File Breakdown

---

airline.sql: Contains all tables and inserts for the Air_Ticket_Reservation_System.

Init1.py: Contains all SQL queries and sends the data to designated .html files.

templates/index.html: This is the home page and contains use cases “View Public Information”, which consists of “Search for Future Flights” and “Get Flight Status”.

1. Search for Future Flights

Query

```
search_flights = 'SELECT * FROM Flight WHERE departure_airport = %s AND arrival_airport = %s
AND trip_type = %s AND departure_date >= DATE(NOW()) AND arrival_date >= DATE(NOW()) AND
departure_date >= %s AND arrival_date <= %s'

cursor.execute(search_flights, (source, destination, trip_type, departure, arrival))
```

Explanation

When a user submits a “Search for Flights” form, the ‘source’, ‘destination’, ‘trip_type’, ‘departure’, and ‘arrival’ are retrieved using the POST method and are used in the above query. Those variables are used in the WHERE clause to SELECT corresponding flights from the Flight Table. The WHERE clause also ensures that a user can only see future flights even if they have selected a departure and arrival date from the past. This was done by using DATE(NOW()).

2. Get Flight Status

Query

```
search_flights = 'SELECT flight_status FROM Flight WHERE airline_name = %s AND flight_num =
%s AND departure_date = %s AND arrival_date = %s'

cursor.execute(search_flights, (airline_name, flight_num, departure, arrival))
```

Explanation

When a user submits a “Get Flight Status” form, the ‘airline_name’, ‘flight_num’, ‘departure’, and ‘arrival’ are retrieved using the POST method and are used in the above query. Those variables are used in the WHERE clause to SELECT the flight_status of corresponding flights from the Flight Table.

templates/customerRegistration.html

1. Customer Registration

Query

```
#Checks if customer already exists
query = 'SELECT * FROM Customer WHERE cus_email = %s'
cursor.execute(query, (email))

#Inserts if customer does not exist
ins = 'INSERT INTO Customer VALUES(%s, %s, MD5(%s), %s, %s, %s, %s, %s, %s, %s, %s, %s)'
cursor.execute(ins, (name, email, password, buildingNum, street, city, state, phone, dob,
passportNum, passportExp, passportCountry))
```

Explanation

When a customer submits a "Customer Registration" form, the 'name', 'email', 'password', 'buildingNum', 'street', 'city', 'state', 'phone', 'dob', 'passportNum', 'passportExp', and 'passportCountry' are retrieved using the POST method and are used in the above query. Those variables are used as VALUES to be INSERTED into the Customer Table.

templates/agentRegistration.html

1. Booking Agent Registration

Query

```
#Checks if booking agent already exists
query = 'SELECT * FROM Booking_Agent WHERE agent_email = %s'
cursor.execute(query, (email))

#Inserts if booking agent does not exist
ins = 'INSERT INTO Booking_Agent (agent_email, agent_password, airline_name) VALUES(%s,
MD5(%s), %s)'
cursor.execute(ins, (email, password, airline))
```

Explanation

When a booking agent submits a "Booking Agent Registration" form, the 'agent_email', 'agent_password', and 'airline_name' are retrieved using the POST method and are used in the above query. Those variables are used as VALUES to be INSERTED into the Booking_Agent Table.

templates/staffRegistration.html

1. Airline Staff Registration

Query

```
#Checks if airline staff already exists
query = 'SELECT * FROM Airline_Staff WHERE staff_username = %s'
cursor.execute(query, (username))

#Inserts if airline staff does not exist
ins = 'INSERT INTO Airline_Staff VALUES(%s, %s, %s, MD5(%s), %s, %s, %s)'
cursor.execute(ins, (firstName, lastName, username, password, dob, phone, airline))
```

Explanation

When an airline staff member submits a "Airline Staff Registration" form, the 'firstName', 'lastName', 'username', 'password', 'dob', 'phone', and 'airline' are retrieved using the POST method and are used in the above query. Those variables are used as VALUES to be INSERTED into the Airline_Staff Table.

templates/customerLogin.html

1. Customer Login

Query

```
query = 'SELECT * FROM Customer WHERE cus_email = %s and cus_password = MD5(%s)'
cursor.execute(query, (email, password))
```

Explanation

When a customer submits a "Customer Login" form, the 'email', and 'password' are retrieved using the POST method and are

used in the above query. Those variables are used to SELECT a corresponding customer in the Customer Table. If there is a result, then the customer is granted access to their personal homepage.

templates/agentLogin.html

1. Booking Agent Login

Query

```
query = 'SELECT * FROM Booking_Agent WHERE agent_email = %s and agent_password = MD5(%s)'
cursor.execute(query, (email, password))
```

Explanation

When a booking agent submits a "Booking Agent Login" form, the 'email', and 'password' are retrieved using the POST method and are used in the above query. Those variables are used to SELECT a corresponding booking agent in the Booking_Agent Table. If there is a result, then the booking agent is granted access to their personal homepage.

templates/staffLogin.html

1. Airline Staff Login

Query

```
query = 'SELECT * FROM Airline_Staff WHERE staff_username = %s and staff_password = MD5(%s)'
cursor.execute(query, (username, password))
```

