Project Overview: The course project for this semester is online Air Ticket Reservation System. There will be two types of users of this system – Customers, and Airline Staff (Administrator). Using this system, customers can search for flights (one way or round trip), purchase flights ticket, view their future flight status or see their past flights etc. Airline Staff will add new airplanes, create new flights, and update flight status. In general, this will be a simple air ticket reservation system.

3 Parts of the Project:

Part 1. Create an ER diagram based on the Project description below. You may work with up to 2 partners or individually for this part.
Part 2. Create a relational database design (relational Schema, write table definitions in SQL, write some queries etc.) based on ER diagram. You may work with up to 2 partners or individually for this part.
Part 3. Develop a web application for the system. You may work with up to 2 partners or individually for this part.

Project Description

There are several airports (Airport), each consisting of a unique code, a name, a city, a country, and an airport type (domestic/international/both).

There are several airlines (Airline), each with a unique name. Each airline owns several airplanes. An airplane (Airplane) consists of the airline that owns it, a unique identification number within that airline, and the number of seats on the airplane, a manufacturing company of that airplane, age of the airplane.

Each airline operates flights (Flight), which consist of the airline operating the flight, a flight number, departure airport, departure date and time, arrival airport, arrival date and time, a base price, and the identification number of the airplane for the flight. Each flight is identifiable using flight number and departure date and time together within that airline.

A ticket (Ticket) can be purchased for a flight by a customer, and will consist of the customer's email address, travel class (it could be either first class, business class, or economy class), the airline name, the flight number, sold_price (may be different from base price of the flight), payment information (including card type - credit/debit, card number, name on card, expiration date), purchase date and time. Each ticket will have a ticket ID number which is unique in this System.

Anyone (including users not signed in) can see flights (future flights) based on the source airport, destination airport, source city, or destination city, departure date for one way (departure and return dates for round trip). Additionally, anyone can see the status (delayed, on time, or

canceled) of the flight based on an airline and flight number combination and arrival or departure time.

There are two types of users for this system: Customer, and Airline Staff.

Customer:

Each Customer has a name, email, password, address (composite attribute consisting of building_number, street, city, state), phone_number, passport_number, passport_expiration, passport_country, and date_of_birth. Each Customer's email is unique, and they will sign into the system using their email address and password

Customers must be logged in to purchase a flight ticket

Customers can purchase a ticket for a flight as long as there is still room on the plane. This is based on the number of tickets already booked for the flight and the seating capacity of the airplane assigned to the flight and customer needs to pay the associated price for that flight. Ticket price of a flight will be determined based on two factors – minimum/base price as set by the airline and additional price which will depend on demand of that flight. If 75% of the capacities is already booked/reserved for that flight, extra 25% will be added with the minimum/base price. Customer can buy tickets using either credit card or debit card. We want to store card information (card number and expiration date and name on the card but not the security code) along with purchased date, time.

Customers will be able to see their future flights or previous flights taken for the airline they logged in.

Customers will be able to rate and comment on their previous flights taken for the airline they logged in

Airline Staff:

Each Airline Staff has a unique username, a password, a first name, a last name, a date of birth, may have more than one phone number, and the airline name that they work for. One Airline Staff only works for one airline

Airline Staff will be able to add new airplanes into the system for the airline they work for

Airline Staff will set flight statuses in the system.

Each Airline Staff can create new flights only for the particular airline that they work for by inserting all necessary information and will set the ticket base price for flight. They will also be able to see all on- time, future, and previous flights for the airline that they work for, as well as a list of passengers for the flights.

In addition, Airline Staff will be able to see a list of all flights a particular Customer has taken only on that particular airline.

Airline Staff will be able to see each flight's average ratings and all the comments and ratings of that flight given by the customers.

Airline Staff will also be able to see the most frequent customer within the last year, see the number of tickets sold each month, see the total amount of revenue earned etc.

Airline Staff can query for how many flights get delayed/on-time etc.

What You Should Do for Part 1:

Design an ER diagram for online Air Ticket Reservation System described above. Draw the ER diagram neatly. You may draw it by hand or using a design tool. Design tool preferred.

