Angela Yeung ay2095
Ketan Vanjani kv932
Sunny Jeong sj3048

Application Use Cases:

View Public Info

Search for Future Flights (One way):

- Inputs from user: Departure airport, arrival airport, departure date

```
query = 'SELECT * FROM Flight WHERE departure_airport = %s and arrival_airport
= %s and departure_date = %s'
cursor.execute(query, (departure_airport, arrival_airport, departure_date)
```

- Query selects all entities from the flight matching the input information from the Flight database
- The HTML outputs the matching flight's airline, departure airport, departure date, departure time, arrival airport, arrival date, and arrival time in a table
- An error message is output if no matching flight is found

Search for Future Flights (Round trip):

- Inputs from user: Departure airport, arrival airport, departure date, return date

```
query1 = 'SELECT * FROM Flight WHERE departure_airport = %s and arrival_airport
= %s and departure_date = %s'
cursor1.execute(query1, (departure_airport, arrival_airport, departure_date))
```

- Query 1 selects all entities from the flight matching the input information from the Flight database as the departing flight in the round trip

```
query2 = 'SELECT * FROM Flight WHERE departure_airport = %s and arrival_airport
= %s and departure_date = %s'
cursor2.execute(query2, (arrival_airport, departure_airport, return_date))
```

- Query2 selects all entities from the flight matching the input information from the Flight database with the arrival airport and departure airport flipped as the return flight
- The HTML outputs the departure flight's airline, departure airport, departure date, departure time, arrival airport, arrival date, and arrival time in a table and outputs the arrival flight's airline, departure airport, departure date, departure time, arrival airport, arrival date, and arrival time in another table
- An error message is output if no matching flight is found

Flight Status:

- Inputs from user: Airline name, flight number, departure date, and arrival date

```
query = 'SELECT * FROM flight WHERE airline = %s and flight_number = %s \
        and departure_date = %s and arrival_date = %s'
cursor.execute(query, (airline, flight_number, departure_date, arrival_date))
```

- Query selects all entities from the flight matching the input information from the Flight database
- The HTML file outputs the status of the matching flight in a table
- An error message is output if no matching flight is found

Register

Customer Register:

- Inputs from user: Username (email), password, name, address (building number, building street name, city, and state), phone number, passport number, passport expiration date, passport country, birthday

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

- This query checks if the email address the user inputs is already in the database
- An error message is output if a matching email is found

```
ins = 'INSERT INTO Customer VALUES(%s, MD5(%s), %s, %s, %s, %s, %s, %s, %s, %s, %s)'
cursor.execute(ins, (username, password, name, building_num, street_name, city, state, phone_num, passport_num, passport_exp, passport_country, birthday))
```

- If the user does not already exist, the ins query is run
- Query inserts a new customer into the Customer table in the database using the inputted information. The input password is run through MD5() for encryption and the encrypted value is stored in the database

Airline Staff Register:

- Inputs from user: Username, password, airline, first name, last name, birthday, email

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

- This query checks if the username the user inputs is already in the database
- An error message is output if a matching username is found

```
ins = 'INSERT INTO airline_staff VALUES(%s, %s, MD5(%s), %s, %s, %s)'
ins2 = 'INSERT INTO airline_staff_email VALUES(%s, %s)'
ins3 = 'INSERT INTO airline_staff_phone_number VALUES(%s, %s)'
cursor.execute(ins, (username, airline, password, first_name, last_name,
birthday))
cursor.execute(ins2, (username, email))
cursor.execute(ins3, (username, phone_num))
```

- If the user does not already exist, the ins, ins2, and ins3 query are run
- Ins inserts a new airline staff into the Airline Staff table in the database using the inputted information. The input password is run through MD5() for encryption and the encrypted value is stored in the database
- Ins2 inserts a new airline staff into the Airline Staff Email database using the inputted information
- Ins3 inserts a new airline staff in the Airline Staff Phone Number database using the inputted information

Login

Customer Login:

- Inputs from user: Username, password

```
query = 'SELECT * FROM Customer WHERE email = %s and pass_word = MD5(%s)'
cursor.execute(query, (username, password))
data = cursor.fetchone()
```

- Query selects all entities from the customer that matches the inputted information from the Customer table. The inputted password is ran through the MD5() function and that value is compared to the encrypted passwords stored in the database
- Those entities are inserted into a variable called data
- If data contains values, then the customer exists in the database and a session is created using the username
- If data contains no values (= None), then the customer has not been registered in the database and an error message is displayed

