Application Use Cases:

Before log in use cases:

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.
      In our search , date is optional. Search parameters are derived from request.form
      In the airport search case,
      If date is not entered,

```
query = '''SELECT DISTINCT airline_name, flight_num,
departure_time , arrival_time, departure_airport,
arrival_airport, status \
                   FROM flight \
                   WHERE status = 'Upcoming' AND
departure_airport = "{}" AND arrival_airport = "{}"
'''.format(departure_airport,arrival_airport)
```

If date is entered,

```
query = '''SELECT DISTINCT airline_name, flight_num,
departure_time , arrival_time, departure_airport,
arrival_airport, status \
               FROM flight \
               WHERE status = 'Upcoming' AND
departure_airport = "{}" AND arrival_airport = "{}" AND
DATE(departure_time) = '{}'
'''.format(departure_airport,arrival_airport,departure_date)
```

We can also include the return date to search for 2 flights to cover two way flights between two places.

```
query = '''SELECT DISTINCT airline_name, flight_num,
departure_time , arrival_time, departure_airport,
arrival_airport, status \
                   FROM flight \
                   WHERE status = 'Upcoming' AND
departure_airport = "{}" AND arrival_airport = "{}" AND
DATE(departure_time) = '{}' '''.format(arrival_airport,
departure_airport,return_date)
```

In the city search case with date entered,

```
query = '''SELECT DISTINCT airline_name,flight_num
,departure_time, arrival_time, departure_airport,
arrival_airport, status  FROM flight \
                       WHERE status = 'Upcoming' AND
departure_airport IN(SELECT airport_name FROM airport WHERE
airport_city = "{}") \
                       AND arrival_airport IN (SELECT
airport_name FROM airport WHERE airport_city = "{}") \
                       AND DATE(departure_time) = '{}'
'''.format (departure_city, arrival_city, departure_date)
```

Results are displayed in a table in html page.

b. Will be able to see the flights status based on flight number, arrival/departure date. We also made date optional in this case

With date entered,

```
query = '''SELECT DISTINCT airline_name, flight_num,
departure_time , arrival_time, departure_airport,
arrival_airport, status \
                   FROM flight \
                   WHERE status = 'Upcoming' AND
airline_name = "{}" AND flight_num = {} AND
DATE(departure_time) = '{}' '''.format(airline_name,
int(flight_num), departure_date)
```

With date entered,

```
query = '''SELECT DISTINCT airline_name, flight_num,
departure_time , arrival_time, departure_airport,
arrival_airport, status \
                   FROM flight \
                   WHERE status = 'Upcoming' AND
airline_name = "{}" AND flight_num = {}
'''.format(airline_name, int(flight_num))
```

c. See a default view that have all flights in the coming week (additional use case we included).

```
today = datetime.now().replace(microsecond=0)
   default_view_date = today + relativedelta(days=+7)

   query = '''SELECT DISTINCT airline_name, flight_num,
departure_time , arrival_time, departure_airport,
arrival_airport, status \
               FROM flight WHERE departure_time > '{}' AND
departure_time <= '{}' '''.format(today,default_view_date)
```

Default view date is derived from datetime.now and relative delta

2. **Register**:3 types of user registrations (Customer, Booking agent, Airline Staff) option via forms.
    a. Check if the user already in the database:
        i. Customer

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

        ii. Booking agent

```
query = '''SELECT * FROM booking_agent WHERE email = '{}'
'''.format(email)
```

        iii. Airline staff

```
query = '''SELECT * FROM airline_staff WHERE username = '{}'
'''.format(username)
```

    b. Insert corresponding data into the database:
        i. Customer

```
sql = '''INSERT INTO customer VALUES
('{}','{}','{}','{}','{}','{}','{}','{}','{}','{}','{}','{}'
)'''.format(email, name, password1, buildingNumber, street,
city, state, phoneNumber, passportNumber,
passportExpiration, passportCountry, dob)
```

        ii. Booking

```
query = '''INSERT INTO booking_agent VALUES ('{}', '{}',
'{}') '''.format(email, password1, agent_id)
```

iii. Airline staff

```
query = '''INSERT INTO airline_staff VALUES ('{}', '{}',
'{}', '{}', '{}', '{}')  '''.format(username, password1,
first_name, last_name, dob, airline_name)
```

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. Customer:

