Mx632 MeiNa Xie
Wz1509 Winnie Zheng

Initial page:

-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).

```
'SELECT flight_num, departure_date_and_time FROM airport natural join arrival WHERE
(name = %s or city = %s) and flight_num in (select flight_num FROM airport natural
join departure where name = %s or city = %s)'
```

This query searches for all the flights and their status based on the given search criteria: flight_num, departure_date_and_time, and arrival_date_and_time. Whoever visits through index page cannot change the status of flights so it can only view the statuses this query produces.

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

```
'SELECT name, flight_num, status FROM flight WHERE name = %s and flight_num = %s and
departure_date_and_time = %s and arrival_date_and_time = %s'
```

This query searches for all the departure flights possible for a specific departure date . The user is able to search up available flights based on either the name of the airport the flight departures and arrives at OR the city in which the flight departures or arrives at. The first two blanks are for the arrival place (city or name of airport) and the second two blanks in the query are for the departure place (city of name of airport). The departure date and time are not accounted for in the query, but rather in our code under search_flights_C().

Customer:

-Login

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

Search if the email and password matches the database for customer

---

-Register

```
'SELECT * FROM customer WHERE email = %s'
```

Finding all the customers that's already registered

```
'INSERT INTO customer VALUES(%s, %s, md5(%s), %s,%s, %s, %s, %s, %s, %s, %s, %s)'
```

Inserting a new customer with its values into the table customer, where the values are email, customer name, password, building number, street name, city, state, phone number, passport id, passport expiration date, passport country, and date of birth respectively.

-View My flights: Provide various ways for the user to see flights information which he/she purchased. The default should be showing for the 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.

```
'SELECT name, flight_num, departure_date_and_time, arrival_date_and_time, status FROM
purchase natural join reserve natural join flight WHERE c_email = %s and
departure_date_and_time - (select now()) > 0'
```

Finding the future flights that he/she bought

-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.

```
'SELECT name, flight_num, departure_date_and_time, arrival_date_and_time, status FROM
purchase natural join reserve natural join flight WHERE c_email = %s and
arrival_date_and_time - (select now()) <=0’
```

Finding all the past flights that the customer has took

```
"INSERT INTO rating values(%s, %s, %s, %s, %s)"
```

Inserting a rating and comment into the table rating for a flight

---

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

```
'SELECT name, flight_num, status FROM flight WHERE name = %s and flight_num = %s and
departure_date_and_time = %s and arrival_date_and_time = %s'
```

This query searches for all the flights and their status based on the given search criteria: flight_num, departure_date_and_time, and arrival_date_and_time. The customer cannot change the status of flights so it can only view the statuses this query produces.

-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).

```
'SELECT flight_num, departure_date_and_time FROM airport natural join arrival WHERE
(name = %s or city = %s) and flight_num in (select flight_num FROM airport natural
join departure where name = %s or city = %s)
```

This query searches for all the departure flights possible for a specific departure date . The user is able to search up available flights based on either the name of the airport the flight departures and arrives at OR the city in which the flight departures or arrives at. The first two blanks are for the arrival place (city or name of airport) and the second two blanks in the query are for the departure place (city of name of airport). The departure date and time are not accounted for in the query, but rather in our code under search_flights_C().

```
'SELECT flight_num, departure_date_and_time FROM airport natural join arrival WHERE
(name = %s or city = %s) and flight_num in (select flight_num FROM airport natural
join departure where name = %s or city = %s)
```

This query searches for all the return flights possible for a specific return date (if the user specifies which date they want to return on). The user is able to search up available flights based on either the name of the airport the flight departures and arrives at OR the city in which the flight departures or arrives at. The first two blanks are for the arrival place (city or name of airport) and the second two blanks in the query are for the departure place (city of name of airport). The departure date and time are not accounted for in the query, but rather in our code under search_flights_C().

- 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.

```
"SELECT * FROM flight natural join reserve as f, ticket as t WHERE t.ticket_ID =
f.ticket_ID and f.flight_num = %s and departure_date_and_time = %s and card_type is
NULL
```

This query queries all the available flights given a specific flight number and a departure date and time (a flight can have multiple tickets available). The card_type is NULL here because if the card_type is NULL, it means that no one has bought the ticket yet.

```
"UPDATE ticket SET card_type = %s, card_number = %s, name = %s, expiration_date = %s,
purhcase_date_and_time = (select NOW()) where ticket_ID = %s"
```

This query inputs the purchase information into the TICKET table. Each ticket row that hasn't been purchased has NULL for the attributes related to purchase info (like card type and card number), so when the customer makes a purchase, it will update the row with the card type, card number, name (on the card), expiration date, purchase_date_and_time, and ticket_ID respectively into the table.

```
"INSERT INTO purchase values(%s, %s, NULL, NULL, NULL, NULL)"
```

Additionally, this query inserts into the PURCHASE table the values ticket_ID, c_email, and four NULL attribute parameters. This is to record the purchase as a direct purchase, with the attributes of the booking agents NULL.

