**REQUIRED Application Use Cases (aka features):**1. **View Public Info**

By default, upcoming flights are shown.

query = "select \* from flight where status = 'Upcoming'"

On clicking specific search, users can enter criteria and input value through forms. The query then becomes:

query = "SELECT \* FROM flight WHERE status = 'Upcoming' and flight.{} = '{}'".format(criteria,input\_value)

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

On clicking register on the front page, the customer is brought to a page with a dropdown selection form. The customer will select the type of user he wants to register for. Depending on the type he selects, he will be brought to the appropriate registration page. After filling in the form, the details will be inserted into the table.

Agent:

sql = '''INSERT INTO booking\_agent VALUES ('{}','{}','{}')'''.format(email, password1, agentID)

Staff:

Need to include a separate insertion into permission that links the agent permission type to None. This allows the staff admin to change his permission.

sql = '''INSERT INTO airline\_staff VALUES ('{}','{}','{}','{}','{}','{}')'''.format(username,password1,first\_name,last\_name,dob,airlineName)

sql2 = '''INSERT INTO permission VALUES ('{}', 'None')'''.format(username)

Customer:

sql = '''INSERT INTO customer VALUES ('{}','{}','{}','{}','{}','{}','{}','{}','{}','{}','{}','{}')'''.format(email, first\_name,

One thing we also implemented was a check for duplicate user ID.

query = "SELECT \* FROM customer WHERE email = '{}'".format(email)

for customer, and similar logic for the other user cases to check the email/username is not taken. This allows us to ensure that no duplicates are entered, and SQL insertion errors are avoided.

|  |
| --- |
|  |
|  |  |

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.

First we display the login page, and then get the data via forms. Then we perform:

query = "SELECT email, password FROM customer WHERE email = '{}'".format(email)

If there is no data, then the email does not exist and we return a message that the user is not found.

If a user is found, we check against the password. If the password is correct, we log the user in. If it is wrong, we return the error message that the password is wrong.

This is used for booking agent and staff as well:

Agent: query = **"SELECT email, password FROM booking\_agent WHERE email = '{}'"**.format(email)

Staff: query = **"SELECT username, password FROM airline\_staff WHERE username = '{}'"**.format(username)

**Customer use cases:**

**View My flights:** Provide various ways for the user to see flights information which he/she purchased. The defaultshouldbeshowingfortheupcomingflights.**Optionally**youmayincludeawayfortheuser tospecifya range of dates, specify destination and/or source airport name or city name etc.

The default case is:

query = "SELECT distinct \* FROM flight natural join ticket natural join purchases WHERE purchases.customer\_email = '{}' and flight.status = 'Upcoming' ".format(user)

To specify a range of dates, there is a specific search option. The user selects from a dropdown the criteria (dates, destination, source airport etc) and the desired value.

Then the specific case:

criteria = request.form.get('criteria')

input\_value = request.form.get('input\_value')

query = "SELECT distinct \* FROM flight natural join ticket natural join purchases WHERE purchases.customer\_email = '{}' and flight.{} = '{}'".format(user, criteria, input\_value)

**Purchase tickets, 3. Search For Flights:** 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.

Customer presses search for flights and sees a table that displays upcoming flights. The query is:

query = "SELECT \* FROM flight WHERE status = 'Upcoming'"

Again, there is a specific search with the specifics highlighted above in “View My Flights” that the customer can use to narrow his options.

Then customers can press purchase, and the system checks if there are tickets.

"SELECT flight.flight\_num,ticket.ticket\_ID FROM (flight natural join ticket) left join purchases on ticket.ticket\_id = purchases.ticket\_id where flight.status = 'Upcoming' and ticket.flight\_num = '{}' and purchases.customer\_email is null".format(flight)

If there are tickets, we insert into the table and inform the customer his ticket number.

