**Public Use Cases**

|  |  |
| --- | --- |
| Use Cases | Queries |
| View Public Info  Search by airports/cities,  view status | Find flight info based on departure date and depart/arrival airports  query1 = 'SELECT \*              FROM future\_flight              WHERE depart\_airport = %s                  and arrival\_airport = %s                  and CONVERT(depart\_date\_time, date) = %s'      cursor.execute(query1, (depAirport, arrAirport, date1))  Optional query for return flights, switch around airport positions  query2 = 'SELECT \*\                  FROM future\_flight\                  WHERE depart\_airport = %s \                      and arrival\_airport = %s                      and (CONVERT(depart\_date\_time, date) = %s'          cursor.execute(query2, (arrAirport, depAirport, date2))  Find flight info based on depart date and depart/arrival cities.  query1 = "SELECT flight\_num, airline\_name, airplane\_id, depart\_date\_time, depart\_airport, arrival\_date\_time, arrival\_airport, base\_price, delay\_status\              FROM future\_flight, airport as d, airport as a\              WHERE depart\_airport = d.airport\_name\                  and d.city = %s\                  and arrival\_airport = a.airport\_name\                  and a.city = %s\                  and CONVERT(depart\_date\_time, date) =  %s"      cursor.execute(query1, (depCity, arrCity, date1))  Optional return flight query, similar change as searching via airport.  query2 = "SELECT flight\_num, airline\_name, airplane\_id, depart\_date\_time, depart\_airport, arrival\_date\_time, arrival\_airport, base\_price, delay\_status\              FROM future\_flight, airport as d, airport as a\              WHERE depart\_airport = d.airport\_name\                  and d.city = %s\                  and arrival\_airport = a.airport\_name\                  and a.city = %s\                  and CONVERT(depart\_date\_time, date) =  %s"          cursor.execute(query2, (arrCity, depCity, date2)) |
|  | Find status of a specific flight using flight number, airline number, and arrival/depart dates  query1 = 'SELECT delay\_status\              FROM future\_flight\              WHERE airline\_name = %s \                  and flight\_num = %s \                  and CONVERT(depart\_date\_time, date) = %s \                  and CONVERT(arrival\_date\_time, date) = %s'      cursor.execute(query1, (airlineName, flightNumber, date1, date2)) |
| Register | Used to check if customer email already exists   query = 'SELECT \* FROM customer WHERE email = %s'      cursor.execute(query, (email))  Insert new customer data into customer  ins = 'insert into customer values(%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s);'          cursor.execute(ins, (email, name, password, building\_number, street, city, state, phone\_number, passport\_expiration, passport\_country, birth\_date))  Check if staff username already exists  query = 'SELECT \* FROM airline\_staff WHERE username = %s'      cursor.execute(query, (username))  Insert staff data along with multiple phone/email entries  ins = 'insert into airline\_staff values(%s, %s, %s, %s, %s,%s)'          p\_ins = 'insert into staff\_phone values(%s, %s)'          e\_ins = 'insert into staff\_email values(%s, %s)'          cursor.execute(ins, (username, airline\_name, password, first\_name, last\_name, birth\_date))          for num in p\_list:              cursor.execute(p\_ins, (username, num.strip()))          for email in e\_list:              cursor.execute(e\_ins, (username, email.strip())) |
| Login | Check if hashed password matches hash in customer  username = request.form['customer\_email']      password = hashlib.md5(request.form['customer\_pw'].encode()).hexdigest()      cursor = conn.cursor()      query = 'SELECT \* FROM customer WHERE email = %s and password = %s'      cursor.execute(query, (username, password)) |
|  | Check if hashed password matches hash in staff  username = request.form['staff\_uname']  password = hashlib.md5(request.form['staff\_pw'].encode()).hexdigest()      cursor = conn.cursor()      query = 'SELECT airline\_name FROM airline\_staff WHERE username = %s and password = %s'      cursor.execute(query, (username, password)) |

