Notes:

1. **Each bullet point is a user case**
2. **Green are queries**
3. **Purple is user input/ output from previous query result/ session information, will specify if it means the latter two**
4. **Orange is explanation**
5. **Past month vs last month: If the current date is 5/16. In here we define past month as 4/16 - 5/16 and last month as 4/1 - 4/30. Same rules for years.**
6. **One column was added into the flight table, called tickets_num, in order to track how many tickets left when buying tickets**

Public Access

- Get to the initial home page

```
# Define a route to Home page
@app.route('/')
def home_page():
    return render_template('index.html')
```

- Get to the universal login page (where users can choose their identities and further use relevant information to log in)

```
# Define route for login
@app.route('/login')
def login():
    return render_template('log_in.html')
```

- Get to the universal login page (where users can choose their identities and further user relevant information to register)

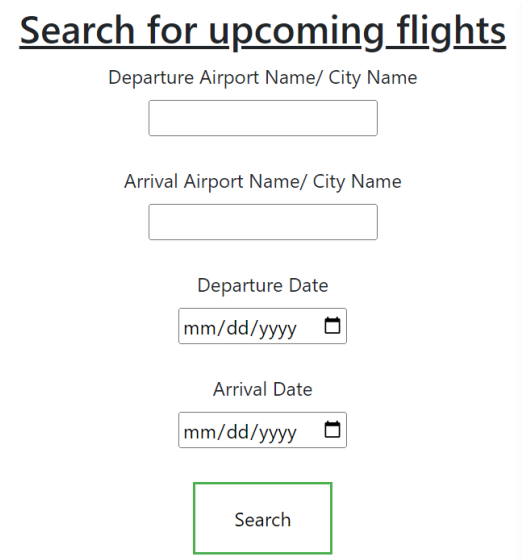
```
# Define route for register
@app.route('/register')
def register():
    return render_template('register.html')
```

- Get to the public search flight page

```
# Define route for checking flights
@app.route('/flight')
def flight():
    return render_template('flight.html')
```

- Public search flight

```
@app.route('/publicSearchFlight', methods=['GET', 'POST'])
def publicSearchFlight():
```



SELECT *
FROM flight
WHERE DATE(departure_time) = start_date(departure date in the interface)
AND DATE(arrival_time) = end_date(arrival date in the interface)
AND departure_airport IN " + depart_airport_str +
" AND arrival_airport IN " + arrive_airport_str
depart_airport_str is & arrive_airport_str the result of codes that:

1) return airport name if the user's input is airport name
2) return corresponding airport name(s) (maybe multiple) if the user's input is a city name

- Public search flight status

```
@app.route('/publicSearchFlightStatus', methods=['GET', 'POST'])
def publicSearchFlightStatus():
```

SELECT airline_name, flight_num, airplane_id, status
FROM flight
WHERE flight_num = flight_number
AND date(departure_time) = departure_date
AND date(arrival_time) = arrive_date

Customers

Home | View My Flights | Search & Purchase Tickets | Track My Spending | Log Out

- Get to customer home page, default will show customer's flights in the next 30 days

```
@app.route('/chome')
def c_home():
    email = session['email']
```

Select customer's flights in the next 30 days:
SELECT *
FROM flight NATURAL JOIN ticket NATURAL JOIN purchases
WHERE customer_email = email (from session)
AND departure_time BETWEEN CURDATE() AND DATE_ADD(CURDATE(), INTERVAL 30 DAY)

Render c_home.html

- Log in authentication

```
@app.route('/customer_login')
def c_login():
    return render_template('c_login.html')
```

Render c_login.html

```
# Authenticates the login of customer
@app.route('/cloginAuth', methods=['GET', 'POST'])
def cloginAuth():
```

1) Render c_home.html with email session and flights information if the user exists and email matches with the password
2) Show an error message if the user does not exist/ wrong password

Authenticate customer:
SELECT * FROM customer
WHERE email = email and password = md5(password)

Select customer's flights in the next 30 days:
SELECT *
FROM flight NATURAL JOIN ticket NATURAL JOIN purchases
WHERE customer_email = email AND departure_time BETWEEN CURDATE() AND DATE_ADD(CURDATE(), INTERVAL 30 DAY)

- Register authentication

```
@app.route('/customer_register')
def c_register():
    return render_template('c_register.html')
```

Render c_register.html

```
@app.route('/cregisterAuth', methods=['GET', 'POST'])
def cregisterAuth():
```

1) Render c_login.html if the user input all valid information and the user does not exist before.
2) Show error message if register information is incomplete/ user already exist