Airline Staff Login:

- Inputs from user: Username, password

```
query = 'SELECT * FROM airline_staff WHERE username = %s and pass_word =
MD5(%s)'
  cursor = conn.cursor()
```

```
cursor.execute(query, (username, password))
data = cursor.fetchone()
```

- Query selects all entities from the customer that matches the inputted information from the Airline Staff table. The inputted password is ran through the MD5() function and that value is compared to the encrypted passwords stored in the database
- Those entities are inserted into a variable called data
- If data contains values, then the airline staff exists in the database and a session is created using the username
- If data contains no values (= None), then the airline staff has not been registered in the database and an error message is displayed

Customer Use Cases:

View My Flights

- Inputs from user: Email

```
query = 'SELECT * FROM Ticket WHERE IF (departure_date >= %s, 1 ,
departure_time >= %s) and c_email = %s'
cursor.execute(query, (curr_date, curr_time, email))
data1 = cursor.fetchall()
```

- Query finds all the future tickets a customer has purchased based off their email address
- An error message is displayed if no future tickets are found

Search For Flights (One Way)

- Uses code from Application Use Case “Search for future flights (one way)

Search for Flights (Round Trip)

- Uses code from Application Use Case “Search for future flights (one way)

Purchase Tickets (One Way)

- Inputs from user: Airline, departure airport, arrival airport, departure date, and departure time, card type, card number, name on card, card expiration date, customer email, current date, and current time

```
query1 = 'SELECT * FROM Flight WHERE airline = %s and departure_airport = %s
and arrival_airport = %s and departure_date = %s and departure_time = %s'
cursor1.execute(query1, (airline, departure_airport, arrival_airport,
departure_date, departure_time))
```

- Query1 is run using the inputted information to find a matching flight

- If a matching flight is found and there are seats left on the flight, the customer's ticket is generated and input into theTicket database

```
query2 = 'INSERT INTO Ticket VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s,
%s, %s, %s)'
cursor2.execute(query2, (t_id, airline, flight_number, departure_date,
departure_time, c_email, base_price, card_type,card_number, card_name,
card_expiration, current_date, current_time))
```

```
num_seats = data1["seatCapacity"] - 1
query3 = 'UPDATE Flight SET seatCapacity = %s WHERE airline = %s and
flight_number = %s and departure_date = %s and departure_time = %s'
cursor3.execute(query3, (num_seats, data1['airline'], data1['flight_number'],
data1['departure_date'], data1['departure_time']))
```

- Query3 updates the seat capacity of the flight after a ticket has been purchased
- If no matching flight is found, an error message is displayed

Purchase Tickets (Round Trip)

- Inputs from user: Airline, flight number (for both flights), departure airport, arrival airport, departure date, and departure time, card type, card number, name on card, card expiration date, customer email, current date, and current time

```
query1 = 'SELECT * FROM Flight WHERE airline = %s and departure_airport = %s
and arrival_airport = %s and flight_number = %s and departure_date = %s and
departure_time = %s'
cursor1.execute(query1, (airline, departure_airport, arrival_airport,
flight_number, departure_date, departure_time))
```

- Query1 finds a departing flight using the inputted information by passing in the parameters into the Flight database

```
query2 = 'SELECT * FROM Flight WHERE airline = %s and departure_airport = %s
and arrival_airport = %s and flight_number = %s and departure_date = %s and
departure_time = %s'
cursor2.execute(query2, (airline, arrival_airport, departure_airport,
flight_number2, return_date, return_time))
```

- Query2 finds a returning flight using the inputted information by passing in the parameters (with the arrival airport now as the departure airport) into the Flight database

- If the remaining seat capacity on the chosen flights is not 0

```
num_seats = data1["seatCapacity"] - 1
query5 = 'UPDATE Flight SET seatCapacity = %s WHERE airline = %s and
flight_number = %s and departure_date = %s and departure_time = %s'
```

```
cursor5.execute(query5, (num_seats, data1['airline'], data1['flight_number'],
data1['departure_date'], data1['departure_time']))
num_seats = data2["seatCapacity"] - 1

query6 = 'UPDATE Flight SET seatCapacity = %s WHERE airline = %s and
flight_number = %s and departure_date = %s and departure_time = %s'
cursor6.execute(query6, (num_seats, data2['airline'], data2['flight_number'],
data2['departure_date'], data2['departure_time']))
```

