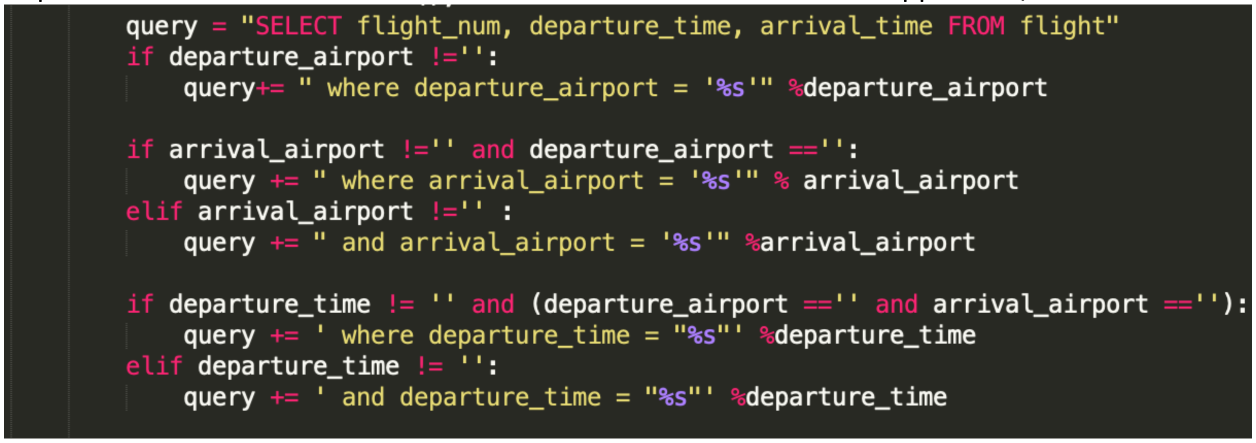
Cases and Queries

Breakdown of use cases and corresponding queries for the final project of Enver Kapetanovic and Oliver Chen. All of the queries can be found in the init1.py file.

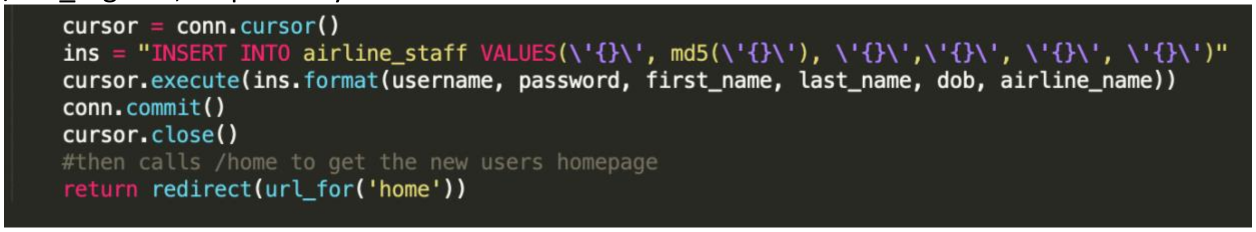
Application Use Cases (AKA features):

1. View Public Info:
   Search and view flights based on arrival airport name and/or a departure airport and/or departure date. The information is filled out as a form Found under app.route /search



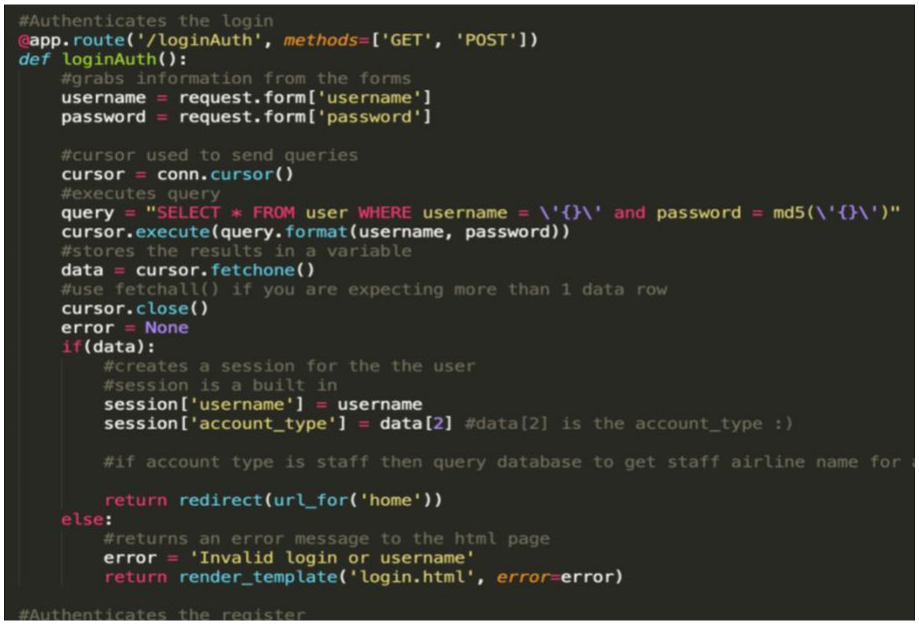
```
query = "SELECT flight_num, departure_time, arrival_time FROM flight"
if departure_airport !='':
    query+= " where departure_airport = '%s'" %departure_airport

if arrival_airport !='' and departure_airport =='':
    query += " where arrival_airport = '%s'" % arrival_airport
elif arrival_airport !='' :
    query += " and arrival_airport = '%s'" %arrival_airport

if departure_time != '' and (departure_airport =='' and arrival_airport ==''):
    query += ' where departure_time = "%s"' %departure_time
elif departure_time != '':
    query += ' and departure_time = "%s"' %departure_time
```

2. Register user:
   User registration queries differs for the 3 account types. Registration first starts at /registerAuth, which checks the database if the user already exists. Given that the user is new, the user is then redirected to a page to fill out details. After they are filled, the respective tables are updated to include the new user. Shown below is the query for airline staff registration, found at app.route /staff_register. Registration for booking agent and customer follow the same logic and can be found at /booking_register and /cus_register, respectively.



```
cursor = conn.cursor()
ins = "INSERT INTO airline_staff VALUES(\'{}\', md5(\'{}\'), \'{}\',\'{}\', \'{}\', \'{}\')"
cursor.execute(ins.format(username, password, first_name, last_name, dob, airline_name))
conn.commit()
cursor.close()
#then calls /home to get the new users homepage
return redirect(url_for('home'))
```

