Shanghai New York University

Project for Databases CSCI-SHU 213

Objective:

The objective of this course project is to provide a realistic experience in the design process of a relational database and corresponding applications. We will focus on conceptual design, logical design, implementation, operation, maintenance of a relational database. We will also implement an associated web based application to communicate with the database (retrieve information, store information etc).

Project Overview:

The course project for this semester is online Air Ticket Reservation System. Using this system, customers can search for flights, purchase flights ticket, view their upcoming flight status or see their past flights etc. There will be three types of users of this system - Customers, Booking Agents and Airline Staff (Administrator). Booking Agents will book flights for other Customers, can get a fixed commission. They can view their monthly reports and get total commission. Airline Staff will add new airplanes, create new flights, and update flight status. In general, this will be simple air ticket reservation system.

3 Parts of the Project:

1. **WORK INDIVIDUALLY** Create an ER diagram based on the description below.

   (Already completed)

2. **WORK INDIVIDUALLY** Create a relational database design (relational Schema, write table definitions in SQL, write some queries etc) based on ER diagram.

   (Already completed)

3. **WORK IN A TEAM OF 2 PERSONS:** Develop a web application for the system.
   You may work with a partner or individually for this part.

Project PART 3:

In Part 3, you'll implement Air Ticket Reservation System as a web based application. **You must use the table definitions that are derived from the E-R diagram that we posted,** 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 (if we provide) so that it matches your table definitions.

REQUIRED Application Use Cases (aka features):

1. **View Public Info:** All users, whether logged in or not, can
    a. Search for upcoming flights based on source city/airport name, destination city/airport name, date.
    b. Will be able to see the flights status based on flight number, arrival/departure date.

2. **Register**: 3 types of user registrations (Customer, Booking agent, Airline Staff) option via forms.

3. **Login: 3 types of user login (Customer, Booking agent, Airline Staff).** User 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 Person 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.

Note: In real applications, members' passwords are stored as md5/other hashes, not as plain text. This keeps the passwords more secure, in case someone is able to break into the system and see the passwords. You can perform the hash using MySQL's md5 function or a library provided with your host language.) Once a user has logged in, reservation system should display his/her home page. 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:

c. Error message if the previous action was not successful,
d. 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.
e. Any other information you'd like to include. For example, you might want to show customer's upcoming 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 showing for the upcoming 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. **Purchase tickets:** Customer chooses a flight and purchase ticket for this flight. You may find it easier to implement this along with a use case to search for flights.

3. **Search for flights:** Search for upcoming flights based on source city/airport name, destination city/airport name, date.

4. **Track My Spending:** Default view will be total amount of money spent in the past year and a bar chart 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 showing month wise money spent within that range. (Bar chart is optional, you can choose to represent the data anyway you like)

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

Booking agent use cases:

After logging in successfully a booking agent may do any of the following use cases:

1. **View My flights:** Provide various ways for the booking agents to see flights information for which he/she purchased on behalf of customers. The default should be showing for the upcoming flights. Optionally you may include a way for the user to specify a range of dates, specify destination and/or source airport name and/or city name etc to show all the flights for which he/she purchased tickets.

2. **Purchase tickets**: Booking agent chooses a flight and purchases tickets for other customers giving customer information. You may find it easier to implement this along with a use case to search for flights. Notice that as described in the previous assignments, the booking agent may only purchase tickets from airlines they work for.

3. **Search for flights:** Search for upcoming flights based on source city/airport name, destination city/airport name, date.

4. **View my commission**: Default view will be total amount of commission received in the past 30 days and the average commission he/she received per ticket booked in the past 30 days and total number of tickets sold by him in the past 30 days. He/she will also have option to specify a range of dates to view total amount of commission received and total numbers of tickets sold.

5. **View Top Customers**: Top 5 customers based on number of tickets bought from the booking agent in the past 6 months and top 5 customers based on amount of commission received in the last year. Show a bar chart showing each of these 5 customers in x-axis and number of tickets bought in y-axis. Show another bar chart showing each of these 5 customers in x-axis and amount commission received in y- axis. (Again, UI/bar chart is optional, the important factor is that you are able to retrieve and display the data.)

6. **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 My flights:** Defaults will be showing all the upcoming 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 or staffs without "Admin" permission from doing this action. Defaults will be showing all the upcoming 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 upcoming to in progress, in progress to delayed etc) via forms. The application should prevent unauthorized users or staffs without "Operator" permission from doing this action.

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 or staffs without "Admin" permission 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 or staffs without "Admin" permission from doing this action. (Additional requirement: Airline Staff with "Admin" permission will be able to add new airports into the system for the airline they work for.)

6. **View all the booking agents:** Top 5 booking agents based on number of tickets sales for the past month and past year. Top 5 booking agents based on the amount of commission received for the last year.

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.

9. **Comparison of Revenue earned:** Draw a pie chart for showing total amount of revenue earned from direct sales (when customer bought tickets without using a booking agent) and total amount of revenue earned from indirect sales (when customer bought tickets using booking agents) in the last month and last year.

10. **View Top destinations:** Find the top 3 most popular destinations for last 3 months and last year.

11. **Grant new permissions:** Grant new permissions to other staffs in the same airline. The application should prevent unauthorized users or staffs without "Admin" permission from doing this action. Initially there should be a staff with "Admin" permission in the database for each airline. Airline staffs registered through the application DO NOT have any permissions at beginning. (Additional requirement: Airline Staff with "Admin" permission will be able to grant new permissions to staffs in the same airline.)

12. **Add booking agents:** Add booking agents that can work for this airline, providing their email address. The application should prevent unauthorized users or staffs without "Admin" permission from doing this action. A booking agent cannot work for any airline (thus cannot purchase tickets) until any staff add then through this action. (Additional requirement: Airline Staffs with "Admin" permission will be able to add booking agents that can work for their airline.)

13. **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. 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.

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. 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.

The user interface should be usable, but it does not need to be fancy.

For testing/debugging you will probably find it useful to execute your SQL queries directly through PHPmyAdmin or using your database provided client program, before incorporating them in your application code.

What you should do

1. Study my solution to Part 2. Drop the tables you defined in Part 2 then install the ones given in my solution to Part 2. (If you defined extra tables or attributes, for additional features, merge them into my solutions to 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. 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. Suggestion: 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.

Complete Final Project Hand in instructions: You will hand in:

- Your source code.
- 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.
- For team projects: A summary of who did what.
- Shortly before the project is due, I'll ask you to sign up for a time slot to demonstrate your project to me. This demo will mainly consist of running application on a bunch of test cases. We may also ask you to explain some of your code. No formal presentation will be expected.

The total project grade will be 30% of your course grade. Part 1 counts for about 20% of the project grade. Part 2 counts for about 20% of the project grade. Part 3 counts for about 60% of the project grade. There may also be a quiz or exam question(s) based on the project.

Instructions for working with a Team

You may work alone or with one teammate.

Teams will be required to submit a work plan, indicating who will do what, and will be required to submit an evaluation at the end. Note that each teammate is expected to contribute roughly equally to each aspect of the project and each teammate is responsible for understanding the entire system. Normally all team members will receive the same grade, but I may deduct points from individuals who are not pulling their weight on a team.