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

    b. Booking Agent

    ```
    query = ''' SELECT * FROM booking_agent WHERE email =
    '{}' and password = '{}' '''.format(email, password)
    ```

    c. Airline Staff

    ```
    query = ''' SELECT password FROM airline_staff WHERE
    username = '{}' and password = '{}' '''.format(username,
    password)
    ```

Customer use cases:

1. **View My flights:**

    The default view case, note that we think even a customer’s flight is delayed, it should also be shown on the home page of the customer, so we did not specify the state as upcoming.

```
query = '''SELECT DISTINCT F.airline_name, F.flight_num,
F.departure_time , F.arrival_time, F.departure_airport,
F.arrival_airport, F.status, \

                   COUNT(F.flight_num) AS ticket_num \

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

                   WHERE F.flight_num = T.flight_num AND
T.ticket_id = P.ticket_id AND P.customer_email = '{}' \

                   GROUP BY
F.airline_name,F.flight_num'''.format(username)
```

With a specified range of date, the query becomes

```
query = '''SELECT DISTINCT F.airline_name, F.flight_num,
F.departure_time , F.arrival_time, F.departure_airport,
F.arrival_airport, F.status, \

                   COUNT(F.flight_num) AS ticket_num \

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

                   WHERE F.flight_num = T.flight_num AND
T.ticket_id = P.ticket_id AND P.customer_email = '{}' \

                       AND (DATE(F.departure_time) BETWEEN "{}"
AND "{}") \

                   GROUP BY
F.airline_name,F.flight_num'''.format(username, start_date,
end_date)
```

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.

   I designed a feature that for each customer only 5 tickets of one flight can be purchased. So it made the case more complicated.

First it detects if a customer have bought over 5 tickets when customer click the purchase button on flight information.

```
'''SELECT COUNT(flight_num) AS ticket_bought FROM ticket NATURAL
JOIN purchases \

                           WHERE customer_email = '{}' AND
flight_num = {} AND airline_name = '{}' \

                           GROUP BY
flight_num,airline_name'''.format(username,purchase_flight_number
,purchase_airline_name)
```

A failure message will be displayed if it is over 5 , and if not, it will judge if there is ticket left.

```
condition_query = '''SELECT COUNT(airline_name) FROM ticket WHERE
airline_name = '{}' AND flight_num = {} GROUP BY \

                               airline_name,flight_num
'''.format(purchase_airline_name,purchase_flight_number)

if condition_result is None:

               query = '''SELECT airline_name, flight_num,
departure_airport, arrival_airport, departure_time, arrival_time,
price, seats AS seats_remaining \

                           FROM airplane NATURAL JOIN flight \

                           WHERE flight_num = {} AND airline_name
= '{}' \

                           GROUP BY flight_num
'''.format(purchase_flight_number,purchase_airline_name)

           else:

               query = '''SELECT airline_name, flight_num,
departure_airport, arrival_airport, departure_time, arrival_time,
price, seats - COUNT(ticket_id) AS seats_remaining \
                           FROM airplane NATURAL JOIN flight
NATURAL JOIN ticket \

                           WHERE flight_num = {} AND airline_name
= '{}' \

                           GROUP BY flight_num \

                           HAVING seats_remaining > 0
'''.format(purchase_flight_number,purchase_airline_name)
```

If there is no result in this query, we display a sold out message, if there is ticket left, we go to the purchase confirmation page.

The customer can at most buy 5 ticket a time, the also need to enter credit card number as payment method. To make sure the buy number does not go over limit.

```
'''SELECT COUNT(flight_num) AS ticket_bought FROM ticket NATURAL
JOIN purchases

WHERE customer_email = '{}' AND flight_num = {} AND airline_name
= '{}'

GROUP BY
flight_num,airline_name'''.format(username,flight_num,airline_nam
e)

'''SELECT A.seats AS remaining_seats FROM airplane AS A, flight
as F

 WHERE F.airline_name = '{}' AND F.flight_num = {} AND
F.airplane_id = A.airplane_id'''.format(airline_name,flight_num)
```

Ticket bought plus ticket purchase number has to be under or equal to five and ticket left must be larger or equal to purchase number, otherwise, an failure message will be displayed and notify users to buy less.

