CS 3083 Introduction to Databases

Air Ticket Reservation System

Mohammed Adrian Ajao
Jonathan Lin

Project Submission Date: 5/07/2021

Project Objective and Overview

The objective of this project is to gain realistic experience of relational database design process. The project focuses on conceptual and logical design, and implementation of relational databases. A web-based application is also created to help create communication with the database.

The system that is going to be designed is an online Air Ticket Reservation System. There will be three users using the system: customers, booking agents, and airline staff/administrator. Customers can use the system to search for flights, purchase tickets, and view future and past flights. Booking agents book flights for customers and receive commission for booking. Airline staff/administrator will be able to add new airplanes, create new flights, and update flight status.

Responsibilities

Mohammed Adrian Ajao was responsible for the middleware design, api design, and front-end development. Jonathan Lin focused on the backend, middleware implementation, constraints, and backend design.

Technologies

NodeJS

AdonisJS - A framework used to build the api for the application. It includes the Lucid ORM which is sometimes used to query objects in the project. It also was used to track database changes. Primarily, it was used for the middleware and authentication.

VueJS - A frontend framework used for the client. Used axios with it to connect with the API.

Running

Both the server and client use .env files.

VUE_APP_API_CONFIG is used to set the api call url for the client in the .env of the client.

The server has a sample .env file that can be used.

Project can be installed with standard node procedures with npm install. AdonisJS has custom commands. Use **node ace serve --watch** to launch the api with watch mode on. The flag can be removed to simply serve the api.

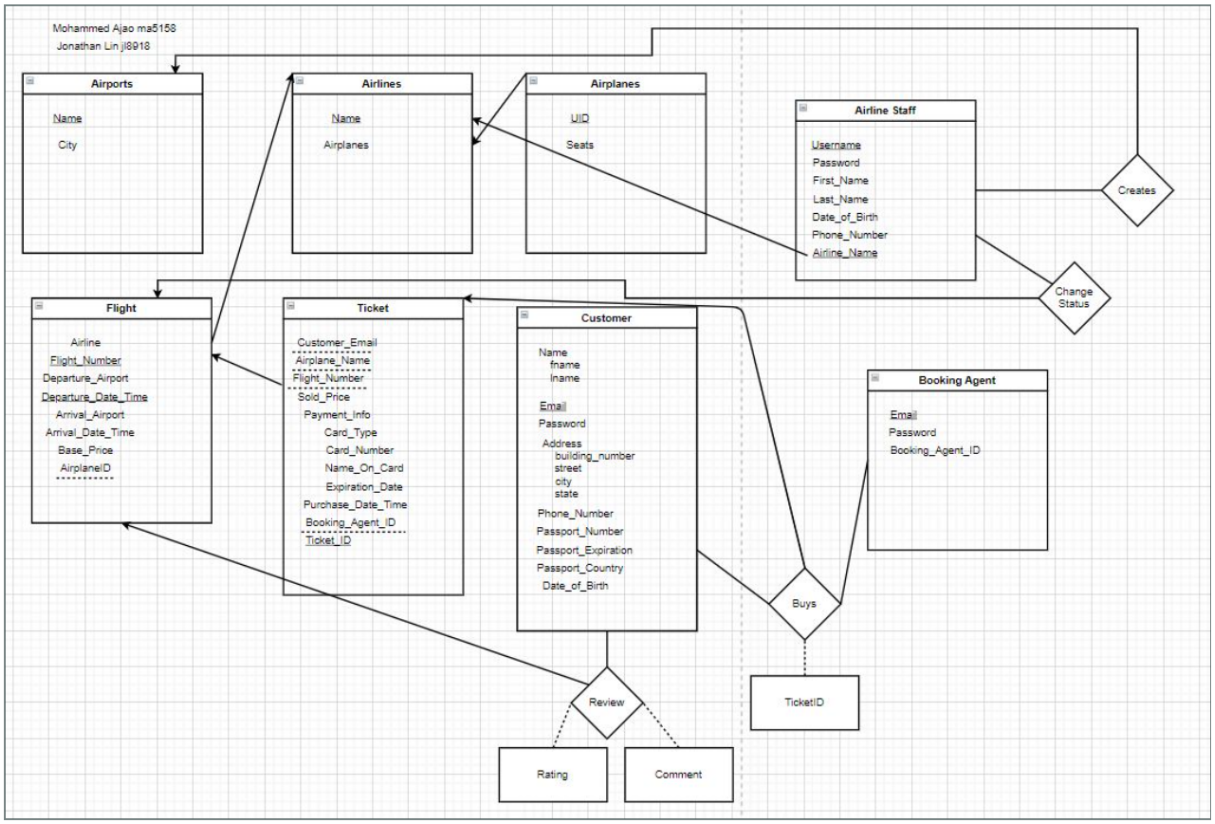
In the client folder, run **npm run serve** in order to run the client in a separate terminal window.