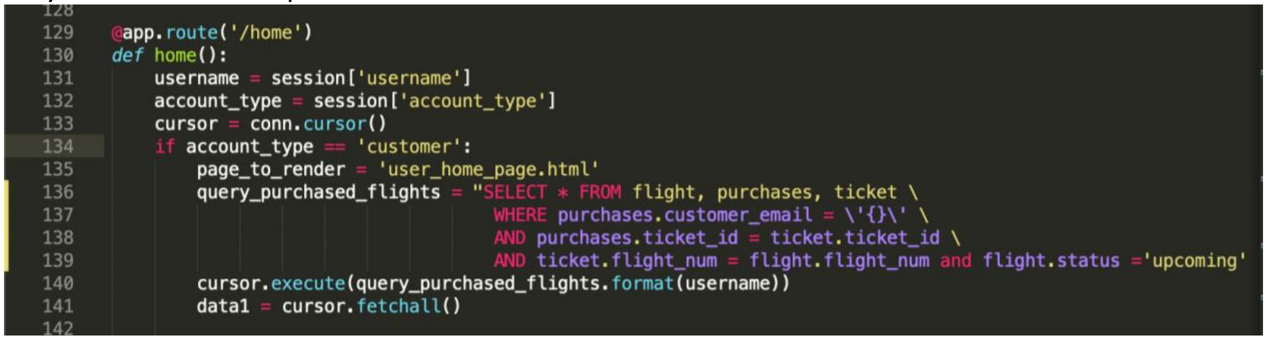
3. Login:
   The login for all account types is done as follows. First we query the "user" table, a table listing all users and account types. Given that the user has an account, the system will then set the session user name and account type, then redirect the user to their specific home page. If the user does exist, an error is shown and login page is shown again.This can be found in app.route /loginAuth



```
#Authenticates the login
@app.route('/loginAuth', methods=['GET', 'POST'])
def loginAuth():
    #grabs information from the forms
    username = request.form['username']
    password = request.form['password']

    #cursor used to send queries
    cursor = conn.cursor()
    #executes query
    query = "SELECT * FROM user WHERE username = \'{}\' and password = md5(\'{}\')"
    cursor.execute(query.format(username, password))
    #stores the results in a variable
    data = cursor.fetchone()
    #use fetchall() if you are expecting more than 1 data row
    cursor.close()
    error = None
    if(data):
        #creates a session for the the user
        #session is a built in
        session['username'] = username
        session['account_type'] = data[2] #data[2] is the account_type :)

        #if account type is staff then query database to get staff airline name for

        return redirect(url_for('home'))
    else:
        #returns an error message to the html page
        error = 'Invalid login or username'
        return render_template('login.html', error=error)

#Authenticates the register
```

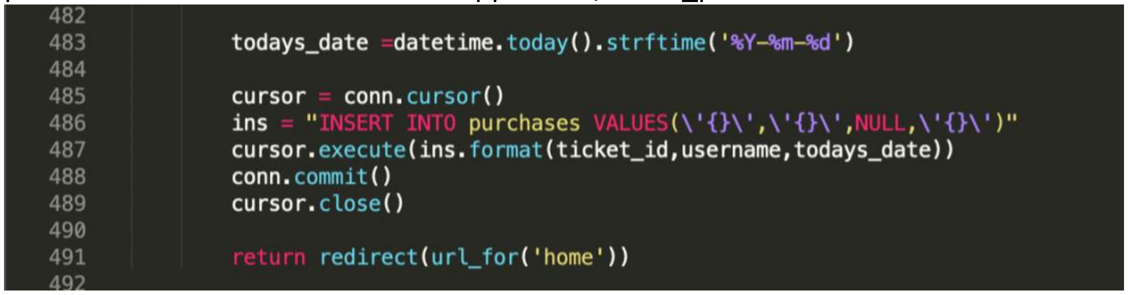
Customer Use Cases:

1. View my flights:
   All information regarding customer upcoming flights is pulled from the database with repect to the custoemers email. This can be found under app.route /home. Searching for my flights is done using the same user_search but with an additional parameter to only search the users purchases.



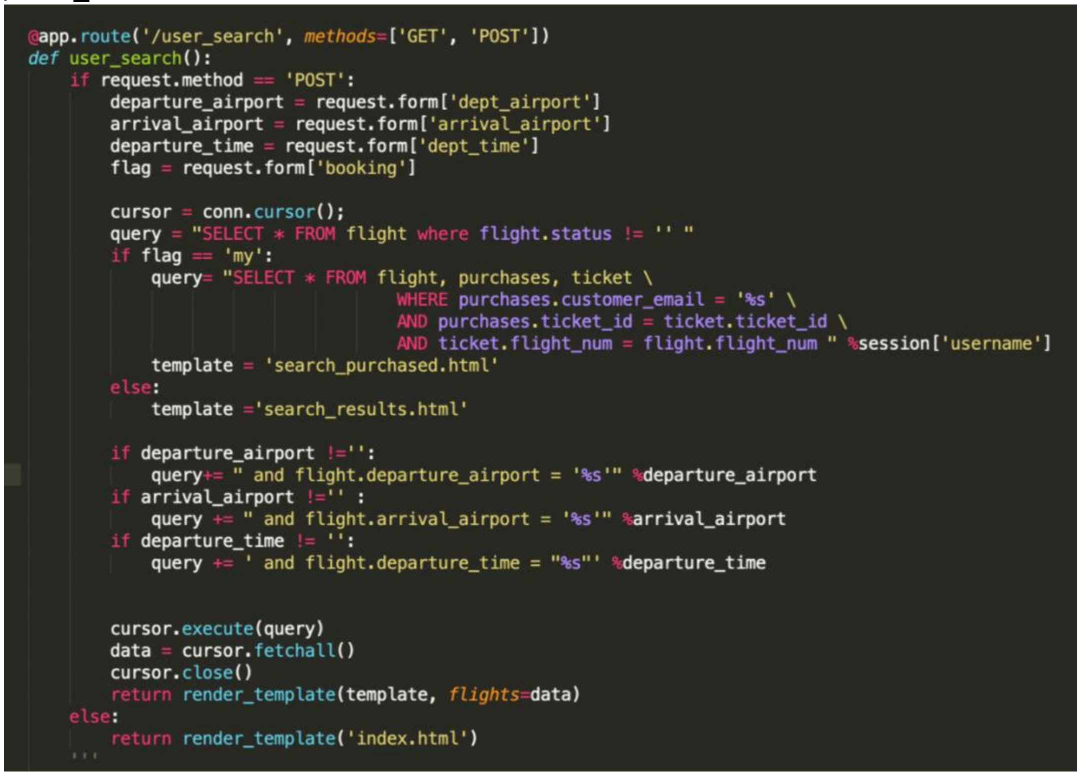
```
@app.route('/home')
def home():
    username = session['username']
    account_type = session['account_type']
    cursor = conn.cursor()
    if account_type == 'customer':
        page_to_render = 'user_home_page.html'
        query_purchased_flights = "SELECT * FROM flight, purchases, ticket \
                                   WHERE purchases.customer_email = \'{}\' \
                                   AND purchases.ticket_id = ticket.ticket_id \
                                   AND ticket.flight_num = flight.flight_num and flight.status ='upcoming'
        cursor.execute(query_purchased_flights.format(username))
        data1 = cursor.fetchall()
```

2. Purchase ticket:
   After a user selects a ticket to buy, the purchases table is then updated to record this purchase. This can be found under app.route /insert_purchase



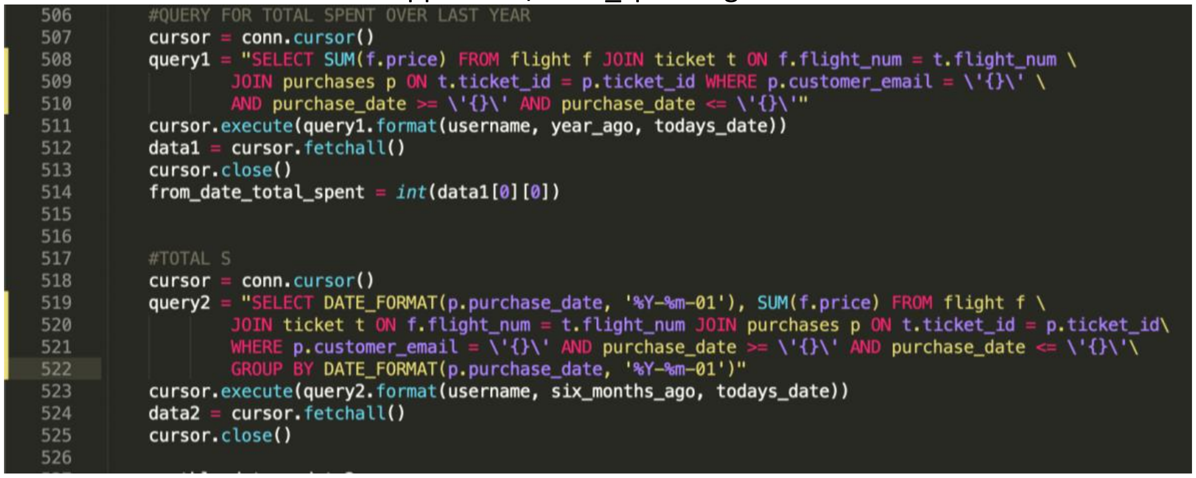
```
            todays_date =datetime.today().strftime('%Y-%m-%d')

            cursor = conn.cursor()
            ins = "INSERT INTO purchases VALUES(\'{}\',\'{}\',NULL,\'{}\')"
            cursor.execute(ins.format(ticket_id,username,todays_date))
            conn.commit()
            cursor.close()

            return redirect(url_for('home'))
```

3. Search for flights:
   There are 2 cases for search, 1 for a general search and 1 for a search of purchased flights. Both versions of can be searched through, the only difference is that purchased flights only searches flights bought by the user. They can be found under app.route /user_search.