Explanation

When an airline staff member submits an "Airline Staff Login" form, the 'username', and 'password' are retrieved using the POST method and are used in the above query. Those variables are used to SELECT a corresponding airline staff member in the Airline_Staff Table. If there is a result, then the airline staff member is granted access to their personal homepage.

templates/customerHome.html

1. View My Flights:

Query (All Flights)

```
# Finds all flights that a customer has purchased from Customer_Purchases Table
find_flights = 'SELECT flight_num, ticket_ID, sold_price FROM Customer_Purchases WHERE
cus_email = %s'
cursor.execute(find_flights, (username))
all_flights = cursor.fetchall()

flightsOccured = []
all_flights_info = []


# Finds additional information about the flights using the Flight Table
for flight in all_flights:

        # Prevents duplicate flight information from being selected
        if (flight['flight_num'] not in flightsOccured):

                find_all_flights_info = 'SELECT * FROM Flight WHERE flight_num = %s'

                cursor.execute(find_all_flights_info, (flight['flight_num']))
                flight_info = cursor.fetchone()
                all_flights_info.append(flight_info)
```

```
            flightsOccured.append(flight['flight_num'])
```

Explanation

When a customer views their personal homepage, they can view all of the flights that they have purchased tickets for. In the first query, for every purchase that a customer has made, their flight_num and ticket_ID is SELECTED from the Customer_Purchases Table. The Customer_Purchases table has limited information about the flight, so the Flight Table is also used. In the second query, all information about a flight that the customer has taken is SELECTED and added to an all_flights_info[] and the flight_num has been added to flightsOccured[] to prevent SELECTING the same flight_data more than once if a customer has multiple tickets for the same flight.

Query (Past Flights)

```
flightsOccured = []
past_flights_info = []


for flight in all_flights:

       # Prevents duplicate flight information from being selected
       if (flight['flight_num'] not in flightsOccured):

              find_past_flights_info = 'SELECT * FROM Flight WHERE flight_num = %s AND
              departure_date < DATE(NOW())'

              cursor.execute(find_past_flights_info, (flight['flight_num']))
              flight_info = cursor.fetchone()
              past_flights_info.append(flight_info)

              flightsOccured.append(flight['flight_num'])
```

Explanation

When a customer views their personal homepage, they can view all of their past flights that they have purchased tickets for. In the first query, for every purchase that a customer has made, their flight_num and ticket_ID is SELECTED from the Customer_Purchases Table. The Customer_Purchases table has limited information about the flight, so the Flight Table is also used. In the second query, all information about a past flight that the customer has taken is SELECTED and added to an past_flights_info[] and the flight_num has been added to flightsOccured[] to prevent SELECTING the same flight_data more than once if a customer has multiple tickets for the same flight.

Query (Future Flights)

```
flightsOccured = []
future_flights_info = []

for flight in all_flights:

       # Prevents duplicate flight information from being selected
       if (flight['flight_num'] not in flightsOccured):

              find_future_flights_info = 'SELECT * FROM Flight WHERE flight_num = %s AND
              departure_date > DATE(NOW())'

              cursor.execute(find_future_flights_info, (flight['flight_num']))
              flight_info = cursor.fetchone()
              future_flights_info.append(flight_info)

              flightsOccured.append(flight['flight_num'])
```

Explanation

When a customer views their personal homepage, they can view all of their past flights that they have purchased tickets for. In the first query, for every purchase that a customer has made, their flight_num and ticket_ID is SELECTED from the Customer_Purchases Table. The Customer_Purchases table has limited information about the flight, so the Flight Table is also used. In the second query, all information about a past flight that the customer has taken is SELECTED and added to an past_flights_info[] and the flight_num has been added to flightsOccured[] to prevent SELECTING the same flight_data more than once if a customer has multiple tickets for the same flight.

2. Search for Flights:

Query

```
search_flights = 'SELECT * FROM Flight WHERE departure_airport = %s AND arrival_airport = %s
AND trip_type = %s AND departure_date >= DATE(NOW()) AND arrival_date >= DATE(NOW()) AND
departure_date >= %s AND arrival_date <= %s'

cursor.execute(search_flights, (source, destination, trip_type, departure, arrival))
```

Explanation

When a customer submits a "Search for Flights" form, the 'source', 'destination', 'trip_type', 'departure', and 'arrival' are retrieved using the POST method and are used in the above query. Those variables are used in the WHERE clause to SELECT corresponding flights from the Flight Table. The WHERE clause also ensures that a user can only see future flights even if they have selected a departure and arrival date from the past. This was done by using DATE(NOW()).