Project Implementation

Part 1

For part 1 of the project, ER diagrams were created for the Air Ticket Reservation System. The ER diagram consisted of Airports, Flight, Airlines, Ticket, Airplanes, Customer, Airline Staff, and Booking Agent.

The following are the entities and their components for the ER diagram:
Airports: name and city
Flight: airline, flight number, departure airport, departure date time, arrival airport, arrival date time, base price, airplaneID
Airlines: name and airplanes
Ticket: customer email, airplane name, flight number, sold price -> payment info, card type, card number, name on card, expiration date, purchase date time, booking agent ID, ticket ID
Airplanes: UID, seats
Customer: name -> fname, lname, email, password, address -> building number, street, city, state, phone number, passport number, passport expiration, passport country, date of birth,
Booking agent: email, password, booking agent ID



The above ER demonstrates the entity relation model for the Air Ticket Reservation System. The initial model of the ER diagram shows how each entity model interacts with other entities.

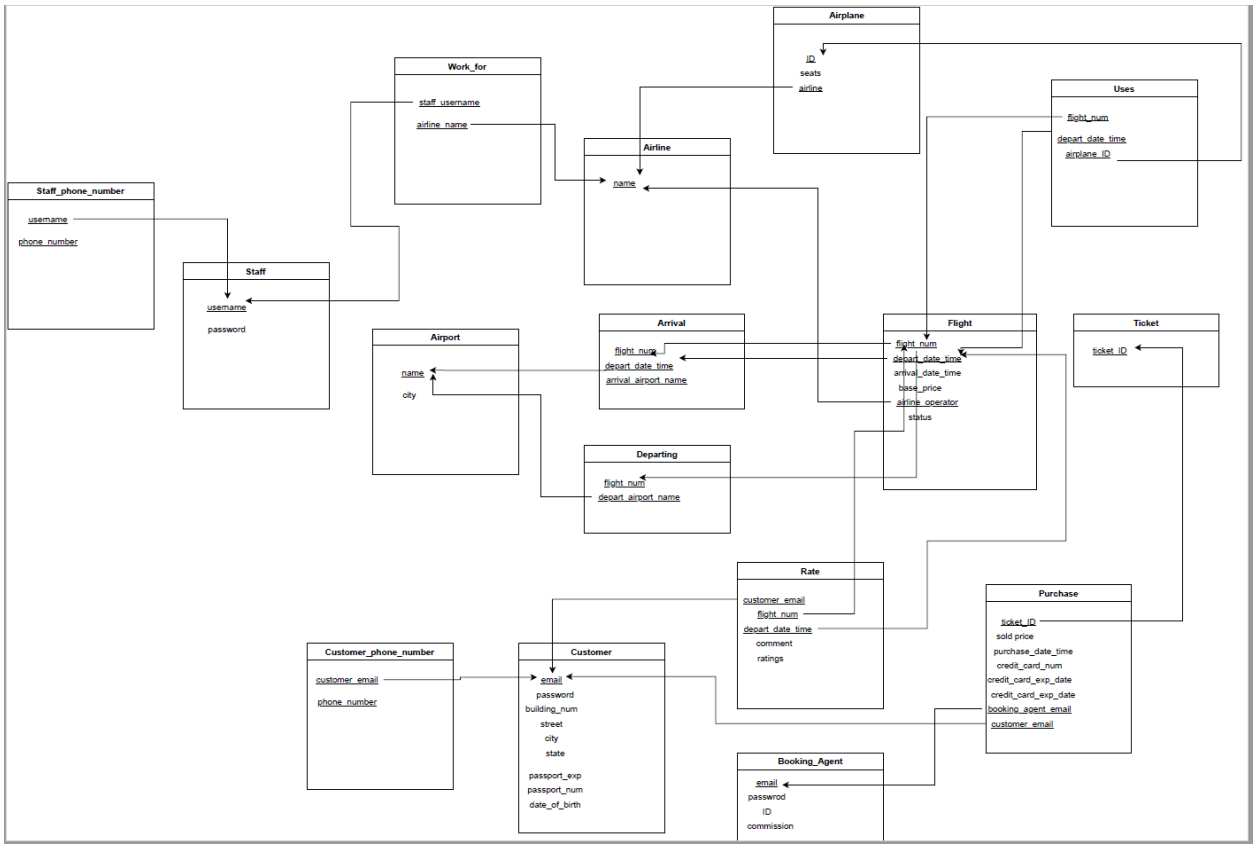
Airline staff can create airports, and airlines as well as change the status of flights. Booking agents and customers can buy flight tickets. Customers can review flight and provide a rating and comment.