-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.

```
"SELECT sold_price as price, purhcase_date_and_time as date FROM ticket natural join
customer WHERE email = %s and (year(CURRENT_TIMESTAMP) - year(purhcase_date_and_time))
* 12 + (month(CURRENT_TIMESTAMP) - month(purhcase_date_and_time)) <= 12"
```

Finding the prices of each ticket he/she bought and the date he/she bought it on for the past 12 months by default. This result is displayed through a bar graph on the customer's home page. The customer's email address (primary key for customer) is used here to query all its past purchases.

He/she will also have an 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.

```
"SELECT sold_price as price, purhcase_date_and_time as date FROM ticket natural join
customer WHERE email = %s and (%s >purhcase_date_and_time and purhcase_date_and_time >
%s)
```

Alternatively, if the customer uses the search bar, s/he can track their total spending using this query above that takes in three parameters: the customer's email, the begin date and the end date for a custom range. The results are displayed on the bar graphs as well.

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

Booking Agent

-Login

```
'SELECT * FROM booking_agent WHERE email = %s and password = md5(%s) and
booking_agent_ID = %s'
```

Look if the email and the password is a match in the database

---

-Register

```
'SELECT * FROM booking_agent WHERE email = %s'
```

Looking if the booking agent already exist

```
'INSERT INTO booking_agent VALUES(%s, md5(%s), %s)'
```

Inserting a new value into the table booking_agent, where the values are email, password, and booking agent id respectively.

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

```
'SELECT name, flight_num, status FROM flight WHERE name = %s and flight_num = %s and
departure_date_and_time = %s and arrival_date_and_time = %s'
```

This query searches for all the flights and their status based on the given search criteria: flight_num, departure_date_and_time, and arrival_date_and_time. The booking agent cannot change the status of flights so it can only view the statuses this query produces.

-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 future 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.

```
'SELECT name, flight_num, departure_date_and_time, arrival_date_and_time, status FROM
purchase natural join reserve natural join flight WHERE b_email = %s and
booking_agent_ID = %s and (select now()) - departure_date_and_time <= 0’
```

Finds the flights that the booking agent booked for the customers

---

- 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 arrival).

```
'SELECT flight_num, departure_date_and_time FROM airport natural join arrival WHERE
(name = %s or city = %s) and flight_num in (select flight_num FROM airport natural
join departure where name = %s or city = %s)
```

This query searches for all the departure flights possible for a specific departure date . The user is able to search up available flights based on either the name of the airport the flight departures and arrives at OR the city in which the flight departures or arrives at. The first two blanks are for the arrival place (city or name of airport) and the second two blanks in the query are for the departure place (city of name of airport). The departure date and time are not accounted for in the query, but rather in our code under search_flights_BA().

```
'SELECT flight_num, departure_date_and_time FROM airport natural join arrival WHERE
(name = %s or city = %s) and flight_num in (select flight_num FROM airport natural
join departure where name = %s or city = %s)
```

This query searches for all the return flights possible for a specific return date (if the user specifies which date they want to return on). The user is able to search up available flights based on either the name of the airport the flight departures and arrives at OR the city in which the flight departures or arrives at. The first two blanks are for the arrival place (city or name of airport) and the second two blanks in the query are for the departure place (city of name of airport). The departure date and time are not accounted for in the query, but rather in our code under search_flights_BA().

- Purchase tickets: Booking agent chooses a flight and purchases tickets for other customers giving customer information and payment information, providing all the needed data, via forms. You may find it easier to implement this along with a use case to search for flights.

```
"SELECT * FROM flight natural join reserve as f, ticket as t WHERE t.ticket_ID =
f.ticket_ID and f.flight_num = %s and departure_date_and_time = %s and card_type is
NULL
```

This query queries all the available flights given a specific flight number and a departure date and time (a flight can have multiple tickets available). The card_type is NULL here because if the card_type is NULL, it means that no one has bought the ticket yet.

```
"UPDATE ticket SET card_type = %s, card_number = %s, name = %s, expiration_date = %s,
purhcase_date_and_time = (select NOW()) where ticket_ID = %s"
```

This query inputs the purchase information into the TICKET table. Each ticket row that hasn't been purchased has NULL for the attributes related to purchase info (like card type and card number), so when the booking agent makes a purchase on behalf of a customer, it will update the row with the card type, card number, name (on the card), expiration date, purchase_date_and_time, and ticket_ID respectively into the table.