Check whether user already exists
SELECT * FROM customer WHERE email = email(from session)
Inser new user information to the database
INSERT INTO customer VALUES (email, name, md5(password), building_number, street, city, state, phone_number, passport_number, passport_expiration, passport_country, date_of_birth))

- View upcoming flights of customers

```
@app.route('/cview')
def c_view():
    return render_template('c_view.html')
```

Render c_view.html

```
@app.route('/cviewshow', methods=['GET', 'POST'])
def c_view_show():
```

Users can search **their own flights** based on
1) departure date & arrival date
2) departure location & arrival location (airport or city name)
3) departure date & arrival date & departure location & arrival location

Select flights based on dates
SELECT *
FROM flight NATURAL JOIN ticket NATURAL JOIN purchases
WHERE customer_email = email(from session) AND departure_time >= departure_date AND arrival_time <= arrival_date

Select flights based on locations
SELECT *
FROM flight NATURAL JOIN ticket NATURAL JOIN purchases WHERE customer_email = email(from session) AND departure_airport IN " \
+ depart_airport_str + " AND arrival_airport IN " + arrive_airport_str
Depart_airport_str & arrive_airport_stre: airport name if input is airport name, airport name(s) of the city if input is city name

Select flights based on dates and locations
SELECT *
FROM flight NATURAL JOIN ticket NATURAL JOIN purchases

WHERE customer_email = email(from session) AND departure_time >= departure_date AND arrival_time <= arrival_date
AND departure_airport IN "+ depart_airport_str + " AND arrival_airport IN " + arrive_airport_str

- Search flights & Purchase Tickets

```
@app.route('/csearch_purchase')
def c_search_purchase():
```

Render c_search_purchase.html

```
@app.route('/csearch', methods = ['GET','POST'])
def c_search():
```

Users can search **all the flights** based on
1) departure date & arrival date
2) departure location & arrival location (airport or city name)
3) departure date & arrival date & departure location & arrival location
Render c_search_purchase.html with search results

Select flights based on dates
SELECT *
FROM flight NATURAL JOIN ticket NATURAL JOIN purchases
WHERE departure_time >= departure_date AND arrival_time <= arrival_date

Select flights based on locations
SELECT *
FROM flight NATURAL JOIN ticket NATURAL JOIN purchases
WHERE departure_airport IN " \
    + depart_airport_str + " AND arrival_airport IN " + arrive_airport_str
Depart_airport_str & arrive_airport_stre: airport name if input is airport name, airport name(s) of the city if input is city name

Select flights based on dates and locations
SELECT *
FROM flight NATURAL JOIN ticket NATURAL JOIN purchases
WHERE departure_time >= departure_date AND arrival_time <= arrival_date
AND departure_airport IN "+ depart_airport_str + " AND arrival_airport IN " + arrive_airport_str

```
@app.route('/cpurchase', methods = ['GET','POST'])
def c_purchase():
```

Users can choose from a collection of existing airline names, a collection of existing flights (using a drop down list) to purchase flights
Render c_search_purchase with either a successfully purchased message or a message with error: flight chosened doesn't match with airline chosened/ no more seats

Find existing airlines and flights:
SELECT airline_name FROM airline
SELECT flight_num FROM flight

Validate flights (Make sure the flight chosened is from the airline chosened:
SELECT * FROM flight WHERE airline_name = airline_name AND flight_num = flight_number

Find the current last ticket ID
SELECT max(ticket_id) FROM ticket

Check whether there are still tickets left:
SELECT *
FROM flight
WHERE airline_name = airline_name AND flight_num = flight_number AND tickets_num > 0

Insert a new ticket to the database
INSERT INTO ticket VALUES (new_ticket_id, airline_name, flight_number)

Insert a new purchase record
INSERT INTO purchases VALUES (new_ticket_id, email(from session), None, CURDATE())

- Track my spending

```
@app.route('/cspending')
def c_spending():
    return render_template('c_spending.html')
```

Render c_spending.html

```
@app.route('/cshowspending', methods =['GET','POST'])
def c_show_spending():
```

User can choose view spending (in a barchart) based on last month/ last year/ time range

Total spending in last month:

SELECT SUM(price) as spending , MONTH(CURDATE())-1 as month
FROM ticket NATURAL JOIN purchases NATURAL JOIN flight
WHERE customer_email = email(from session) AND purchase_date BETWEEN DATE_ADD(LAST_DAY(DATE_ADD(CURDATE(), INTERVAL - 2 MONTH)), INTERVAL 1 DAY) and LAST_DAY(DATE_ADD(CURDATE(), INTERVAL - 1 MONTH))"

