.place(relx=0.1, rely=0.4, relwidth=0.8, relheight=0.4)


    lb1 = Label(labelFrame, text="User Name : ", bg='#F8F9F9',
fg='black')

    lb1.place(relx=0.05, rely=0.02, relheight=0.08)


    userName = Entry(labelFrame)

    userName.place(relx=0.3, rely=0.02, relwidth=0.62, relheight=0.08)


    lb4 = Label(labelFrame, text="User Password", bg='#F8F9F9',
fg='black')

    lb4.place(relx=0.05, rely=0.60, relheight=0.08)


    userPassword = Entry(labelFrame)

    userPassword.place(relx=0.3, rely=0.60, relwidth=0.62,
relheight=0.08)


    SubmitBtn = Button(root, text="SUBMIT", bg='#82E0AA', fg='black',
command=lambda: userRegister(args))

    SubmitBtn.place(relx=0.28, rely=0.9, relwidth=0.18,
relheight=0.08)


    quitBtn = Button(root, text="QUIT", bg='#EC7063', fg='black',
command=root.destroy)

    quitBtn.place(relx=0.53, rely=0.9, relwidth=0.18, relheight=0.08)


    root.mainloop()
```

AeroManageX_DeleteUser.py

```python
from tkinter import *
```

```python
from tkinter import messagebox
import mysql.connector


mypass = "Savfish4$"
mydatabase = "mydb"
con = mysql.connector.connect(
    host="localhost",
    user="root",
    password=mypass,
    database=mydatabase
)
cur = con.cursor()


def delete():
    global userInfo, Canvas1, con, cur, root

    root = Tk()
    root.title("Delete")
    root.minsize(width=400, height=400)
    root.geometry("600x500")

    Canvas1 = Canvas(root)
    Canvas1.config(bg="#FDFEFE")
    Canvas1.pack(expand=True, fill=BOTH)

    headingFrame1 = Frame(root, bg="#FDFEFE", bd=5)
    headingFrame1.place(relx=0.25, rely=0.1, relwidth=0.5,
relheight=0.13)

    headingLabel = Label(headingFrame1, text="Delete An Existing
User", bg='#FDFEFE', fg='black', font=15)
```

```python
    headingLabel.place(relx=0, rely=0, relwidth=1, relheight=1)


    labelFrame = Frame(root, bg='#FDFEFE')

    labelFrame.place(relx=0.1, rely=0.3, relwidth=0.8, relheight=0.5)


    lb1 = Label(labelFrame, text="Enter user name: ", bg='#FDFEFE',
fg='black')

    lb1.place(relx=0.05, rely=0.5)


    userInfo = Entry(labelFrame)

    userInfo.place(relx=0.3, rely=0.5, relwidth=0.62)


    SubmitBtn = Button(root, text="Delete", bg='#d1ccc0', fg='black',
command=deleteUser)

    SubmitBtn.place(relx=0.28, rely=0.9, relwidth=0.18,
relheight=0.08)


    quitBtn = Button(root, text="Quit", bg='#f7f1e3', fg='black',
command=root.destroy)

    quitBtn.place(relx=0.53, rely=0.9, relwidth=0.18, relheight=0.08)


    root.mainloop()


def deleteUser():

    userName = userInfo.get()

    print(userName)

    getUserId = "SELECT user_id FROM user WHERE user_name = %s"

    print(getUserId)

    deleteByUserName = "DELETE FROM user WHERE user_name = %s"


    try:

        # Execute the getUserId query
```

```python
        cur.execute(getUserId, (userName,))

        result = cur.fetchone()


        # Check if the user was found

        if result:

            user_id = result[0]


            # Execute the deleteJunctionTableRecords query

            deleteJunctionTableRecords = "DELETE FROM user WHERE
user_id = %s"

            cur.execute(deleteJunctionTableRecords, (user_id,))


            # Execute the deleteByUserName query

            cur.execute(deleteByUserName, (userName,))


            # Commit the changes

            con.commit()


            # Display a message

            messagebox.showinfo("Information", "Record Deleted")

        else:

            messagebox.showinfo("Error", "User not found")


    except Exception as e:

        messagebox.showinfo("Error", str(e))

        print(f"Failed SQL Query: {getUserId} with parameters:
{userName}")
```

AeroManageX_ShowAllRecord.py