Additionally, this query needs to be executed:

```
"INSERT INTO purchase values(%s, %s, %s, %s, %s)"
```

This query inserts into the PURCHASE table the values ticket_ID, c_email, email (of booking agent), ID (of booking agent), and the commission for the booking agent. This way, we can keep track of whether each purchase was handled through the help of an agent or if the customer purchased directly.

- 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.

```
"SELECT (sold_price * commission / 100) as profit FROM purchase natural join ticket
where b_email = %s and booking_agent_ID = %s and ((year(CURRENT_TIMESTAMP) -
year(purhcase_date_and_time)) * 365 + (month(CURRENT_TIMESTAMP) -
month(purhcase_date_and_time)) * 30  + day(CURRENT_TIMESTAMP) -
day(purhcase_date_and_time)) <= 30"
```

Default 30 day commission the booking agent earned

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 (sold_price * commission / 100) as price FROM ticket natural join purchase
WHERE b_email = %s and booking_agent_ID = %s and (%s >purhcase_date_and_time and
purhcase_date_and_time > %s) "
```

First, this query takes in the booking agent ID and email (the primary keys of a booking agent) along with the custom date range (in relation to when a ticket is purchased).

-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.

```
"SELECT count(ticket_ID) as tickets, c_email FROM ticket natural join purchase WHERE
b_email = %s and booking_agent_ID = %s and ((year(CURRENT_TIMESTAMP) -
year(purhcase_date_and_time)) * 12 + (month(CURRENT_TIMESTAMP) -
month(purhcase_date_and_time))) <= 6 GROUP BY c_email ORDER BY tickets DESC LIMIT 5"
```

Finding the top five customers that bought tickets from the booking agent, and if there's not 5, just shows the number of customers he/she had.

Show another bar chart showing each of these 5 customers in x-axis and amount commission received in y- axis.

```
"SELECT sum(sold_price * commission / 100) as profit, c_email FROM ticket natural join
purchase WHERE b_email = %s and booking_agent_ID = %s and ((year(CURRENT_TIMESTAMP) -
year(purhcase_date_and_time)) * 12 + (month(CURRENT_TIMESTAMP) -
month(purhcase_date_and_time))) <= 12 GROUP BY c_email ORDER BY profit DESC LIMIT 5"
```

Finding the top five customers that gave him/her the most commission, and if there's not 5, it just shows how ever many customers he/she had.

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

Airline Staff

-Login

```
'SELECT * FROM airline_staff WHERE username = %s and password = md5(%s)'
```

Look if the username and password is a match in the database

```
'SELECT name FROM works WHERE username = %s'
```

Getting the airline name that the airline staff works for

-Register

```
'SELECT * FROM airline_staff WHERE username = %s'
```

Looking if there’s a match of the username and password in the database

```
'INSERT INTO airline_staff VALUES(%s, md5(%s), %s, %s, %s)'
```

Inserting values into the table of airline staff, where the values are username, password, first name, last name, and date of birth respectively.

```
"INSERT INTO works VALUES(%s, %s)"
```

Inserting the relation between airline staff and airline into works, where values are username and airline name respectively.

```
"INSERT into staff_phone_number values(%s, %s)"
```

Inserting into the table of staff’s phone number with values username, and 10 digit phone number respectively

---

-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).

```
SELECT flight_num, departure_date_and_time FROM airport natural join arrival WHERE
(name = %s or city = %s) and flight_num in (select flight_num FROM airport natural
join departure where name = %s or city = %s)
```

This query searches for all the departure flights possible for a specific departure date . The user is able to search up available flights based on either the name of the airport the flight departures and arrives at OR the city in which the flight departures or arrives at. The first two blanks are for the arrival place (city or name of airport) and the second two blanks in the query are for the departure place (city of name of airport). The departure date and time are not accounted for in the query, but rather in our code under search_flights_AS().

