List of Use Cases and Queries

1. View Public Info:

All users, whether logged in or not, can
a. **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).

Departure flights:

```
query = "select * from flight natural join airplane, airport as A, airport as B \
        where flight.dept_from = A.name and flight.arr_at = B.name \
        and (A.name = %s or A.city = %s) and (B.name = %s or B.city = %s) \
        and date(dept_time) = %s "
cursor.execute(query, (dept_from, dept_from, arr_at, arr_at, dept_date))
```

Return flights:

```
query2 = "select * from flight natural join airplane, airport as A, airport as B \
         where flight.dept_from = A.name and flight.arr_at = B.name \
         and (A.name = %s or A.city = %s) and (B.name = %s or B.city = %s) \
         and date(dept_time) = %s "
cursor.execute(query2, ( arr_at, arr_at, dept_from, dept_from, return_date))
```

For each flight:
get the current sold ticket number for determining the current price

```
queryTicketNum = "select count(*) from ticket natural join flight natural join
             airplane where airline_name = %s and flight_num = %s and
             dept_time=%s"
cursor.execute(queryTicketNum, ( each['airline_name'], each['flight_num'],
each['dept_time']))
```

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

```
    if dept_date and arr_date:
        query = "select * from flight \
            where airline_name = %s and flight_num = %s and date(dept_time) = %s and
            date(arr_time) = %s"
        cursor.execute(query, (airline_name, flight_num, dept_date, arr_date))
    elif dept_date:
        query = "select * from flight \
            where airline_name = %s and flight_num = %s and date(dept_time) = %s"
        cursor.execute(query, (airline_name, flight_num, dept_date))
    elif arr_date:
        query = "select * from flight \
            where airline_name = %s and flight_num = %s and date(arr_time) = %s"
        cursor.execute(query, (airline_name, flight_num, arr_date))
```

2. Register:

Customer:

1. Check if the customer already exists by fetching the customer with the email input

```
query = 'SELECT * FROM customer WHERE email = %s'
cursor.execute(query, (email))
```

2. If not, create an account for this customer with form inputs and a hashed password

```
ins = 'INSERT INTO customer VALUES(%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)'
cursor.execute(ins, (email, password, name, building_num, street, city, state, phone_num, passport_num, passport_expr, passport_country, DOB, 0))
```

Agent:

1. Check if the agent email already exists by fetching the agent with the email input

```
query = 'SELECT * FROM booking_agent WHERE email = %s'
cursor.execute(query, (email))
```

2. If not, check if the agent id already exists

```
query = 'SELECT * FROM booking_agent WHERE agent_id = %s'
cursor.execute(query, (agent_id))
```

3. If not, create an account for this agent with form inputs and a hashed password

```
ins = 'INSERT INTO booking_agent VALUES(%s, %s, %s, %s)'
cursor.execute(ins, (email, password, agent_id, 0))
```

Staff:

1. Check if the staff already exists by fetching the staff with the username input

```
query = 'SELECT * FROM airline_staff WHERE username = %s'
cursor.execute(query, (username))
```

2. If not, create an account for this staff with form inputs and a hashed password, also store all his/her phone numbers

```
ins = 'INSERT INTO airline_staff VALUES(%s, %s, %s, %s, %s, %s)'
cursor.execute(ins, (username, password, first_name, last_name, DOB, airline_name))
if ";" in phone_number:
    phone_number_lst = phone_number.split(";")
    for item in phone_number_lst:
        ins = 'INSERT INTO staff_phone VALUES(%s, %s)'
        cursor.execute(ins, (username, item.strip()))
    else:
        ins = 'INSERT INTO staff_phone VALUES(%s, %s)'
        cursor.execute(ins, (username, phone_number))
```

3. Login:

Customer:

1. Try to fetch customer information with the email input
```
query = 'SELECT * FROM customer WHERE email = %s'
cursor.execute(query, (email))
data = cursor.fetchone()
```
2. If successfully fetched, compare the encoded password with the password input
```
if(sha256_crypt.verify(password, data['password']))
```
3. Else, display error message

Agent:

