Use Cases Explanation

Databases Final Project

Connie Li, Chenyi Xu

May 20, 2022

Start Page1. **View Public Info:** All users, whether logged in or not, can

1. Search for upcoming flights based on source city/airport name, destination city/airport name, date.

SELECT airline\_name, flight\_num, departure\_airport, departure\_time, arrival\_airport, arrival\_time, price, status, airplane\_id

FROM flight, airport AS A, airport AS B

WHERE departure\_airport = A.airport\_name AND arrival\_airport = B.airport\_name

AND status = 'Upcoming' AND A.airport\_city %s AND departure\_airport %s

AND B.airport\_city %s AND arrival\_airport %s AND departure\_time %s"

% (input1, input2, input3, input4, input5)

Description:

This query allows viewers to search the upcoming flights based on any combinations of source city, departure airport, destination city, arrival airport and departure time. Blank input is also accepted as no specific requirement for this feature.

Explanation:

We joined table flight, and airport (twice) so as to match the city for departure airport and arrival airport respectively. Then, we get the table of upcoming flights that satisfies user’s requirement(s). If there’s no input, we use ‘LIKE’ so that there will be no restriction on the flights with that feature. \ and ‘’ are used to make sure that the datatypes are correct. The output is all the information of qualified flights.

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

SELECT airline\_name, flight\_num, departure\_time, arrival\_time, status

FROM flight

WHERE flight\_num %s AND departure\_time %s AND arrival\_time %s"

% (input1, input2, input3)

Description:

This query allows viewers to search for flight status based on any combinations of flight number, arrival date and departure date. Blank input is also accepted as no specific requirement for this feature. The output is all the information of qualified flights, because users can know which flight is the one that they are looking for if more than one flight meets the requirement. This might happen, since both airline name and flight number are the key attributes of a flight, but we only have the input for flight number as required. Also, arrival time and departure time in flight is accurate to the time, while the input only contains the date.

Explanation:

We basically search from the flight table with all the requirements from the user by using the where clause. If there’s no input, we use ‘LIKE’ so that there will be no restriction on the flights with that feature. \ and ‘’ are used to make sure that the datatypes are correct.

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

SELECT \* FROM customer WHERE email = %s (1)

INSERT INTO customer VALUES(%s, %s, md5(%s), %s, %s, %s, %s, %s, %s, %s, %s, %s) (2)

SELECT \* FROM booking\_agent WHERE email = %s (3)

INSERT INTO booking\_agent VALUES(%s, md5(%s), %s) (4)

SELECT \* FROM airline\_staff WHERE username = %s (5)

INSERT INTO airline\_staff VALUES(%s, md5(%s), %s, %s, %s, %s) (6)

1, 3, 5 searches according to the information customers/agents/staffs input to see if there already exists such an account. If so, an error message will be presented. If not, 2, 4, 6 will be executed to add the new information to the database, in which the password is saved after executing the md5 function.

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.

SELECT \* FROM customer WHERE email = %s and password = md5(%s)

SELECT \* FROM booking\_agent WHERE email = %s and password = md5(%s)

SELECT \* FROM airline\_staff WHERE username = %s and password = md5(%s)

These three functions select user information according to the input from the users. If there’s no result, an error message will be displayed on the webpage.

**Customer use cases:**

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

Homepage for customers:

1. **View My flights:** The homepage for customers directly displays detailed information on the flights that he/she purchased.

SELECT \* FROM flight NATURAL JOIN ticket NATURAL JOIN purchases

WHERE `status` = 'Upcoming'

AND customer\_email = %s

ORDER BY departure\_time;

1. **Additional Function: View My ticket requests:** The homepage also displays the ticket requests that he/she made and its status (pending/success/failed).

SELECT DISTINCT \* FROM ticket\_request NATURAL JOIN ticket

WHERE customer\_email = %s;

In the scroll down bar “Actions” in the menu bar on the homepage there are two available subpages:

Ticket Search/Purchase Page for customers:

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

SELECT \* FROM flight NATURAL JOIN ticket

WHERE status = 'Upcoming'

AND ticket\_id NOT IN (SELECT ticket\_id FROM purchases)

ORDER BY departure\_time;

1. **Filter by city**

SELECT \* FROM flight NATURAL JOIN ticket

WHERE status = 'Upcoming'

AND ticket\_id NOT IN (SELECT ticket\_id FROM purchases)

AND departure\_airport IN (SELECT airport\_name FROM airport WHERE airport\_city = %s)

AND arrival\_airport IN (SELECT airport\_name FROM airport WHERE airport\_city = %s)