```
'SELECT flight_num, departure_date_and_time FROM airport natural join arrival WHERE
(name = %s or city = %s) and flight_num in (select flight_num FROM airport natural
join departure where name = %s or city = %s)
```

This query searches for all the return flights possible for a specific return date (if the user specifies which date they want to return on). The user is able to search up available flights based on either the name of the airport the flight departures and arrives at OR the city in which the flight departures or arrives at. The first two blanks are for the arrival place (city or name of airport) and the second two blanks in the query are for the departure place (city of name of airport). The departure date and time are not accounted for in the query, but rather in our code under search_flights_AS().

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

```
'SELECT name, flight_num, status, departure_date_and_time FROM flight WHERE name = %s
and flight_num = %s and departure_date_and_time = %s and arrival_date_and_time = %s'
```

This query searches for all the flights, its departure_date_and_time, and its status based on the given search criteria: flight_num, departure_date_and_time, and arrival_date_and_time. The departure_date_and_time is queried here because this query will be used to display all the flights that the airline staff can also change the status of later on. Because of that, both primary keys flight_num and departure_date_and_time need to be present to update the status in the next query.

- View flights: Defaults will be showing all the future flights operated by the airline he/she works for the next 30 days.

```
"SELECT DISTINCT flight_num, departure_date_and_time FROM flight NATURAL JOIN reserve
WHERE name = %s and ((year(departure_date_and_time) - year(CURRENT_TIMESTAMP)) * 12 +
(month(departure_date_and_time) - month(CURRENT_TIMESTAMP))) <= 1"
```

Finding the flights that the airline staff works for in the future month, and the value is airline name.

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.

```
'SELECT DISTINCT flight_num, departure_date_and_time FROM airport natural join arrival
WHERE (name = %s or city = %s) and flight_num in (select flight_num FROM airport
natural join departure where name = %s or city = %s) and departure_date_and_time >= %s
and departure_date_and_time <= %s'
```

Finding the flights that the airline staff works for in a specific range of time.

He/she will be able to see all the customers of a particular flight.

```
"SELECT DISTINCT c_email FROM purchase natural join reserve natural join flight WHERE
flight_num = %s and departure_date_and_time = %s"
```

Finding distinct customers that bought this flight, in case they bought more than one ticket

---

- 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.

```
"insert into flight values(%s, %s, %s, %s, %s, %s)"
```

This query is executed to allow for the airline staff to create a new flight and insert it into the FLIGHT table with the values of flight_num, departure_date_and_time, name(of airline), arrival_date_and_time, base_price (of ticket), and status (on time or delayed), respectively.

```
"insert into arrival values(%s, %s, %s)"
```

This query is executed to allow for the airline staff to create a new flight and insert it into the ARRIVAL table with the values of flight_num, departure_date_and_time, and arrival (airport name), respectively. During the creation of a new flight, the airline staff must also provide both the arrival airport name and the departure airport name.

```
"insert into departure values(%s, %s, %s)"
```

This query is executed to allow for the airline staff to create a new flight and insert it into the DEPARTURE table with the values of flight_num, departure_date_and_time, and departure (airport name), respectively. During the creation of a new flight, the airline staff must also provide both the arrival airport name and the departure airport name.

```
"insert into has values(%s, %s, %s)"
```

This query is executed to allow for the airline staff to create a new flight and insert it into the HAS table the values flight_num, departure_date_and_time, and ID_num(of airplane), respectively. This is because during the creation of a new flight, the flight must be made for a certain airplane.

-Change Status of flights: He or she changes a flight status (from on-time to delayed or vice versa) via forms.

```
"UPDATE flight SET status = %s WHERE flight_num = %s and departure_date_and_time = %s
and name = %s"
```

This query is executed to allow for the airline staff to change the status of a certain flight. The airline staff must search for the flight using the flight_num and the departure_date_and_time (since those are the primary keys of a flight) and then is able to directly change the status. The update query above has the values status (either “on time” or delayed”), flight_num, and departure_date_and_time respectively.

-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.

```
"insert into airplane values(%s, %s)"
```

This query is executed to allow for the airline staff to create a new airline into the system. It inserts both the ID number of the new airplane as well as the number of seats. Those are the only two attributes. Then, we run this query to below:

```
"insert into owns values(%s, %s)"
```

This query will take the name of the airline that airline staff works for (which is recorded in the session dictionary) along with the new ID num of the new airplane and add it to the owns table so that the new airplane now has an airline that it belongs to.

```
Select ID_num from owns natural join works where username = %s
```

This query is executed when the airline staff is on the confirmation page where s/he has just created a new airplane. This query selects all the ID number of all the airplanes of the airline that the specific airline staff works for. (username is the airline staff's username).

-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.

```
"insert into airport values(%s, %s)"
```

This query is executed to allow the airline staff user to create a new airport. Only two attributes are needed: the name of the airport as well as the city the airport is located in.

-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.