For part 1, the ER diagram was successfully created, however, an update was needed for part 2 due to the fact that the ER diagram was incomplete. Professor Dey provided guidance on the proper ER diagram so the current ER diagram was modified to match the correct one.

Part 2

For part 2 of the project, a relational database design was completed with relational schema, table definition in SQL, as well queries. The relational database design was based on the updated ER diagram.

The ER diagram was the same one shown in class by Professor Dey, with the updated ER diagram, the relational schema was designed in the picture below:



SQL was also written for part 2, the code for creating the tables based off the relational schema is below:

Tables:

```
CREATE TABLE Airline (
    name VARCHAR(255) PRIMARY KEY
);

CREATE TABLE Staff (
```

```
    username VARCHAR(255) PRIMARY KEY,
    password VARCHAR(30) NOT NULL,
    airline VARCHAR(255) NOT NULL,
    FOREIGN KEY (airline) REFERENCES Airline(name)
);

CREATE TABLE Works_for(
    staff_username VARCHAR(255),
    airline VARCHAR(255),
    FOREIGN KEY (airline) REFERENCES Airline(name),
    FOREIGN KEY (staff_username) REFERENCES Staff(username),
    PRIMARY KEY (airline, staff_username)
);

CREATE TABLE Staff_phone_number (
    number CHAR(10),
    username VARCHAR(255),
    FOREIGN KEY (username) REFERENCES Staff(username),
    PRIMARY KEY(username, number)
);

CREATE TABLE Airplane (
    ID int PRIMARY KEY,
    seat_capacity int,
    airline VARCHAR(255) NOT NULL,
    FOREIGN KEY (airline) REFERENCES Airline(name)
);

CREATE TABLE Airport (
    name VARCHAR(255) PRIMARY KEY,
    city VARCHAR(255)
);

CREATE TABLE Flight (
    flight_num int UNIQUE,
    depart_date_time DATETIME UNIQUE,
    arrival_date_time DATETIME,
    base_price int,
    status int,
    airline VARCHAR(255) NOT NULL,
```

```
    PRIMARY KEY(flight_num, depart_date_time),
    FOREIGN KEY (airline) REFERENCES Airline(name)
);

CREATE TABLE Uses (
    flight_num int,
    depart_date_time DATETIME,
    airplane_id int,
    FOREIGN KEY (airplane_id) REFERENCES Airplane(ID),
        FOREIGN KEY (flight_num, depart_date_time) REFERENCES Flight(flight_num,
depart_date_time),
    PRIMARY KEY (flight_num, depart_date_time, airplane_id)
);


CREATE TABLE Arrival_airport (
    flight_num int,
    arrival_airport_name VARCHAR(255),
    depart_date_time DATETIME,
    FOREIGN KEY (flight_num, depart_date_time) REFERENCES Flight(flight_num,
depart_date_time),
    FOREIGN KEY (arrival_airport_name) REFERENCES Airport(name),
    PRIMARY KEY (flight_num, depart_date_time, arrival_airport_name)
);

CREATE TABLE Departure_airport (
    flight_num int,
    depart_airport_name VARCHAR(255),
    depart_date_time DATETIME,
    FOREIGN KEY (flight_num, depart_date_time) REFERENCES Flight(flight_num,
depart_date_time),
    FOREIGN KEY (depart_airport_name) REFERENCES Airport(name),
    PRIMARY KEY(flight_num, depart_date_time, depart_airport_name)
);

CREATE TABLE Customer (
    email VARCHAR(255) PRIMARY KEY,
    password VARCHAR(30) NOT NULL,
    building_num int,
    street VARCHAR(255),
```

```
    city VARCHAR(255),
    state VARCHAR(255),
    passport_num int,
    passport_exp DATE,
    date_of_birth DATE
);

CREATE TABLE Customer_phone_number (
    number CHAR(10),
    email VARCHAR(255),
    FOREIGN KEY (email) REFERENCES Customer(email),
    PRIMARY KEY(email, number)
);

CREATE TABLE Booking_agent (
    email VARCHAR(255) PRIMARY KEY,
    password VARCHAR(30) NOT NULL,
    ID int UNIQUE
);

CREATE TABLE Ticket (
    ticket_id int PRIMARY KEY
);

CREATE TABLE Has (
    flight_num int,
    ticket_id int,
    FOREIGN KEY (flight_num) REFERENCES Flight(flight_num),
    FOREIGN KEY (ticket_id) REFERENCES Ticket(ticket_id),
    PRIMARY KEY (flight_num, ticket_id)
);

CREATE TABLE Purchases (
    sold_price int,
    purchase_date_time DATETIME,
    credit_card_num CHAR(16),
    credit_card_exp_date DATE,
    credit_card_name VARCHAR(255),
    ticket_id int NOT NULL,
    customer_email VARCHAR(255),
```

```
    booking_agent_id int NULL,
    FOREIGN KEY (booking_agent_id) REFERENCES Booking_agent(ID),
    FOREIGN KEY (customer_email) REFERENCES Customer(email),
    FOREIGN KEY (ticket_id) REFERENCES Ticket(ticket_id),
    PRIMARY KEY(ticket_id)
);

CREATE TABLE Flight_rating (
        comment VARCHAR(255),
    rating int,
    customer_email VARCHAR(255) NOT NULL,
    flight_num int NOT NULL,
    flight_depart_date_time DATETIME NOT NULL,
    FOREIGN KEY (customer_email) REFERENCES Customer(email),
    FOREIGN KEY (flight_num, flight_depart_date_time) REFERENCES Flight(flight_num,
depart_date_time),
    PRIMARY KEY(flight_num, flight_depart_date_time, customer_email)
);
```