If the purchase is successful, we insert values into database, but we need to determine ticket id.

```
query = '''SELECT MAX(ticket_id)+1 AS new_ticket_id FROM
ticket'''
```

```
query1 = ''' INSERT INTO ticket (ticket_id,
airline_name,flight_num) VALUES('{}','{}',{});
'''.format(ticket_id,airline_name,flight_num)

query2 = '''INSERT INTO purchases (ticket_id, customer_email,
booking_agent_id, purchase_date, credit_card_number, figure)
VALUES ('{}','{}',NULL,'{}','{}','{}')
'''.format(ticket_id,username,purchase_date,ccn,price)
```

We display the flight information of the purchase afterwards and automatically go back to home page after 3 seconds.

```
if len(ticket_num_list) == 1:

                query = '''SELECT * FROM purchases WHERE ticket_id
= {}'''.format(ticket_num_list[0])

            else:

                ticket_numbers = tuple(ticket_num_list)

                query = '''SELECT * FROM purchases WHERE ticket_id
IN {}'''.format(ticket_numbers)
```

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

   Cases are similar with that in public info search. But we let customers see the price of the flight after they log in.

```
query = '''SELECT DISTINCT airline_name, flight_num,
departure_time , arrival_time, departure_airport,
arrival_airport, status \

                    FROM flight \
                   WHERE status = 'Upcoming' AND
departure_airport = "{}" AND arrival_airport = "{}" AND
DATE(departure_time) = '{}'
'''.format(departure_airport,arrival_airport,departure_date)

query = '''SELECT DISTINCT airline_name,flight_num
,departure_time, arrival_time, departure_airport,
arrival_airport, status  FROM flight \

                           WHERE status = 'Upcoming' AND
departure_airport IN(SELECT airport_name FROM airport WHERE
airport_city = "{}") \

                           AND arrival_airport IN (SELECT
airport_name FROM airport WHERE airport_city = "{}") \

                           AND DATE(departure_time) = '{}'
'''.format (departure_city, arrival_city, departure_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.
    a. Find the total amount of money spent in the past year

    ```
    query = '''SELECT sum(price) FROM flight NATURAL JOIN
    ticket NATURAL JOIN purchases WHERE customer_email =
    '{}' AND purchase_date >= '{}'
    '''.format(session['customer'], last_year)
    ```

    b. Find the total amount of money spent in each of the months, by defaut 6 months

    ```
    query = '''SELECT sum(price) FROM flight NATURAL JOIN
    ticket NATURAL JOIN purchases WHERE customer_email =
    '{}' AND MONTH(purchase_date) = '{}' And
    YEAR(purchase_date) = '{}'
    '''.format(session['customer'], month, year)
    ```

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.

   Default view

```
query = '''SELECT DISTINCT F.airline_name, F.flight_num,
F.departure_time , F.arrival_time, F.departure_airport,
F.arrival_airport, F.status, \

                   P.customer_email AS customer_email\

                   FROM flight as F, ticket as T, purchases as
P,booking_agent as B \

                   WHERE F.flight_num = T.flight_num AND
T.ticket_id = P.ticket_id AND B.booking_agent_id =
P.booking_agent_id AND B.email = '{}' \

                   GROUP BY
F.airline_name,F.flight_num'''.format(username)
```

   With time range entered,

```
query = '''SELECT DISTINCT F.airline_name, F.flight_num,
F.departure_time , F.arrival_time, F.departure_airport,
F.arrival_airport, F.status, \

                   P.customer_email as customer_email \
                   FROM flight as F, ticket as T, purchases as P,
booking_agent as B \

                   WHERE F.flight_num = T.flight_num AND
T.ticket_id = P.ticket_id AND B.booking_agent_id =
P.booking_agent_id AND B.email = '{}' \

                       AND (DATE(F.departure_time) BETWEEN "{}"
AND "{}") \

                   GROUP BY
F.airline_name,F.flight_num'''.format(username, start_date,
end_date)
```

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.

   The logic is similar with customer case. The limit of ticket that an agent can bought of a flight is 30, but it is still 5 for the same customer. The query to determine if there is ticket left is the same, so it does not need to be included below. To make sure agent does not purchase for more than 30 tickets.

```
condition_query2= '''SELECT COUNT(flight_num) AS ticket_bought
FROM ticket NATURAL JOIN purchases NATURAL JOIN booking_agent\

                           WHERE email = '{}' AND flight_num = {}
AND airline_name = '{}' \

                           GROUP BY
flight_num,airline_name'''.format(username,flight_num,airline_nam
e)
```

   After the verification process, we need to insert information to database.

