CSDS 341
Project Final Report
Airlines Management Database System
Orion Follett, Alexander Harding Bradley, Sanhita Kumari, Vivek Kapur
April 20, 2021

1. Overview

Airlines often need to coordinate hundreds of flights, and thousands of passengers, staff, and luggage, all on a daily basis. The purpose of an airline management system is to provide a single point of truth where airline personnel can gain access to accurate information concerning the scheduling of flights, booking of seats, managing ticket sales, and maintaining restrictions (constraints) as required. This system can also provide records of past flights, and be used to plan future trips.

The backbone of a good airline management system is a well designed database. Our project will involve the core database that any airline would need to maintain, as well as an interface to access the contents of that database. Our web interface is meant for internal use so there is access to add flights and search through all customers, rather than being restricted to a single customer's view.

2. Database Requirements Specification

(Notation for the entire report, bolded is primary key, underlined is foreign key)

a. Plane:

i. **Data**: Each plane will have a plane id (which uniquely identifies a plane owned or loaned to the airline), the plane model, the number of seats in the plane, the capacity for bags, the total amount of weight allowed on the plane, and the range of the plane (based on how much fuel is in the plane).

ii. **Integrity constraints**:

All values must be non negative

b. Passenger:

i. **Data**: Every passenger in the system will have a customer id (which uniquely identifies a customer of the airline), a name related to the passenger, and a username for the passenger

ii. **Queries**:

1. You should be able to list all of the customers upcoming trips.
    a. Frequency: High
2. You should be able to list all of the customer name and ids and usernames who are taking a certain flight as well as who have a certain first and last name
    a. Frequency: medium

iii. **Integrity constraints**:

1. Customer full name must be less than 100 characters

c. Pilot:
   i. **Data**: Each pilot will have an employee id (which uniquely identifies a certain employee working for the airline), and the name of this employee
   ii. **Queries**:
      1. You should be able to list the pilot's ids based on their first name and last name, for later registering a flight
         a. Frequency: medium
      2. You should be able to list the pilot's names and ids who are not currently flying an airplane for a flight
         a. Frequency: low

d. Airport:
   i. **Data**: Each airport will have an airport id (which uniquely identifies a certain airport which the airline arranges flights to or from), the name of this airport, the city that this airport is located in, and the number of gates in the airport.
   ii. **Queries**:
      1. You should be able to list the customer names ids of customers who are leaving from a certain airport on a given date.
   iii. **Integrity constraints**:
      1. There must not be two airports entered into the database with the exact same name in the exact same city

e. Bag:
   i. **Data**: Every bag in the system will store a bag id (which uniquely identifies the bag), the weight of the bag, the type of bag (whether it is a check-in bag or a carry-on bag), and the id of the trip which the bag will be on the plane for.
   ii. **Queries**:
      1. You should be able to list the bags that are currently checked in for a given trip
         a. Frequency: medium
   iii. **Integrity constraints**:
      1. There can be at most two carry-on bags and at most two check-in bags associated with a given customer's trip.

f. Trip:
   i. **Data**: Every trip in the system will represent a specific customer on a given flight. The trip will have a trip id (which uniquely identifies the trip), the id of the flight which the customer is taking, the customer id of the customer on the trip, the seat type of the customer (whether they are sitting in economy, business, a pilot's seat, or a flight attendant's seat)
   ii. **Queries**:
      1. Trips is used for any query that requires information about a customers flights
   iii. **Integrity constraints**:
      1. A customer may only book a maximum of ten future trips
      2. A customer cannot reserve two trips for the same flight

g. Flight:

i. **Data**: Each flight in the system will have a flight id (which uniquely identifies a flight), the id of the plane which the flight will take place on, the id of the airport which the flight will leave from, the id of the airport which the flight will arrive in, the departure time/date of the flight, the arrival time/date of the flight,

ii. **Queries**:

1. You should be able to list the number of a given seat class of seats which are currently taken for a given flight.
    a. Frequency: low
2. You should be able to list the customers and customer ids of customers who are going on a given flight
    a. Frequency: high
3. You should be able to list the number of total economy seats and the number of total business seats on the plane used for a given flight
    a. Frequency: medium

iii. **Integrity constraints**:

1. There cannot be a pilot reserved for multiple flights at the same time
2. A plane cannot be reserved for multiple flights at the same time

3. ER Data Model Design

PRIMARY KEYS ARE BOLDED

Strong Entities:

Plane(**pid: 4 digit numeric**, model: char[10], noBusinessSeats: 3 digit numeric, noEconomySeats: 3 digit numeric, carryOnCapacity: 3 digit numeric, checkInCapacity: 3 digit numeric, maxWeight: 6 digit numeric, range: 6 digit numeric)

Passenger(**cid: 4 digit numeric**, cname: char[20], username: char[20], password: char[20])

Pilot(**pilotID: 4 digit numeric**, pname: char[20])

Airport(**aid: 4 digit numeric**, aname: char[20], city: char[20], numGates: 3 digit numeric)

Weak entities:

Bag(registeredOrder: 1 digit numeric, bagType: char[20], weight: 3 digit numeric)

Identifying relationship type for Bag: Owns
Primary key for Section: (registeredOrder)

Identifying strong entity type for Bag: Customer

Relationships:

If not specified, it's assumed that the entities have partial participation within the relationship

FlysOn:

Attributes: (price: 4 digit numeric)

Description:

-Connects many passengers to a single plane

-Many to one relationship

Flys:

Attributes: None

Description:

- Connects one pilot to one plane
- One to One
- Plane has total participation in pilot

Departs:

Attributes: (departureTime: ETIME type, departureGate: 3 digit numeric, distance: 4 digit numeric)

Description:

- connects plane to departure airport
- Many to One

Arrives:

Attributes: (arrivalTime: ETIME type, arrivalGate: 3 digit numeric)

Description

- connect plane to arrival airport
- Many to one

Owns:

Attributes: None

- passenger to a bag
- one to many

Constraint Satisfaction:

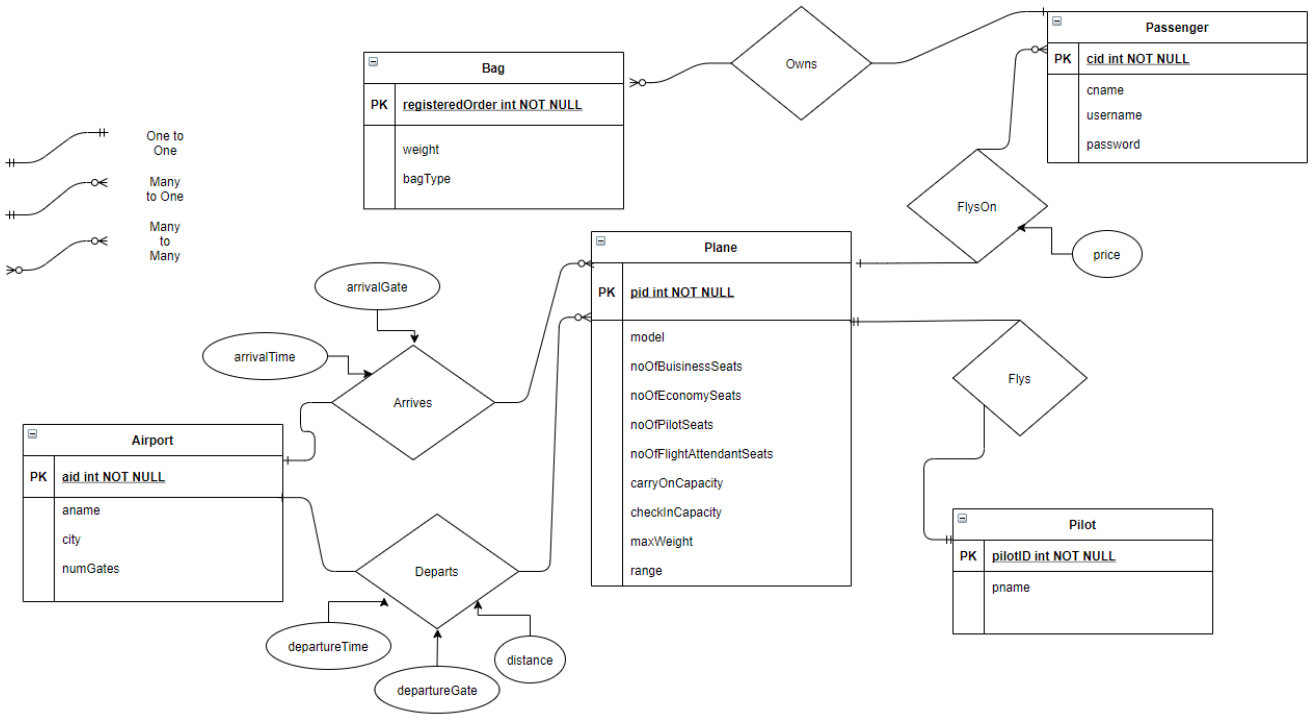
Cardinality: (Mentioned in ER Diagram)