1. Try to fetch agent information with the email input
```
query = 'SELECT * FROM booking_agent WHERE email = %s'
cursor.execute(query, (email))
data = cursor.fetchone()
```
2. If successfully fetched, compare the encoded password with the password input
```
if(sha256_crypt.verify(password, data['password']))
```
3. Else, display error message

Staff:

1. Try to fetch staff information with the username input
```
query = 'SELECT * FROM 'SELECT * FROM airline_staff WHERE username = %s'
cursor.execute(query, (username))
data = cursor.fetchone()
```
2. If successfully fetched, compare the encoded password with the password input
```
if(sha256_crypt.verify(password, data['password']))
```
3. Else, display error message

Customer use cases:

1. View My Flights:

Default: view future flights:

```
query = "select * from ticket natural join flight natural join airport as A, airport as B where cust_email = %s and dept_time > %s and dept_from = A.name and arr_at = B.name"
cursor.execute(query, (email, current_date))
```

2. Search for Flights:

Same as search flights in Public.
(check if the flight is full by comparing the number of airplane seats and the number of sold tickets)

3. Purchase Tickets:

1. Check if the customer has already purchased a ticket for the flight

```
query = "select count(*) from rates where cust_email = %s and
((airline_name,flight_num, dept_time) = (%s, %s, %s) or (airline_name,flight_num,
dept_time) = (%s, %s, %s)) "
cursor.execute(query, (cust_email, airline_name, flight_num, dept_time,
airline_name2, flight_num2, dept_time2))
num = cursor.fetchone()
```

2. If not, insert flight data and detailed purchase information

```
query = "insert into ticket values (%s, %s, %s, %s, %s, %s, %s, null, %s, %s,
%s, %s)"
cursor.execute(query, (str(datetime.datetime.now().timestamp()), current_price,
card_type, card_num, name_on_card, expr_date, time, session['email'], airline_name,
flight_num, dept_time))
```

4. Give Ratings and Comment on previous flights:

Fetch all previous flights based on customer email:

```
query = "select * from ticket natural join flight natural join airport as A,
airport as B where cust_email = %s and dept_time < %s and dept_from = A.name
and arr_at = B.name"
cursor.execute(query, (session['email'], datetime.datetime.now()))
```

Give Comments to a certain ticket:

1. Fetch data of a certain ticket

```
query = "select * from ticket where ticket_id = %s"
cursor.execute(query, (ticket_id))
ticket = cursor.fetchone()
```

1. Fetch rating of this flight given by this customer

```
query = "select * from rates where cust_email = %s and airline_name = %s and
flight_num = %s and dept_time = %s"
cursor.execute(query, (session['email'], ticket['airline_name'],
ticket['flight_num'],ticket['dept_time']))
data = cursor.fetchone()
```

2. If there is no previous rating of this certain customer & flight, add rating and comment

```
if data == None:
    query = "insert into rates values (%s, %s, %s, %s, %s, %s)"
    cursor.execute(query, (session['email'], ticket['airline_name'],
ticket['flight_num'],ticket['dept_time'], rate, comment))
```

3. Else, update rating and comment

```
query = "update rates set rate = %s, comments = %s where cust_email = %s and
airline_name = %s and flight_num = %s and dept_time = %s"
```

```
        cursor.execute(query, (rate, comment, session['email'], ticket['airline_name'],
ticket['flight_num'], ticket['dept_time']))
```

Display detailed information of a certain ticket:

1. Fetch detailed data linked with airport of a certain ticket

```
        query = "select * from ticket natural join flight natural join airport as A,
airport as B where ticket_id = %s and dept_from = A.name and arr_at = B.name"
        cursor.execute(query, (ticket_id))
        data = cursor.fetchone()
```

2. Fetch the current rating and comment

```
        query = "select * from rates where cust_email = %s and airline_name = %s and
flight_num = %s and dept_time = %s"
        cursor.execute(query, (session['email'], data['airline_name'],
data['flight_num'], data['dept_time']))
        rates = cursor.fetchone()
```