3. Purchase Tickets:

Query

```
for ticket_ID in ticket_IDs:

        search_tickets = 'SELECT is_Purchased FROM Ticket WHERE ticket_ID = %s'
        cursor.execute(search_tickets, (ticket_ID))
        is_Purchased = cursor.fetchone()
        is_Purchased = is_Purchased['is_Purchased']

        if(is_Purchased == "No"):

                change_ticket_status = 'UPDATE Ticket SET is_purchased = "Yes" WHERE
                ticket_ID = %s AND is_purchased = "No"'
                cursor.execute(change_ticket_status, (ticket_ID))
                conn.commit()

                # Find the base_price and airplane_ID for the flight
                find_flight_info = 'SELECT base_price, airplane_ID FROM Flight WHERE
                flight_num = %s'
                cursor.execute(find_flight_info, (flight_purchase))
                flight_info = cursor.fetchone()
                base_price = flight_info['base_price']
                airplane_ID = flight_info['airplane_ID']

                # Using the airplane_ID, find the seats on that plane
                find_seats = 'SELECT seats FROM Airplane WHERE airplane_ID = %s'
                cursor.execute(find_seats, (airplane_ID))
                seats = cursor.fetchone()
                seats = int(seats['seats'])

                # Count the number of tickets purchased for the flight
                count_tickets_purchased = 'SELECT COUNT(ticket_ID) AS tickets_purchased FROM
                Customer_Purchases WHERE flight_num = %s'
                cursor.execute(count_tickets_purchased, (flight_purchase))
                tickets_purchased = cursor.fetchone()
                tickets_purchased = int(tickets_purchased['tickets_purchased'])

                # Determine the capacity of the flight by dividing the number of tickets
```

```
purchased by the number of seats on the airplane
capacity = (tickets_purchased / seats) * 100

# If the capacity of the flight> 70, the sold price is 1.2 * base_price
if(capacity > 70):
       sold_price = base_price * 1.2
else:
       sold_price = base_price

insert_purchase = 'INSERT INTO CUSTOMER_PURCHASES (cus_email, ticket_ID,
flight_num, sold_price, card_type, card_num, card_name, card_exp_date,
purchase_date, purchase_time, agent_ID) VALUES (%s, %s, %s, %s, %s, %s, %s,
%s, DATE(NOW()), TIME(NOW()), NULL)'

cursor.execute(insert_purchase, (username, ticket_ID, flight_purchase,
sold_price, card_type, card_num, card_name, card_exp))
conn.commit()
```

Explanation

Because a customer can purchase multiple tickets, we use a for loop to conduct each ticket purchase. First, we determine if a ticket has been purchased. If the ticket has not been purchased, the ticket's 'isPurchased' attribute is changed from 'Yes' to 'No'. Next, we find the base price and airplane ID associated with the flight that we are buying a ticket for. We then use the airplane ID to find out how many seats are on the plane and also find the number of tickets purchased for that flight. After that, we determine the capacity by dividing tickets purchased by seats, if the capacity is greater than 70, the sold price is multiplied by 1.2, otherwise the sold price remains the same as the base price. Once the sold price is determined, the purchase is inserted into the 'Customer_Purchases' Table.

4. Give Ratings and Comment on Previous Flights:

Query

```
# Checks if review has been completed
search_reviews = 'SELECT * FROM Review WHERE flight_num = %s AND cus_email = %s'
cursor.execute(search_reviews, (flight_to_rate, username))
isFlightRated = cursor.fetchall()

# Inserts review if it does not exist
submit_flight_rating = 'INSERT INTO Review(cus_email, flight_num, rating, comment)
VALUES(%s, %s, %s, %s)'
cursor.execute(submit_flight_rating, (username, flight_rated, rating, comment))
conn.commit()
```

Explanation

In the past flights table listed on the customer's home page, there is a column that asks if the user wants to submit a review. If they click yes, 'flight_to_rate' (flight_num) will be retrieved using the POST Method and will be used to SELECT a review from the Review Table that has the corresponding flight_num and cus_email. If there are no results, a review form will be revealed for the customer to submit a review.

When the user submits "Review Previously Taken Flights", 'flight_rated', 'rating', and 'comment' are retrieved using the POST Method and are used as VALUES to INSERT INTO the Review Table.

5. Track My Spending:

Query (Year Expenses)

```
calc_year_expenses = 'SELECT SUM(sold_price) AS year_expenses FROM Customer_Purchases WHERE
cus_email = %s AND purchase_date >= DATE_SUB(DATE(NOW()), INTERVAL 1 YEAR)'

cursor.execute(calc_year_expenses, (username))
year_expenses = cursor.fetchone()
```

Explanation

This query calculates the SUM of the prices that a customer has paid for each ticket that they have purchased within INTERVAL of one year.

Query (Six Month Expenses for Bar Chart)

```
calc_six_month_expenses = 'SELECT SUM(sold_price) AS month_expense, MONTHNAME(purchase_date)
AS month FROM Customer_Purchases WHERE cus_email = %s AND purchase_date >=
DATE_SUB(DATE(NOW()), INTERVAL 6 MONTH) GROUP BY month ORDER BY DATE(NOW())'

cursor.execute(calc_six_month_expenses, (username))
six_month_expenses = cursor.fetchall()
```

Explanation

This query calculates the SUM of the prices that a customer has paid for each ticket that they have purchased within INTERVAL of 6 months. The month in which each of these purchases was made is also SELECTED so that the bar chart can the customer's expenses month wise.

Query (Expenses Within a Range)

```
calc_date_expenses = 'SELECT SUM(sold_price) AS date_expenses FROM Customer_Purchases WHERE
cus_email = %s AND purchase_date >= %s AND purchase_date <= %s'

cursor.execute(calc_date_expenses, (username, start_date, end_date))
date_expenses = cursor.fetchone()
```

Explanation

This query calculates the SUM of the prices that a customer has paid for each ticket that they have purchased within the range specified by the customer.