Primary key constraints for each entity/relationship: (Mentioned in ER Diagram)

Foreign key constraints:

- None (there are no foreign keys)

ER Diagram:



4. Relational Model

Relations:

(**Primary Keys are bold**, Foreign Keys are underlined)

Plane(**pid**, model, noBusinessSeats, noEconomySeats, carryOnCapacity, checkInCapacity, maxWeight, range)

Passenger(**cid**, cname, username, password)

Pilot(**pilotID**, pname)

Airport(**aid**, aname, city, numGates)

Bag(**bid**, weight, bagType, tid)

Trip(**tid**, fid, cid, ticketPrice)

Flight(**fid**, pid, pilotID, startAid, destinationAid, departureTime, distance, arrivalTime, departureGate, arrivalGate)

Integrity Constraints:
Customer : username is unique
Flight: arrival time after departure time, distance less than or equal to associated plane range

```
CREATE TABLE IF NOT EXISTS "plane" (
  pid SERIAL PRIMARY KEY,
  model VARCHAR(100) NOT NULL,
  noBusinessSeats INTEGER NOT NULL,
  noEconomySeats INTEGER NOT NULL,
  carryOnCapacity INTEGER NOT NULL,
  checkInCapacity INTEGER NOT NULL,
  maxWeight INTEGER NOT NULL,
  range INTEGER NOT NULL
);

CREATE TABLE IF NOT EXISTS "passenger" (
  cid SERIAL PRIMARY KEY,
  cname VARCHAR(100) NOT NULL,
  username VARCHAR(15) NOT NULL,
  password VARCHAR(25) NOT NULL
);

CREATE TABLE IF NOT EXISTS "pilot" (
  pilotID SERIAL PRIMARY KEY,
  pname VARCHAR(50) NOT NULL,
);

CREATE TABLE IF NOT EXISTS "airport" (
  aid SERIAL PRIMARY KEY,
  aname VARCHAR(50) NOT NULL,
  city VARCHAR(25) NOT NULL,
  numGates INTEGER NOT NULL
);
```

```
CREATE TABLE IF NOT EXISTS "flight" (
  fid SERIAL PRIMARY KEY,
  pid INTEGER NOT NULL REFERENCES "plane",
  pilotID INTEGER NOT NULL REFERENCES "pilot",
  startAid INTEGER NOT NULL REFERENCES "airport",
  destinationAid INTEGER NOT NULL REFERENCES "airport",
  distance INTEGER NOT NULL,
  departureGate INTEGER NOT NULL,
  arrivalGate INTEGER NOT NULL
);

CREATE TABLE IF NOT EXISTS "trip" (
  tid SERIAL PRIMARY KEY,
  fid INTEGER NOT NULL REFERENCES "flight",
  cid INTEGER NOT NULL REFERENCES "passenger",
  ticketPrice INTEGER NOT NULL
);

CREATE TABLE IF NOT EXISTS "bag" (
  bid SERIAL PRIMARY KEY,
  bagType VARCHAR(50) NOT NULL,
  weight VARCHAR(25) NOT NULL,
  tid INTEGER NOT NULL REFERENCES "trip"
);
```