The code for SQL queries to test the code to see if the database returns the correct information:
Queries:

```
SELECT * FROM flight where depart_date_time > NOW();

SELECT * FROM flight where status=1;

SELECT FROM Airplane WHERE airline="China Eastern";

select credit_card_name FROM Purcahses WHERE booking_agent_id IS NOT NULL;
SELECT credit_card_name FROM Purchases;

SELECT * FROM Airplanne WHERE airline="China Eastern";
```

The initial information inserted into the database to provide it data that can be used for queries:
Insert Into Tables:

```
INSERT INTO AIRLINE VALUES ("China Eastern");
INSERT INTO Airport VALUES ("JFK", "NYC"), ("PVG", "Shanghai");
INSERT INTO Customer VALUES ("majao@nyu.edu", "password123", 370, "Jay St", "NYC",
"NY", 123, "2015-11-12 10:20:19", "2005-03-22 02:00:00");
```

```
INSERT INTO Customer VALUES ("jlin@nyu.edu", "password123", 370, "Jay St", "NYC",
"NY", 123, "2015-11-13 10:20:19", "2005-03-27 02:00:00");
INSERT INTO Booking_agent VALUES ("ratan@nyu.edu", "password123", 456);
INSERT INTO Airplane VALUES (1234, 100, (SELECT name FROM Airline WHERE
name="China Eastern"));
INSERT INTO Airplane VALUES (4567, 78, (SELECT name FROM Airline WHERE
name="China Eastern"));
INSERT INTO Staff VALUES ("AshKetchup", "password123", (SELECT name FROM Airline
WHERE name="China Eastern"));
INSERT INTO Flight VALUES (1, "2003-08-14 18:08:04", "2003-08-31 18:08:04", 500, 0,
(SELECT name FROM Airline WHERE name="China Eastern"));
INSERT INTO Flight VALUES (1, "2003-08-14 18:08:04", "2003-08-21 18:08:04", 5399, 1,
(SELECT name FROM Airline WHERE name="China Eastern"));

INSERT INTO TICKET VALUES (789);
INSERT INTO Ticket VALUES (676);
INSERT INTO PURCHASES VALUES (7500, "2021-04-30 18:08:04", 1231233456,
"2027-08-14", "Mohammed Ajao", (SELECT ticket_id FROM Ticket WHERE ticket_id=789),
(SELECT email FROM Customer WHERE email="majao@nyu.edu"));
INSERT INTO PURCHASES VALUES (500, "2003-07-14 18:08:04", 1231233456,
"2005-08-14", "Jonathan Lin", (SELECT ticket_id FROM Ticket WHERE ticket_id=676),
(SELECT email FROM Customer WHERE email="jlin@nyu.edu"));
```

Part 3: Overview

For part 3, the goal is to combine everything and develop a web application for the Air Ticket Reservation System. For the web application, it was decided that we will use javascript and NodeJS for designing and implementing the web application. The client side uses vuejS for the implementation. Adonis for the middleware and Lucid ORM are also utilized to aid the development.

The basic logic flow of the application is that there are client side verification checks that follow server side validations. The API verifies the required data that the user is requesting is present and if it is, the data is presented. The web application is not mobile compatible. Airlines need to be implemented manually by the developer to restrict the airline staff will be able to register to “official” airlines registered on the system programmed by the developer, that way the airlines will stay consistent as staff access to airline creation/deletion would be chaotic for the application.