Query (Expenses Within a Range for Bar Chart)

```
calc_range_month_expenses = 'SELECT SUM(sold_price) AS month_expense,
MONTHNAME(purchase_date) AS month FROM Customer_Purchases WHERE cus_email = %s AND
purchase_date >= %s AND purchase_date <= %s GROUP BY month ORDER BY DATE(NOW())'

cursor.execute(calc_range_month_expenses, (username, start_date, end_date))
range_month_expenses = cursor.fetchall()
```

Explanation

This query calculates the SUM of the prices that a customer has paid for each ticket that they have purchased within the range specified by the customer. The month in which each of these purchases was made is also SELECTED so that the bar chart can the customer's expenses month wise.

templates/agentHome.html

1. View My Flights:

Query

```
# Finds all flights that a booking agent has purchased from Customer_Purchases Table
find_flights = 'SELECT cus_email, flight_num, ticket_ID FROM Customer_Purchases WHERE
```

```
agent_ID = %s'
cursor.execute(find_flights, (agent_ID))
all_flights = cursor.fetchall()

flightsOccured = []
all_flights_info = []

# Finds additional information about each flight that isn't stored in the Cutomer_Purchases
for flight in all_flights:

        # Prevents duplicate flight information from being selected
        if (flight['flight_num'] not in flightsOccured):

                find_all_flights_info = 'SELECT * FROM Flight WHERE flight_num = %s'

                cursor.execute(find_all_flights_info, (flight['flight_num']))
                flight_info = cursor.fetchone()
                all_flights_info.append(flight_info)

                flightsOccured.append(flight['flight_num'])
```

Explanation

When a booking agent views their personal homepage, they can view all of the flights that they have purchased tickets for on behalf of customers. In the first query, for every purchase that a booking agent has made, their flight_num and ticket_ID is SELECTED from the Customer_Purchases Table. The Customer_Purchases table has limited information about the flight, so the Flight Table is also used. In the second query, all information about a flight that the customer has taken is SELECTED and added to an all_flights_info[] and the flight_num has been added to flightsOccured[] to prevent SELECTING the same flight_data more than once if a booking agent has multiple tickets for the same flight.

Query

```
flightsOccured = []
past_flights_info = []

for flight in all_flights:
        # Prevents duplicate flight information from being selected
        if (flight['flight_num'] not in flightsOccured):

                find_past_flights_info = 'SELECT * FROM Flight WHERE flight_num = %s AND
                departure_date < DATE(NOW())'

                cursor.execute(find_past_flights_info, (flight['flight_num']))
                flight_info = cursor.fetchone()
                past_flights_info.append(flight_info)

                flightsOccured.append(flight['flight_num'])
```

Explanation

When a booking agent views their personal homepage, they can view all of the flights that they have purchased tickets for on behalf of customers. In the first query, for every purchase that a booking agent has made, their flight_num and ticket_ID is SELECTED from the Customer_Purchases Table. The Customer_Purchases table has limited information about the flight, so the Flight Table is also used. In the second query, all information about a past flight that the customer has taken is SELECTED and added to an past_flights_info[] and the flight_num has been added to flightsOccured[] to prevent SELECTING the same flight_data more than once if a booking agent has multiple tickets for the same flight.

Query

```
flightsOccured = []
future_flights_info = []
```

```
for flight in all_flights:
       # Prevents duplicate flight information from being selected
       if (flight['flight_num'] not in flightsOccured):

              find_future_flights_info = 'SELECT * FROM Flight WHERE flight_num = %s AND
              departure_date > DATE(NOW())'

              cursor.execute(find_future_flights_info, (flight['flight_num']))
              flight_info = cursor.fetchone()
              future_flights_info.append(flight_info)

              flightsOccured.append(flight['flight_num'])
```

Explanation

When a booking agent views their personal homepage, they can view all of the flights that they have purchased tickets for on behalf of customers. In the first query, for every purchase that a booking agent has made, their flight_num and ticket_ID is SELECTED from the Customer_Purchases Table. The Customer_Purchases table has limited information about the flight, so the Flight Table is also used. In the second query, all information about a future flight that the customer has taken is SELECTED and added to an past_flights_info[] and the flight_num has been added to flightsOccured[] to prevent SELECTING the same flight_data more than once if a booking agent has multiple tickets for the same flight.

2. Search for Flights:

Query

```
search_flights = 'SELECT * FROM Flight WHERE departure_airport = %s AND arrival_airport = %s
AND trip_type = %s AND departure_date >= DATE(NOW()) AND arrival_date >= DATE(NOW()) AND
departure_date >= %s AND arrival_date <= %s'

cursor.execute(search_flights, (source, destination, trip_type, departure, arrival))
query_flights = cursor.fetchall()
```

Explanation

When a customer submits a "Search for Flights" form, the 'source', 'destination', 'trip_type', 'departure', and 'arrival' are retrieved using the POST method and are used in the above query. Those variables are used in the WHERE clause to SELECT corresponding flights from the Flight Table. The WHERE clause also ensures that a user can only see future flights even if they have selected a departure and arrival date from the past. This was done by using DATE(NOW()).