Total spending in last year:
SELECT SUM(price) as spending
FROM ticket NATURAL JOIN purchases NATURAL JOIN flight
WHERE customer_email = email(from session) AND YEAR(purchase_date) = YEAR(CURDATE())-1

Each month spending in last year:
SELECT SUM(price) as spending, YEAR(purchase_date) as year, MONTH(purchase_date) as month
FROM ticket NATURAL JOIN purchases NATURAL JOIN flight
WHERE customer_email = email(from session) AND YEAR(purchase_date) = YEAR(CURDATE())-1
GROUP BY MONTH(purchase_date), YEAR(purchase_date)

Total spending in the time range:
SELECT SUM(price) as spending FROM ticket NATURAL JOIN purchases NATURAL JOIN flight WHERE customer_email = email(from session) AND purchase_date BETWEEN start_date AND end_date

Each month spending in the time range:
SELECT SUM(price) as spending, YEAR(purchase_date) as year, MONTH(purchase_date) as month
FROM ticket NATURAL JOIN purchases NATURAL JOIN flight
WHERE customer_email = email(from session) AND purchase_date BETWEEN start_date AND end_date
GROUP BY MONTH(purchase_date), YEAR(purchase_date)

Booking Agent

Home | View My Flights | Search for Flights & Purchase Tickets | View my Commission | View Top5 Customers | Log Out

- Get to booking agent home page

```
@app.route('/bhome')
def b_home():
```

Default will be showing upcoming flights bought by booking agents

Select upcoming flights bought by booking agent:
SELECT *
FROM purchases NATURAL JOIN ticket NATURAL JOIN booking_agent JOIN flight USING (flight_num)
WHERE booking_agent.email = email(from session) AND flight.status ='Upcoming'

- Log in authentication

```
@app.route('/booking_agent_login')
def b_login():
    return render_template('b_login.html')
```

Render b_login.html

```
# Authenticates the login of customer
@app.route('/bloginAuth', methods=['GET', 'POST'])
def bloginAuth():
```

1) Render b_home.html with email session and flights information if the user exists and email matches with the password
2) Show an error message if the user does not exist/ wrong password

Authenticate agent:
SELECT * FROM booking_agent
WHERE email = email and password = md5(password)

Select upcoming flights bought by agents:
SELECT *
FROM purchases NATURAL JOIN ticket NATURAL JOIN booking_agent JOIN flight USING (flight_num)
WHERE booking_agent.email = email AND flight.status ='Upcoming'

- Register authentication

```
@app.route('/booking_agent_register')
def b_register():
    return render_template('b_register.html')
```

Render b_register.html

```
#Authenticates the register of booking agent
@app.route('/bregisterAuth', methods=['GET', 'POST'])
def bregisterAuth():
```

1) Render b_login.html if the user input all valid information and the user does not exist before.

2) Show error message if register information is incomplete/ user already exist

Check whether user (email/ booking_agent id) already exists
SELECT * FROM booking_agent WHERE email = email(from session)
SELECT * FROM booking_agent WHERE booking_agent_id = booking_agent_id
Inser new agent information to the database
INSERT INTO booking_agent VALUES(email(from session), md5(password), booking_agent_id)

- View upcoming flights (bought by booking agents)

```
@app.route("/bview")
def b_view():
    return render_template("b_view.html")
```

Render b_view.html

```
@app.route("/bviewshow", methods=['GET', 'POST'])
def b_view_show():
```

Agentsrs can search **flights that they bought** based on
1) departure date & arrival date
2) departure location & arrival location (airport or city name)
3) departure date & arrival date & departure location & arrival location

Select flights based on dates
SELECT *
FROM purchases NATURAL JOIN ticket NATURAL JOIN booking_agent JOIN flight USING (flight_num)
WHERE booking_agent.email = email(from session) AND departure_time >= departure_date AND arrival_time <= arrival_date

Select flights based on locations
SELECT *
FROM purchases NATURAL JOIN ticket NATURAL JOIN booking_agent JOIN flight USING (flight_num)
WHERE booking_agent.email = email(from session) AND departure_airport IN " + depart_airport_str + " AND arrival_airport IN " + arrive_airport_str
Depart_airport_str & arrive_airport_stre: airport name if input is airport name, airport name(s) of the city if input is city name

Select flights based on dates and locations
SELECT *
FROM purchases NATURAL JOIN ticket NATURAL JOIN booking_agent JOIN flight USING (flight_num)

WHERE booking_agent.email = email(from session) AND departure_time >= departure_date AND arrival_time <= arrival_date AND departure_airport IN " + depart_airport_str + " AND arrival_airport IN " + arrive_airport_str