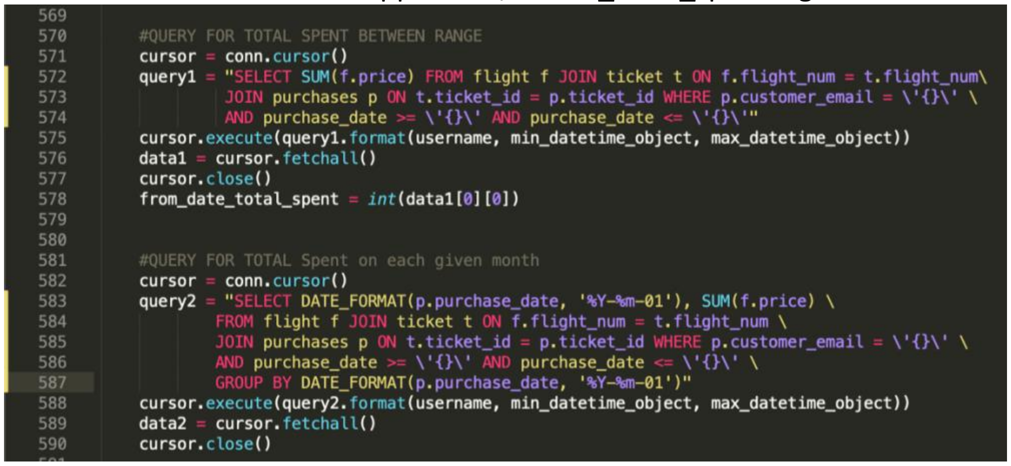
```python
@app.route('/user_search', methods=['GET', 'POST'])
def user_search():
    if request.method == 'POST':
        departure_airport = request.form['dept_airport']
        arrival_airport = request.form['arrival_airport']
        departure_time = request.form['dept_time']
        flag = request.form['booking']

        cursor = conn.cursor();
        query = "SELECT * FROM flight where flight.status != '' "
        if flag == 'my':
            query= "SELECT * FROM flight, purchases, ticket \
                        WHERE purchases.customer_email = '%s' \
                        AND purchases.ticket_id = ticket.ticket_id \
                        AND ticket.flight_num = flight.flight_num " %session['username']
            template = 'search_purchased.html'
        else:
            template ='search_results.html'

        if departure_airport !='':
            query+= " and flight.departure_airport = '%s'" %departure_airport
        if arrival_airport !='' :
            query += " and flight.arrival_airport = '%s'" %arrival_airport
        if departure_time != '':
            query += ' and flight.departure_time = "%s"' %departure_time


        cursor.execute(query)
        data = cursor.fetchall()
        cursor.close()
        return render_template(template, flights=data)
    else:
        return render_template('index.html')
    '''
```

4. Track my spending:
   Two queries are done, one looking at total money spent over the past 6 months and over the past year. The data is then inserted into an array to be presented in bar graphs. The code can be found under app.route /track_spending.



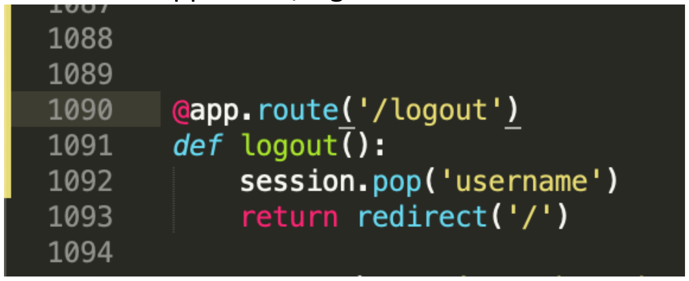
```python
    #QUERY FOR TOTAL SPENT OVER LAST YEAR
    cursor = conn.cursor()
    query1 = "SELECT SUM(f.price) FROM flight f JOIN ticket t ON f.flight_num = t.flight_num \
            JOIN purchases p ON t.ticket_id = p.ticket_id WHERE p.customer_email = \'{}\' \
            AND purchase_date >= \'{}\' AND purchase_date <= \'{}\'"
    cursor.execute(query1.format(username, year_ago, todays_date))
    data1 = cursor.fetchall()
    cursor.close()
    from_date_total_spent = int(data1[0][0])


    #TOTAL S
    cursor = conn.cursor()
    query2 = "SELECT DATE_FORMAT(p.purchase_date, '%Y-%m-01'), SUM(f.price) FROM flight f \
            JOIN ticket t ON f.flight_num = t.flight_num JOIN purchases p ON t.ticket_id = p.ticket_id\
            WHERE p.customer_email = \'{}\' AND purchase_date >= \'{}\' AND purchase_date <= \'{}\'\
            GROUP BY DATE_FORMAT(p.purchase_date, '%Y-%m-01')"
    cursor.execute(query2.format(username, six_months_ago, todays_date))
    data2 = cursor.fetchall()
    cursor.close()

```

User can also set range of dates to track total spending between them. The queries are as follows and found under app.route /search_track_spending



```
    #QUERY FOR TOTAL SPENT BETWEEN RANGE
    cursor = conn.cursor()
    query1 = "SELECT SUM(f.price) FROM flight f JOIN ticket t ON f.flight_num = t.flight_num\
             JOIN purchases p ON t.ticket_id = p.ticket_id WHERE p.customer_email = \'{}\' \
             AND purchase_date >= \'{}\' AND purchase_date <= \'{}\'"
    cursor.execute(query1.format(username, min_datetime_object, max_datetime_object))
    data1 = cursor.fetchall()
    cursor.close()
    from_date_total_spent = int(data1[0][0])


    #QUERY FOR TOTAL Spent on each given month
    cursor = conn.cursor()
    query2 = "SELECT DATE_FORMAT(p.purchase_date, '%Y-%m-01'), SUM(f.price) \
            FROM flight f JOIN ticket t ON f.flight_num = t.flight_num \
            JOIN purchases p ON t.ticket_id = p.ticket_id WHERE p.customer_email = \'{}\' \
            AND purchase_date >= \'{}\' AND purchase_date <= \'{}\' \
            GROUP BY DATE_FORMAT(p.purchase_date, '%Y-%m-01')"
    cursor.execute(query2.format(username, min_datetime_object, max_datetime_object))
    data2 = cursor.fetchall()
    cursor.close()
```