```
query = '''SELECT MAX(ticket_id)+1 AS new_ticket_id FROM
ticket'''
```

```
prepare_query = '''SELECT booking_agent_id FROM booking_agent
WHERE email = '{}' '''.format(username)

query1 = ''' INSERT INTO ticket (ticket_id,
airline_name,flight_num) VALUES ('{}','{}',{});
'''.format(ticket_id,airline_name,flight_num)

query2 = '''INSERT INTO purchases (ticket_id, customer_email,
booking_agent_id, purchase_date, credit_card_number, figure)
VALUES ('{}','{}',{},'{}','{}','{}')
'''.format(ticket_id,cus_email,agent_id, purchase_date,ccn,price)
```

And display success message,

```
if len(ticket_num_list) == 1:

               query = '''SELECT * FROM purchases WHERE ticket_id
= {}'''.format(ticket_num_list[0])

           else:

               ticket_numbers = tuple(ticket_num_list)

               query = '''SELECT * FROM purchases WHERE ticket_id
IN {}'''.format(ticket_numbers)
```

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

   Same with customer case, so not included here.

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.
   a. By default, past 30 days

```
query = '''SELECT SUM(flight.price)*0.1,
COUNT(ticket.ticket_id) FROM ticket NATURAL JOIN flight
NATURAL JOIN purchases, booking_agent

               WHERE purchases.booking_agent_id =
booking_agent.booking_agent_id

               AND booking_agent.email='{}'

               AND purchases.purchase_date >= '{}'
'''.format(session['agent'], last_month)
```

b. If user specified a range of dates

```
query = '''SELECT SUM(flight.price)*0.1,
COUNT(ticket.ticket_id) FROM ticket NATURAL JOIN flight
NATURAL JOIN purchases, booking_agent

               WHERE purchases.booking_agent_id =
booking_agent.booking_agent_id

               AND booking_agent.email='{}'

               AND purchases.purchase_date >= '{}' AND
purchases.purchase_date <= '{}'
'''.format(session['agent'], start_date, end_date)
```

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.

```
#query Top five customer with number of tickets

query ='''SELECT purchases.customer_email,
COUNT(ticket.ticket_id) FROM purchases NATURAL JOIN ticket
NATURAL JOIN flight, booking_agent WHERE
booking_agent.booking_agent_id = purchases.booking_agent_id
```

```
AND booking_agent.email = '{}' AND purchases.purchase_date >=
'{}' GROUP BY
purchases.customer_email'''.format(session['agent'],
last_month)
```

```
#query Top five customer with commissions

query ='''SELECT purchases.customer_email, SUM(flight.price)
FROM purchases NATURAL JOIN ticket NATURAL JOIN flight,
booking_agent WHERE booking_agent.booking_agent_id =
purchases.booking_agent_id AND booking_agent.email = '{}' AND
purchases.purchase_date >= '{}' GROUP BY
purchases.customer_email'''.format(session['agent'],
last_year)
```

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:
First we update the airline and permission information in to session right after log in

```
query = '''SELECT airline_name FROM airline_staff WHERE username
='{}' '''.format(session['staff'])
query = '''SELECT permission_type FROM airline_staff NATURAL JOIN
permission WHERE username = '{}' '''.format(session['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.
Default view,

```
query = '''SELECT DISTINCT airline_name, flight_num,
departure_time , arrival_time, departure_airport,
arrival_airport, status \
                   FROM flight WHERE airline_name = '{}'  AND
(DATE(departure_time) BETWEEN "{}" AND
"{}")'''.format(staff_airline_name, start_date, end_date)
```
To view flight detail, the staff can click on the detail button and enter a new page for detail and remaining seats.

```
query1 = '''SELECT P.customer_email, T.ticket_id,
P.purchase_date, P.booking_agent_id FROM ticket AS T, purchases
as P \
               WHERE T.ticket_id = P.ticket_id AND T.airline_name
= '{}' AND T.flight_num = '{}'
'''.format(detail_airline_name,detail_flight_num)
query2 = '''SELECT A.seats - COUNT(T.ticket_id) AS
remaining_seats FROM airplane AS A, ticket as T, flight as F \
               WHERE F.airline_name = '{}' AND F.flight_num = {}
AND F.airplane_id = A.airplane_id AND T.airline_name =
F.airline_name \
               AND T.flight_num = F.flight_num
'''.format(detail_airline_name,detail_flight_num)
```
If there has not been tickets created for the flight, seats remaining is derived from

```
query2 = '''SELECT A.seats AS remaining_seats FROM airplane AS A,
flight as F \
                   WHERE F.airline_name = '{}' AND F.flight_num =
{} AND F.airplane_id =
A.airplane_id'''.format(detail_airline_name,detail_flight_num)
```

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.