|  |  |
| --- | --- |
| Customer Use Cases |  |
| View Flights | Check for info future flights booked by this specific customer using email.  email = session['username']      cursor = conn.cursor();      query = 'SELECT flight\_num, airline\_name, airplane\_id, depart\_date\_time, depart\_airport, arrival\_date\_time, arrival\_airport, delay\_status FROM ticket natural join future\_flight WHERE email = %s'      cursor.execute(query, (email))  Can use date ranges and/or airports(optional) to find future/old flights booked by the customer.  cursor = conn.cursor();      query = 'SELECT flight\_num, airline\_name, airplane\_id, depart\_date\_time, depart\_airport, arrival\_date\_time, arrival\_airport, base\_price, delay\_status \              FROM flight natural join ticket \              WHERE email = %(name)s \                  and (depart\_airport = %(dep)s or %(dep)s = "") \                  and (arrival\_airport = %(arr)s or %(arr)s = "") \                  and (CONVERT(depart\_date\_time, date) between %(d1)s and %(d2)s);' |
| Search for flights | Queries are identical to public search use case. |
| Purchase Tickets | Check if desired flight is full using open\_flight view.  query1 = "SELECT \*\                  FROM open\_flight\                  WHERE flight\_num = %s and depart\_date\_time = %s"      cursor.execute(query1, (flight\_num, depart\_date\_time))  Check if booked seats exceed 60% of max capacity to determine ticket price.   query2 = "SELECT count(email)/seating\_capacity as ratio\                      FROM ticket natural join open\_flight natural join airplane\                      WHERE flight\_num = %s \                          and depart\_date\_time = %s"          cursor.execute(query2, (flight\_num, depart\_date\_time))          cursor.close()          data2 = float(cursor.fetchone()['ratio'])          price = float(data1[0]['base\_price'])          if data2 >= 0.6:              price \*= 1.2  Find vacant ticket with the desired flight number  query = "SELECT ticket\_id\              FROM ticket\              WHERE flight\_num = %s and depart\_date\_time = %s"      cursor.execute(query, (flight\_num, depart\_date\_time))  Update ticket values with customer info  ins = "UPDATE ticket\              SET sold\_price = %s, email = %s, card\_type = %s, card\_number = %s, card\_name = %s, expire\_date = %s, depart\_date\_time=depart\_date\_time \              WHERE ticket\_id = %s" |
| Cancel Trip | Check if time difference between present selected flight is greater than 24 hours.  query1 = "SELECT \* \              FROM future\_flight \              WHERE flight\_num = %s \                  and depart\_date\_time = %s \                  and (TIMESTAMPDIFF(HOUR, NOW(), depart\_date\_time) > 24)"  Update ticket with matching customer email to remove their information.  ins = "UPDATE ticket\                  SET sold\_price = NULL, email = NULL, card\_type = NULL, card\_number = NULL, card\_name = NULL, expire\_date = NULL, depart\_date\_time=depart\_date\_time, purchase\_date\_time = NULL \                  WHERE flight\_num = %s and email = %s" |
| Rate/Comment on previous flights | Ensure flight has occurred or is occurring by comparing depart date with current date.  query1 = "SELECT \* \              FROM future\_flight \              WHERE flight\_num = %s \                  and depart\_date\_time = %s \                  and (TIMESTAMPDIFF(HOUR, NOW(), depart\_date\_time) < 0)"  Check if consumer already made a rating for that specific flight.  query2 = "SELECT \* \              FROM rate \              WHERE flight\_num = %s \                  and depart\_date\_time = %s \                  and email = %s"  Insert customer and flight data into rate.  ins = "INSERT into rate values(%s, %s, %s, %s, %s)"      cursor.execute(ins, (username, flight\_num, depart\_date\_time, rating, comment)) |