5. *Create the database*

We are using PostgreSQL for our database. We did not encounter any issues installing it, as we simply followed instructions on how to install it which worked out fine. As we used it, we had to learn the minor differences between the SQL that we had learned and the SQL required for this database management system. This was our main issue we encountered in using the DBMS.

6. SQL Queries

Note: we use {} to denote data which is given dynamically.

Query: Select all of the passengers registered in the database with the given first name and last name.

```
SELECT * FROM passenger WHERE firstname = {firstname} AND lastname = {lastname}
```

Query: List all of the passenger name and ids who are flying out of a certain airport on a certain date (given the airport name and date, where date is formatted like the following example: “2020-04-20”):

SELECT DISTINCT C.cname, C.cid
FROM Customer C, Trip T, Flight F, Airport A
WHERE A.aname = {airportName} AND A.aid = F.startAid AND F.departureTime >= ‘{date}T00:00:00.000’ AND F.departureTime <= ‘{date}T23:59:99.999’ AND T.fid = F.fid AND T.cid = C.cid

Query: Select all of the passengers registered in the database with the given first name and last name and username.

SELECT * FROM passenger WHERE firstname = {firstname} AND lastname = {lastname} AND username = {username}

Query: Select all of the flights currently in the database with the given departure airport, arrival airport, and a range for the departure dates.

SELECT * FROM flight WHERE startaid = {departureairport} AND destinationaid = {arrivalairport} AND departureDate > {departuredatebegin} AND departureDate < {departuredateend}

Query: Create a flight instance, given all necessary flight information, and return the new instance’s fid

INSERT INTO flight (pid, pilotid, startaid, destinationaid, departuregate, arrivalgate, departuredate, arrivaldate) VALUES ({planeid}, {pilotid}, {departureairport}, {arrivalairport}, {departuregate}, {arrivalgate}, {departuredatetime}, {arrivaldatetime}) RETURNING fid

Query: List all of a customer’s future trips (given cid)

SELECT * FROM passenger WHERE firstname = $1 AND lastname = $2 AND username = $3
SELECT T.tid
FROM Trip T, Flight F
WHERE T.cid = {cid} AND T.fid = F.fid AND F.departureTime > NOW()

Query: Create a passenger instance, given all necessary passenger information, and return the new instance’s cid

INSERT INTO passenger (username, firstname, lastname) VALUES ({username}, {firstname}, {lastname}) RETURNING cid

Query: List all of the plane ids and models of planes with a range greater than the specified range and not on any flight for a given date:

SELECT P.pid, P.model

FROM Plane P
WHERE ({range} < P.range) AND NOT EXISTS (SELECT * FROM Flight F WHERE F.departureTime >= '{date}T00:00:00.000' AND F.departureTime <= '{date}T23:59:99.999' AND F.pid = P.pid)

Query: Create an airport instance, given all necessary airport information, and return the new instance's aid

INSERT INTO airport (aname, city, numgates, country) VALUES (${airportname} {airportcity}, {airportnumgates}, {airportcountry}) RETURNING aid

Query: Create a pilot instance, given all necessary pilot information, and return the new instance's pilotid

INSERT INTO pilot (firstname, lastname, username) VALUES ({firstname}, {lastname}, {username}) RETURNING pilotid

Query: List the names and ids of all of the customers who are going on a given flight.