ORDER BY departure\_time;

1. **Filter by airport**

SELECT \* FROM flight NATURAL JOIN ticket

WHERE status = 'Upcoming'

AND ticket\_id NOT IN (SELECT ticket\_id FROM purchases)

AND departure\_airport = %s

AND arrival\_airport = %s

ORDER BY departure\_time;

1. **Filter by time**

**\*The page will display all the flights that depart before the input depart time and arrive before the input arrive time\***

SELECT \* FROM flight NATURAL JOIN ticket

WHERE status = 'Upcoming'

AND ticket\_id NOT IN (SELECT ticket\_id FROM purchases)

AND departure\_time < %s

AND arrival\_time < %s

ORDER BY departure\_time;

1. **Purchase tickets:** Customer can purchase tickets by entering the ticket id and click “buy” button to finish the purchase.

INSERT INTO purchases(ticket\_id, customer\_email, booking\_agent\_id, purchase\_date)

VALUES(%s, %s, NULL, CURRENT\_DATE);

DELETE FROM ticket\_request

WHERE (ticket\_id = %s AND customer\_email = %s);

1. **Additional Function: Request tickets:** Customer can post ticket requests by entering the ticket id and click “request” button to upload the request.

INSERT INTO ticket\_request(ticket\_id, customer\_email, request\_status)

VALUES(%s, %s, 'pending');

Spending Profile Page for customers:

1. **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.

**a. Total amount of money spent in the past year**

SELECT SUM(price) spending

FROM ticket NATURAL JOIN purchases NATURAL JOIN flight

WHERE customer\_email = %s

AND purchase\_date

BETWEEN DATE\_SUB(CURRENT\_DATE(),INTERVAL 1 YEAR)

AND CURRENT\_DATE;

**b. Month wise money spent for last 6 months**

SELECT MONTH(purchase\_date) `month`, SUM(price) spending

FROM ticket NATURAL JOIN purchases NATURAL JOIN flight

WHERE customer\_email = %s

AND purchase\_date

BETWEEN DATE\_SUB(CURRENT\_DATE(),INTERVAL 6 MONTH)

AND CURRENT\_DATE

GROUP BY MONTH(purchase\_date)

ORDER BY MONTH(purchase\_date);

1. **Filter by certain time range**

SELECT SUM(price) spending

FROM ticket NATURAL JOIN purchases NATURAL JOIN flight

WHERE customer\_email = %s

AND purchase\_date

BETWEEN %s AND %s;

In the menu bar on the homepage there is a button for logout:

1. **Logout:** The session is destroyed and the general home page is displayed.

page3image14017152

**Booking agent use cases:**

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

Homepage for agents:

1. **View My flights:** The homepage for agents directly displays detailed information on the flights that he/she purchased for his/her customers.

SELECT \* FROM flight NATURAL JOIN ticket NATURAL JOIN purchases NATURAL JOIN booking\_agent

WHERE status = 'Upcoming’

AND email = %s

ORDER BY departure\_time;

In the scroll down bar “Actions” in the menu bar on the homepage there are two available subpages:

Ticket Search/Purchase Page for customers:

1. **Search for flights:** Search for upcoming flights based on source city/airport name, destination city/airport name, date. \*Agents can only view the flights under the airline they work for\*

SELECT \* FROM flight NATURAL JOIN ticket

WHERE status = 'Upcoming’

AND airline\_name IN

(SELECT airline\_name FROM booking\_agent\_work\_for WHERE email = %s)

AND ticket\_id NOT IN (SELECT ticket\_id FROM purchases)

ORDER BY departure\_time;

1. **Additional Function: View ticket requests**: Booking agent can view the ticket requests posted by customers. \*Agents may only view requests on the tickets from airlines they work for\*

SELECT DISTINCT \* FROM ticket\_request NATURAL JOIN ticket

WHERE airline\_name IN

(SELECT airline\_name FROM booking\_agent\_work\_for WHERE email = %s)

AND request\_status = 'pending';

1. **Purchase tickets**: Booking agent can purchases tickets for other customers by entering ticket id and customer email and click “book” to finish the booking. “Booking agent can only purchase tickets from airlines they work for\*

INSERT INTO purchases(ticket\_id, customer\_email, booking\_agent\_id, purchase\_date)

VALUES(%s, %s,

(SELECT booking\_agent\_id FROM booking\_agent WHERE email = %s),

CURRENT\_DATE);

1. **View my commission**: Agents can view the total amount of commission they 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.