| Track Spending | Find total spending of consumer in past year  query1 = "SELECT sum(sold\_price) as total\              FROM ticket\              WHERE email = %s \                  and CONVERT(purchase\_date\_time, date) between DATE\_ADD(CURDATE(), INTERVAL -1 YEAR) and CURDATE();"  Find monthly spending of consumer in past 6 months, use date\_format to identify purchase month in the tickets.  query2 = "SELECT date\_format(purchase\_date\_time, '%%M') as month, sum(sold\_price) as m\_spend \              FROM ticket WHERE email = %s \                  and CONVERT(purchase\_date\_time, date) between DATE\_ADD(CURDATE(), INTERVAL -6 MONTH) and CURDATE() \              GROUP by date\_format(purchase\_date\_time, '%%M')"  Find total spending of consumer based on given date range  query1 = "SELECT sum(sold\_price) as total\              FROM ticket\              WHERE email = %s \                  and CONVERT(purchase\_date\_time, date) between %s and %s;"  Find monthly spending of consumer within given date range  query2 = "SELECT date\_format(purchase\_date\_time, '%%M') as month, sum(sold\_price) as m\_spend \              FROM ticket WHERE email = %s \                  and CONVERT(purchase\_date\_time, date) between %s and %s \              GROUP by date\_format(purchase\_date\_time, '%%M')" |
| Logout | No queries needed, destroy session info upon logout. |

**Staff Use Cases**

|  |  |
| --- | --- |
| View Flights | Find flight info on all flights with the user’s airline name (use session information).  airline = session['airline']      cursor = conn.cursor();      query = 'SELECT \* FROM future\_flight WHERE airline\_name = %s'      cursor.execute(query, (airline)) |
| Create New Flights | Use flight number and depart date to check if flight already exists.  query1 = 'SELECT \* FROM flight WHERE flight\_num = %s and depart\_date\_time = %s'  Insert flight info into flight.  ins1 = 'INSERT into flight values(%s, %s, %s, %s, %s, %s, %s, %s, %s)'          cursor.execute(ins1, (flight\_num, depart\_date\_time, airplane\_id, airline, depart\_airport, arrival\_airport, arrival\_date\_time, base\_price, delay\_status))  Find seating capacity of plane to determine number of tickets that must be created.  query2 = 'SELECT seating\_capacity \                  FROM airplane WHERE airplane\_id = %s'  Find highest existing ticket ID value.   query3 = 'SELECT ticket\_id \                  FROM ticket\                  ORDER BY ticket\_id DESC\                  LIMIT 1'  Use the two previous values to determine the amount of tickets to be inserted.  data3 = int(cursor.fetchone()['ticket\_id'])          data3 += 1          for i in range(data3, data3+data2):              ins2 = 'INSERT into ticket values(%s, NULL, %s, %s, NULL, NULL, NULL, NULL, NULL, NULL)'              cursor.execute(ins2, (i, flight\_num, depart\_date\_time)) |
| Change Status of flights | Update delay\_status of flight.   ins = 'UPDATE flight \              SET delay\_status = %s, depart\_date\_time = depart\_date\_time\              WHERE flight\_num = %s and depart\_date\_time = %s'  Acquire info on modified flight to display on website.  displayQuery = 'SELECT \* \                  FROM flight WHERE flight\_num = %s and depart\_date\_time = %s' |
| Add airplane to system | Check if plane already exists in airplane.   query = 'SELECT \* FROM airplane WHERE airplane\_id = %s and airline\_name = %s'  Add in plane info as a new entry for airplane.  ins = 'INSERT into airplane values(%s, %s, %s, %s, %s)'          cursor.execute(ins, (airplane\_id, airline, seat\_capacity, manufacturing\_company, age)) |
| Add new airport to system | Check if airport already exists.  query = 'SELECT \* FROM airport WHERE name = %s'  Add in info for the new entry in airport.  query1 = 'INSERT into aiport values(%s, %s, %s, %s)'          cursor.execute(query1, (name, city, country, port\_type)) |
| View flight ratings | Determine average ratings of a specific flight.  query1 = 'SELECT avg(rating\_level) as avg\                      FROM rate\                      WHERE flight\_num = %s and depart\_date\_time = %s'  Find information on individual customers and their rating info for the specific flight.   query2 = 'SELECT name, email, rating\_level, comment\              FROM customer natural join rate\              WHERE flight\_num = %s and depart\_date\_time = %s;' |
