

## 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('%s', md5('%s'), '%s', '%s', '%s', '%s')"
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 respect to the customers 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.

```

128
129 @app.route('/home')
130 def home():
131     username = session['username']
132     account_type = session['account_type']
133     cursor = conn.cursor()
134     if account_type == 'customer':
135         page_to_render = 'user_home_page.html'
136         query_purchased_flights = "SELECT * FROM flight, purchases, ticket \
137                                   WHERE purchases.customer_email = \'{ }\' \
138                                   AND purchases.ticket_id = ticket.ticket_id \
139                                   AND ticket.flight_num = flight.flight_num and flight.status = 'upcoming'"
140         cursor.execute(query_purchased_flights.format(username))
141         data1 = cursor.fetchall()
142

```

### 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

```

482
483     todays_date =datetime.today().strftime('%Y-%m-%d')
484
485     cursor = conn.cursor()
486     ins = "INSERT INTO purchases VALUES(\'{ }\', \'{ }\', NULL, \'{ }\'"
487     cursor.execute(ins.format(ticket_id, username, todays_date))
488     conn.commit()
489     cursor.close()
490
491     return redirect(url_for('home'))
492

```

### 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.

```
@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.

```
506 #QUERY FOR TOTAL SPENT OVER LAST YEAR
507 cursor = conn.cursor()
508 query1 = "SELECT SUM(f.price) FROM flight f JOIN ticket t ON f.flight_num = t.flight_num \
509         JOIN purchases p ON t.ticket_id = p.ticket_id WHERE p.customer_email = '{0}' \
510         AND purchase_date >= '{0}' AND purchase_date <= '{0}'"
511 cursor.execute(query1.format(username, year_ago, todays_date))
512 data1 = cursor.fetchall()
513 cursor.close()
514 from_date_total_spent = int(data1[0][0])
515
516
517 #TOTAL S
518 cursor = conn.cursor()
519 query2 = "SELECT DATE_FORMAT(p.purchase_date, '%Y-%m-01'), SUM(f.price) FROM flight f \
520         JOIN ticket t ON f.flight_num = t.flight_num JOIN purchases p ON t.ticket_id = p.ticket_id \
521         WHERE p.customer_email = '{0}' AND purchase_date >= '{0}' AND purchase_date <= '{0}' \
522         GROUP BY DATE_FORMAT(p.purchase_date, '%Y-%m-01')"
523 cursor.execute(query2.format(username, six_months_ago, todays_date))
524 data2 = cursor.fetchall()
525 cursor.close()
526
```

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`

```
569
570 #QUERY FOR TOTAL SPENT BETWEEN RANGE
571 cursor = conn.cursor()
572 query1 = "SELECT SUM(f.price) FROM flight f JOIN ticket t ON f.flight_num = t.flight_num \
573 JOIN purchases p ON t.ticket_id = p.ticket_id WHERE p.customer_email = '{}'\ \
574 AND purchase_date >= '{}'\ AND purchase_date <= '{}'"
575 cursor.execute(query1.format(username, min_datetime_object, max_datetime_object))
576 data1 = cursor.fetchall()
577 cursor.close()
578 from_date_total_spent = int(data1[0][0])
579
580
581 #QUERY FOR TOTAL Spent on each given month
582 cursor = conn.cursor()
583 query2 = "SELECT DATE_FORMAT(p.purchase_date, '%Y-%m-01'), SUM(f.price) \
584 FROM flight f JOIN ticket t ON f.flight_num = t.flight_num \
585 JOIN purchases p ON t.ticket_id = p.ticket_id WHERE p.customer_email = '{}'\ \
586 AND purchase_date >= '{}'\ AND purchase_date <= '{}'\ \
587 GROUP BY DATE_FORMAT(p.purchase_date, '%Y-%m-01')"
588 cursor.execute(query2.format(username, min_datetime_object, max_datetime_object))
589 data2 = cursor.fetchall()
590 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`

```
1087
1088
1089
1090 @app.route('/logout')
1091 def logout():
1092     session.pop('username')
1093     return redirect('/')
1094
```