- Search flights & Purchase Tickets

```
@app.route('/bsearch_purchase')
def b_search_purchase():
```

Render b_search_purchase.html

```
@app.route('/bsearch', methods=['GET', 'POST'])
def b_search():
```

Agentss can search **all the flights** based on
1) departure date & arrival date
2) departure location & arrival location (airport or city name)
3) departure date & arrival date & departure location & arrival location
Render c_search_purchase.html with search results

Select flights based on dates
SELECT *
FROM purchases NATURAL JOIN ticket NATURAL JOIN booking_agent JOIN flight USING (flight_num)
WHERE departure_time >= departure_date AND arrival_time <= arrival_date

Select flights based on locations
SELECT *
FROM purchases NATURAL JOIN ticket NATURAL JOIN booking_agent JOIN flight USING (flight_num)
WHERE departure_airport IN " + depart_airport_str + " AND arrival_airport IN " + arrive_airport_str
Depart_airport_str & arrive_airport_stre: airport name if input is airport name, airport name(s) of the city if input is city name

Select flights based on dates and locations
SELECT *
FROM purchases NATURAL JOIN ticket NATURAL JOIN booking_agent JOIN flight USING (flight_num)
WHERE departure_time >= departure_date AND arrival_time <= arrival_date AND departure_airport IN " + depart_airport_str + " AND arrival_airport IN " + arrive_airport_str

```
@app.route('/bpurchase', methods=['GET', 'POST'])
def b_purchase():
```

Users can choose from a collection of existing airline names especially that agents work for, a collection of existing flights (using a drop down list) to purchase flights. Render b_search_purchase with either a successfully purchased message or a message with error: flight chosened doesn't match with airline chosened/ no more seats/ customers haven't registered

Find existing airlines that booking works for and corresponding flights:
SELECT airline_name FROM booking_agent_work_for WHERE email = email(from session)
SELECT flight_num FROM flight WHERE airline_name IN + airline_for_agent (not user input, but derived from the last query)

Validate flights (Make sure the flight chosened is from the airline chosened):
SELECT * FROM flight WHERE airline_name = airline_name AND flight_num = flight_number

Validate customers: (Make sure customer exists)
SELECT * FROM customer WHERE email = email(from session)

Find booking agent ID:
SELECT booking_agent_id FROM booking_agent WHERE email = email(from session)

Find the current last ticket ID
SELECT max(ticket_id) FROM ticket

Check whether there are still tickets left:
SELECT *
FROM flight
WHERE airline_name = airline_name AND flight_num = flight_number AND tickets_num > 0

Insert a new ticket to the database
INSERT INTO ticket VALUES (new_ticket_id, airline_name, flight_number)

Insert a new purchase record
INSERT INTO purchases VALUES (new_ticket_id, email(from session), booking_agent_id (not user input, but derived from find booking agent ID query), CURDATE())

- View commission

```
@app.route('/bcommission')
def b_commission():
```

Render b_commission.html, Default will be showing the total commission, average commission, total tickets sold in the past 30 days.

Find the booking agent ID:
SELECT booking_agent_id
FROM booking_agent
WHERE email = email(from session)

Find total commission, average commission, total tickets sold in the past 30 days:
SELECT sum(price)*0.1, avg(price)*0.1, count(ticket_id)
FROM ticket NATURAL JOIN purchases NATURAL JOIN flight
WHERE booking_agent_id = booking_agent_id (derived from the last query) AND purchase_date between DATE_ADD(NOW(), INTERVAL -'30' DAY) and NOW()

```
@app.route('/bcommissionwdate', methods=['GET', 'POST'])
def b_commission_with_date():
```

Render b_commission.html, user can specify a time range to search for their commission.

Find total commission, average commission, total tickets sold in theday range:
SELECT sum(price)*0.1, avg(price)*0.1, count(ticket_id)
FROM ticket NATURAL JOIN purchases NATURAL JOIN flight
WHERE booking_agent_id = booking_agent_id (derived from the find booking agent query) AND purchase_date >= start_date AND purchase_date <= end_date

- View Top 5 customers

```
@app.route('/btopcustomer')
def b_topcustomer():
```

Render b_topcustomer.html, showing the results in a barchat.