When you do this, think about: which information should be represented as attributes, which as entity sets or relationship sets? Are any of the entity sets weak entity sets? If so, what is the identifying strong entity set? What is the primary keys (or discriminant) of each entity set? What are the cardinality constraints on the relationship sets? Do you need to use ternary relationship sets or aggregation? You may find it useful to read the Project Part3 descriptions but not required

What You Should Do for Part 2

1. Following the techniques we studied, derive a relational schema diagram from the Part 1's ER diagram. Remember to underline primary keys and use arrows from the referencing schema to the referenced schema to indicate foreign key constraints

2. Write and execute SQL CREATE TABLE statements to create the tables. Choose reasonable types for the attributes.

3. Write and execute INSERT statements to insert data representing one airline's air ticket reservation system. As for example, you can insert data in the appropriate tables as follows or you can insert data for another airline or your own make up airline:

a. One Airline name "China Eastern"
b. At least Two airports named "JFK" in NYC and "PVG" in Shanghai.
c. Insert at least three customers with appropriate names and other attributes.
d. Insert at least three airplanes.
e. Insert At least One airline Staff working for China Eastern
f. Insert several flights with on-time, and delayed statuses
g. Insert some tickets (each travel class should have at least one ticket) for corresponding flights

4. Write SQL queries for executing following queries and show the results in your file (SQL query and corresponding answers):

a. Show all the future flights in the system.

b. Show all of the delayed flights in the system.

c. Show the customer names who bought the tickets.

d. Show all the airplanes owned by the airline (such as "China Eastern") You may find it useful to read the Project Part3 descriptions but not required.

What You Should Do for Part 3

In Part 3, you'll implement Air Ticket Reservation System as a web-based application. You must use the table definitions that you created for part 2 (derived from the E-R diagram) unless you need to make some small additions/modifications to support your additional features. If you do modify the table definitions, you will be responsible for translating the test data/test scenarios (in case we provide) so that it matches your table definitions.

REQUIRED Application Use Cases (aka features):

Home page when not logged-in:

When the user is not logged-in, the following cases should be available in the home page:

1. View Public Info: All users, whether logged in or not, can

a. Search for future flights based on source city/airport name, destination city/airport name, departure date for one way (departure and return dates for round trip).

b. Will be able to see the flights status based on airline name, flight number, arrival/departure date.

2. Register: 2 types of user registrations (Customer, and Airline Staff) option via forms

3. Login: 2 types of user login (Customer, and Airline Staff). Users enters their username (email address will be used as username) - x, and password - y, via forms on login page. This data is sent as POST parameters to the login-authentication component, which checks whether there is a tuple in the corresponding user's table with username=x and the password = md5(y) :

A. If so, login is successful. A session is initiated with the member's username stored as a session variable. Optionally, you can store other session variables. Control is redirected to a component that displays the user's home page.

B. If not, login is unsuccessful. A message is displayed indicating this to the user

Once a user has logged in, reservation system should display his/her home page according to user's role. Also, after other actions or sequences of related actions, are executed, control will return to component that displays the home page. The home page should display an error message if the previous action was not successful.

Some mechanism for the user to choose the use case he/she wants to execute: You may choose to provide links to other URLs that will present the interfaces for other use cases, or you may include those interfaces directly on the home page

Any other information you'd like to include: For example, you might want to show customer's future flights information on the customer's home page, or you may prefer to just show them when he/she does some of the following use cases.

Customer use cases: After logging in successfully a user(customer) may do any of the following use cases:

1. View My flights: Provide various ways for the user to see flights information which he/she purchased. The default should be shown for future flights. Optionally you may include a way for the user to specify a range of dates, specify destination and/or source airport name or city name etc.
2. Search for flights: Search for future flights (one way or round trip) based on source city/airport name, destination city/airport name, dates (departure or return).
3. Purchase tickets: Customer chooses a flight and purchase ticket for this flight, providing all the needed data, via forms. You may find it easier to implement this along with a use case to search for flights.
4. Cancel Trip: Customer chooses a purchased ticket for a flight that will take place more than 24 hours in the future and cancel the purchase. After cancellation, the ticket will no longer belong to the customer. The ticket will be available again in the system and purchasable by other customers.
5. Give Ratings and Comment on previous flights: Customer will be able to rate and comment on their previous flights (for which he/she purchased tickets and already took that flight) for the airline they logged in.
6. Track My Spending: Default view will be total amount of money spent in the past year and a bar chart/table showing month wise money spent for last 6 months. He/she will also have option to specify a range of dates to view total amount of money spent within that range and a bar chart/table showing month wise money spent within that range.
7. Logout: The session is destroyed and a "goodbye" page or the login page is displayed.