3. Purchase Tickets:

Query

```
for ticket_ID in ticket_IDs:

       search_tickets = 'SELECT is_Purchased FROM Ticket WHERE ticket_ID = %s'
       cursor.execute(search_tickets, (ticket_ID))
       is_Purchased = cursor.fetchone()
       is_Purchased = is_Purchased['is_Purchased']

       if(is_Purchased == "No"):

              change_ticket_status = 'UPDATE Ticket SET is_purchased = "Yes" WHERE
              ticket_ID = %s AND is_purchased = "No"'
              cursor.execute(change_ticket_status, (ticket_ID))
              conn.commit()

              # Find the base_price and airplane_ID for the flight
              find_flight_info = 'SELECT base_price, airplane_ID FROM Flight WHERE
              flight_num = %s'
              cursor.execute(find_flight_info, (flight_purchase))
              flight_info = cursor.fetchone()
```

```
base_price = flight_info['base_price']
airplane_ID = flight_info['airplane_ID']

# Using the airplane_ID, find the seats on that plane
find_seats = 'SELECT seats FROM Airplane WHERE airplane_ID = %s'
cursor.execute(find_seats, (airplane_ID))
seats = cursor.fetchone()
seats = int(seats['seats'])

# Count the number of tickets purchased for the flight
count_tickets_purchased = 'SELECT COUNT(ticket_ID) AS tickets_purchased FROM
Customer_Purchases WHERE flight_num = %s'
cursor.execute(count_tickets_purchased, (flight_purchase))
tickets_purchased = cursor.fetchone()
tickets_purchased = int(tickets_purchased['tickets_purchased'])

# Determine the capacity of the flight by dividing the number of tickets
purchased by the number of seats on the airplane
capacity = (tickets_purchased / seats) * 100

# If the capacity of the flight> 70, the sold price is 1.2 * base_price
if(capacity > 70):
       sold_price = base_price * 1.2
else:
       sold_price = base_price

insert_purchase = 'INSERT INTO CUSTOMER_PURCHASES (cus_email, ticket_ID,
flight_num, sold_price, card_type, card_num, card_name, card_exp_date,
purchase_date, purchase_time, agent_ID) VALUES (%s, %s, %s, %s, %s, %s, %s,
%s, DATE(NOW()), TIME(NOW()), %s)'

cursor.execute(insert_purchase, (cus_email, ticket_ID, flight_purchase,
sold_price, card_type, card_num, card_name, card_exp, agent_ID))
conn.commit()
```

Explanation

Because a booking agent can purchase multiple tickets for a customer, we use a for loop to conduct each ticket purchase. First, we determine if a ticket has been purchased. If the ticket has not been purchased, the ticket's 'isPurchased' attribute is changed from 'Yes' to 'No'. Next, we find the base price and airplane ID associated with the flight that we are buying a ticket for. We then use the airplane ID to find out how many seats are on the plane and also find the number of tickets purchased for that flight. After that, we determine the capacity by dividing tickets purchased by seats, if the capacity is greater than 70, the sold price is multiplied by 1.2, otherwise the sold price remains the same as the base price. Once the sold price is determined, the purchase is inserted into the 'Customer_Purchases' Table.

4. View my Commission:

Query

```
calc_commission = 'SELECT SUM(sold_price) AS commission, COUNT(ticket_ID) AS tickets_sold
FROM Customer_Purchases WHERE agent_ID = %s AND purchase_date >= DATE_SUB(DATE(NOW()),
INTERVAL 30 DAY)'
cursor.execute(calc_commission, (agent_ID))
commission_info = cursor.fetchone()
commission = commission_info['commission']

if (commission == None):
       commission='0.0'
       comm_per_ticket = '0.0'
       tickets_sold = 0
else:
       commission = int(commission) * 0.1
       tickets_sold = int(commission_info['tickets_sold'])
       comm_per_ticket = commission / tickets_sold
```

Explanation

This query calculates the SUM of the prices that a customer has paid for each ticket that they have purchased within the INTERVAL of 30 days and multiplies that amount by 0.1 to determine the commission. This query also COUNTS the number of tickets purchased by a booking agent on behalf of the customer within the INTERVAL of 30 days. Using the commission made and tickets sold within that month, the commission per ticket average was calculated.

5. View Top Customers:

Query (Top 5 Customers Within the Past 6 Months Based on Tickets)

```
find_customers = 'SELECT cus_email, COUNT(ticket_ID) as tickets FROM Customer_Purchases
WHERE agent_ID = %s AND purchase_date >= DATE_SUB(DATE(NOW()), INTERVAL 6 MONTH) GROUP BY
cus_email ORDER BY tickets DESC LIMIT 0, 5'

cursor.execute(find_customers, (agent_ID))
top_customers_month = cursor.fetchall()
```

Explanation

This query COUNTS how many tickets that the booking agent has purchased on behalf of the customer using the Customer_Purchases Table within the INTERVAL of 6 months and SELECTS the 5 customers with the 5 highest ticket counts.