### 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`

```

152
153
154     elif account_type == 'booking_agent':
155         page_to_render = 'booking_home_page.html'
156         query2 = "SELECT * FROM flight, purchases, booking_agent, ticket WHERE booking_agent.email = '{0}'\n\
157                 AND purchases.booking_agent_id = booking_agent.booking_agent_id \
158                 AND purchases.ticket_id = ticket.ticket_id and flight.flight_num = ticket.flight_num"
159         cursor.execute(query2.format(username))
160         data1 = cursor.fetchall()
161

```

## 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.

```
447
448     cursor = conn.cursor()
449     ins = "INSERT INTO purchases VALUES(\''{}'\',\''{}'\',\''{}'\',\''{}'\')".format(ticket_id, customer_username, agent_id, today's_date)
450     cursor.execute(ins)
451     conn.commit()
452     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`

```

348 @app.route('/agent_search', methods=['GET', 'POST'])
349 def agent_search():
350     departure_airport = request.form['departure_airport']
351     arrival_airport = request.form['arrival_airport']
352     departure_time = request.form['dept_time']
353     max_date = request.form['max_date']
354     min_date = request.form['min_date']
355     flag = request.form['booking']
356
357     cursor = conn.cursor()
358     if flag == "my":
359         query = "SELECT * from flight, purchases, booking_agent, ticket\
360             WHERE booking_agent.email = '%s' and booking_agent.booking_agent_id \
361             = purchases.booking_agent_id and flight.flight_num = ticket.flight_num\
362             and purchases.ticket_id = ticket.ticket_id" % session['username']
363         page_to_render = 'agent_search_results.html'
364     else:
365         query = "SELECT * FROM flight"
366         if departure_airport == '' and arrival_airport == '' and departure_time == '' \
367             and max_date == '' and min_date == '':
368             query += ' where flight.status = "upcoming"'
369         else:
370             query += ' where flight.status != ""'
371         page_to_render = 'agent_search_results_all.html'
372     if departure_airport != '':
373         query += " and flight.departure_airport = '%s'" % departure_airport
374     if arrival_airport != '':
375         query += " and flight.arrival_airport = '%s'" % arrival_airport
376     if departure_time != '':
377         query += " and flight.departure_time = '%s'" % departure_time
378     if max_date != '' and min_date != '':
379         query += ' and (flight.departure_time >= "%s" %min_date'
380         query += ' and flight.departure_time <= "%s")' % max_date
381     elif max_date != '' and min_date == '':
382         query += ' and flight.departure_time <= "%s"' % max_date
383     elif min_date != '' and max_date == '':
384         query += ' and flight.departure_time >= "%s"' % min_date
385     cursor.execute(query)
386     data = cursor.fetchall()
387     cursor.close()
388     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

```
623 my_commission = '10%'
624
625 cursor = conn.cursor()
626 query1 = "SELECT Count(price) FROM flight f JOIN ticket t ON f.flight_num = t.flight_num \
627         JOIN purchases p ON t.ticket_id = p.ticket_id WHERE booking_agent_id = '{0}' \
628         AND p.purchase_date >= '{1}' AND p.purchase_date <= '{2}'"
629 cursor.execute(query1.format(booking_ID, past, max_date))
630 data1 = cursor.fetchall()
631 cursor.close()
632 count = int(data1[0][0])
633
634 cursor = conn.cursor()
635 query2 = "SELECT sum(price) FROM flight f JOIN ticket t ON f.flight_num = t.flight_num \
636         JOIN purchases p ON t.ticket_id = p.ticket_id WHERE booking_agent_id = '{0}' \
637         AND p.purchase_date >= '{1}' AND p.purchase_date <= '{2}'"
638 cursor.execute(query2.format(booking_ID, past, max_date))
639 data2 = cursor.fetchall()
640 cursor.close()
641 total = int(data2[0][0])
642 total = total * 0.1
643
644 cursor = conn.cursor()
645 query3 = "SELECT avg(price) FROM flight f JOIN ticket t ON f.flight_num = t.flight_num \
646         JOIN purchases p ON t.ticket_id = p.ticket_id WHERE booking_agent_id = '{0}' \
647         AND p.purchase_date >= '{1}' AND p.purchase_date <= '{2}'"
648 cursor.execute(query3.format(booking_ID, past, max_date))
649 data3 = cursor.fetchall()
650 cursor.close()
651 average = int(data3[0][0])
652 average = average * 0.1
653
```