If not, we inform the customer that there are no tickets left.

email = session['customer']

query = '''INSERT INTO purchases VALUES ('{}','{}',null,'{}')'''.format(ticket\_ID, email, today)

**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.

Total amount of money spent from last year: We set the dates we need and query the total amount of money spent in the past year.

last\_year = date.today() + relativedelta(months=-12)

query = "SELECT sum(flight.price) FROM ticket natural join purchases natural join flight WHERE purchases.customer\_email = '{}' and purchases.purchase\_date >= '{}'".format(userPrimaryKey,last\_year)

For monthly data, we get the tickets sold in each month by initializing a month and year value. Then we query the tickets sold in that month. The default is starting today and going back 6 months.

Monthly data:

for i in range(default):

month\_x\_axis = (the\_date + relativedelta(months=-i)).month

year\_x\_axis = (the\_date + relativedelta(months=-i)).year

x\_axis = str(month\_x\_axis) + "/" + str(year\_x\_axis)

query2 = "SELECT sum(flight.price) as month\_spend FROM ticket natural join purchases natural join flight WHERE purchases.customer\_email = '{}' and MONTH(purchases.purchase\_date) = '{}' and YEAR(purchases.purchase\_date) = '{}'".format(userPrimaryKey, month\_x\_axis, year\_x\_axis)

If the customer specifies the starting date and duration, the value of the date and duration will be passed into variables “the\_date” and “default” respectively.

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

page3image100284096

**Booking agent use cases:**

**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. Optionallyyou 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.

We query the default: upcoming flights purchased on behalf of customers.

query = "SELECT \* FROM flight natural join ticket natural join purchases natural join booking\_agent WHERE booking\_agent.email = '{}' and flight status = 'Upcoming'".format(user)

If a criteria and input value are specified by clicking specific search, just as in the customer’s case, there will be a criteria and an input value variable, and the query becomes:

query = "SELECT \* FROM flight natural join ticket natural join purchases natural join booking\_agent WHERE booking\_agent.email = '{}' and flight.{} = '{}'".format(user, criteria, input\_value)

**Purchase tickets, Search for flights**: 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.

Implemented in the same way as customer, up until the purchase is registered.

However, instead of immediately registering the purchase, we first check to see if the agent is from the correct airline.

query = "SELECT airline\_name from booking\_agent\_work\_for where email = '{}'".format(session['booking\_agent'])

Then we check if there are tickets.

check = "SELECT airline\_name from ticket where ticket\_ID = '{}'".format(ticket\_ID)

Only if these two checks are passed will we commit the buying of ticket.

query = "INSERT INTO purchases VALUES ('{}','{}','{}','{}')".format(ticket\_ID, email, id, today)

**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.

**The assumption is the commission is a standard 10%**.

We initialize the\_date as today’s date, and default as 1.

query\_email = "select sum(flight.price)\*0.1 as total\_commission, sum(flight.price)/300 as average\_commission, count(\*) as salecount from purchases natural join booking\_agent natural join flight natural join ticket where booking\_agent.email = '{}' group by booking\_agent.email".format(user)

Otherwise the customer will specify a starting date and the duration in months and submit the information through forms.

query\_email = "select sum(flight.price)\*0.1 as total\_commission, sum(flight.price)/{} as average\_commission, count(\*) as salecount from purchases natural join booking\_agent natural join flight natural join ticket where booking\_agent.email = '{}' and purchases.purchase\_date > '{}' and purchases.purchase\_date < '{}' group by booking\_agent.email ".format(

number\_days,user, start\_date, the\_date)

**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.

We first initialize variables six\_months\_ago and one\_year\_ago that correspond to the respective dates. To find top 5 based on tickets bought,

query\_ticketsbought = " select customer\_email,count(\*) from purchases natural join booking\_agent where booking\_agent.email = '{}' and purchases.purchase\_date <= '{}' and purchases.purchase\_date > '{}' group by customer\_email order by count(\*) desc limit 5;".format(user, date.today(),six\_months\_ago)