```
"SELECT rating, comment, email FROM rating WHERE flight_num = %s and
departure_date_and_time = %s"
```

Finding the rating comment and who did the rating for a particular flight

```
"SELECT AVG(rating) as avg FROM rating WHERE flight_num = %s and
```

```
departure_date_and_time = %s"
```

Find the average rating of that particular flight

-View all the booking agents:

```
"Select email from booking_agent WHERE email is not NULL"
```

Finding all the booking agents

Top 5 booking agents based on number of tickets sales for the past month

```
"SELECT count(ticket_ID) as tickets, b_email, booking_agent_ID FROM ticket natural
join purchase WHERE (b_email is not NULL) and ((year(CURRENT_TIMESTAMP) -
year(purhcase_date_and_time)) * 12 + (month(CURRENT_TIMESTAMP) -
month(purhcase_date_and_time))) <= 1 GROUP BY b_email, booking_agent_ID ORDER BY
tickets DESC LIMIT 5"
```

Finding the top 5 booking agents in the past month by tickets sold, if not enough, just the number of booking agents we have there

and past year.

```
"SELECT count(ticket_ID) as tickets, b_email, booking_agent_ID FROM ticket natural
join purchase WHERE (b_email is not NULL) and ((year(CURRENT_TIMESTAMP) -
year(purhcase_date_and_time)) * 12 + (month(CURRENT_TIMESTAMP) -
month(purhcase_date_and_time))) <= 12 GROUP BY b_email, booking_agent_ID ORDER BY
tickets DESC LIMIT 5"
```

Finding the top 5 booking agents in the past year by tickets sold, if not enough, just the number of booking agents we have there

---

Top 5 booking agents based on the amount of commission received for the last year.

```
"SELECT sum(sold_price * commission / 100) as profit, b_email, booking_agent_ID FROM
ticket natural join purchase WHERE (b_email is not NULL) and ((year(CURRENT_TIMESTAMP)
- year(purhcase_date_and_time)) * 12 + (month(CURRENT_TIMESTAMP) -
month(purhcase_date_and_time))) <= 12 GROUP BY b_email, booking_agent_ID ORDER BY
profit DESC LIMIT 5"
```

Finding the top 5 booking agents in the past year by commission, if not enough, just the number of booking agents we have there

---

-View frequent customers: Airline Staff will also be able to see the most frequent customer within the last year.

```
"Select c_email from flight as f , reserve as r , ticket as t, purchase as p where
f.flight_num = r.flight_num and f.name = %s and  r.ticket_ID = t.ticket_ID and
t.ticket_ID = p.ticket_ID group by c_email order by count(t.ticket_ID) DESC"
```

This query retrieves all the c_emails based on the number of tickets they have bought in descending order. Because there might be multiple customers who bought the most number of tickets, we accounted for the edge case in python. This way, if more than one customer bought the most amount of tickets, both of their names will pop up.

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

```
"Select f.flight_num from flight as f , reserve as r , ticket as t, purchase as p
where f.flight_num = r.flight_num and f.name = %s and r.ticket_ID = t.ticket_ID and
t.ticket_ID = p.ticket_ID and p.c_email = %s"
```

Finding flights that a particular customer bought at that 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.

```
"SELECT purhcase_date_and_time as date FROM ticket as t, reserve as r, flight as f
WHERE t.ticket_ID = r.ticket_ID and r.flight_num = f.flight_num and f.name = %s and
(%s >purhcase_date_and_time and purhcase_date_and_time > %s) "
```

Finding the dates of the tickets sold for that airline in a specific range of dates, and in python, we look at how many values it's in each month, because each date returned by the query is a ticket sold.

-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.

-View Top destinations: Find the top 3 most popular destinations for last 3 months

```
"Select a.name, count(t.ticket_ID) as tickets from ticket as t, reserve as r, flight
as f, arrival as a where t.ticket_ID = r.ticket_ID and r.flight_num = f.flight_num and
f.flight_num = a.flight_num and t.card_type is not None and ((year(CURRENT_TIMESTAMP)
- year(t.purhcase_date_and_time)) * 12 + (month(CURRENT_TIMESTAMP) -
month(t.purhcase_date_and_time))) <= 3 group by a.name order by tickets desc limit 3”
```

Finding the top three airports that most tickets are bought to fly to in the last three months

and last year (based on tickets already sold)

```
“Select a.name, count(t.ticket_ID) as tickets from ticket as t, reserve as r, flight
as f, arrival as a where t.ticket_ID = r.ticket_ID and r.flight_num = f.flight_num and
f.flight_num = a.flight_num and t.card_type is not None and ((year(CURRENT_TIMESTAMP)
- year(t.purhcase_date_and_time)) * 12 + (month(CURRENT_TIMESTAMP) -
month(t.purhcase_date_and_time))) <= 12 group by a.name order by tickets desc limit 3”
```

Finding the top three airports that most tickets are bought to fly to in the last year

---

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