SELECT C.cname, C.cid
FROM Trip T, Customer C
WHERE T.fid = {fid} AND C.cid = T.cid

Query: Create a trip instance, given all necessary trip information, and return the new instance's pilotid

INSERT INTO trip (fid, cid, ticketprice) VALUES ({flightid}, {passengerid}, {ticketprice}) RETURNING tid

Query: Delete a flight instance given the flight's fid

First: DELETE FROM trip WHERE fid = {fid}
Then: DELETE FROM flight WHERE fid = {fid}

Query: List the ids and bag types of bags currently associated with a given trip.

SELECT B.bid, B.BagType
FROM Bag B
WHERE b.tid = {tid}
Query:Delete a trip instance given the trip's tid

DELETE FROM trip WHERE tid = {tid}

Query: List the trips registered in the database, given the customer and the flight number

SELECT * FROM trip WHERE cid = {cid} AND fid = {fid}

Query: Give the number of taken seats on the plane for a given flight in the given seat class.

```
SELECT COUNT(*)
FROM Trip T, Flight F
WHERE T.fid = {fid} AND T.seatType = {seatClass}
```

Query: Give the list of flights between the given range of departure dates

```
SELECT * FROM flight WHERE departureDate > {departuredates} AND departureDate < {departuredates}
```

Query: Give the list of flights flying between the given start airport and destination airport

```
SELECT * FROM flight WHERE startaid = {startaid} AND destinationaid = {destinationaid}
```

Query: List all of the available pilot id's and names who are currently not on a flight.

```
SELECT E.pilotID, E.name
FROM Pilot E
WHERE NOT EXISTS (SELECT * FROM Flight F WHERE F.pilotID = E.pilotID AND F.departureTime < NOW()
AND F.arrivalTime > NOW())
```

7. Integrity Constraints

1. A customer can only make a maximum of 10 upcoming trips

```
CREATE TRIGGER MaxTrips
BEFORE INSERT ON Trip
FOR EACH ROW
IF (SELECT COUNT(*) FROM Trip T, Flight F WHERE T.cid = NEW.cid AND T.fid = F.fid AND
        F.departureTime >= NOW()) = 10 THEN CALL RAISE_APPLICATION_ERROR (3001,
        'Max future trips reserved!');
END IF;
```

2. There cannot be a pilot reserved for multiple flights at the same time

```
CREATE TRIGGER ConcurrentPilotFlight
BEFORE INSERT ON Flight
FOR EACH ROW
IF (SELECT COUNT(*) FROM Flight F WHERE F.pilotID = NEW.pilotID AND (F.arrivalTime <
        NEW.departureTime AND F.departureTime > NEW.departureTime) OR
        (NEW.arrivalTime < F.departureTime AND NEW.departureTime > F.departureTime) OR
        (F.departureTime < NEW.departureTime AND F.arrivalTime > NEW.arrivalTime) OR
        (NEW.departureTime < F.departureTime AND NEW.arrivalTime > F.arrivalTime)) = 1
        THEN CALL_RAISE_APPLICATION_ERROR(3001, 'Flight cannot be created, pilot
        scheduled for other flight at same time!'')
END IF;
```

3. A plane cannot be reserved for multiple flights at the same time

```
CREATE TRIGGER ConcurrentPlaneFlight
BEFORE INSERT ON Flight
FOR EACH ROW
IF (SELECT COUNT(*) FROM Flight F WHERE F.pid = NEW.pid AND (F.arrivalTime <
        NEW.departureTime AND F.departureTime > NEW.departureTime) OR
        (NEW.arrivalTime < F.departureTime AND NEW.departureTime > F.departureTime) OR
        (F.departureTime < NEW.departureTime AND F.arrivalTime > NEW.arrivalTime) OR
        (NEW.departureTime < F.departureTime AND NEW.arrivalTime > F.arrivalTime)) = 1
        THEN CALL RAISE_APPLICATION_ERROR(3001, ‘Flight cannot be created, plane
        scheduled for other flight at same time!’’)
END IF;
```