5. Track My Spending:

Fetch the total spending within the given range:

```
        query = "SELECT COALESCE( SUM(sold_price), 0) as total_spending FROM ticket
WHERE purchase_time > %s AND purchase_time < %s AND cust_email = %s"
        cursor.execute(query, (from_date, to_date, session['email']))
```

For each month in the range, fetch the total spending:

```
        query = "SELECT COALESCE( SUM(sold_price), 0)  as monthly_spending FROM ticket
WHERE purchase_time > %s and purchase_time < %s AND cust_email = %s"
        cursor.execute(query, (temp_date, this_date, session['email']))
```

6. Log out:

```
session.clear()
```

Agent use cases:

1. View My Flights:

Default: view future flights:

```
        query = "select * from ticket natural join flight natural join airport as A,
airport as B where agent_email = %s and dept_time > %s and dept_from = A.name and
arr_at = B.name"
        cursor.execute(query, (email, current_date))
```

2. Search for flights:

Same as customer

3. Purchase Tickets:

1. Check if customer email exists

```
query = "select count(*) from customer where email = %s"
cursor.execute(query, cust_email)
```

2. If so, buy tickets successfully with detailed purchase information

```
query = "insert into ticket values (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)"
cursor.execute(query, (str(datetime.datetime.now().timestamp()), current_price, card_type, card_num, name_on_card, expr_date, time, session['email'], cust_email, airline_name, flight_num, dept_time))
```

4. View My Commission:

Default, fetch total commission for the last 30 days:

```
query = "SELECT IFNULL(SUM(sold_price) , 0) as total_price, IFNULL(COUNT(*) ,0) as ticket_num FROM ticket WHERE DATE(purchase_time) BETWEEN NOW() - INTERVAL 30 DAY AND NOW() + INTERVAL 1 DAY AND agent_email = %s"
cursor.execute(query, (session['email']))
```

Alternative, fetch total commission within a time range:

```
query = "SELECT IFNULL(SUM(sold_price) , 0) as total_price, IFNULL(COUNT(*) ,0) as ticket_num FROM ticket WHERE DATE(purchase_time) BETWEEN NOW() - INTERVAL 30 DAY AND NOW() + INTERVAL 1 DAY AND agent_email = %s"
cursor.execute(query, (session['email']))
```

5. View Top Customers:

Top customers with ticket number in the past 6 month:

```
query = "SELECT cust_email, count(*) as num FROM ticket WHERE DATE(purchase_time) BETWEEN NOW() - INTERVAL 6 MONTH AND NOW() + INTERVAL 1 DAY AND agent_email = %s GROUP BY cust_email ORDER BY num DESC LIMIT 5"
cursor.execute(query, (session['email']))
```

Top customers with total spending in the past year:

```
query = "SELECT cust_email, SUM(sold_price) as sum FROM ticket WHERE DATE(purchase_time) BETWEEN NOW() - INTERVAL 1 YEAR AND NOW() + INTERVAL 1 DAY AND agent_email = %s GROUP BY cust_email ORDER BY sum DESC LIMIT 5"
cursor.execute(query, (session['email']))
```

6. Log out:

Same as customer

Staff use cases:

1. View Flights:

Default, future flights:

```
        query = 'SELECT * FROM flight WHERE airline_name = %s AND DATE(dept_time)
BETWEEN DATE(CURRENT_TIMESTAMP) \
        AND DATE(CURRENT_TIMESTAMP) + INTERVAL 30 DAY'
        cursor.execute(query, (airline_name))
```

Flights within a given range:

```
        query = "SELECT * FROM flight NATURAL JOIN airplane,airport as A, airport as B\
        where airline_name = %s AND date(dept_time) >= %s AND date(dept_time) <= %s \
        and flight.dept_from = A.name and flight.arr_at = B.name and (A.name = %s or
A.city = %s) and (B.name = %s or B.city = %s)"
        cursor.execute(query, (airline_name, start_date, end_date, dept_from,
dept_from, arr_at, arr_at))
```

2. Create Flights:

1. Check if flight already exists with the airline name, flight number, and departure time

```
        query = "SELECT * FROM flight WHERE (airline_name, flight_num, dept_time) =
(%s, %s, %s)"
        cursor.execute(query, (airline_name, flight_num, dept_time))
```