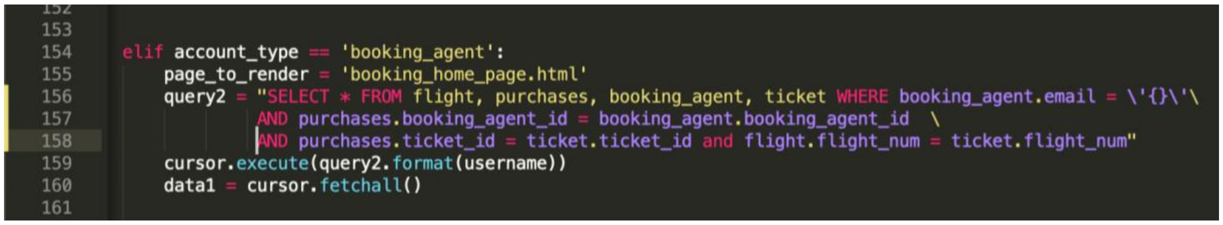
5. Logout:
   Logging out returns the user to the homepage, and ends the current session. This can be seen under app.route /logout



```
@app.route('/logout')
def logout():
    session.pop('username')
    return redirect('/')
```

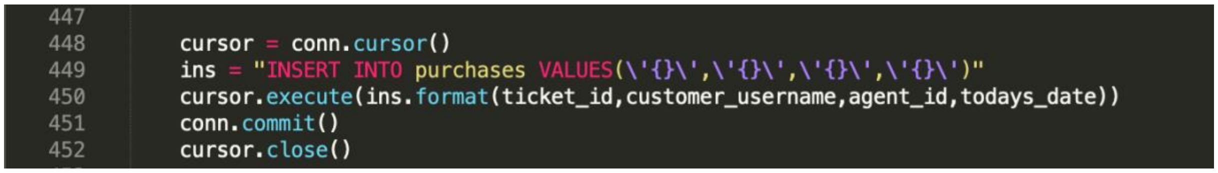
Booking Agent Use Cases:

1. View My flights:
   Similar to the view my flight for customers, booking agent view my flights queries the database and finds all flights bought by the booking agent. This can also be searched with the search feature. The code below can be found under app.route /home



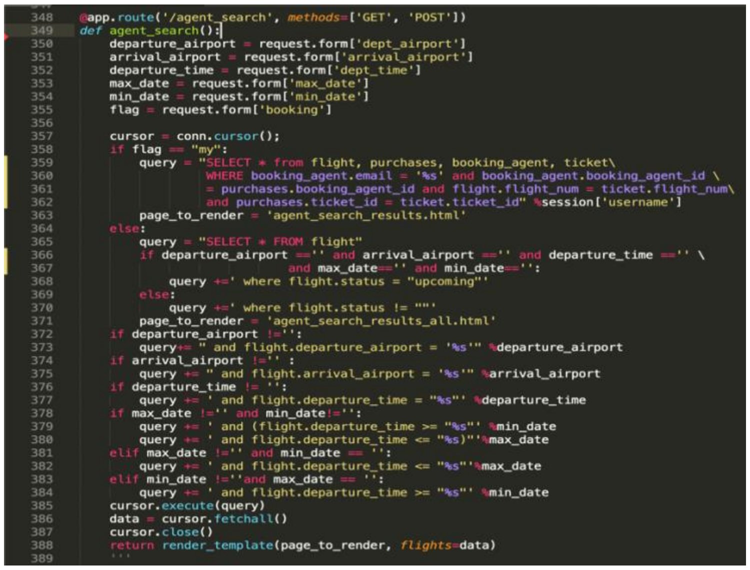
```
elif account_type == 'booking_agent':
    page_to_render = 'booking_home_page.html'
    query2 = "SELECT * FROM flight, purchases, booking_agent, ticket WHERE booking_agent.email = \'{}\'\
                AND purchases.booking_agent_id = booking_agent.booking_agent_id  \
                AND purchases.ticket_id = ticket.ticket_id and flight.flight_num = ticket.flight_num"
    cursor.execute(query2.format(username))
    data1 = cursor.fetchall()
```

2. Purchase Ticket:
   Same as customer purchase ticket, except that the booking agents ID is also updated into the purchases table. The agent can also choose between buying tickets for existing users and new users. For new users, the agent fills out the customer information and creates a new user, then buys him a ticket. The code shown below can be found under app.route /agent_insert_purchase, however this took more steps as there were many different cases.



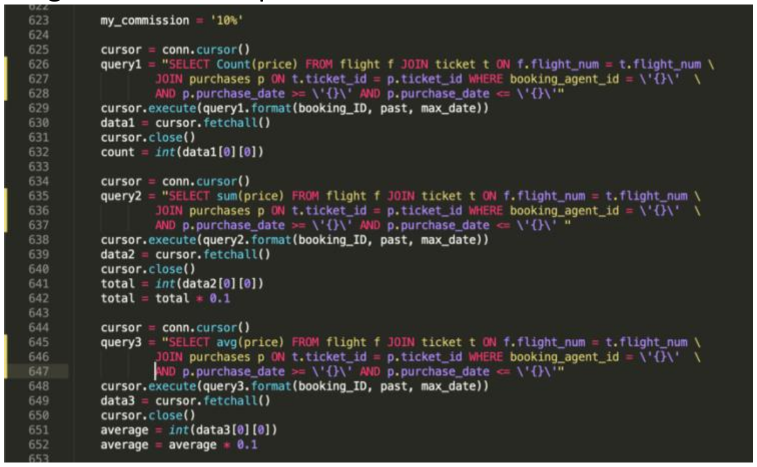
```
        cursor = conn.cursor()
        ins = "INSERT INTO purchases VALUES(\'{}\',\'{}\',\'{}\',\'{}\')"
        cursor.execute(ins.format(ticket_id,customer_username,agent_id,todays_date))
        conn.commit()
        cursor.close()
```

3. Search for Flights:
   Similar to customer search. Can search my purchases and all flights, by selecting the corresponding button on the web app. The code can be found under app.route /agent_search



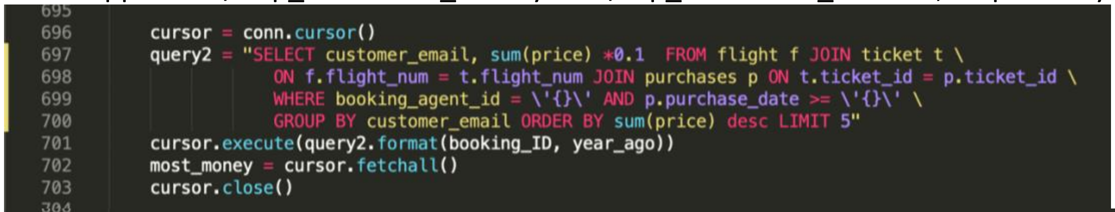
```
@app.route('/agent_search', methods=['GET', 'POST'])
def agent_search():
    departure_airport = request.form['dept_airport']
    arrival_airport = request.form['arrival_airport']
    departure_time = request.form['dept_time']
    max_date = request.form['max_date']
    min_date = request.form['min_date']
    flag = request.form['booking']

    cursor = conn.cursor();
    if flag == "my":
        query = "SELECT * from flight, purchases, booking_agent, ticket\
                 WHERE booking_agent.email = '%s' and booking_agent.booking_agent_id \
                 = purchases.booking_agent_id and flight.flight_num = ticket.flight_num\
                 and purchases.ticket_id = ticket.ticket_id" %session['username']
        page_to_render = 'agent_search_results.html'
    else:
        query = "SELECT * FROM flight"
        if departure_airport ==''  and arrival_airport =='' and departure_time =='' \
                            and max_date=='' and min_date=='':
            query +=' where flight.status = "upcoming"'
        else:
            query +=' where flight.status != ""'
        page_to_render = 'agent_search_results_all.html'
    if departure_airport !='':
        query+= " and flight.departure_airport = '%s'" %departure_airport
    if arrival_airport !='' :
        query += " and flight.arrival_airport = '%s'" %arrival_airport
    if departure_time != '':
        query += ' and flight.departure_time = "%s"' %departure_time
    if max_date !='' and min_date!='':
        query += ' and (flight.departure_time >= "%s"' %min_date
        query += ' and flight.departure_time <= "%s)"'%max_date
    elif max_date !='' and min_date == '':
        query += ' and flight.departure_time <= "%s"'%max_date
    elif min_date !=''and max_date == '':
        query += ' and flight.departure_time >= "%s"' %min_date
    cursor.execute(query)
    data = cursor.fetchall()
    cursor.close()
    return render_template(page_to_render, flights=data)
    '''
```