Find the booking agent ID:
SELECT booking_agent_id FROM booking_agent
WHERE email = email(from session)

Select top5 customers of tickets in the past six months (not include current month)
SELECT customer_email as Customer , count(ticket_id) as Tickets
FROM purchases

WHERE booking_agent_id = booking_agent_id (derived from the find booking agent query) AND purchase_date between
DATE_ADD(LAST_DAY(DATE_ADD(CURDATE(), INTERVAL - 7 MONTH)), INTERVAL 1 DAY) and LAST_DAY(DATE_ADD(CURDATE(), INTERVAL - 1 MONTH))
GROUP BY customer_email
ORDER BY count(ticket_id) desc
LIMIT 5

Select top5 customers of commission in the last year (2021)
SELECT customer_email as Customer, sum(price)*0.1 as Commission
FROM ticket NATURAL JOIN purchases NATURAL JOIN flight
WHERE booking_agent_id = %s AND
YEAR(purchase_date) = YEAR(CURDATE())-1
GROUP BY customer_email
ORDER BY sum(price) desc
LIMIT 5

Airline staff

- Get to staff home page

```
@app.route('/ahome')
def a_home():
```

Render a_home.html. Default will be showing the next 30 days upcoming flights from the airline that the staff works for

Find airline that the staff works for
SELECT airline_name FROM airline_staff WHERE username = username(from session)

Select next 30 days upcoming flights from the airline that agents works for:
SELECT flight_num, airplane_id, airline_name, departure_airport, arrival_airport, departure_time, arrival_time, price
FROM flight
WHERE airline_name = airline_name AND departure_time BETWEEN CURDATE() AND DATE_ADD(CURDATE(), INTERVAL 30 DAY)

- Log in authentication

```python
@app.route('/staff_login')
def a_login():
    return render_template('a_login.html')
```

Render a_login.html

```python
# Authenticates the login of customer
@app.route('/aloginAuth', methods=['GET', 'POST'])
def aloginAuth():
```

1) Render a_home.html with username session if the user exists and email matches with the password
2) Show an error message if the user does not exist/ wrong password

Authenticate agent:
SELECT * FROM airline_staff
WHERE username = username(from session) and password = md5(password)

Find airline that the staff works for
SELECT airline_name FROM airline_staff WHERE username = username(from session)

Select next 30 days upcoming flights from the airline that agents works for:
SELECT flight_num, airplane_id, airline_name, departure_airport, arrival_airport, departure_time, arrival_time, price
FROM flight
WHERE airline_name = airline_name AND departure_time BETWEEN CURDATE() AND DATE_ADD(CURDATE(), INTERVAL 30 DAY)

- Register authentication

```python
@app.route('/staff_register')
def a_register():
    return render_template('a_register.html')
```

Render a_register.html

```python
#Authenticates the register of airline_staff
@app.route('/aregisterAuth', methods=['GET', 'POST'])
def aregisterAuth():
```

1) Render a_login.html if the user input all valid information and the user does not exist before.
2) Show error message if register information is incomplete/ user already exist/ input airlin does not exist

Check whether user (username) already exists
SELECT * FROM airline_staff WHERE username= username(from session)
Check whe therniput airline exists
SELECT airline_name FROM airline WHERE airline_name = airline_name
Inser new staff information to the databaseINSERT INTO airline_staff VALUES (username, md5(password), first_name, last_name, date_of_birth, airline_name)

- View upcoming flights of the airline that this staff works in

```
@app.route("/aview")
def a_view():
    return render_template("a_view.html")
```

Render a_view.html

```
@app.route("/aviewshow", methods=['GET', 'POST'])
def a_view_show():
```

Staff can search flights that belongs to the airline they work for based on
1) departure date & arrival date
2) departure location & arrival location (airport or city name)
3) departure date & arrival date & departure location & arrival location

Select airline name that the staff works for
SELECT airline_name FROM airline_staff WHERE username = usernamel(from session)

Select flights based on dates
SELECT flight_num, airplane_id, airline_name, departure_airport, arrival_airport, departure_time, arrival_time, price
FROM flight
WHERE airline_name = airline_name AND departure_time >= departure_date AND arrival_time <= arrival_date

Select flights based on locations
SELECT flight_num, airplane_id, airline_name, departure_airport, arrival_airport, departure_time, arrival_time, price
FROM flight WHERE departure_airport IN " + depart_airport_str + " AND arrival_airport IN " + arrive_airport_str + " AND airline_name = airline_name "
Depart_airport_str & arrive_airport_stre: airport name if input is airport name, airport name(s) of the city if input is city name

Select flights based on dates and locations
SELECT flight_num, airplane_id, airline_name, departure_airport, arrival_airport, departure_time, arrival_time, price
FROM flight WHERE airline_name = airline_name AND departure_time >= departure_date AND arrival_time <= arrival_date
AND departure_airport IN " + depart_airport_str + " AND arrival_airport IN " + arrive_airport_str