To find based on commission,

query\_commission = "select purchases.customer\_email,sum(flight.price)\*0.1 as commission from purchases natural join booking\_agent natural join flight natural join ticket where booking\_agent.email = '{}' and purchases.purchase\_date > '{}' and purchases.purchase\_date <= '{}' group by purchases.customer\_email order by count(\*) desc limit 5;".format(user, one\_year\_ago, date.today())

**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:

**We have front end restrictions in that people can only see buttons corresponding to the use cases that staff with their permissions can perform. But there are also backend checks to ensure 2 layers of checks.**

query = "SELECT permission\_type FROM permission WHERE username = '{}'".format(session['staff'])

And then check using:

if permission[0] == 'Admin' or permission[0] == 'Both':

**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.

The airline that the staff belongs to is queried.

query = "SELECT airline\_name from airline\_staff where username = '{}'".format(username)

Then we search for flights.

query = "SELECT \* FROM flight WHERE airline\_name = '{}' and flight.departure\_time > '{}'".format(airline[0], one\_month\_ago)

After that, the staff can enter specifics like the customer and agent can through a form. The values are once again passed onto a “criteria” and “input value” variable.

The resulting query is:

query = "SELECT \* FROM flight WHERE airline\_name = '{}' and flight.{} = '{}'".format(airline[0], criteria, input\_value)

**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.

First thing is that the airline staff can only create flights for his airline. Therefore the airline field of flight is filled in by the backend using:

query2 = "SELECT airline\_name FROM airline\_staff WHERE username = '{}'".format(session['staff'])

The remaining variables are obtained through forms. We insert using

uery = "INSERT INTO flight VALUES ('{}','{}','{}','{}','{}','{}','{}','{}','{}')".format(airline\_name,flight\_num,departure\_airport,departure\_time,arrival\_airport,arrival\_time,price,status,airplane\_ID)

One important aspect is that we needed to generate the appropriate number of seats using a function generate\_seats.

First we find out the number of seats.

query = "SELECT airplane.seats from airplane natural join flight where flight.airplane\_ID = '{}' and flight.airline\_name = '{}'".format(airplane\_ID,airline\_name)

Then we insert the appropriate number of tickets into the system.

query = "INSERT INTO ticket VALUES ('{}','{}','{}')".format(random.randint(10,999999),airline\_name,flight\_num)

**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.

Operator submits a form specifying the flight and new status.

After performing permissions check, we update the table. Airline name is retrieved via backend:

query2 = "SELECT airline\_name FROM airline\_staff WHERE username = '{}'".format(session['staff'])

Then table is updated with the remaining form values.

query = "UPDATE flight SET status = '{}' WHERE airline\_name = '{}' and flight\_num = '{}'".format(new\_status,airline\_name,flight\_number)

**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.

Again the airline\_name variable retrieved via the backend query.

query2 = "SELECT airline\_name FROM airline\_staff WHERE username = '{}'".format(session['staff'])

The remaining variables are obtained via forms.

query = "INSERT INTO airplane VALUES ('{}','{}','{}')".format(airline\_name,airplane\_ID,seats)

**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.)

We perform a request for information via forms, then we insert into the table.

query = "INSERT INTO airport VALUES ('{}','{}')".format(airport\_name,airport\_city)

**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.

Once again we check which airline the staff is from.

query = "SELECT airline\_name from airline\_staff where username = '{}'".format(username)

We find the top 5 based on ticket sales using this query:

modified\_date = date.today() + relativedelta(months=-duration)

sort\_query = "select purchases.booking\_agent\_id, sum(flight.price) as sales from ticket natural join flight natural join purchases where flight.airline\_name = '{}' and purchases.booking\_agent\_id is not null and purchases.purchase\_date > '{}' group by purchases.booking\_agent\_id order by sales desc limit 5;".format(airline[0],modified\_date)

Modified date allows us to specify from how many months ago do we start counting. For past month, we set to 1. For past year, 12.

Same for commission.

query = "select purchases.booking\_agent\_id, sum(flight.price) as sales from ticket natural join flight natural join purchases where flight.airline\_name = '{}' and purchases.booking\_agent\_id is not null and purchases.purchase\_date > '{}' group by purchases.booking\_agent\_id order by sales desc limit 5;".format(airline[0],one\_year\_ago)

But because it is only 12 months ago, we just use the variable one\_year\_ago.