4. View my commission:
   There are 3 queries to provide the various commission data regarding number of tickets, total commission and average commission. Date ranges can be added to specify date range of data. The queries are as follows and can be found under app.route /earnings

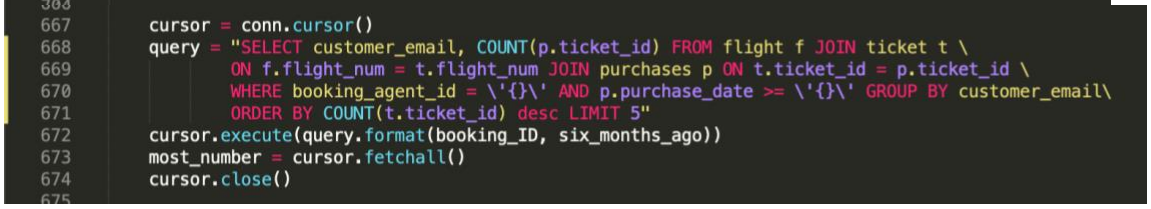


```
my_commission = '10%'

cursor = conn.cursor()
query1 = "SELECT Count(price) FROM flight f JOIN ticket t ON f.flight_num = t.flight_num \
        JOIN purchases p ON t.ticket_id = p.ticket_id WHERE booking_agent_id = \'{}\'  \
        AND p.purchase_date >= \'{}\' AND p.purchase_date <= \'{}\'"
cursor.execute(query1.format(booking_ID, past, max_date))
data1 = cursor.fetchall()
cursor.close()
count = int(data1[0][0])

cursor = conn.cursor()
query2 = "SELECT sum(price) FROM flight f JOIN ticket t ON f.flight_num = t.flight_num \
        JOIN purchases p ON t.ticket_id = p.ticket_id WHERE booking_agent_id = \'{}\'  \
        AND p.purchase_date >= \'{}\' AND p.purchase_date <= \'{}\' "
cursor.execute(query2.format(booking_ID, past, max_date))
data2 = cursor.fetchall()
cursor.close()
total = int(data2[0][0])
total = total * 0.1

cursor = conn.cursor()
query3 = "SELECT avg(price) FROM flight f JOIN ticket t ON f.flight_num = t.flight_num \
        JOIN purchases p ON t.ticket_id = p.ticket_id WHERE booking_agent_id = \'{}\'  \
        AND p.purchase_date >= \'{}\' AND p.purchase_date <= \'{}\'"
cursor.execute(query3.format(booking_ID, past, max_date))
data3 = cursor.fetchall()
cursor.close()
average = int(data3[0][0])
average = average * 0.1
```

5. Top Customers:
   Top customers can be viewed based on 2 metrics, by total commission and by number of tickets purchased. We separate this into two different pages, one for commission and one for numbers of tickets. The corresponding queries are as follows and can be found under app.route / top_customers_money and /top_customers_number, respectively.



```
cursor = conn.cursor()
query2 = "SELECT customer_email, sum(price) *0.1  FROM flight f JOIN ticket t \
            ON f.flight_num = t.flight_num JOIN purchases p ON t.ticket_id = p.ticket_id \
            WHERE booking_agent_id = \'{}\' AND p.purchase_date >= \'{}\' \
            GROUP BY customer_email ORDER BY sum(price) desc LIMIT 5"
cursor.execute(query2.format(booking_ID, year_ago))
most_money = cursor.fetchall()
cursor.close()
```

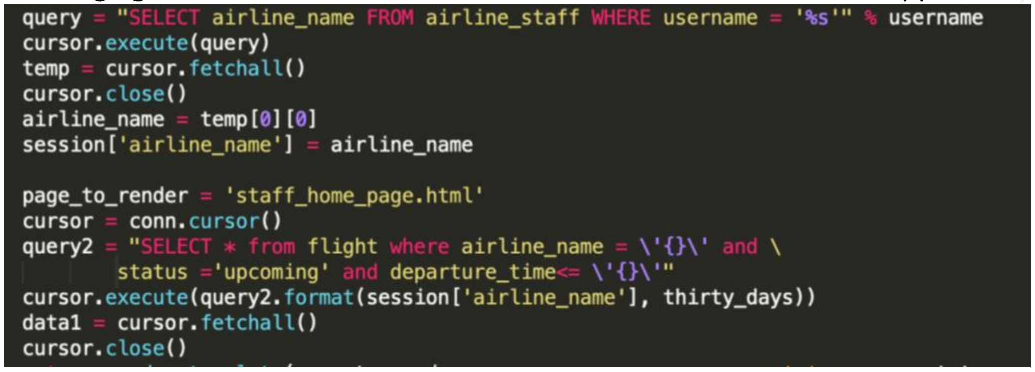


```
cursor = conn.cursor()
query = "SELECT customer_email, COUNT(p.ticket_id) FROM flight f JOIN ticket t \
        ON f.flight_num = t.flight_num JOIN purchases p ON t.ticket_id = p.ticket_id \
        WHERE booking_agent_id = \'{}\' AND p.purchase_date >= \'{}\' GROUP BY customer_email\
        ORDER BY COUNT(t.ticket_id) desc LIMIT 5"
cursor.execute(query.format(booking_ID, six_months_ago))
most_number = cursor.fetchall()
cursor.close()
```

6. Logout:
   Same as for customer

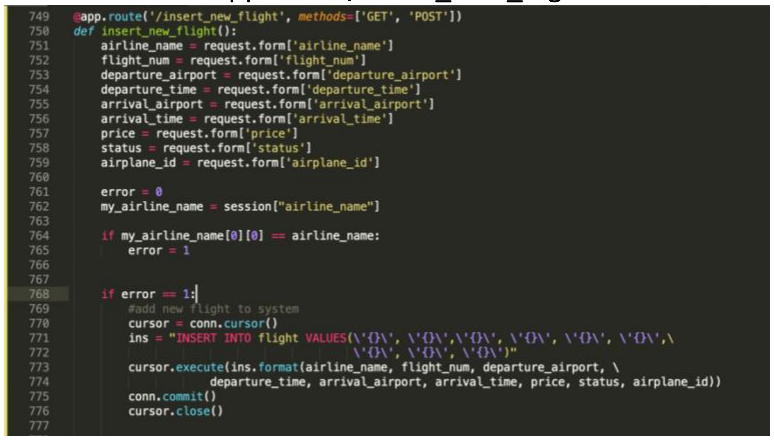
Airline Staff Use Cases:

1. View my flights:
   Same as for booking agent and customer except that airline name is equal to that of the booking agent. This can also be searched. Code can be found under app.route /home