2. If not, add flight

```
        query = "INSERT INTO flight VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s)"
        cursor.execute(query, (airline_name, flight_num, dept_time, arr_time,\
           base_price, flight_status, dept_from, arr_at, airplane_id))
```

3. Change Flight Status:

1. Get current flight status

```
        query = "SELECT * FROM flight WHERE airline_name = %s AND flight_num = %s AND
dept_time = %s"
        cursor.execute(query, (airline_name, flight_num, dept_time))
```

2. Update flight status

```
        query = "UPDATE flight SET flight_status = %s WHERE (airline_name, flight_num,
dept_time) = (%s, %s, %s)"
        cursor.execute(query, (new_status, airline_name, flight_num, dept_time))
```

4. Add Airplane:

1. Get all the airplanes of this airline

```
query = "SELECT * FROM airplane WHERE airline_name = %s"
cursor.execute(query,(airline_name))
```

2. Check duplicate

```
query = "SELECT * FROM airplane WHERE (airline_name, airplane_id) = (%s, %s)"
cursor.execute(query, (airline_name, airplane_id))
```

3. Add airplane

```
query = "INSERT INTO airplane VALUES (%s, %s, %s)"
cursor.execute(query,(airline_name,airplane_id,seats))
```

5. View Flight Rating:

Find Average Rate of One Flight:

```
query = "SELECT airline_name,flight_num, dept_time, AVG(rate) as avg_rate \
FROM rates \
WHERE (airline_name,flight_num, dept_time) = (%s, %s, %s)"
cursor.execute(query, (airline_name,flight_num, dept_time))
```

Find All the Ratings of One Flight:

```
query = "SELECT airline_name,flight_num, dept_time, cust_email, rate, comments\
    FROM rates \
    WHERE (airline_name,flight_num, dept_time) = (%s, %s, %s) "
cursor.execute(query, (airline_name,flight_num, dept_time))
```

6.View Top Booking-agents:

Top 5 booking agents by ticket sales for the past month:

```
query = "SELECT  agent_email, COUNT(*) AS total_sales FROM ticket \
WHERE agent_email IS NOT NULL AND DATE(purchase_time) BETWEEN NOW() - INTERVAL
30 DAY AND NOW() + INTERVAL 1 DAY\
GROUP BY agent_email ORDER BY total_sales DESC LIMIT 5"
```

Top 5 booking agents by ticket sales for the past year:

```
query = "SELECT  agent_email, COUNT(*) AS total_sales FROM ticket \
    WHERE agent_email IS NOT NULL AND DATE(purchase_time) BETWEEN NOW() -
INTERVAL 1 YEAR AND NOW() + INTERVAL 1 DAY\
    GROUP BY agent_email ORDER BY total_sales DESC LIMIT 5"
```

Top 5 booking agents by total commission for the past year:

```
query = "SELECT  agent_email, COUNT(*) AS total_sales FROM ticket \
```

```
        WHERE agent_email IS NOT NULL AND DATE(purchase_time) BETWEEN NOW() - INTERVAL
30 DAY AND NOW() + INTERVAL 1 DAY\
        GROUP BY agent_email ORDER BY total_sales DESC LIMIT 5"
```

7.View Frequent Customers:

Most frequent customer(s) for the past year:

```
        query = "SELECT cust_email, COUNT(*) AS travel_times FROM ticket WHERE
airline_name = %s AND DATE(purchase_time) BETWEEN NOW() - INTERVAL 1 YEAR AND NOW() +
INTERVAL 1 DAY GROUP BY cust_email"
        cursor.execute(query,(airline_name))
        data1 = cursor.fetchall()
        max_times = 0
        for each in data1:
            if each["travel_times"] > max_times:
                max_times = each["travel_times"]

        query2 = "SELECT cust_email, COUNT(*) AS travel_times FROM ticket WHERE
airline_name = %s GROUP BY cust_email HAVING travel_times = %s"
        cursor.execute(query2,(airline_name, max_times))
        data = cursor.fetchall()
```

* since mysql here does not support select clause in select clause, and considering the possibility that there may be more than one most frequent customer, we chose to find out the most frequent customer(s) using python instead of pure sql.