4. A customer must not reserve multiple trips for the same flight

```
CREATE TRIGGER DuplicateReservations
BEFORE INSERT ON Trip
FOR EACH ROW
IF (SELECT COUNT(*) FROM Trip T WHERE T.fid = NEW.fid AND T.cid = NEW.cid) = 1 THEN
CALL
        RAISE_APPLICATION_ERROR (3001, 'Multiple reservations for same customer cannot
        be on same flight!');
END IF;
```

5. There cannot be multiple airports both with the same name and the same city

```
CREATE TRIGGER SameAirportCharacteristics
BEFORE INSERT ON Airport
FOR EACH ROW
IF (SELECT COUNT(*) FROM Airport A WHERE A.aname =  NEW.aname AND A.city =
        New.city) = 1 THEN CALL RAISE_APPLICATION_ERROR (3001, 'Cannot be two
        airports with exact same name in same city!');
END IF;
```

6. A customer can have at most two check-in bags and two carry-on bags for a given trip

```
CREATE TRIGGER MaxBags
BEFORE INSERT ON Bag
FOR EACH ROW
IF (SELECT COUNT(*) FROM Bag B B.tid = NEW.tid AND B.bagType = NEW.bagType) = 2
        THEN CALL RAISE_APPLICATION_ERROR (3001, 'You have reached the limit of this
        bagType on this trip!');
END IF;
```

7. Customer name must be less than or equal to 100 characters.

This is satisfied in the creation of the table, which specifies that the entire customer name has a limit of 100 characters

8. Relational Database Design--Applying the Database DesignTheory(Ch 8 of your textbook). Perform normalization by applying all those algorithms you have mastered in the class. More specifically, define your functional dependencies, and find the minimal set of f.d.s. Are all your relations in BCNF or 3NF? If not, apply the algorithms you have seen in the class to decompose and make them BCNF/3NF. Note that your decompositions should be lossless and dependency preserving.

Relations: (primary key bolded, foreign key underlined)
Plane(**pid**, model, noBusinessSeats, noEconomySeats, carryOnCapacity, checkInCapacity, maxWeight, range)

Passenger(**cid**, cname, username, password)

Pilot(**pilotID**, pname)

Airport(**aid**, aname, city, numGates)

Bag(**bid**, weight, bagType, tid)

Trip(**tid**, fid, cid, ticketPrice)

Flight(**fid**, pid, pilotID, startAid, destinationAid, departureTime, distance, arrivalTime, departureGate, arrivalGate)

Functional Dependencies:
Plane:
F = {
Pid → model, noBusinessSeats, noEconomySeats, carryOnCapacity, checkInCapacity, maxWeight, range
}

Passenger:
F = {Cid → cname, username, password}

Airport:
F = {Aid → aname, city, numGates}

Bag:
F = {Bid→ weight, bagType, tid}

Pilot:
F = {pilotID → pname}

Trip:
F = {Tid → fid, cid, seatType, ticketPrice;
fid, cid → tid, seatType, ticketPrice}

Flight:
F = {fid → pid, pilotID, startAid, destinationAid, departureTime, distance, arrivalTime, departureGate, arrivalGate

departureTime, departureGate → fid, pid, pilotID, startAid, destinationAid, distance, arrivalTime, arrivalGate

arrivalTime, arrivalGate → fid, pid, pilotID, startAid, destinationAid, distance, arrivalTime, arrivalGate

pid, departureTime → fid, pilotID, startAid, destinationAid, distance, arrivalTime, departureGate, arrivalGatejob

pid, arrivalTime → fid, pilotID, startAid, destinationAid, distance, arrivalTime, departureGate, arrivalGate

pilotID, departureTime → fid, pid, startAid, destinationAid, distance, arrivalTime, departureGate,

pilotID, arrivalTime → fid, pid, startAid, destinationAid, distance, departureTime, departureGate,
}

9. No changes made to the relations. Since every left hand side of every non redundant functional dependency is a candidate key, (every other attribute in the relationship can be derived from them), our database conforms to BCNF, and subsequently, it also conforms to the 3NF.