Airline Staff use cases: After logging in successfully an airline staff may do any of the following use cases:

1. View flights: Defaults will be showing all the future flights operated by the airline he/she works for the next 30 days. He/she will be able to see all the current/future/past flights operated by the airline he/she works for based range of dates, source/destination airports/city etc. He/she will be able to see all the customers of a particular flight.
2. Create new flights: He or she creates a new flight, providing all the needed data, via forms. The application should prevent unauthorized users from doing this action. Defaults will be showing all the future flights operated by the airline he/she works for the next 30 days.
3. Change Status of flights: He or she changes a flight status (from on-time to delayed or vice versa) via forms.

4. Add airplane in the system: He or she adds a new airplane, providing all the needed data, via forms. The application should prevent unauthorized users from doing this action. In the confirmation page, she/he will be able to see all the airplanes owned by the airline he/she works for.

5. Add new airport in the system: He or she adds a new airport, providing all the needed data, via forms. The application should prevent unauthorized users from doing this action.

6. View flight ratings: Airline Staff will be able to see each flight's average ratings and all the comments and ratings of that flight given by the customers.

7. View frequent customers: Airline Staff will also be able to see the most frequent customer within the last year. In addition, Airline Staff will be able to see a list of all flights a particular Customer has taken only on that particular airline.

8. View reports: Total amounts of ticket sold based on range of dates/last year/last month etc. Month wise tickets sold in a bar chart/table.

9. View Earned Revenue: Show total amount of revenue earned from ticket sales in the last month and last year.

10. View Earned Revenue by travel class: Show total amount of revenue earned from ticket sales by each travel class.

11. View Top destinations: Find the top 3 most popular destinations for last 3 months and last year (based on tickets already sold).

12. Logout: The session is destroyed and a "goodbye" page or the login page is displayed

Additional Requirements: You should implement Air ticket reservation system as a web-based application. If you want to use a DBMS other than MySQL, SQLserver, Oracle, MongoDB or to use a programming language other than Python/Flask, Java/JDBC/Servlets, PHP, C#, node.js, or JavaScript please check with me first. You will need to bring the host computer to the demo/test session at the end of the semester or make the application available remotely over the web

Enforcing complex constraints: Your air ticket reservation system implementation should prevent users from doing actions they are not allowed to do. For example, system should prevent users who are not authorized to do so from adding flight information. This should be done by querying the database to check whether the user is an airline staff or not before allowing him to create the flight. You may also use the interface to help the user to avoid violating the constraints. However, you should not rely solely on client-side interactions to enforce the constraint, since a user could bypass the client-side interface and send malicious http requests

Session Management: When a user logs in, a session should be initiated; relevant session variables should be stored. When the member logs out, the session should be terminated. Each component executed after the login component should authenticate the session and retrieve the user's pid from a stored session variable. (If you're using Python/Flask, you can follow the model in the Flask examples presented to do this.)

You must use prepared statements if your programming language supports them. (This is the style used in Flask; if you're using PHP, use the MySQLi interface; if you're using Java/JDBC, use the PreparedStmt class.) If your programming language does not support prepared statements, Free form inputs (i.e., text entered through text boxes) that is incorporated into SQL statements should be validated or cleaned to prevent SQL injection.

You should take measures to prevent cross-site scripting vulnerabilities, such as passing any text that comes from users through htmlspecialchars or some such function, before incorporating it into the html that air ticket reservation system produces.

The user interface should be usable, but it does not need to be fancy. For each type of users, you need to implement different home pages where you only show relevant use cases for that type of users and you should not show/combine all the use cases in one page.

Tips and suggestions for Part 3:

1. Use the tables you defined/created in Part 2. (If you defined extra tables or attributes, for additional features, merge them into Part 2.)
2. Before you start coding, think about what each component will do. If there are commonalities among many of the use cases, think about how you will modularize your code.
3. Implement the web-based application for Air Ticket Reservation System. For each component: a. Using some sample data, write the queries executed by the component, and test them. b. Write the application code for the component.
4. Test the component with additional values, including values that are not valid input.
5. Implement and test the components one at a time. When a component is ready, add links to it to your home page (or enhance the home page with the interface of the new component.) You will get partial credit if some of your features work, even if others have not been implemented.
6. For testing/debugging you will probably find it useful to execute your SQL queries directly through PHPmyAdmin (if you use MySQL) or using your database provided client program, before incorporating them in your application code

Complete Final Project Hand in instructions:

You will hand in:

- Your source codes. (Details about whether to zip it, etc., will be provided.)
- A list of the files in your application and what's in each file. (E.g. "homepage.phpscript to generate home page".)
- A separate file that lists all of the use cases and the queries executed by them (with brief explanation). This should be well organized and readable. It should be detailed enough to give readers a good idea of how your application works, without making them dig through all the code