- View customers of a particular flight

```
@app.route("/asearchcusbyflight", methods=['GET', 'POST'])
def a_search_cus_by_flight():
```

Find the airline name that the staff works for
SELECT airline_name FROM airline_staff WHERE username = usernamel(from session)

Check whether the flight number user choose is valid (flight of a particular airline)
SELECT * FROM flight WHERE airline_name = airline_name(from query result, not user input) AND flight_num = flight_number
Search customers of a particular flight
SELECT customer_email, ticket_id
FROM flight NATURAL JOIN ticket NATURAL JOIN purchases
WHERE airline_name = airline_name(from query result, not user input) AND flight_num = flight_number

- Create new flights & Change the status of flights

```
@app.route("/atwochange", methods=['GET', 'POST'])
def a_change_create_flights():
```

Render a_change_create_flight.html
1) User with ‘Admin’ will see create new flights section but not change status section
2) User with ‘Operator’ will see change status of flights but not create new flights
3) User without both permission will see a message saying you don’t have the permission to create flights or change status

Check user’s permission
SELECT *
FROM permission
WHERE username = usernamel(from session)

```
@app.route("/acreate", methods=['GET', 'POST'])
def a_create_flight():
```

Render a_change_create_flight.html with either a successfully created message or error message includes:

1) Airplane ID doesn't exist
2) Departure/Arrival airport doesn't exist
3) Same departure and arrival airport
4) Flight number existed
5) Invalid dates

Find Airline Name:
SELECT airline_name FROM airline_staff WHERE username = usernamel(from session)

Find Airplane ID:
SELECT airplane_id FROM airplane WHERE airline_name = usernamel(from session)

Find airport name:
SELECT airport_name FROM airport

Find flight number:
SELECT flight_num FROM flight WHERE airline_name = airline name(from query result, not user input)

Insert new flight information:
INSERT INTO flight VALUES (airline_name(from query result, not user input), flight_number, departure_airport, departure_date, departure_time, arrival_airport, arrival_date, arrival_time, price, status, airplaneid, seats)

```
@app.route("/achange", methods=["GET", "POST"])
def a_change_status():
```

Render a_change_create_flight.html with either a successfully created message or error message: Status after changing and before changing is the same.

Find Airline Name:
SELECT airline_name FROM airline_staff WHERE username = usernamel(from session)

Find flight number:

SELECT flight_num FROM flight WHERE airline_name = airline name(from query result, not user input)

Check this current flight status
SELECT status FROM flight WHERE flight_num = flight_number

Update flight status
Update flight SET status = new_status WHERE flight_num = flight_number

- Add airplane & Add airport

```
@app.route("/atwoadd")
def a_add_airplane_airport():
```

Render a_add_airplane_airport.html
Show add airplane and airport section only if the user is an 'Admin'

Check user's permission:
SELECT *
FROM permission
WHERE username = usernamel(from session)

```
@app.route("/aplusairplane", methods=["GET", "POST"])
def a_add_airplane():
```

Render a_add_airplane_airport.html with either a successfully created message or error message: This airplane already exists.

Find Airline Name:
SELECT airline_name FROM airline_staff WHERE username = usernamel(from session)

Check Airplane ID:
SELECT * FROM airplane WHERE airline_name = airline_name(from query result, not user input) AND airplane_id = airline_id

Insert the new airplane:
INSERT INTO airplane VALUES (airline_name(from query result, not user input), airplane_id, seats)

Select all the airplane of the airline that the staff works for:
SELECT airplane_id, seats FROM airplane WHERE airline_name = airline_name(from query result, not user input)

```
@app.route("/aplusairport", methods=["GET", "POST"])
def a_add_airport():
```

Render a_add_airplane_airport.html with either a successfully created message or error message: This airport already exists.

Check whether the airport exists:
SELECT * FROM airport WHERE airport_name = airport_name

Insert new airport:
INSERT INTO airport VALUES (airport_name, airport_city)

- View top 5 booking agent

```
@app.route('/aviewagent')
def a_view_booking_agent():
```

Render a_booking_agent.html with below information in three bar charts:

Select Top5 agents based on number of tickets sold in the past month:
SELECT booking_agent.email, booking_agent_id, COUNT(ticket_id) AS ticket
FROM booking_agent NATURAL JOIN purchases NATURAL JOIN ticket AS t, airline_staff
WHERE purchase_date between DATE_ADD(NOW(), INTERVAL -'1' MONTH) and NOW() and username = usernamel(from session) and t.airline_name = airline_staff.airline_name
GROUP BY booking_agent_id
ORDER BY COUNT(ticket_id) DESC
LIMIT 5