10. DBMS Implementation.Start using the DBMS asap. Create your schema, constraints and queries. Populate your database, and run/test your queries, constraints. Develop and test your stored procedures. Summarize the main components of your code here with proper explanations. Discuss any problems encountered and how you have solved them. If some of the problems require the team to extensively redesign everything, the team may choose to point them out, and not implement them. If some of the stored procedures are too elaborate, scale down your design, explain your decisions, and implement the scaled down version.

There are several main components for our code. Seed.js is what initially populated our database with several flights, customers, pilots, bags, etc. One consideration that had to be made while creating the seed file was that the order of the creation of entities mattered. For example we could not make a trip entity before we had already made at least a customer and a flight to go on that trip. Index.js is one of our main files for the web

application. It has the basic functionality such as querying the database, setting the views, and so on. The Pages folder has all of the html files that our project uses. The Dynamic Pages folder has some ejs files. The ejs files are a combination of html and javascript and are used for some of the searching and displaying of entities throughout the application.

One of the main issues we ran into was constantly redesigning our web application and database in order to make a polished final product that was user-friendly and worked well. In order to do this, we decided to focus the web application for a solely-employee-based interface with the necessary functionality for the employee to register flights, pilots, airports, customers, and trips, and to search for all of those entities via various combinations of SQL statements, and finally to delete and update those parameters. While the web application showed a lot of our databases' functionality, it didn't provide an interface for all of the SQL queries that we created for further searching flights and customers according to unique filters, but we wrote out in our SQL query section which you can use to test on the database itself.

11.Application Implementation.Code transactions (and eb forms, web services, if any). Discuss any problems encountered, and how they are resolved.

We ran into many small problems while implementing the application. One of the biggest issues we ran into while implementing the web application was displaying information from the database. Specifically, we had a problem with rendering a dynamic ejs file once we had retrieved that data that we wanted to send to the front end for filtered data. We started out by trying to implement a page that can search for flights. We realized that the problem was that we weren't using the pool.query method correctly, which was causing the errors. Once we figured out what the issue was behind rendering the ejs files, we used a similar approach to display information on other pages.

Another problem we encountered was that we had to design entire pages of html/ css for each individual page on the web application. While this doesn't sound like a lot, given that we had individual pages for the searching of each of our entities, pages for a details of a decent amount of our entities, and pages to create each kind of entity for our database, it was hard to ensure that each of these pages were working properly and were correctly linked to each other via navigation bar/ other areas. Nevertheless, we spent time customizing each page and looking to make sure that all of the links/ information on the page was specific to the kind of data each page was meant to be showing.

One last problem we encountered was with writing code in order to populate our database. We needed to write a program which would enter randomized code, to show what our application would look like with a lot of entries for each entity in our database. We spent a lot of time rewriting and testing the code so that the program would properly populate our database, which meant constantly dropping the information in the database. Nevertheless, we were able to successfully populate each of our entities within the database.

12.Conclusions.List the contributions of each project team member to the project. And, discuss here anything else you want to say, and conclude.

Vivek: Setting up the DBMS, writing SQL queries, backend and middleware

Alex:
-Functional dependencies, middleware, frontend and UI, DRS, SQL queries,integrity constraints

Sanhita:
- ER Diagram, schemas, backend with setting up sql queries, presentation, frontend pages

Orion:
-Create table commands, translating ER diagram to relational database, Appendices, some front end

Conclusions:

Databases are oftentimes the foundation of a web application. Good database design can make the creation of a web application much simpler. Through this project, we learned about how to design and implement a database-backed web application from scratch. Since we had never done this before, there were a lot of obstacles in our way towards creating a functional database and web application in the first place. We had problems with how to correctly represent the information we were hoping to include in our database, problems creating our database on amazon web services, and problems creating the middleware and frontend in order to provide a user interface for the hypothetical employees to more easily interact with our database.