- Query5 and Query6 updates the seat capacity of the flight after a ticket has been purchased

```
query3 = 'INSERT INTO Ticket VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s,
%s, %s, %s)'
cursor3.execute(query3, (t_id2, airline, flight_number2, return_date,
return_time, c_email, base_price2, card_type, card_number, card_name,
card_expiration, current_date, current_time))

query4 = 'INSERT INTO Ticket VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s,
%s, %s, %s)'
cursor4.execute(query4, (t_id, airline, flight_number, departure_date,
departure_time, c_email, base_price, card_type, card_number, card_name,
card_expiration, current_date, current_time))
```

- Using the flights found by query1 and query2, the ticket the user chooses is inserted into the Ticket database as purchased by them
- If no matching flights are found, an error message is displayed

<u>Cancel Trip</u>

- Input from user: Ticket ID

```
query1 = 'SELECT * FROM Ticket WHERE t_id = %s'
cursor1.execute(query1, t_id)
departure_date = cursor1.fetchone()
```

- Query1 selects the departure date of the matching ticket from the Ticket database
- If the ticket does not exist, an error message is displayed
- If the ticket exists

```
time_diff = (departure_date['departure_date'] - curr_date).days
query2 = 'SELECT * FROM Ticket WHERE (t_id = %s and %s >= 1)'
cursor2.execute(query2, (t_id, time_diff))
```

- Query2 checks that the selected ticket's departure date is more than 24 hours from the current time
- If the selected ticket's departure date is more than 24 hours ahead,

```
query4 = 'SELECT seatCapacity from Flight WHERE airline = %s and
flight_number = %s and departure_date = %s and departure_time = %s'
cursor4.execute(query4, (airline, flight_number, departure_date,
departure_time))
data4 = cursor4.fetchone()

num_seats = data4['seatCapacity'] + 1
query5 = 'UPDATE Flight SET seatCapacity = %s WHERE airline = %s and
flight_number = %s and departure_date = %s and departure_time = %s'
cursor5.execute(query5, (num_seats, airline, flight_number,
departure_date, departure_time))
```

- Query4 selects the current amount of available seats from flight associated with the ticket
- Query5 updates the amount of available seats from the flight by adding one more available seat

```
query3 = 'DELETE FROM Ticket WHERE (t_id = %s and %s >= 1)'
cursor3.execute(query3, (t_id, time_diff))
cursor3.close()
```

- Query3 deletes the ticket from the Ticket table and returns the ticket as available for purchase in the database
- If the selected ticket's departure date is less than 24 hours ahead, than an error message is displayed saying that the user cannot cancel the flight within 24 hours of its departure

<u>Give Ratings and Comments</u>

- Input from users: Ticket ID, customer email, rating (0-5), and comments

```
query1 = 'SELECT * FROM Ticket WHERE t_id = %s'
cursor1.execute(query1, (t_id))
departure_info = cursor1.fetchone()
flight_num = departure_info['flight_number']
departure_date = departure_info['departure_date']
departure_time = departure_info['departure_time']
```

- Query1 selects all the entities and information associated with the matching ticket to the imputed Ticket ID
- The flight number, departure date, and departure time is taken from the result of query1
- If the query does not return anything, the ticket does not exist and an error message is displayed

```
date_diff = departure_date - curr_date
query2 = 'SELECT * FROM Ticket WHERE (t_id = %s and %s <= 0)'
cursor2.execute(query2, (t_id, date_diff))
data1 = cursor2.fetchone()
```

- Query2 checks that the customer has taken the flight the customer wants to leave a rating on (aka there is a difference between the flight's departure time and the current time)
- If this requirement is not met, then an error message stating that the customer cannot give a rating at this time is displayed
- If this requirement is met,

```
query3 = 'INSERT into Rates VALUES (%s, %s, %s, %s, %s, %s)'
cursor3.execute(query3, (flight_num, departure_date, departure_time, email,
rating, comment))

query4 = 'SELECT flight_number, rating, comments FROM Rates WHERE flight_number
= %s and departure_date = %s and departure_time = %s and email = %s'
cursor4.execute(query4, (flight_num, departure_date, departure_time, email))
data2 = cursor4.fetchall()
```

- Query3 inserts the rating into the Rates table
- The rating is then displayed on the web application using query4 as a table using the associated HTML file