Select Top5 agents based on number of tickets sold in the past year:
SELECT booking_agent.email, booking_agent_id, COUNT(ticket_id) AS ticket
FROM booking_agent NATURAL JOIN purchases NATURAL JOIN ticket AS t, airline_staff
WHERE purchase_date between DATE_ADD(NOW(), INTERVAL -'12' MONTH) and NOW() and username = usernamel(from session) and t.airline_name = airline_staff.airline_name
GROUP BY booking_agent_id
ORDER BY COUNT(ticket_id) DESC
LIMIT 5

Select Top5 agents based on commission earned in the past year:
SELECT email, booking_agent_id, sum(price) * 0.1 as commission

FROM booking_agent NATURAL JOIN purchases NATURAL JOIN flight NATURAL JOIN ticket AS T, airline_staff
WHERE username = usernamel(from session) and airline_staff.airline_name = T.airline_name and YEAR(purchase_date) = YEAR(CURDATE())-1
GROUP BY email, booking_agent_id
ORDER BY commission DESC
LIMIT 5

- View frequent customers

```
@app.route('/aviewcustomer')
def a_view_customer():
```

Render a_customers.html

Select the most frequent customer in the last year (who bought the most tickets)
SELECT customer_email , COUNT(customer_email) as ticket
FROM purchases NATURAL JOIN ticket as t, airline_staff
WHERE airline_staff.username = usernamel(from session) AND airline_staff.airline_name = t.airline_name AND YEAR(purchase_date) = YEAR(CURDATE())-1
GROUP BY customer_email
ORDER BY COUNT(customer_email) DESC

```
@app.route('/aviewcustomerflight', methods=['GET', 'POST'])
def a_view_customer_flight():
```

Render a_customers.html. Show a list of flights a particular customer took

Check whether the input customer exists in the database:
SELECT * FROM customer WHERE email = usernamel(from session)

Select flights for the user:
SELECT *
FROM purchases NATURAL JOIN ticket as t JOIN flight using(flight_num), airline_staff
WHERE airline_staff.username = usernamel(from session) AND airline_staff.airline_name = t.airline_name AND customer_email = customer_email

- View ticket reports

```
@app.route('/areport')
def a_report():
    return render_template("a_report.html")
```

Render a_report.html

```
@app.route('/areportshow', methods = ['GET','POST'])
def a_show_report():
```

Render a_report.html with tickets sold in last month/ last year/ time range. Month wise tickets in a bar chart.

Total amount of tickets in the last month:
SELECT COUNT(ticket_id) as ticket , MONTH(CURDATE())-1 as month
FROM ticket NATURAL JOIN purchases NATURAL JOIN airline_staff
WHERE username = usernamel(from session) AND purchase_date BETWEEN DATE_ADD(LAST_DAY(DATE_ADD(CURDATE(), INTERVAL - 2 MONTH)), INTERVAL 1 DAY) and LAST_DAY(DATE_ADD(CURDATE(), INTERVAL - 1 MONTH))

Monthly tickets in the last year:
SELECT COUNT(ticket_id) as ticket , YEAR(purchase_date) as year, MONTH(purchase_date) as month
FROM ticket NATURAL JOIN purchases NATURAL JOIN airline_staff
WHERE username = usernamel(from session) AND YEAR(purchase_date) = YEAR(CURDATE())-1
GROUP BY MONTH(purchase_date), YEAR(purchase_date)

Monthly tickets in the time range:
SELECT COUNT(ticket_id) as ticket, YEAR(purchase_date) as year, MONTH(purchase_date) as month
FROM ticket NATURAL JOIN purchases NATURAL JOIN airline_staff WHERE username = usernamel(from session) AND purchase_date BETWEEN start_date AND end_date
GROUP BY MONTH(purchase_date), YEAR(purchase_date)

- View revenue comparison

```
@app.route('/arevenue')
def a_revenue():
```

Render a_revenue.html

```
@app.route('/arevenueshow', methods = ['GET','POST'])
def a_revenue_show():
```

Render a_revenue,html with two pie charts (direct sales vs indirect sales in the last month and last yeat)

Find the airline name:
SELECT airline_name FROM airline_staff WHERE username = usernamel(from session)

Direct Sales in the last month:
SELECT SUM(price) as revenue_cus
FROM ticket NATURAL JOIN purchases NATURAL JOIN flight
WHERE airline_name = airline_name(from query result, not user input) AND booking_agent_id IS NULL AND purchase_date BETWEEN DATE_ADD(LAST_DAY(DATE_ADD(CURDATE(), INTERVAL - 2 MONTH)), INTERVAL 1 DAY) and LAST_DAY(DATE_ADD(CURDATE(), INTERVAL - 1 MONTH))