   We validate the permission using information in session. First we want to make sure the creation is legal.

   To make sure airports exist

```
query = '''SELECT airport_name FROM airport'''
```

To make sure flight number has no duplicate within the airline. Also we make other case handling like you can't use an airplane when it is operating another flight. So we need most information in flight.

```
query = '''SELECT * FROM flight'''
```

After the validation, the insertion is

```
query = '''INSERT INTO flight VALUES
('{}',{},'{}','{}','{}','{}',{},'{}',{} )
'''.format(airline_name,int(flight_num),departure_airport,departu
re_time, arrival_airport,arrival_time,Decimal(price),status,
int(airlane_id))
```

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.

   We validate the permission using information in session.

   To make sure the flight exists,

```
query = '''SELECT flight_num FROM flight WHERE airline_name ='{}'
'''.format(airline_name)
```

   After this validation, update is,

```
query = '''UPDATE flight SET status = '{}' WHERE flight_num = {}
AND airline_name = '{}'
'''.format(status,flight_num,airline_name)
```

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.

We validate the permission using information in session. Then we confirm there is no duplicate plane id.

```
query = '''SELECT airplane_id FROM airplane'''
```

If validation success, the insertion is

```
query = '''INSERT INTO airplane VALUES ('{}',{},{})
'''.format(session['airline_name'],airplane_id,seats)
```

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.)

   We validate the permission using information in session. Then check duplicate.

```
query = '''SELECT airport_name FROM airport'''
```

   If validation success, insertion is

```
query = '''INSERT INTO airport VALUES ('{}','{}')
'''.format(airport_name,airport_city)
```

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.

```
#query Top five agent with number of tickets last month

       query ='''SELECT booking_agent.email, COUNT(purchases.ticket_id) FROM
purchases, booking_agent, booking_agent_work_for

               WHERE booking_agent.booking_agent_id =
purchases.booking_agent_id AND booking_agent.email =
booking_agent_work_for.email

               AND purchases.purchase_date >= '{}' AND
booking_agent_work_for.airline_name = '{}'
```

```
               GROUP BY booking_agent.email'''.format(last_month,
session['airline_name'])
```

```
#query Top five agent with number of tickets last year

       query ='''SELECT booking_agent.email, COUNT(purchases.ticket_id) FROM
purchases, booking_agent, booking_agent_work_for

               WHERE booking_agent.booking_agent_id =
purchases.booking_agent_id AND booking_agent.email =
booking_agent_work_for.email

               AND purchases.purchase_date >= '{}' AND
booking_agent_work_for.airline_name = '{}'

               GROUP BY booking_agent.email'''.format(last_year,
session['airline_name'])
```

```
#query Top five agent with commissions

       query ='''SELECT booking_agent.email, SUM(flight.price) FROM
purchases NATURAL JOIN ticket NATURAL JOIN flight, booking_agent NATURAL
JOIN booking_agent_work_for

               WHERE booking_agent.booking_agent_id =
purchases.booking_agent_id AND booking_agent_work_for.airline_name = '{}'
AND purchases.purchase_date >= '{}'

               GROUP BY
booking_agent.email'''.format(session['airline_name'], 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.

   First, for the most frequent customer, check if there is customer for the airline in last year,

```
query = '''SELECT COUNT(ticket_id) FROM purchases NATURAL JOIN
ticket NATURAL JOIN flight

WHERE airline_name = '{}' AND purchases.purchase_date >= '{}'
```

```
GROUP BY customer_email'''.format(airline_name, start_date,
airline_name,start_date)
```

Then, display a message if there is no customer, run the query below if there is,

```
query = '''SELECT customer_email FROM purchases NATURAL JOIN
ticket NATURAL JOIN flight

                   WHERE airline_name = '{}' AND
purchases.purchase_date >= '{}' GROUP BY customer_email

                   HAVING COUNT(ticket_id)>=all(SELECT
COUNT(ticket_id) FROM purchases NATURAL JOIN ticket NATURAL JOIN
flight

                   WHERE airline_name = '{}' AND
purchases.purchase_date >= '{}'

                   GROUP BY
customer_email)'''.format(airline_name,start_date,airline_name,st
art_date)
```

Second, display the customer info by entering email

```
query = '''SELECT email FROM customer'''
```

If customer's email is not in this result, display error message, else, run the query below,

```
query = '''SELECT flight_num, ticket_id, purchase_date FROM
ticket NATURAL JOIN purchases WHERE customer_email = '{}'
'''.format(customer_email)
```

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.

