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.