All flights a particular Customer has taken:

```
        query = "SELECT airline_name, flight_num, dept_time, purchase_time, sold_price,
cust_email FROM ticket WHERE cust_email = %s"
        cursor.execute(query,(email))
```

8.View Reports:

Total Sales for the Past Month:

```
        query = "SELECT DATE(NOW()) - INTERVAL 1 MONTH AS curr_prev, DATE(NOW()) AS
current, COUNT(*) AS total_sales FROM ticket WHERE date(purchase_time) between
DATE(NOW()) - INTERVAL 1 MONTH AND DATE(NOW()) + INTERVAL 1 DAY and airline_name = %s"
        cursor.execute(query, (airline_name))
```

Total Sales within a Given Range:

```
        query = "SELECT COUNT(*) as total_sales FROM ticket WHERE date(purchase_time)
>= %s AND date(purchase_time) <= %s and airline_name = %s"
        cursor.execute(query, (from_date, to_date, airline_name))
```

Total Sales for the Past Year:

```
        query = "SELECT DATE(NOW()) AS current, DATE(NOW()) - INTERVAL 1 YEAR AS
curr_prev , COUNT(*) as total_sales FROM ticket WHERE date(purchase_time) between
DATE(NOW()) - INTERVAL 1 YEAR AND DATE(NOW()) + INTERVAL 1 DAY and airline_name = %s"
        cursor.execute(query, (airline_name))
```

Sales by Month:

```
        query = "SELECT YEAR(purchase_time) as year, MONTH(purchase_time) as month,
COUNT(*) as total_sales \
    FROM ticket WHERE date(purchase_time) >= %s AND date(purchase_time) <= %s and
airline_name = %s GROUP BY year, month ORDER BY year, month ASC"
    cursor.execute(query, (from_date, to_date, airline_name))
```

9.Comparison of Revenue Earned:

Revenue comparison for the past month:

```
        query_direct = "SELECT SUM(sold_price) as total_price FROM ticket \
            WHERE agent_email IS NULL and DATE(purchase_time) BETWEEN DATE(NOW()) -
INTERVAL 1 MONTH and DATE(NOW())"
        query_indirect = "SELECT SUM(sold_price) as total_price FROM ticket \
            WHERE agent_email IS NOT NULL and DATE(purchase_time) BETWEEN DATE(NOW()) -
INTERVAL 1 MONTH and DATE(NOW())"
```

Revenue comparison for the past year:

```
        query_direct = "SELECT SUM(sold_price) as total_price FROM ticket \
                WHERE agent_email IS NULL and DATE(purchase_time) BETWEEN DATE(NOW()) -
INTERVAL 1 YEAR and DATE(NOW())"
        query_indirect = "SELECT SUM(sold_price) as total_price FROM ticket \
                WHERE agent_email IS NOT NULL and DATE(purchase_time) BETWEEN
DATE(NOW()) - INTERVAL 1 YEAR and DATE(NOW())"
```

10. View Top Destinations:

Top Three Destinations for the Past Three Months:

```
        query = "SELECT arr_at, city, count(*) as visit_time \
                FROM ticket NATURAL JOIN flight as S, airport \
                WHERE S.arr_at  = airport.name AND DATE(purchase_time) BETWEEN NOW() -
INTERVAL 3 MONTH and NOW() + INTERVAL 1 DAY \
                GROUP BY arr_at ORDER BY visit_time DESC LIMIT 3"
```

Top Three Destinations for the Past Year:

```
        query = "SELECT arr_at, city, count(*) as visit_time \
                    FROM ticket NATURAL JOIN flight as S, airport \
                    WHERE S.arr_at  = airport.name AND DATE(purchase_time) BETWEEN
NOW() - INTERVAL 1 YEAR and NOW() + INTERVAL 1 DAY\
                    GROUP BY arr_at ORDER BY visit_time DESC LIMIT 3"
```

11. Log out:

Same as customer