#### 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.

```
695
696 cursor = conn.cursor()
697 query2 = "SELECT customer_email, sum(price) *0.1 FROM flight f JOIN ticket t \
698         ON f.flight_num = t.flight_num JOIN purchases p ON t.ticket_id = p.ticket_id \
699         WHERE booking_agent_id = '{0}' AND p.purchase_date >= '{1}' \
700         GROUP BY customer_email ORDER BY sum(price) desc LIMIT 5"
701 cursor.execute(query2.format(booking_ID, year_ago))
702 most_money = cursor.fetchall()
703 cursor.close()
704
705
706 cursor = conn.cursor()
707 query = "SELECT customer_email, COUNT(p.ticket_id) FROM flight f JOIN ticket t \
708         ON f.flight_num = t.flight_num JOIN purchases p ON t.ticket_id = p.ticket_id \
709         WHERE booking_agent_id = '{0}' AND p.purchase_date >= '{1}' GROUP BY customer_email\
710         ORDER BY COUNT(t.ticket_id) desc LIMIT 5"
711 cursor.execute(query.format(booking_ID, six_months_ago))
712 most_number = cursor.fetchall()
713 cursor.close()
714
715
```

#### 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 = '{0}' and \
        status='upcoming' and departure_time<= '{0}'"
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`

```

749 @app.route('/insert_new_flight', methods=['GET', 'POST'])
750 def insert_new_flight():
751     airline_name = request.form['airline_name']
752     flight_num = request.form['flight_num']
753     departure_airport = request.form['departure_airport']
754     departure_time = request.form['departure_time']
755     arrival_airport = request.form['arrival_airport']
756     arrival_time = request.form['arrival_time']
757     price = request.form['price']
758     status = request.form['status']
759     airplane_id = request.form['airplane_id']
760
761     error = 0
762     my_airline_name = session["airline_name"]
763
764     if my_airline_name[0][0] == airline_name:
765         error = 1
766
767     if error == 1:
768
769         #add new flight to system
770         cursor = conn.cursor()
771         ins = "INSERT INTO flight VALUES('"+airline_name+"','"+flight_num+"','"+departure_airport+"','"+departure_time+"','"+arrival_airport+"','"+arrival_time+"','"+price+"','"+status+"','"+airplane_id+"')"
772         cursor.execute(ins)
773         conn.commit()
774         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`.

```

736
737 @app.route('/change_status', methods=['GET', 'POST'])
738 def change_status():
739     flight_num = request.form['flight_num']
740     airline_name = session['airline_name']
741     status = request.form['status']
742     cursor = conn.cursor();
743     query = "UPDATE flight SET status = '{}' where flight_num = '{}' and airline_name = '{}'"
744     cursor.execute(query.format(status, flight_num, airline_name))
745     conn.commit()
746     cursor.close()
747     return redirect(url_for('home'))
748

```

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`.

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

5. Add airport:

Information is collected then inserted into airport. Code can be seen at `app.route /insert_new_airport`.

```
@app.route('/insert_new_airport', methods=['GET', 'POST'])\ndef insert_new_airport():\n    airport_name = request.form['airport_name']\n    city = request.form['city']\n\n    cursor = conn.cursor()\n    ins = "INSERT INTO airport VALUES(\'{}\', \'{}\')"\n    cursor.execute(ins.format(airport_name, city))\n    conn.commit()\n    cursor.close()\n    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`

```
865\n866     cursor = conn.cursor()\n867     query = 'SELECT booking_agent_id, count(ticket_id) \\  
868             from flight natural join ticket natural join purchases where booking_agent_id is NOT NULL\  
869             and purchase_date >= \'{}\' and airline_name= \'{}\' group by booking_agent_id \\  
870             order by count(ticket_id) desc limit 5'\n871     cursor.execute(query.format(one_month, airline_name))\n872     top_month = cursor.fetchall()\n873     cursor.close()\n874\n875     cursor = conn.cursor()\n876     query2 = 'SELECT booking_agent_id, count(ticket_id) \\  
877             from flight natural join ticket natural join purchases where booking_agent_id is NOT NULL\  
878             and purchase_date >= \'{}\' and airline_name= \'{}\' group by booking_agent_id \\  
879             order by count(ticket_id) desc limit 5'\n880     cursor.execute(query2.format(one_year, airline_name))\n881     top_year = cursor.fetchall()\n882     cursor.close()\n883\n884     cursor = conn.cursor()\n885     query2 = 'SELECT booking_agent_id, sum(price) *0.1 \\  
886             from flight natural join ticket natural join purchases where booking_agent_id is NOT NULL\  
887             and purchase_date >= \'{}\' and airline_name= \'{}\' group by booking_agent_id \\  
888             order by sum(price) desc limit 5'\n889     cursor.execute(query2.format(one_year, airline_name))\n890     top_commission = cursor.fetchall()\n891     cursor.close()\n892
```