```python
from tkinter import *

from tkinter.ttk import Treeview
```

```python
from tkinter import messagebox



import mysql.connector



mypass = "Savfish4$"

mydatabase="mydb"

con = mysql.connector.connect(

        host="localhost",

        user="root",

        password=mypass,

        database=mydatabase

    )

cur = con.cursor()



def showAll(userId):

    class A(Frame):

        def __init__(self, parent):

            Frame.__init__(self, parent)

            self.CreateUI()

            self.LoadTable()

            self.grid(sticky=(N, S, W, E))

            parent.grid_rowconfigure(0, weight=1)

            parent.grid_columnconfigure(0, weight=1)


        def CreateUI(self):

            tv = Treeview(self)

            tv['columns'] = ('user_id', 'user_name', 'role_id')
```

```python
        tv.heading('#0', text='user_id', anchor='center')

        tv.column('#0', anchor='center')

        tv.heading('#1', text='user_name', anchor='center')

        tv.column('#1', anchor='center')

        tv.heading('#2', text='role_id', anchor='center')

        tv.column('#2', anchor='center')

        tv.grid(sticky=(N, S, W, E))

        self.treeview = tv

        self.grid_rowconfigure(0, weight=1)

        self.grid_columnconfigure(0, weight=1)


    def LoadTable(self):
        cur.execute("SELECT * FROM usercontacts WHERE user_id =
%s", (userId,))

        result = cur.fetchall()

        user_id = ""

        user_name = ""

        role_id = ""

        for i in result:

            user_id = i[0]

            user_name = i[1]

            role_id = i[2]

            self.treeview.insert("", 'end', text=user_id,

                        values=(user_name, role_id))

root = Tk()

root.title("Your contacts")

A(root)
```

13. Conclusion and Future work

Reflection

We learned a lot about database design, application development, and the value of user experience while working on this project. SQL code taught us how to turn abstract ideas into actual structures, from building the Entity-Relationship models to implementing the database in MySQL. Assuring data integrity and improving system efficiency grew dependent on performance optimizations and normalization checks. Creating the graphical user interface (GUI) required careful planning and user-friendly design, which was an exciting challenge. The pages of the GUI, which included the login, main menu, search, insertion, modification, deletion, and data display, were linked together to offer the user smooth functioning and navigation. To demonstrate these views' functionality, links to the database, and the code behind each page's features, comprehensive design descriptions, visualizations, and source code were used in their implementation.

Enhancements

In order to improve this project even more, a few things could be added:

1) User roles and access levels should be divided into a more comprehensive login system for user authentication and authorization.
2) Data validation is the process of applying extensive validation checks for user inputs in order to guarantee data confidentiality and accuracy.
3) Reporting and Analytics: Including a function for creating reports and examining patterns in data.
4) Including graphical data representations for enhanced comprehension and analysis is known as interactive visualization.
5) Real-time Updates: To allow for immediate data updates, enable real-time synchronization between the database and the GUI.
6) Mobile responsiveness: Making the graphical user interface (GUI) more accessible on mobile devices.

The program would become more adaptable, safe, and user-friendly by adding these features, meeting the needs and preferences of a larger spectrum of users.

14. References

1. Airline Suite. (n.d.). *Airline Suite Reviews - Pros & Cons, Ratings & More*. GetApp.
   https://www.getapp.com/operations-management-software/a/airline-suite/reviews/
2. *Aviation Maintenance Operations & Aircraft Management Software: RAMCO systems*. Aviation
   Maintenance Operations & Aircraft Management Software | Ramco Systems. (n.d.).
   https://www.ramco.com/products/aviation-
   software/?utm_source=capterra&utm_medium=capterra_ppc&utm_campaign=capterra_aviation&utm_ch
   annel=capterra
3. AvPro. (n.d.-a). *Aircraft Maintenance Software: Wellington FL: AvPro software*. avpro.
   https://www.avprosoftware.com/
4. AvPro. (n.d.). *Fly Safer. work smarter.* C.A.L.M. Systems | Airline Suite | Aviation Management Software.
   https://info.gartnerdigitalmarkets.com/calm-systems-gdm-lp?utm_source=capterra
5. Trustradius. (2021, June 16). *Pros and cons of Ramco ERP 2023*. TrustRadius.
   https://www.trustradius.com/products/ramco-erp/reviews?qs=pros-and-cons#reviews