Nevertheless, we eventually managed to overcome these obstacles and make a web application that interacted with our database through searching with various parameters, inserting, deleting, and updating, all through our web application. Given how little we all knew about databases as well as how to set up a database and corresponding front-end at the beginning of this semester, we believe that this project was a success and we took away a lot from this experience.

13.**Appendix 1. Installation Manual.**Write an Installation Manual that explains how one can compile and implement your system.

- Clone our repo: git clone https://github.com/vkapur2202/airlineDB.git

- In order to connect to the database, create a file called ".env" in the directory where you cloned the repo, in the file put the text "CONNECTION_STRING = 'postgresql://postgres:airlines@airline-management-db.cb7isu5t5kyq.us-east-2.rds.amazonaws.com:5432/airlinesDB' "(without the outermost quotation marks)

- run "npm install" in the directory to download the necessary dependencies (you may need to install npm which comes with node.js, to install node.js, check this link: https://nodejs.org/en/, details vary depending on your system)

- finally run “node index.js” by default the it should run on port 3000 though this may vary, regardless, the output of the node index.js command will tell you which port to check. In your browser, navigate to http://localhost:3000/ this should bring you to the main page. See the users manual for further instruction.

14.**Appendix 2. Users manual**.Write a Users Manual that explains to a naïve user how to start using your application.

Users Manual:

The site should be relatively straightforward to use. At the top there is a navbar where you can see flights, airports, pilots, customers, etc. Clicking on any of these will reveal a drop down menu that shows the available options.

Available pages are listed below. These are all available through the navbar on the main landing page.

1.Search Flights: Can enter a combination of departure airport ID, arrival airport ID, departure time and arrival time, it will output flights that fit the search parameters, click on a flight to take you to the flight detail view for more detailed information on the flight. You can also delete a flight from the detail view.

2. Register Flights: Enter information into the fields, will add flight to the database

3. Register New Customer: Enter information into the fields, will add customer to the database

4. Add Customer to Flight: Enter the customer’s username and the flight ID for the flight you want to register them for. Also fill in the number of bags. This will create a trip entity in the database associating that customer with that flight.

5. Search Customer Trips: Enter the customer credentials, a list of associated flights will appear.

6. Search Customers: Enter the customer’s name, email or combination of both, matching customers will be displayed.

7. Search Pilots: Similar to Search Customer, enter pilot credentials to view a pilot, at first all the pilots will be displayed (up to a maximum number).

8. Register Pilot: Enter pilot credentials and the pilot will be added to the database.

9. Search Airport: Enter either, city, country, or the airport name and a list of airports that match the criteria will be displayed.

10. Register Airport: Enter information about the airport and a new airport will be created.

15.**Appendix 3. Programmers Manual.** Write a Programmers Manual that explains the classes, functions, etc and their functionality. Post your report, data, etc., on your Canvas group. Supply any password info that I may need to run your application.

Programmers Manual:

Pages Folder -> HTML files for our webpages, self explanatory HTML code.

Dynamic Pages Folder -> EJS files that form our more dynamic webpages
The files in here are all of the Search and Detail pages, for example Flight Search, Airport Search, Airport detail view, Flight detail view, etc. The EJS files have a combination of HTML and Javascript inside them. The Javascript generally requests a query from the database while the HTML sets up the format and look of the webpage.

Index.js -> main entry point where program is run and setup, this is the middleware of the web application and is where the structure of the website is defined, such as the various links that are used and what file each link should load.

Seed.js -> populates the database with random data for testing purposes.