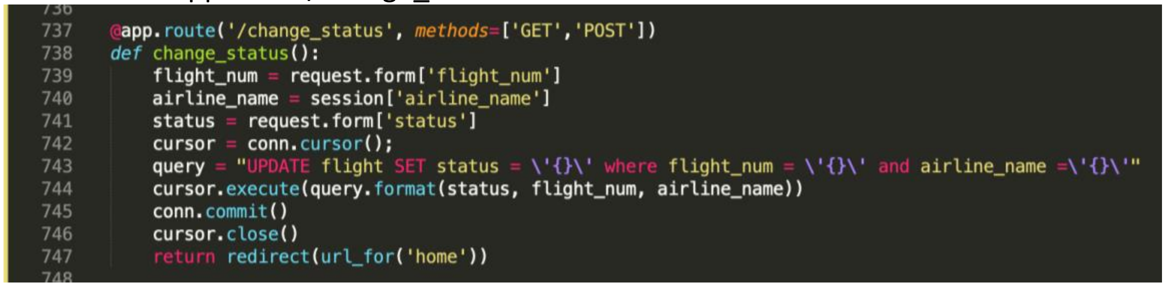
```
query = "SELECT airline_name FROM airline_staff WHERE username = '%s'" % username
cursor.execute(query)
temp = cursor.fetchall()
cursor.close()
airline_name = temp[0][0]
session['airline_name'] = airline_name

page_to_render = 'staff_home_page.html'
cursor = conn.cursor()
query2 = "SELECT * from flight where airline_name = \'{}\' and \
          status ='upcoming' and departure_time<= \'{}\'"
cursor.execute(query2.format(session['airline_name'], thirty_days))
data1 = cursor.fetchall()
cursor.close()
```

2. Create new flights:
   Booking agent fills in information about new flight then is inserted into flights. Code can be found under app.route /insert_new_flight



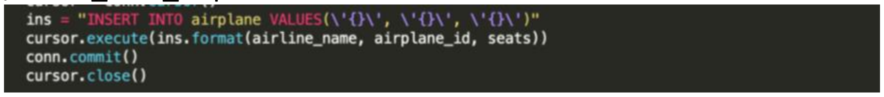
```
@app.route('/insert_new_flight', methods=['GET', 'POST'])
def insert_new_flight():
    airline_name = request.form['airline_name']
    flight_num = request.form['flight_num']
    departure_airport = request.form['departure_airport']
    departure_time = request.form['departure_time']
    arrival_airport = request.form['arrival_airport']
    arrival_time = request.form['arrival_time']
    price = request.form['price']
    status = request.form['status']
    airplane_id = request.form['airplane_id']

    error = 0
    my_airline_name = session["airline_name"]

    if my_airline_name[0][0] == airline_name:
        error = 1


    if error == 1:
        #add new flight to system
        cursor = conn.cursor()
        ins = "INSERT INTO flight VALUES(\'{}\', \'{}\',\'{}\', \'{}\', \'{}\', \'{}\',\
                                        \'{}\', \'{}\', \'{}\')"
        cursor.execute(ins.format(airline_name, flight_num, departure_airport, \
                    departure_time, arrival_airport, arrival_time, price, status, airplane_id))
        conn.commit()
        cursor.close()
```

3. Change flight status:
   Status of flight can be changes based on flight number and airline name. Code can be found under app.route /change_status.



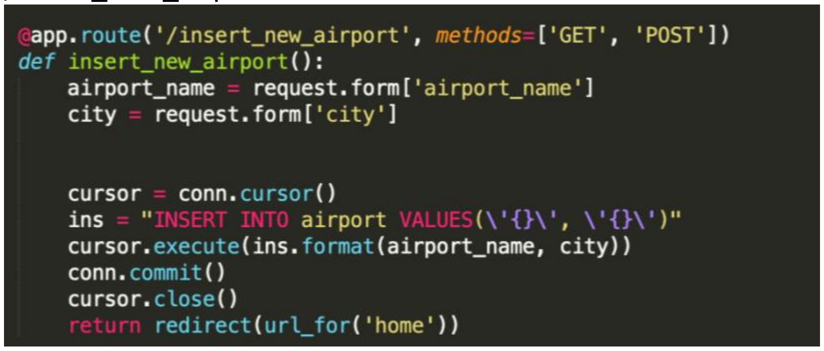
```
@app.route('/change_status', methods=['GET','POST'])
def change_status():
    flight_num = request.form['flight_num']
    airline_name = session['airline_name']
    status = request.form['status']
    cursor = conn.cursor();
    query = "UPDATE flight SET status = \'{}\' where flight_num = \'{}\' and airline_name =\'{}\'"
    cursor.execute(query.format(status, flight_num, airline_name))
    conn.commit()
    cursor.close()
    return redirect(url_for('home'))
```

4. Add airplane:
   Given that the staff is trying to add an airplane to their company's fleet, the information is collected then inserted into airplane. The code can be seen at app.route /insert_new_airplane.



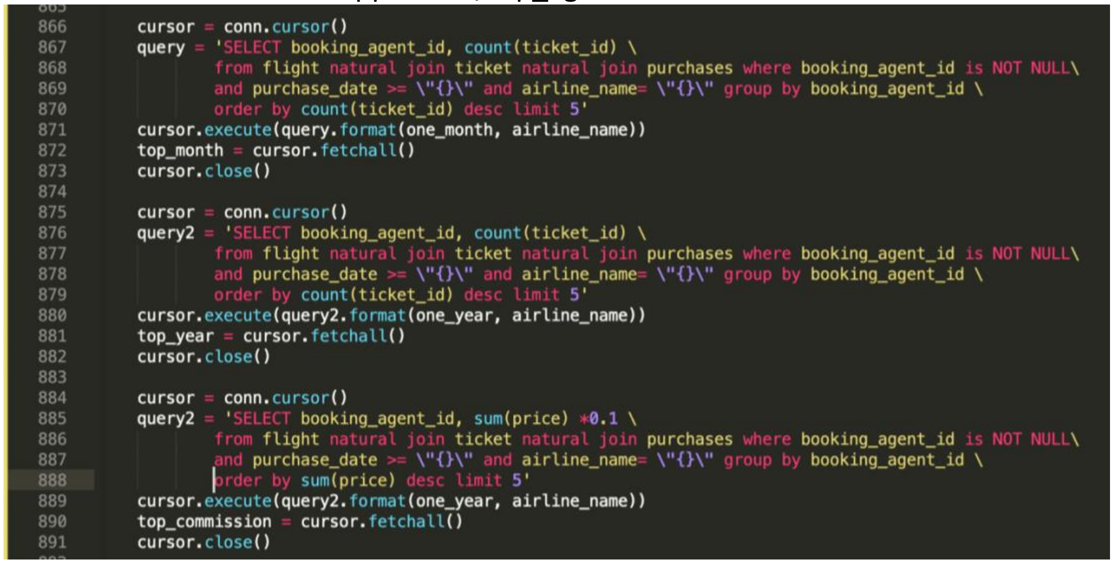
```python
ins = "INSERT INTO airplane VALUES(\'{}\', \'{}\', \'{}\')"
cursor.execute(ins.format(airline_name, airplane_id, seats))
conn.commit()
cursor.close()
```

5. Add airport:
   Information is collected then inserted into airport. Code can be seen at app.route /inster_new_airport.