Track My Spending

- Input from user: Customer email, start date, end date

- If no start date, end date (time range) is inserted,

```
query1 = 'SELECT t_id, sold_price FROM Ticket WHERE c_email = %s and (%s -
curr_date) <= 4450'
cursor1.execute(query1, (email, current_date))
data1 = cursor1.fetchall()
```

- Query1 selects all the tickets purchased by the customer and their purchasing prices in the past 6 months

```
query3 = 'SELECT SUM(sold_price) as total_sum FROM Ticket WHERE c_email = %s
and (%s - curr_date) <= 4450'
cursor3.execute(query3, (email, current_date))
data3 = cursor3.fetchall()
```

- Query3 calculates the total amount the customer spent on tickets in the last 6 months and the HTML displays the total amount in a table

- If a specific time range is inserted into the web application,

```
query2 = 'SELECT t_id, sold_price FROM Ticket WHERE c_email = %s and curr_date
>= %s and curr_date <= %s'
cursor2.execute(query2, (email2, startDate, endDate))
data2 = cursor2.fetchall()
```

```
query4 = 'SELECT SUM(sold_price) as total_sum FROM Ticket WHERE c_email = %s
and (curr_date >= %s and curr_date <= %s)'
cursor4.execute(query4, (email2, startDate, endDate))
data4 = cursor4.fetchall()
```

- Query2 selects the ticket ID and ticket price of all the tickets purchased by the customer during the specified period
- Query4 calculated the total amount the customer spent on tickets during the time period and the HTML file prints out the summed value
- If the date range is invalid, an error message is displayed

Log Out

- If the log out link is clicked, the session is cleared and the user is redirected to the home page

Airline Staff Use Cases:

View Flights

- User Input: staff's airline

```
query1 = 'SELECT * FROM Flight WHERE airline = %s and departure_date <= %s \
          and departure_date >= %s'
 cursor1.execute(query1, (airline, one_month_later, curr_date))
```

- Query1 selects all the future flights scheduled within a 1 month range from the current date that has the same airline as the staff's associated airline

- User Input: staff's airline, start date, end date

```
query2 = 'SELECT * FROM Flight WHERE airline = %s and departure_date <= %s and
departure_date >= %s and departure_airport = %s and arrival_airport = %s'
cursor2.execute(query2, (airline2, endDate, startDate, source_airport,
dest_airport))
```

- Query2 selects all the flights scheduled with the same airline as the staff and within a specific range of time inputted by the staff.

- User Input: flight number

```
query3 = 'SELECT c_email FROM Flight NATURAL JOIN Ticket WHERE flight_number =
%s'
cursor3.execute(query3, flight_number)
```

- Query3 selects all user emails of the customers that have at least one ticket from a specific flight.

Create New Flights

- User Input: staff airline, flight airline, flight number, departure date, departure time, arrival airport, arrival date, arrival time base price, airplane ID, flight status

```
query = 'INSERT INTO Flight VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s,
(SELECT num_of_seats FROM Airplane WHERE airplane_id = %s))'
cursor.execute(query, (airline, flight_number, departure_airport,
departure_date, departure_time, arrival_airport, arrival_date, arrival_time,
base_price, airplane_id, flight_status, airplane_id))
```

- Query inserts in a new instance of the flight into the Flight table using the inputted information and gets the number of seats available for purchase from the Airplane table

Change Status of Flights

- User Input: staff airline, flight airline, flight number, departure date, departure time, flight status

```
query = 'UPDATE Flight SET flight_status = %s WHERE airline = %s and
flight_number = %s and departure_date = %s and departure_time = %s'
cursor.execute(query, (flight_status, airline, flight_number, departure_date,
departure_time))
```

- Query updates the flight status to the inputted status after finding the matching flight using the inputted information
- If a matching flight cannot be found, an error message is displayed

Add airplane into the System

- User input: Airline, airplane ID, number of seats, manufacturing company, and age

```
query = 'INSERT INTO Airplane VALUES (%s, %s, %s, %s, %s)'
cursor.execute(query, (airline, airplane_id, num_of_seats,
manufacturing_company, age))
```

- Query inserts a new airplane using the inputted information into the Airplane table
- If the staff's airline does not match the airplane's airline, an error message is displayed and no airplane is inserted into the database