SELECT SUM(price) total\_com, AVG(price) avg\_com, COUNT(\*) number\_tick

FROM ticket NATURAL JOIN purchases NATURAL JOIN flight NATURAL JOIN booking\_agent

WHERE email = %s

AND purchase\_date

BETWEEN DATE\_SUB(CURRENT\_DATE(),INTERVAL 30 DAY) \

AND CURRENT\_DATE;

**Filter by time range**

SELECT COALESCE(SUM(price),0) AS total\_com, COALESCE(AVG(price),0) AS avg\_com, COUNT(\*) number\_tick

FROM ticket NATURAL JOIN purchases NATURAL JOIN flight

WHERE booking\_agent\_id = %s

AND purchase\_date

BETWEEN %s AND %s;

1. **View Top Customers**: Agents can view their 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.

SELECT customer\_email, COUNT(\*) num\_of\_tick

FROM ticket NATURAL JOIN purchases NATURAL JOIN flight NATURAL JOIN booking\_agent

WHERE email = %s

AND purchase\_date

BETWEEN DATE\_SUB(CURRENT\_DATE(),INTERVAL 6 MONTH)

AND CURRENT\_DATE

GROUP BY customer\_email

ORDER BY num\_of\_tick DESC

LIMIT 5;

SELECT customer\_email, SUM(price) total\_com

FROM ticket NATURAL JOIN purchases NATURAL JOIN flight NATURAL JOIN booking\_agent

WHERE email = %s

AND purchase\_date

BETWEEN DATE\_SUB(CURRENT\_DATE(),INTERVAL 1 YEAR)

AND CURRENT\_DATE

GROUP BY customer\_email

ORDER BY total\_com DESC

LIMIT 5;

1. **Logout:** The session is destroyed and the general home page is displayed.

**Airline Staff use cases:**

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

Homepage for airline staff:

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.

SELECT A.airline\_name, flight\_num, departure\_airport, departure\_time,

arrival\_airport, arrival\_time, price, status, airplane\_id

FROM flight AS A JOIN airline\_staff AS B

WHERE A.airline\_name = B.airline\_name AND B.username = %s

AND CURRENT\_DATE <= departure\_time

AND departure\_time <= DATE\_ADD(CURRENT\_DATE(),INTERVAL 30 DAY) AND status = ‘Upcoming’

SELECT F.airline\_name, flight\_num, departure\_airport, departure\_time,

arrival\_airport, arrival\_time, price, status, airplane\_id FROM flight AS F JOIN airport AS A JOIN airport AS B JOIN airline\_staff AS S

WHERE departure\_airport = A.airport\_name AND arrival\_airport = B.airport\_name AND F.airline\_name = S.airline\_name

AND S.username = %s AND A.airport\_city %s AND departure\_airport %s

AND B.airport\_city %s AND arrival\_airport %s AND departure\_time %s AND departure\_time %s

The first query searches for all the information about the flights from the flight and airline\_staff table which satisfies airline equals to the airline that the current staff is in, the status is ‘Upcoming’, and the time period is the next 30 days, displaying as the default on the corresponding webpage.

The second query searches for all the information from the flight and airport table (twice) according to the staff’s input: range of dates, source/destination airports/city. The logic is exactly the same as the two search functions on the start page, so that the staff can enter any combination about the above features.

1. **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.

SELECT A.airline\_name, flight\_num, departure\_airport, departure\_time,

arrival\_airport, arrival\_time, price, status, airplane\_id

FROM flight AS A JOIN airline\_staff AS B

WHERE A.airline\_name = B.airline\_name AND B.username = %s

AND CURRENT\_DATE <= departure\_time

AND departure\_time <= DATE\_ADD(CURRENT\_DATE(),INTERVAL 30 DAY) AND status = ‘Upcoming’

The query searches for all the information about the flight by combining flight and airline\_staff table, and see whether the output satisfies the features that the staff enters, as well as whether the flight is from the staff’s airline with upcoming status in the next 30 days.

SELECT \* FROM permission WHERE username = %s AND permission\_type = 'Admin'

Check permission table to see whether this staff has admin permission.

SELECT \* FROM flight WHERE airline\_name = %s AND flight\_num = %s

Select the flights with key values in the input and see if the flight already exists.

INSERT INTO flight VALUES(%s, %s, %s, %s, %s, %s, %s, %s, %s)

Add the new flight to the flight table if the inputs are all valid.

If any of the errors mentioned above arise, an error message will be displayed.

1. **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.

SELECT \* FROM permission WHERE username = %s AND permission\_type = 'Operator'

Check permission table to see whether this staff has operator permission.