**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.

This is used to query the most frequent customers, sorted accordingly from the most to the least number of tickets bought.

query = "select purchases.customer\_email, count(\*) from purchases natural join ticket where ticket.airline\_name = '{}' and purchases.purchase\_date > '{}' group by purchases.customer\_email order by count(\*) desc;".format(airline, one\_year\_ago)

By clicking on “specifics”, the staff is brought to a page showing the list of all flights.

query = "SELECT distinct \* FROM flight natural join ticket natural join purchases WHERE purchases.customer\_email = '{}' and flight.airline\_name = '{}'".format(user, airline)

**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.

Again we initialize date variables, the default showing ticket sales from a year ago.

The query is:

query\_email = "select count(\*) as number\_sold from purchases natural join flight natural join ticket where flight.airline\_name = '{}' and purchases.purchase\_date > '{}' and purchases.purchase\_date <= '{}'".format(

airline, start\_date, the\_date)

For the total tickets. The customer can specify the date he wants, modifying the start\_date variable and the end date.

Then for the monthly tickets:

query2 = "SELECT count(\*) as number\_sold FROM ticket natural join purchases natural join flight WHERE flight.airline\_name = '{}' and MONTH(purchases.purchase\_date) = '{}' and YEAR(purchases.purchase\_date) = '{}'".format(airline, month\_x\_axis, year\_x\_axis)

**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.

To find the amount earned by customer without booking agent:

query\_customer = "SELECT sum(flight.price) as sales FROM ticket natural join flight natural join purchases WHERE ticket.airline\_name = '{}' and purchases.booking\_agent\_id is null and purchases.purchase\_date > '{}'".format(airline, the\_date)

And with agent:

query\_agent = "SELECT sum(flight.price) as sales FROM ticket natural join purchases natural join flight WHERE ticket.airline\_name = '{}' and purchases.booking\_agent\_id is not null and purchases.purchase\_date > '{}'".format(airline, the\_date)

**ViewTopdestinations:**Find the top 3 most popular destinations for last 3 months and last year.

By default,

query = "select count(\*) as dest\_count, flight.arrival\_airport as dest from flight where flight.airline\_name = '{}' and flight.departure\_time > '{}' group by flight.arrival\_airport order by dest\_count desc limit 3;".format(airline, three\_months\_ago)

For one year:

query = "select count(\*) as dest\_count, flight.arrival\_airport as dest from flight where flight.airline\_name = '{}' and flight.departure\_time > '{}' group by flight.arrival\_airport order by dest\_count desc limit 3;".format(airline, one\_year\_ago)

**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.)

First we perform the second-layer check to make sure the permissions are correct.

query = "SELECT permission\_type FROM permission WHERE username = '{}'".format(session['staff'])

Then we display all the staff affiliated with the airline. We left-join to display those with no permissions.

query = "SELECT \* FROM airline\_staff left join permission on airline\_staff.username = permission.username where airline\_staff.airline\_name = '{}'".format(airline)

Then from a drop down menu, the admin selects the new permission and submits through forms. Then the update is performed.

query = "UPDATE permission SET permission\_type = '{}' WHERE username = '{}'".format(new\_status, username)

**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.)

First we perform the second-layer check to make sure the permissions are correct.

query = "SELECT permission\_type FROM permission WHERE username = '{}'".format(session['staff'])

Then we display agents with no airline.

query = "SELECT booking\_agent.\* FROM booking\_agent left join booking\_agent\_work\_for on booking\_agent.email = booking\_agent\_work\_for.email where booking\_agent\_work\_for.airline\_name is null"

With the “add this guy” button, the agent is added.

query = "INSERT INTO booking\_agent\_work\_for values ('{}','{}')".format(email,airline[0])

13. **Logout:** The session is destroyed and a “goodbye” page or the login page is displayed.