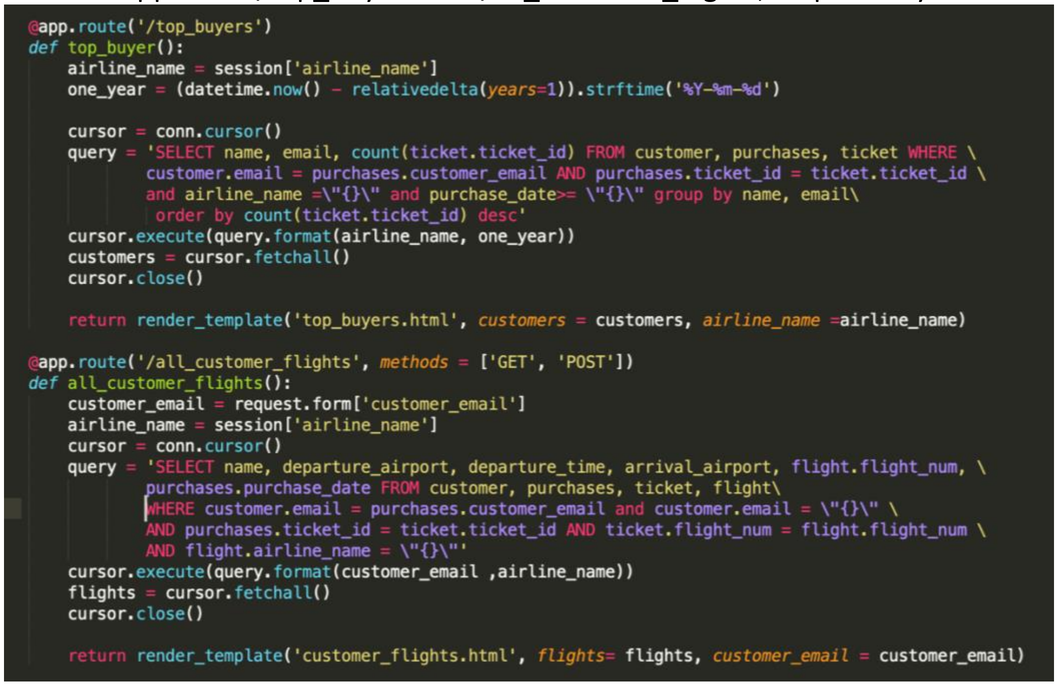
```python
@app.route('/insert_new_airport', methods=['GET', 'POST'])
def insert_new_airport():
    airport_name = request.form['airport_name']
    city = request.form['city']


    cursor = conn.cursor()
    ins = "INSERT INTO airport VALUES(\'{}\', \'{}\')"
    cursor.execute(ins.format(airport_name, city))
    conn.commit()
    cursor.close()
    return redirect(url_for('home'))
```

6. Top 5 booking agents:
   Top booking agents are queried and pulled from the database based on various metrics. The code can be found at app.route /top_agent



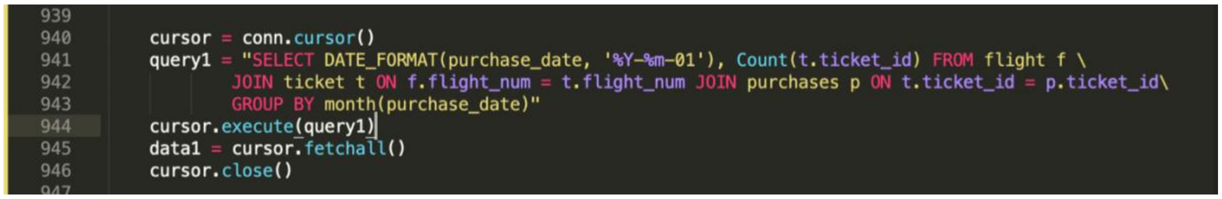
```python
    cursor = conn.cursor()
    query = 'SELECT booking_agent_id, count(ticket_id) \
            from flight natural join ticket natural join purchases where booking_agent_id is NOT NULL\
            and purchase_date >= \"{}\" and airline_name= \"{}\" group by booking_agent_id \
            order by count(ticket_id) desc limit 5'
    cursor.execute(query.format(one_month, airline_name))
    top_month = cursor.fetchall()
    cursor.close()

    cursor = conn.cursor()
    query2 = 'SELECT booking_agent_id, count(ticket_id) \
            from flight natural join ticket natural join purchases where booking_agent_id is NOT NULL\
            and purchase_date >= \"{}\" and airline_name= \"{}\" group by booking_agent_id \
            order by count(ticket_id) desc limit 5'
    cursor.execute(query2.format(one_year, airline_name))
    top_year = cursor.fetchall()
    cursor.close()

    cursor = conn.cursor()
    query2 = 'SELECT booking_agent_id, sum(price) *0.1 \
            from flight natural join ticket natural join purchases where booking_agent_id is NOT NULL\
            and purchase_date >= \"{}\" and airline_name= \"{}\" group by booking_agent_id \
            order by sum(price) desc limit 5'
    cursor.execute(query2.format(one_year, airline_name))
    top_commission = cursor.fetchall()
    cursor.close()
```

7. View Frequent Customers:
   Airline customers are shown in descending order based on number of tickets purchased by the customer. For each customer you are able to go in and view all of the customers flights with the airline staff's airline. Code for top customers and customer flights can be found at app.route /top_buyers and /all_customer_flights, respectively.

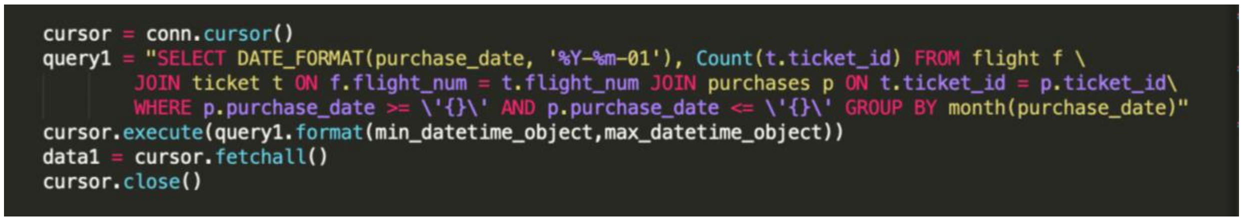


```
@app.route('/top_buyers')
def top_buyer():
    airline_name = session['airline_name']
    one_year = (datetime.now() - relativedelta(years=1)).strftime('%Y-%m-%d')

    cursor = conn.cursor()
    query = 'SELECT name, email, count(ticket.ticket_id) FROM customer, purchases, ticket WHERE \
            customer.email = purchases.customer_email AND purchases.ticket_id = ticket.ticket_id \
            and airline_name =\"{}\" and purchase_date>= \"{}\" group by name, email\
             order by count(ticket.ticket_id) desc'
    cursor.execute(query.format(airline_name, one_year))
    customers = cursor.fetchall()
    cursor.close()

    return render_template('top_buyers.html', customers = customers, airline_name =airline_name)

@app.route('/all_customer_flights', methods = ['GET', 'POST'])
def all_customer_flights():
    customer_email = request.form['customer_email']
    airline_name = session['airline_name']
    cursor = conn.cursor()
    query = 'SELECT name, departure_airport, departure_time, arrival_airport, flight.flight_num, \
            purchases.purchase_date FROM customer, purchases, ticket, flight\
            WHERE customer.email = purchases.customer_email and customer.email = \"{}\" \
            AND purchases.ticket_id = ticket.ticket_id AND ticket.flight_num = flight.flight_num \
            AND flight.airline_name = \"{}\"'
    cursor.execute(query.format(customer_email ,airline_name))
    flights = cursor.fetchall()
    cursor.close()

    return render_template('customer_flights.html', flights= flights, customer_email = customer_email)
```

8. View reports:
   Data on counts of ticket per month is pulled from the database using query shown below. Date ranges can also be changed to change the range of dates of the reports. Code can be found under app.route /view_report and /search_report