Add new airport into the System

- User input: Airport name, city, country, and type (international, domestic, both)

```
query2 = 'SELECT * FROM Airport WHERE name = %s'
cursor2.execute(query2, (name))
```

- Query2 checks if the airport already exists in the database
- If the airport already exists, an error message is displayed
- If the airport does not already exist,

```
query = 'INSERT INTO Airport VALUES (%s, %s, %s, %s)'
cursor.execute(query, (name, city, country, type))
```

- Query inserts a new airport into the Airport table in the database using the inputting values

View Flight Ratings

- Input from user: Flight number, departure date, and departure time

```
query1 = 'SELECT email, rating, comments FROM Rates WHERE flight_number = %s
and departure_date = %s and departure_time = %s'
cursor1.execute(query1, (flight_number, departure_date, departure_time))
```

- Query1 selects the customer email, rating, and comments from Rates using the matching instance using the inputted information
- The HTML file outputs each rating for the matching flight in a table

```
query2 = 'SELECT flight_number, AVG(rating) as avg_rating FROM Rates WHERE
flight_number = %s and departure_date = %s and departure_time = %s'
cursor2.execute(query2, (flight_number, departure_date, departure_time))
```

- Query2 calculated the average rating of the matching flight from the Rates table
- The HTML file outputs the average rating

View Frequent Customers

- Input from user: Airline and customer email

```
query1 = 'SELECT a.* from (SELECT c_email, COUNT(c_email) as flight_occurance
FROM ticket WHERE curr_date >= %s and curr_date <= %s and airline = %s GROUP BY
c_email) a WHERE flight_occurance in (SELECT MAX(flight_occurance) FROM (SELECT
c_email, COUNT(c_email) as flight_occurance FROM Ticket WHERE curr_date >= %s
and curr_date <= %s and airline = %s GROUP BY c_email) a)'
cursor1.execute(query1, (past_year, curr_date, airline, past_year, curr_date,
airline))
```

- Query1 selects the email and amount of tickets bought of the most frequent customer in the past year

```
query2 = 'SELECT * FROM ticket WHERE c_email = %s and airline = %s'
cursor2.execute(query2, (customer, airline2))
```

- Query2 selects the tickets a customer has purchased from an airline
- The HTML file outputs the details of each ticket
- If the imputed values do not match, an error message is displayed to indicate that there are no matching tickets purchased

View Reports

- Inputs from user: Start date and end date of time interval

```
query1 = 'SELECT COUNT(t_id) as num_tickets from Ticket WHERE curr_date >= %s
and curr_date <= %s'
cursor1.execute(query1, (startDate, endDate))
```

- Query1 selects the number of tickets purchased during the time period

```
query2 = 'SELECT month(curr_date) AS month, COUNT(t_id) as num_ticket from
Ticket WHERE curr_date >= %s and curr_date <= %s GROUP BY month(curr_date)'
cursor2.execute(query2, (startDate, endDate))
```

- Query2 selects the month and number of tickets purchased that month for every month in the input time interval
- The HTML file outputs the data in a table

View Earned Revenue

```
query1 = 'SELECT SUM(sold_price) as Total_Revenue_Past_Year FROM Ticket WHERE
curr_date >= %s and curr_date <= %s'
cursor1.execute(query1, (past_year, curr_date))
```

- As the default, query1 calculates the total revenue in the past year by summing the prices of the Tickets
- The HTML file outputs this value

```
query2 = 'SELECT SUM(sold_price) as Total_Revenue_Past_Month FROM Ticket WHERE
curr_date >= %s and curr_date <= %s'
cursor2.execute(query2, (past_month, curr_date))
```

- If a date range is specified, query2 calculates the total revenue in the time range by summing the prices of the Tickets
- The HTML file outputs the total value

Add Multiple Phone Numbers

- Input from user: Username and phone number

```
query1 = 'SELECT * from airline_staff_phone_number where phone_number = %s'
cursor1.execute(query1, (phone_num))
data1 = cursor1.fetchall()
```

- Query1 checks if the inputted phone number already exists in the database for the user
- If the phone number already exists, an error message is displayed

- If the phone number does not already exist,

```
query2 = 'INSERT INTO airline_staff_phone_number VALUES(%s, %s)'
cursor2.execute(query2, (username, phone_num))
```

- The phone number is inserted into the Airline Staff Phone Number table with its associated username

Log Out

- If the log out link is clicked, the session is cleared and the staff user is redirected to the home page