Indirect sales in the last month:
SELECT SUM(price)*0.9 as revenue_agent
FROM ticket NATURAL JOIN purchases NATURAL JOIN flight
WHERE airline_name = airline_name(from query result, not user input) AND booking_agent_id IS NOT NULL AND purchase_date BETWEEN DATE_ADD(LAST_DAY(DATE_ADD(CURDATE(), INTERVAL - 2 MONTH)), INTERVAL 1 DAY) and LAST_DAY(DATE_ADD(CURDATE(), INTERVAL - 1 MONTH))

Direct Sales in the last year:
SELECT SUM(price) as revenue_cus
FROM ticket NATURAL JOIN purchases NATURAL JOIN flight
WHERE airline_name = airline_name(from query result, not user input) AND booking_agent_id IS NULL AND YEAR(purchase_date) = YEAR(CURDATE())-1

Indirect sales in the last year:
SELECT SUM(price)*0.9 as revenue_agent
FROM ticket NATURAL JOIN purchases NATURAL JOIN flight
WHERE airline_name = airline_name(from query result, not user input) AND booking_agent_id IS NOT NULL AND YEAR(purchase_date) = YEAR(CURDATE())-1

- View top 3 destinations

```
@app.route('/adestination', methods=['GET','POST'])
def a_destination():
```

Render a_destination.html

Top3 arrival city in the past 3 months

SELECT airport_city as destination, COUNT(ticket_id)
FROM purchases NATURAL JOIN ticket NATURAL JOIN flight as t, airport
WHERE t.arrival_airport = airport.airport_name AND purchase_date between
DATE_ADD(LAST_DAY(DATE_ADD(CURDATE(), INTERVAL - 4 MONTH)),
INTERVAL 1 DAY) and LAST_DAY(DATE_ADD(CURDATE(), INTERVAL - 1
MONTH))
GROUP BY airport_city
ORDER BY COUNT(ticket_id) DESC
LIMIT 3

Top3 arrival city in the past year
SELECT airport_city as destination, COUNT(ticket_id)
FROM purchases NATURAL JOIN ticket NATURAL JOIN flight as t, airport
WHERE t.arrival_airport = airport.airport_name AND YEAR(purchase_date) =
YEAR(CURDATE())-1
GROUP BY airport_city
ORDER BY COUNT(ticket_id) DESC
LIMIT 3

- Grant new permission

```
@app.route('/apermission', methods=['GET','POST'])
def a_permission():
```

Render a_permission.html. A message saying that you don't have the permission to grant permission will show up if the user is not Admin:

Check users' permissions:
SELECT * FROM permission WHERE username = usernamel(from session)

```
@app.route('/apermissionstart', methods=['GET','POST'])
def a_permission_start():
```

Render a_permission.html with either a succuesffuly granted message or error message includes:

1) Staff doesn't exist in this database
2) Staff not in the same airline as you do
3) Duplicate granting

Check whether this staff exists
SELECT * FROM airline_staff WHERE username = staff_username

Find user's airline:

SELECT airline_name FROM airline_staff WHERE username = usernamel(from session)

Check whether they work in the same airline
SELECT username FROM airline_staff WHERE username = staff_username
AND airline_name = airline_name(from query result, not user input)

Check whether this staff already have this permission:
SELECT username FROM permission WHERE username = staff_username AND permission_type = permission_type

Grant permission:
INSERT INTO permission VALUES (staff_username, permission_type)

- Add booking agent

```
@app.route('/aaddagent')
def a_add_booking_agent():
```

Rendera_add_booking_agent.html. A message saying that you don't have the permission to grant permission will show up if the user is not Admin:

Check users' permissions:
SELECT * FROM permission WHERE username = usernamel(from session)

```
@app.route('/aaddagentstart', methods=['GET', 'POST'])
def a_add_booking_agent_start():
```

Rendera_add_booking_agent.html with either a succuesffuly granted message or error message includes:

4) Agent doesn't exist in this database
5) Agent already worked for this airline

Check whether this agent has registered:
SELECT * FROM booking_agent WHERE email = booking_agent_email

Find user's airline:
SELECT airline_name FROM airline_staff WHERE username = usernamel(from session)

Check agent's airline
SELECT airline_name FROM booking_agent_work_for WHERE email = booking_agent_email AND airline_name = airline_name(from query result, not user input)

Inset new airline for an agent:

INSERT INTO booking_agent_work_for VALUES (booking_agent_email, airline_name)