Query (Top 5 Customers Within the Past Year Based on Commission)

```
find_customers = 'SELECT cus_email, SUM(sold_price) * 0.1 as cus_commission FROM
Customer_Purchases WHERE agent_ID = %s AND purchase_date >= DATE_SUB(DATE(NOW()), INTERVAL 1
YEAR) GROUP BY cus_email ORDER BY cus_commission DESC LIMIT 0, 5'

cursor.execute(find_customers, (agent_ID))
top_customers_year = cursor.fetchall()
```

Explanation

This query calculates the SUM of the commission that the booking agent has made by purchasing tickets on behalf of the customer using the Customer_Purchases Table within the INTERVAL of one year and SELECTS the 5 customers who have earned the booking agent the 5 highest commissions.

templates/staffHome.html

1. View Flights:

Query (Past Flights)

```
find_past_flights = 'SELECT * FROM Flight WHERE airline_name = %s AND departure_date >=
DATE_SUB(DATE(NOW()), INTERVAL 30 DAY) AND departure_date <= DATE(NOW())'

cursor.execute(find_past_flights, (airline_name))
past_flights = cursor.fetchall()
```

Explanation

This query SELECTS all past flights that are under the airline staff member's airline within the INTERVAL of 30 days.

Query (Future Flights)

```
Finds flights that will depart within the next 30 days
```

```
find_future_flights = 'SELECT * FROM Flight WHERE airline_name = %s AND departure_date >=
DATE(NOW()) AND departure_date <= DATE_ADD(DATE(NOW()), INTERVAL 30 DAY)'

cursor.execute(find_future_flights, (airline_name))
future_flights = cursor.fetchall()
```

Explanation

This query SELECTS all future flights that are under the airline staff member's airline within the INTERVAL of 30 days.

2. Search for Flights

Query

```
search_flights = 'SELECT * FROM Flight WHERE departure_airport = %s AND arrival_airport = %s
AND trip_type = %s AND departure_date >= DATE(NOW()) AND arrival_date >= DATE(NOW()) AND
departure_date >= %s AND arrival_date <= %s'
cursor.execute(search_flights, (source, destination, trip_type, departure, arrival))
query_flights = cursor.fetchall()
```

Explanation

When an airline staff member submits a "Search for Flights" form, the 'source', 'destination', 'trip_type', 'departure', and 'arrival' are retrieved using the POST method and are used in the above query. Those variables are used in the WHERE clause to SELECT corresponding flights from the Flight Table. The WHERE clause also ensures that a user can only see future flights even if they have selected a departure and arrival date from the past. This was done by using DATE(NOW()).

3. View Customers on Flights

Query

```
search_flights = 'SELECT cus_email FROM Customer_Purchases WHERE flight_num = %s'

cursor.execute(search_flights, (customer_flight_select))
customers = cursor.fetchall()
```

Explanation

When a customer searches for a flight and the flight data is displayed in a table, there is a column that asks if the user wants to "View Customers". If they click yes, customer_flight_select (flight_num) will be retrieved using the POST Method and will be used to SELECT all customers from the Customer_Purchases Table that has the corresponding flight_num.

4. Create New Flights:

Query

```
insert_flight = 'INSERT INTO Flight (airline_name, airplane_ID, departure_airport,
departure_date, departure_time, arrival_airport, arrival_date, arrival_time, base_price,
flight_status, trip_type) VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)'

cursor.execute(insert_flight, (airline_name, airplane_ID, departure_airport, departure_date,
departure_time, arrival_airport, arrival_date, arrival_time, base_price, flight_status,
trip_type))
conn.commit()
```

Explanation

When an airline staff member submits a "Add New Flight" form, the 'airline_name', 'airplane_ID', 'departure_airport', 'departure_date', 'departure_time', 'arrival_airport', 'arrival_date', 'arrival_time', 'base_price', 'flight_status', and 'trip_type' are retrieved using the POST method and are used in the above query. Those variables are used as VALUES to be INSERTED into the Flight Table.

5. Change Status of Flights:

Query

```
update_flight = 'UPDATE Flight SET flight_status = %s WHERE flight_num = %s'

cursor.execute(update_flight, (flight_status, flight_select))
conn.commit()
```

Explanation

In the future flights table listed in the airline staff members personal homepage, there is a column that asks the airline staff member if they want to change the flight status. If they click "Change Flight Status", 'flight_select" (flight_num) and 'flight status' will be retrieved using the POST Method and will be used to update the 'flight_status' of 'flight_select'.

6. Add Airplane in the System:

Query

```
insert_airplane = 'INSERT INTO AIRPLANE (airline_name, seats) VALUES (%s, %s)'
cursor.execute(insert_airplane, (airline_name, seats))
conn.commit()
```

Explanation

When an airline staff member submits an "Add Airplane" form, the 'seats' are retrieved using the POST method and are used in the above query. Those variables are used AS VALUES to INSERT INTO the Airplane Table.

7. View Flight Ratings and Reviews:

Query

```
find_flight_review = 'SELECT * FROM REVIEW WHERE flight_num = %s'
cursor.execute(find_flight_review, (review_flight_select))
reviews = cursor.fetchall()

if (reviews == ()):
	reviews = "No Reviews"
		avg_rating = 0
	else:
		find_avg_rating = 'SELECT AVG(rating) as avg_rating FROM REVIEW WHERE
		flight_num = %s'
		cursor.execute(find_avg_rating, (review_flight_select))
		avg_rating = cursor.fetchone()
```

Explanation

When a customer searches for a flight and the flight data is displayed in a table, there is a column that asks if the user wants to "View Ratings and Reviews". If they click yes, 'review_flight_select' (flight_num) will be retrieved using the POST Method and will be used to SELECT all reviews from the Review Table that has the corresponding flight_num. Additionally, if there are results, the avg_rating of the reviews is calculated. If not, the avg_rating is 0.