The code will not be shown in the report due to the lengthiness. However, screenshots of some of the use cases, explanation of the code, and queries will be listed.

Project 3: Files
Some calls or titles or files may not be explained as their title or the context around them is deemed to suffice for their usage.

Client

- Public > index.html
    - File is responsible for the root of the single page application (SPA) and initializes static content like scripts or base stylesheets.
- Src > main.js
    - Hooks Vue application together. Launch point
- Src > Axios.js
    - Config for axios.js
- Src > App.vue
    - Main content of the VueJS application and root of the logic in terms of visible content. Imports navbar content
- Src > Components > Navbar.vue
    - Handles display logic for navigation on the site and hides elements to restrict pages from users.
- Src > router > index.js
    - Handles navigation logic and restricts access to pages. Determines page urls and components to load at those urls for the SPA
- Src > Store
    - Modules > User.js
        - Handles state using Vuex for the user in the client
    - Handles state of the application. Mainly client-side authentication
- Views
    - BADashboard.vue
        - Booking Agent Dashboard. Allows agents to see their commission, customers, customer/booked flights, and top customers.
        - API Calls:
            - GET: /agent/acf
                - Gets booked flights. ACF = Agent-customer-flights
            - GET: /purchases/top-customers/
            - POST: /purchases/agent-commission
    - CustomerProfile.vue

        - Customers can see what they spent in a range and their spending for the year initially. On a bar chart, they see their monthly spending for the past 6 months.
        - API Calls:
            - POST: /purchases/get-spending
            - POST: /purchases/get-monthly-spending
- FlightDisplay.vue
    - Displays all basic flight data. Ratings, comments, allows for comments, allows for payment, allows for hiring agents, allows for updating flights, and lists customers who bought tickets, allows agents to buy tickets for customers
    - API Calls:
        - POST: /ratings/create
        - POST: /flights/update
        - POST: /purchases/create
        - GET: /flights/
        - POST: /ratings/flight
        - POST: /flights/getCustomers
        - GET: /agents
        - POST: /purchases/getClients
            - Gets clients of an agent to buy for.
- Flights.vue
    - Allows users to search flights based on times, airports, cities
    - API Calls:
        - POST: /flights/search
        - GET: /airports
        - GET: /flights
- Login.vue
    - POST: /login
- Register.vue
    - POST: /register
- StaffDashboard.vue
    - Allows staff to see their revenue, flights bought, create flights, create airports, create airplanes, see tickets bought in a time range, see tickets sold in a range, see a revenue chart of their source gains, top clients and agents in different ranges, top locations
        - POST: /purchases/revenue-data
            - Gets revenue data
        - POST: /purchases/get-tickets-thru
            - Gets tickets in a range ordered by months and year

- POST: /flights/flight-cust-air
    - Gets flights by customer and airline
- POST: /flights/create
- GET: /staff/getAirline
- POST: /airports/create
- GET: /airports/get-top-airports/
- GET: /agents/top-agents/
- GET: /airports
- GET: /staff/getAirplanes
- GET: /staff/getTopCustomers/
- GET: /purchases/ticket-report/

Server

- Database > Migrations Folder
    - This defines the tables to be created. Essentially CREATE SQL statements in an ORM form. The migrations are used to rollback mistakes in code/development without dropping the entire database and tracking changes.
- Start > Routes.ts
    - Defines routes for the API and links the URLs with the controllers that handle the logic. Also defines middleware to use with the controllers
- App > Models
    - Defines each table as a model for the ORM to used for querying. Rather than raw SQL queries, the ORM allows for easier querying and predefined relationships to be instantiated into the response data
- App > Controllers
    - Each controllers handles the creation, showcase of an individual, updating, and querying of the primary model they are related or named after. Here are controllers that have functions that stray from this commonality
    - AuthController
        - Handles login, logout and registration logic
    - BookingAgentsController
        - Gets top agents based on dates and clients
    - FlightsController
        - Handles search for flights, getting flights by airline, getting customers of flights
    - PurchasesController

        - Gets people who purchased tickets, tickets sold in range, monthly ticket report, revenue sources for airlines, top clients, commission data, spending data
    - StaffController
        - Get airline from staff, get airplanes from staff, get staff's airline's top customers

Conclusion

In conclusion the project objectives were completed successfully. The Air Ticket Reservation System worked successfully with the given project descriptions and objectives. All parts of the project progression were also completed successfully. We were able to learn a lot about designing and implementing relational databases. We were able to use what we learned in the classroom and make it into something that we can use. Overall, we are very happy with the product we were able to create.