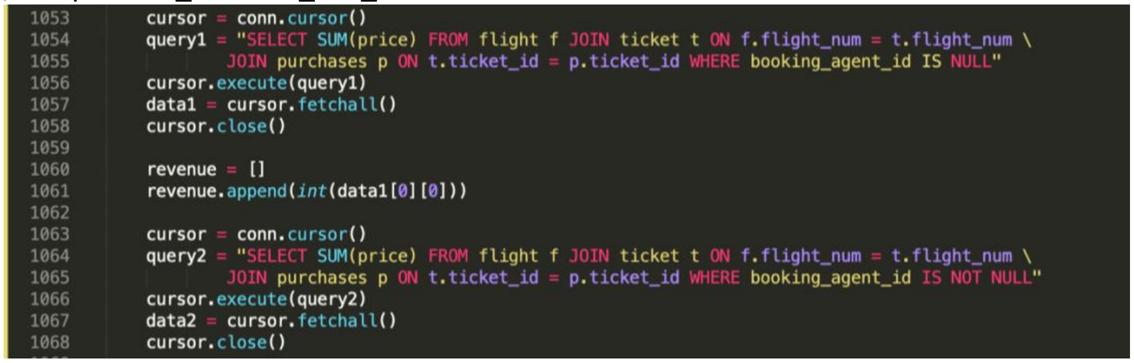


```

    cursor = conn.cursor()
    query1 = "SELECT DATE_FORMAT(purchase_date, '%Y-%m-01'), Count(t.ticket_id) FROM flight f \
            JOIN ticket t ON f.flight_num = t.flight_num JOIN purchases p ON t.ticket_id = p.ticket_id\
            GROUP BY month(purchase_date)"
    cursor.execute(query1)
    data1 = cursor.fetchall()
    cursor.close()
```



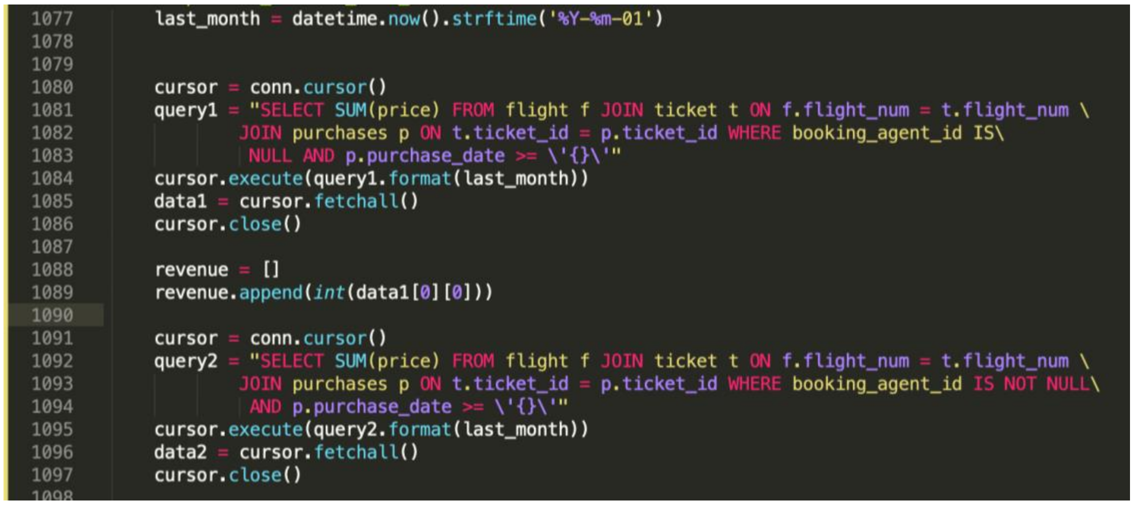
```
cursor = conn.cursor()
query1 = "SELECT DATE_FORMAT(purchase_date, '%Y-%m-01'), Count(t.ticket_id) FROM flight f \
        JOIN ticket t ON f.flight_num = t.flight_num JOIN purchases p ON t.ticket_id = p.ticket_id\
        WHERE p.purchase_date >= \'{}\' AND p.purchase_date <= \'{}\' GROUP BY month(purchase_date)"
cursor.execute(query1.format(min_datetime_object,max_datetime_object))
data1 = cursor.fetchall()
cursor.close()
```

9. Comparison of revenue earned:
   Pie chart shows the total amount of revenue earned by direct sales and indirect sales, in the last month and year. Code can found under app.route /comparision_revenue and /comparision_revenue_last_month



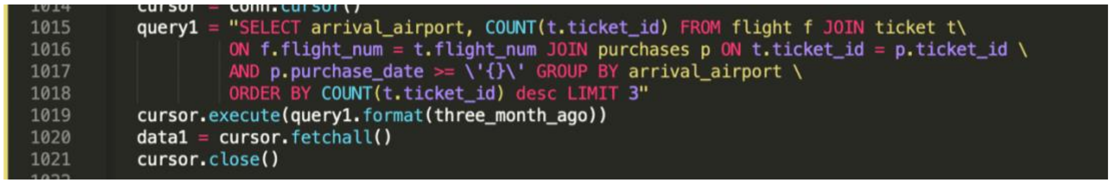
```
    cursor = conn.cursor()
    query1 = "SELECT SUM(price) FROM flight f JOIN ticket t ON f.flight_num = t.flight_num \
            JOIN purchases p ON t.ticket_id = p.ticket_id WHERE booking_agent_id IS NULL"
    cursor.execute(query1)
    data1 = cursor.fetchall()
    cursor.close()

    revenue = []
    revenue.append(int(data1[0][0]))

    cursor = conn.cursor()
    query2 = "SELECT SUM(price) FROM flight f JOIN ticket t ON f.flight_num = t.flight_num \
            JOIN purchases p ON t.ticket_id = p.ticket_id WHERE booking_agent_id IS NOT NULL"
    cursor.execute(query2)
    data2 = cursor.fetchall()
    cursor.close()
```

   One month

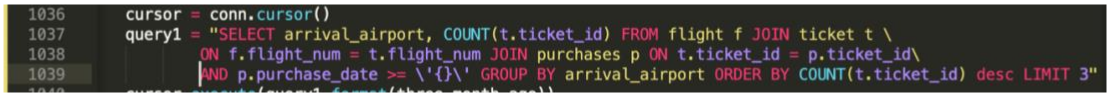


```
    last_month = datetime.now().strftime('%Y-%m-01')


    cursor = conn.cursor()
    query1 = "SELECT SUM(price) FROM flight f JOIN ticket t ON f.flight_num = t.flight_num \
            JOIN purchases p ON t.ticket_id = p.ticket_id WHERE booking_agent_id IS\
             NULL AND p.purchase_date >= \'{}\'"
    cursor.execute(query1.format(last_month))
    data1 = cursor.fetchall()
    cursor.close()

    revenue = []
    revenue.append(int(data1[0][0]))

    cursor = conn.cursor()
    query2 = "SELECT SUM(price) FROM flight f JOIN ticket t ON f.flight_num = t.flight_num \
            JOIN purchases p ON t.ticket_id = p.ticket_id WHERE booking_agent_id IS NOT NULL\
             AND p.purchase_date >= \'{}\'"
    cursor.execute(query2.format(last_month))
    data2 = cursor.fetchall()
    cursor.close()
```

10. View top destinations:
    Shows the top destination of the last 3 months and last year. The queries are as follows and can be found at app.route /view_top_destination and /top_year_des, respectively.



```
    query1 = "SELECT arrival_airport, COUNT(t.ticket_id) FROM flight f JOIN ticket t\
            ON f.flight_num = t.flight_num JOIN purchases p ON t.ticket_id = p.ticket_id \
            AND p.purchase_date >= \'{}\' GROUP BY arrival_airport \
            ORDER BY COUNT(t.ticket_id) desc LIMIT 3"
    cursor.execute(query1.format(three_month_ago))
    data1 = cursor.fetchall()
    cursor.close()
```



```
    cursor = conn.cursor()
    query1 = "SELECT arrival_airport, COUNT(t.ticket_id) FROM flight f JOIN ticket t \
            ON f.flight_num = t.flight_num JOIN purchases p ON t.ticket_id = p.ticket_id\
            AND p.purchase_date >= \'{}\' GROUP BY arrival_airport ORDER BY COUNT(t.ticket_id) desc LIMIT 3"
```

11. Logout:
    Same as for customer and booking agent.