8. View all the Booking Agents:

Query (Top 5 Booking Agents: Based on Ticket Sales Within the Last Month)

```
find_top_agents = 'SELECT Customer_Purchases.agent_ID, COUNT(purchase_ID) AS purchases FROM
Customer_Purchases, Booking_Agent WHERE Customer_Purchases.agent_ID = Booking_Agent.agent_ID
AND Booking_Agent.airline_name = %s AND purchase_date >= DATE_SUB(DATE(NOW()), INTERVAL 1
```

```
MONTH) GROUP BY agent_ID ORDER BY purchases DESC LIMIT 0, 5'

cursor.execute(find_top_agents, (airline_name))
top_agents_month = cursor.fetchall()
```

Explanation

This query COUNTS how many tickets that the booking agents have made on behalf of the customer using the Customer_Purchases Table within the INTERVAL of 6 months and SELECTS the 5 booking agents with the 5 highest ticket counts.

Query (Top 5 Booking Agents: Based on Ticket Sales Within the Last Year)

```
find_top_agents = 'SELECT Customer_Purchases.agent_ID, COUNT(purchase_ID) AS purchases FROM
Customer_Purchases, Booking_Agent WHERE Customer_Purchases.agent_ID = Booking_Agent.agent_ID
AND Booking_Agent.airline_name = %s AND purchase_date >= DATE_SUB(DATE(NOW()), INTERVAL 1
YEAR) GROUP BY agent_ID ORDER BY purchases DESC LIMIT 0, 5'

cursor.execute(find_top_agents, (airline_name))
top_agents_year = cursor.fetchall()
```

Explanation

This query COUNTS how many tickets that the booking agents have made on behalf of the customer using the Customer_Purchases Table within the INTERVAL of one year and SELECTS the 5 booking agents with the 5 highest ticket counts.

Query (Top 5 Booking Agents: Based on Commission Within the Last Year)

```
find_top_agents = 'SELECT Customer_Purchases.agent_ID, SUM(sold_price) * 0.1 AS commission
FROM Customer_Purchases, Booking_Agent WHERE Customer_Purchases.agent_ID =
Booking_Agent.agent_ID AND Booking_Agent.airline_name = %s AND purchase_date >=
DATE_SUB(DATE(NOW()), INTERVAL 1 YEAR) GROUP BY agent_ID ORDER BY commission DESC LIMIT 0,
5'

cursor.execute(find_top_agents, (airline_name))
top_agents_commission = cursor.fetchall()
```

Explanation

This query calculates the SUM of the commission that the booking agents have made by purchasing tickets on behalf of the customer using the Customer_Purchases Table within the INTERVAL of one year and SELECTS the 5 booking agents who have earned the booking agent the 5 highest commissions.

9. View Frequent Customers:

Query

```
find_customers = 'SELECT cus_email, COUNT(purchase_ID) AS purchases FROM Customer_Purchases,
Flight WHERE Customer_Purchases.flight_num = Flight.flight_num AND Flight.airline_name = %s
AND purchase_date >= DATE_SUB(DATE(NOW()), INTERVAL 1 YEAR) GROUP BY cus_email ORDER BY
purchases DESC LIMIT 0, 1'

cursor.execute(find_customers, (airline_name))
top_customer = cursor.fetchone()
```

Explanation

This query COUNTS all the tickets that have been purchased by each customer using the Customer_Purchases Table within the

INTERVAL of one year and SELECTS the customer that has bought the most tickets.

10. View Reports:

Query (Tickets Sold in the Past Month)

```
find_tickets_sold = 'SELECT COUNT(ticket_ID) AS tickets_sold from Customer_Purchases, Flight
WHERE Customer_Purchases.flight_num = Flight.flight_num AND Flight.airline_name = %s AND
purchase_date >= DATE_SUB(DATE(NOW()), INTERVAL 1 MONTH)'

cursor.execute(find_tickets_sold, (airline_name))
tickets_sold_month = cursor.fetchone()
```

Explanation

This query COUNTS the number of tickets told within the past month using the Customer_Purchases and Flight Table within the INTERVAL of one month.

Query (Tickets Sold Within the Past Year)

```
find_tickets_sold = 'SELECT COUNT(ticket_ID) AS tickets_sold from Customer_Purchases, Flight
WHERE Customer_Purchases.flight_num = Flight.flight_num AND Flight.airline_name = %s AND
purchase_date >= DATE_SUB(DATE(NOW()), INTERVAL 1 YEAR)'

cursor.execute(find_tickets_sold, (airline_name))
tickets_sold_year = cursor.fetchone()
```

Explanation

This query COUNTS the number of tickets told within the past month using the Customer_Purchases and Flight Table within the INTERVAL of one year.