```
query = '''SELECT COUNT(ticket_id) FROM ticket NATURAL JOIN purchases WHERE
airline_name = '{}' AND MONTH(purchase_date) = '{}' And YEAR(purchase_date)
= '{}' '''.format(session['airline_name'], month, year)
```

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.

```
#Direct Sales last month

query ='''SELECT SUM(price) FROM purchases NATURAL JOIN ticket NATURAL JOIN
flight WHERE airline_name = '{}' AND booking_agent_id is Null AND
purchases.purchase_date >= '{}' '''.format(session['airline_name'],
last_month)

#Direct Sales last year

query ='''SELECT SUM(price) FROM purchases NATURAL JOIN ticket NATURAL JOIN
flight WHERE airline_name = '{}' AND booking_agent_id is Null AND
purchases.purchase_date >= '{}' '''.format(session['airline_name'],
last_year)

#Indirect Sales last month

query ='''SELECT SUM(price) FROM purchases NATURAL JOIN ticket NATURAL JOIN
flight WHERE airline_name = '{}' AND booking_agent_id is not Null AND
purchases.purchase_date >= '{}' '''.format(session['airline_name'],
last_month)



#Indirect Sales last year

query ='''SELECT SUM(price) FROM purchases NATURAL JOIN ticket NATURAL JOIN
flight WHERE airline_name = '{}' AND booking_agent_id is not Null AND
purchases.purchase_date >= '{}' '''.format(session['airline_name'],
last_year)
```

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

```
 #Query Top 3 destination last month

query ='''SELECT airport.airport_name, COUNT(ticket.ticket_id) FROM
purchases NATURAL JOIN ticket NATURAL JOIN flight, airport WHERE
airline_name = '{}' AND flight.arrival_airport = airport.airport_name AND
```

```
purchases.purchase_date >= '{}' GROUP BY
airport.airport_name'''.format(session['airline_name'], last_month)

#Query Top 3 destination last year

query ='''SELECT airport.airport_name, COUNT(ticket.ticket_id) FROM
purchases NATURAL JOIN ticket NATURAL JOIN flight, airport WHERE
airline_name = '{}' AND flight.arrival_airport = airport.airport_name AND
purchases.purchase_date >= '{}' GROUP BY
airport.airport_name'''.format(session['airline_name'], 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.)
    a. Check if there exist the airline staff in the system

    ```
    query='''SELECT * FROM airline_staff WHERE airline_name = '{}' AND
    username = '{}' '''.format(session['airline_name'], username)
    ```

    b. Authorize "Admin" Permission

    ```
    #Check if already has the permission

    query='''SELECT * FROM permission WHERE username = '{}' AND
    permission_type = 'Admin' '''.format(username)

    #If not, create permission

    query = '''INSERT INTO `permission`(`username`, `permission_type`)
    VALUES ('{}','Admin')'''.format(username)
    ```

    c. Authorize "Operator" Permission

    ```
    #Check if already has the permission

    query='''SELECT * FROM permission WHERE username = '{}' AND
    permission_type = 'Operator' '''.format(username)
    ```

```
#If not, create permission

query = '''INSERT INTO `permission`(`username`, `permission_type`)
VALUES ('{}','Operator')'''.format(username)
```

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.)

```
#Check if the booking agent exist in the system
query='''SELECT * FROM booking_agent WHERE email = '{}' '''.format(email)

#Check if the booking agent has already been added
query='''SELECT * FROM booking_agent_work_for WHERE email = '{}' AND
airline_name = '{}' '''.format(email, session['airline_name'])

#Add booking agents
query = '''INSERT INTO `booking_agent_work_for`(`email`, `airline_name`)
VALUES ('{}','{}')'''.format(email, session['airline_name'])
```

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