UPDATE flight SET status = %s WHERE airline\_name = %s AND flight\_num = %s

Update the flight status according to the input if all the input are valid.

If any of the errors mentioned above arise, an error message will be displayed.

1. **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.

SELECT \* FROM permission WHERE username = %s AND permission\_type = 'Admin'

Check permission table to see whether this staff has admin permission.

SELECT \* FROM airline\_staff WHERE username = %s AND airline\_name = %s

Check airline\_staff table to see if the airline\_name input is valid.

SELECT \* FROM airplane WHERE airline\_name = %s AND airplane\_id = %s AND seats = %s

Check airplane table to see if the airplane already exists.

INSERT INTO airplane VALUES(%s, %s, %s)

Insert new airplane into the airplane table.

If any of the errors mentioned above arise, an error message will be displayed.

1. **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.)

SELECT \* FROM permission WHERE username = %s AND permission\_type = 'Admin'

Check permission table to see whether this staff has admin permission.

SELECT \* FROM airline\_staff WHERE username = %s AND airline\_name = %s

Check airline\_staff table to see if the airline\_name input is valid.

SELECT \* FROM airport WHERE airport\_name = %s AND airport\_city = %s

Check airport table to see if the airport already exists.

INSERT INTO airport VALUES(%s, %s)

Insert new airport into the airport table.

If any of the errors mentioned above arise, an error message will be displayed.

1. **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.

CREATE VIEW agent\_month AS

SELECT ticket\_id, email

FROM purchases AS P JOIN booking\_agent AS B

WHERE P.booking\_agent\_id = B.booking\_agent\_id

AND purchase\_date BETWEEN DATE\_SUB(CURRENT\_DATE(),INTERVAL 30 DAY) AND CURRENT\_DATE

Create view to select all the tickets purchased by all the agents in the required time interval.

SELECT email, COUNT(ticket\_id) FROM agent\_month GROUP BY email

ORDER BY COUNT(ticket\_id) DESC LIMIT 5

Select email and ticket number sold by the count function from the view agent\_month and get the first 5 after grouping it by email which is the key value of the agents and ordering it in descending order according to the number of tickets.

SELECT email, COUNT(ticket\_id) FROM agent\_year GROUP BY email

ORDER BY COUNT(ticket\_id) DESC LIMIT 5

Select email and ticket number sold by the count function from the view agent\_year, which is exactly the same logic as view agent\_month except for the time, and get the first 5 after grouping it by email which is the key value of the agents and ordering it in descending order of the number of tickets.

CREATE VIEW agent\_year\_total AS

SELECT P.ticket\_id, email, price \* 0.1 AS commission

FROM purchases AS P JOIN booking\_agent AS B JOIN ticket AS T JOIN flight AS F

WHERE P.booking\_agent\_id = B.booking\_agent\_id AND P.ticket\_id = T.ticket\_id

AND T.airline\_name = F.airline\_name AND T.flight\_num = F.flight\_num

AND purchase\_date BETWEEN DATE\_SUB(CURRENT\_DATE(),INTERVAL 1 YEAR) AND CURRENT\_DATE

Create view to generate the total commission of all the tickets by joining the purchase, booking\_agent, ticket and flight to get the price of each ticket. The email of the agent is displayed along with the ticket price multiplied by 0.1 named as commission.

SELECT email, SUM(commission) FROM agent\_year\_total GROUP BY email

ORDER BY COUNT(ticket\_id) DESC LIMIT 5

Select email and total commission by the sum function from the view agent\_year\_total and get the first 5 after grouping it by email which is the key value of the agents and ordering it in descending order according to the total commission.

1. **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.

CREATE VIEW customer\_year AS

SELECT ticket\_id, customer\_email, name

FROM purchases AS P JOIN customer AS C

WHERE customer\_email = email

AND purchase\_date BETWEEN DATE\_SUB(CURRENT\_DATE(),INTERVAL 1 YEAR) AND CURRENT\_DATE

Create view to get all the tickets that has been purchased within the given time along with the customer’s email by combining the purchase and customer table.

SELECT customer\_email, name, COUNT(ticket\_id) FROM customer\_year

GROUP BY customer\_email ORDER BY COUNT(ticket\_id) DESC LIMIT 1

Get the most frequent customer by counting how many tickets they bought in the past year from view customer\_year, and then group it by customer\_email and order it in descending order according to the total number of tickets a customer has bought and only keep the first one.