| View frequent customers | Create two temp relations, one holding the total number of flights (for the specific airline) taken by each customer, and another containing the highest number of flights taken by a customer. Compare the two values to find the most frequent customer.  query5 = "WITH flightCount(email, amount) as (\                      SELECT email, count(ticket\_ID)\                      FROM ticket natural join flight\                      WHERE airline\_name = %s \                          and email is not NULL\                          and CONVERT(purchase\_date\_time, date) between DATE\_ADD(CURDATE(), INTERVAL -1 YEAR) and CURDATE() \                      GROUP by email  ),\                  mostFlights(flights) as (\                      SELECT max(amount)\                      FROM flightCount)\                  SELECT name, flights\                  FROM (customer natural join flightCount), mostFlights\                  WHERE flightCount.amount = mostFlights.flights" |
| View reports | Finds the total number of tickets sold in the past month using purchase dates.  query1 = "SELECT count(ticket\_ID) as total\                  FROM ticket natural join flight\                  WHERE airline\_name = %s\                      and email is not null\                      and CONVERT(purchase\_date\_time, date) between DATE\_ADD(CURDATE(), INTERVAL -1 MONTH) and CURDATE()"  Modify the previous query to find tickets sold in the past year.  query1 = "SELECT count(ticket\_ID) as total\                  FROM ticket natural join flight\                  WHERE airline\_name = %s\                      and email is not null\                      and CONVERT(purchase\_date\_time, date) between DATE\_ADD(CURDATE(), INTERVAL -1 YEAR) and CURDATE()"  Do the same to find tickets sold in a given date range.  query1 = "SELECT count(ticket\_ID) as total\                  FROM ticket natural join flight\                  WHERE airline\_name = %s\                      and email is not null\                      and CONVERT(purchase\_date\_time, date) between %s and %s;"  Find the monthly number of tickets sold in the past 6 months using purchase date again.  query2 = "SELECT date\_format(purchase\_date\_time, '%%M') as month, count(ticket\_ID) as m\_sold\                  FROM ticket natural join flight\                  WHERE airline\_name = %s\                      and email is not null\                      and CONVERT(purchase\_date\_time, date) between DATE\_ADD(CURDATE(), INTERVAL -1 YEAR) and CURDATE()\                  GROUP BY date\_format(purchase\_date\_time, '%%M');"      cursor.execute(query2, (airline))      data2 = cursor.fetchall()  Same thing, but only for one month.      query2 = "SELECT date\_format(purchase\_date\_time, '%%M') as month, count(ticket\_ID) as m\_sold\                  FROM ticket natural join flight\                  WHERE airline\_name = %s\                      and email is not null\                      and CONVERT(purchase\_date\_time, date) between DATE\_ADD(CURDATE(), INTERVAL -1 MONTH) and CURDATE()\                  GROUP BY date\_format(purchase\_date\_time, '%%M');"  Modify the query again to find monthly ticket sales within the date range.     query2 = "SELECT date\_format(purchase\_date\_time, '%%M') as month, count(ticket\_ID) as m\_sold\                  FROM ticket natural join flight\                  WHERE airline\_name = %s\                      and email is not null\                      and CONVERT(purchase\_date\_time, date) between %s and %s\                  GROUP BY date\_format(purchase\_date\_time, '%%M');" |
| View revenue | Use sum() and purchase dates to find the airline’s total revenue in the past year.  query3 = "SELECT sum(sold\_price) as rev\                  FROM ticket natural join flight\                  WHERE airline\_name = %s\                      and email is not null\                      and CONVERT(purchase\_date\_time, date) between DATE\_ADD(CURDATE(), INTERVAL -1 YEAR) and CURDATE();"  Modify query to find revenue in the past month.  query4 = "SELECT sum(sold\_price) as rev\                  FROM ticket natural join flight\                  WHERE airline\_name = %s\                      and email is not null\                      and CONVERT(purchase\_date\_time, date) between DATE\_ADD(CURDATE(), INTERVAL -1 MONTH) and CURDATE();" |
| Logout | No queries needed, destroy all session info during logout. |