## 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`

```
939
940     cursor = conn.cursor()
941     query1 = "SELECT DATE_FORMAT(purchase_date, '%Y-%m-01'), Count(t.ticket_id) FROM flight f \
942             JOIN ticket t ON f.flight_num = t.flight_num JOIN purchases p ON t.ticket_id = p.ticket_id \
943             GROUP BY month(purchase_date)"
944     cursor.execute(query1)
945     data1 = cursor.fetchall()
946     cursor.close()
947
```

```
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

```
1053 cursor = conn.cursor()
1054 query1 = "SELECT SUM(price) FROM flight f JOIN ticket t ON f.flight_num = t.flight_num \
1055         JOIN purchases p ON t.ticket_id = p.ticket_id WHERE booking_agent_id IS NULL"
1056 cursor.execute(query1)
1057 data1 = cursor.fetchall()
1058 cursor.close()
1059
1060 revenue = []
1061 revenue.append(int(data1[0][0]))
1062
1063 cursor = conn.cursor()
1064 query2 = "SELECT SUM(price) FROM flight f JOIN ticket t ON f.flight_num = t.flight_num \
1065         JOIN purchases p ON t.ticket_id = p.ticket_id WHERE booking_agent_id IS NOT NULL"
1066 cursor.execute(query2)
1067 data2 = cursor.fetchall()
1068 cursor.close()
```

#### One month

```
1077 last_month = datetime.now().strftime('%Y-%m-01')
1078
1079
1080 cursor = conn.cursor()
1081 query1 = "SELECT SUM(price) FROM flight f JOIN ticket t ON f.flight_num = t.flight_num \
1082         JOIN purchases p ON t.ticket_id = p.ticket_id WHERE booking_agent_id IS \
1083         NULL AND p.purchase_date >= '{}/{}'"
1084 cursor.execute(query1.format(last_month))
1085 data1 = cursor.fetchall()
1086 cursor.close()
1087
1088 revenue = []
1089 revenue.append(int(data1[0][0]))
1090
1091 cursor = conn.cursor()
1092 query2 = "SELECT SUM(price) FROM flight f JOIN ticket t ON f.flight_num = t.flight_num \
1093         JOIN purchases p ON t.ticket_id = p.ticket_id WHERE booking_agent_id IS NOT NULL \
1094         AND p.purchase_date >= '{}/{}'"
1095 cursor.execute(query2.format(last_month))
1096 data2 = cursor.fetchall()
1097 cursor.close()
1098
```

#### 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.

```
1014 cursor = conn.cursor()
1015 query1 = "SELECT arrival_airport, COUNT(t.ticket_id) FROM flight f JOIN ticket t \
1016         ON f.flight_num = t.flight_num JOIN purchases p ON t.ticket_id = p.ticket_id \
1017         AND p.purchase_date >= '{}/{}' GROUP BY arrival_airport \
1018         ORDER BY COUNT(t.ticket_id) desc LIMIT 3"
1019 cursor.execute(query1.format(three_month_ago))
1020 data1 = cursor.fetchall()
1021 cursor.close()
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036 cursor = conn.cursor()
1037 query1 = "SELECT arrival_airport, COUNT(t.ticket_id) FROM flight f JOIN ticket t \
1038         ON f.flight_num = t.flight_num JOIN purchases p ON t.ticket_id = p.ticket_id \
1039         AND p.purchase_date >= '{}/{}' GROUP BY arrival_airport ORDER BY COUNT(t.ticket_id) desc LIMIT 3"
1040 cursor.execute(query1.format(three_month_ago))
1041 data1 = cursor.fetchall()
1042 cursor.close()
```

#### 11. Logout:

Same as for customer and booking agent.