Query (Tickets Sold Within the Past Year for Bar Chart)

```
find_tickets_sold = 'SELECT COUNT(ticket_ID) AS tickets_sold, MONTHNAME(purchase_date) AS
month FROM Customer_Purchases, Flight WHERE Customer_Purchases.flight_num =
Flight.flight_num AND Flight.airline_name = %s AND purchase_date >= DATE_SUB(DATE(NOW()),
INTERVAL 1 YEAR) GROUP BY month ORDER BY DATE(NOW())'

cursor.execute(find_tickets_sold, (airline_name))
tickets_sold_yearly_graph = cursor.fetchall()
```

Explanation

This query COUNTS the number of tickets told within the past month using the Customer_Purchases and Flight Table within the INTERVAL of one month. The month in which each of these purchases was made is also SELECTED so that the bar chart can show the tickets sold month wise.

11. Comparison of Revenue Earned:

Query

```
find_direct_revenue_month = 'SELECT SUM(sold_price) AS direct_revenue from
Customer_Purchases, Flight WHERE Customer_Purchases.flight_num = Flight.flight_num AND
Flight.airline_name = %s AND purchase_date >= DATE_SUB(DATE(NOW()), INTERVAL 1 MONTH) AND
agent_ID IS NULL'
```

```
cursor.execute(find_direct_revenue_month, (airline_name))
direct_revenue_month = cursor.fetchone()
```

Explanation

This query finds the SUM of the prices of tickets purchased by customers without a booking agent using the Customer_Purchases and Flight Table within the INTERVAL of one month using the Customer_Purchases and Flight Table.

Query

```
find_direct_revenue_year = 'SELECT SUM(sold_price) AS direct_revenue from
Customer_Purchases, Flight WHERE Customer_Purchases.flight_num = Flight.flight_num AND
Flight.airline_name = %s AND purchase_date >= DATE_SUB(DATE(NOW()), INTERVAL 1 YEAR) AND
agent_ID IS NULL '

cursor.execute(find_direct_revenue_year, (airline_name))
direct_revenue_year = cursor.fetchone()
```

Explanation

This query finds the SUM of the prices of tickets purchased by customers without a booking agent using the Customer_Purchases and Flight Table within the INTERVAL of one year using the Customer_Purchases and Flight Table.

Query

```
find_indirect_revenue_month = 'SELECT SUM(sold_price) AS indirect_revenue from
Customer_Purchases, Flight WHERE Customer_Purchases.flight_num = Flight.flight_num AND
Flight.airline_name = %s  AND purchase_date >= DATE_SUB(DATE(NOW()), INTERVAL 1 MONTH) AND
agent_ID IS NOT NULL'

cursor.execute(find_indirect_revenue_month, (airline_name))
indirect_revenue_month = cursor.fetchone()
```

Explanation

This query finds the SUM of the prices of tickets purchased by customers with a booking agent using the Customer_Purchases and Flight Table within the INTERVAL of one month using the Customer_Purchases and Flight Table.

Query

```
find_indirect_revenue_year = 'SELECT SUM(sold_price) AS indirect_revenue from
Customer_Purchases, Flight WHERE Customer_Purchases.flight_num = Flight.flight_num AND
Flight.airline_name = %s  AND purchase_date >= DATE_SUB(DATE(NOW()), INTERVAL 1 YEAR) AND
agent_ID IS NOT NULL'

cursor.execute(find_indirect_revenue_year, (airline_name))
indirect_revenue_year = cursor.fetchone()
```

Explanation

This query finds the SUM of the prices of tickets purchased by customers with a booking agent using the Customer_Purchases and Flight Table within the INTERVAL of one year using the Customer_Purchases and Flight Table.

12. View Top Destinations:

Query (Top 3 Destinations Within the Past Month)

```
find_top_destinations_month = 'SELECT Flight.arrival_airport, COUNT(Flight.arrival_airport)
AS tickets FROM Flight, Customer_Purchases WHERE Flight.flight_num =
Customer_Purchases.flight_num AND Flight.airline_name = %s AND purchase_date >=
DATE_SUB(DATE(NOW()), INTERVAL 3 MONTH)GROUP BY Flight.arrival_airport ORDER BY tickets DESC
LIMIT 0, 3'

cursor.execute(find_top_destinations_month, (airline_name))
top_destinations_month = cursor.fetchall()
```

Explanation

This query finds the top 3 destinations within the INTERVAL of 3 months (based on tickets already sold) using the Customer_Purchases and Flight Table.

Query (Top 3 Destinations Within the Past Year)

```
find_top_destinations_year = 'SELECT Flight.arrival_airport, COUNT(Flight.arrival_airport)
AS tickets FROM Flight, Customer_Purchases WHERE Flight.flight_num =
Customer_Purchases.flight_num AND Flight.airline_name = %s AND purchase_date >=
DATE_SUB(DATE(NOW()), INTERVAL 1 YEAR) GROUP BY Flight.arrival_airport ORDER BY tickets DESC
LIMIT 0, 3'

cursor.execute(find_top_destinations_year, (airline_name))
top_destinations_year = cursor.fetchall()
```

Explanation

This query finds the top 3 destinations within the INTERVAL of one year (based on tickets already sold) using the Customer_Purchases and Flight Table.

templates/staffConfirmation.html: The airline staff member is able to see all of the airplanes owned by the airline they work for after adding a new airplane to the system.

static/styles/styles.css: Contains css styling for all .html pages.