SELECT F.airline\_name, F.flight\_num, departure\_airport, departure\_time,

arrival\_airport, arrival\_time, price, status, airplane\_id

FROM purchases as P, ticket AS T, flight AS F, airline\_staff AS S

WHERE P.ticket\_id = T.ticket\_id AND T.airline\_name = F.airline\_name AND T.flight\_num = F.flight\_num

AND F.airline\_name = S.airline\_name AND customer\_email = %s AND S.username = %s

Get all the information of the flight the customer has taken of the particular airline the staff is in by joining the purchases, ticket, flight and airline\_staff tables, and match the features of the input from the staff, which contains customer email. Username of the staff is also store in a variable so as to get the correct airline from the session[‘username’] and match it with S.username.

1. **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.

SELECT COUNT(ticket\_id) FROM purchases \

WHERE purchase\_date BETWEEN %s AND %s

SELECT MONTH(purchase\_date) `month`, COUNT(ticket\_id) `times` \

FROM purchases WHERE purchase\_date BETWEEN %s AND %s \

GROUP BY MONTH(purchase\_date) \

ORDER BY MONTH(purchase\_date)

The first query gets the number of tickets from the requested date, and the second query gets the month and number of tickets, order and group by month, which will then be turned into a dictionary for graphing.

1. **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.

CREATE VIEW revenue\_month\_direct AS

SELECT T.ticket\_id, price

FROM purchases as P, ticket AS T, flight AS F

WHERE P.ticket\_id = T.ticket\_id AND T.airline\_name = F.airline\_name AND T.flight\_num = F.flight\_num

AND P.booking\_agent\_id IS NOT NULL

AND purchase\_date BETWEEN DATE\_SUB(CURRENT\_DATE(),INTERVAL 1 MONTH) AND CURRENT\_DATE

SELECT SUM(price) AS MD FROM revenue\_month\_direct

SELECT SUM(price) AS MI FROM revenue\_month\_indirect

SELECT SUM(price) AS YD FROM revenue\_year\_direct

SELECT SUM(price) AS YI FROM revenue\_year\_indirect

The first query is an example of how the view revenue\_month\_direct is created. We get all the tickets satisfying the time requirement and ordered by a booking agent along with its price, by combining the purchases, ticket and flight table. The other three views are created in the same logic. Then, we use another query to get the sum of the prices purchased directly/indirectly and by month/year accordingly.

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

CREATE VIEW destination\_3month AS

SELECT airport\_city, P.ticket\_id

FROM purchases AS P, ticket AS T, flight AS F, airport AS A

WHERE P.ticket\_id = T.ticket\_id AND T.airline\_name = F.airline\_name AND T.flight\_num = F.flight\_num

AND F.arrival\_airport = A.airport\_name

AND purchase\_date BETWEEN DATE\_SUB(CURRENT\_DATE(),INTERVAL 3 MONTH) AND CURRENT\_DATE

SELECT airport\_city, COUNT(ticket\_id) FROM destination\_3month \

GROUP BY airport\_city ORDER BY COUNT(ticket\_id) DESC LIMIT 3

SELECT airport\_city, COUNT(ticket\_id) FROM destination\_year \

GROUP BY airport\_city ORDER BY COUNT(ticket\_id) DESC LIMIT 3

We first create a view to get all the airport city along with ticket\_id within the given time frame by combining the purchases, ticket, flight, and airport tables. The example is for 3 months and we create the annual version with the same logic. Then, we get the top three airport\_city by ordering according to the ticket number with the count function and only take the first 3.

1. **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.)

SELECT \* FROM permission WHERE username = %s AND permission\_type = \'Admin\'

Check permission table to see whether this staff has admin permission.

SELECT airline\_name FROM airline\_staff WHERE username = %s

Use this query twice, with agent/staff username separately and compare the result, in order to see if they are in the same airline.

SELECT \* FROM permission WHERE username = %s AND permission\_type = %s

See if the other staff is already an Admin

INSERT INTO permission VALUES(%s, %s)

Insert new staff and the permission type into the permission table.

1. **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.)

SELECT \* FROM permission WHERE username = %s AND permission\_type = \'Admin\'

Check permission table to see whether this staff has admin permission.

SELECT \* FROM airline\_staff WHERE username = %s AND airline\_name = %s

Check if the staff and agent are in the same airline.

SELECT \* FROM booking\_agent\_work\_for WHERE email = %s AND airline\_name = %s

Check if the booking\_agent is already working for the airline.

INSERT INTO booking\_agent\_work\_for VALUES(%s, %s)

Insert the input into the booking\_agent\_work table.

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