Functions & Queries in main.py

Alibi Nauanov (an3502) & Sayon Biswas (sb7239)

Yellow background is route/functions

Dark red is explanation

Create tables:

```
CREATE TABLE Airlines (
   airline_name VARCHAR(20),
   PRIMARY KEY(airline_name)
);

CREATE TABLE AirlineStaff (
   staff_username VARCHAR(20),
   staff_password VARCHAR(20),
   first_name VARCHAR(20),
   last_name VARCHAR(20),
   date_of_birth DATE,
   airline_name VARCHAR(20),
   FOREIGN KEY(airline_name) REFERENCES Airlines(airline_name) ON DELETE
CASCADE
);

CREATE TABLE Airplanes (
   airline_name VARCHAR(20),
   airplane_id CHAR(5),
   total_seats SMALLINT,
   PRIMARY KEY(airline_name, airplane_id),
   FOREIGN KEY(airline_name) REFERENCES Airlines(airline_name) ON DELETE
CASCADE
);

CREATE TABLE Airports (
   airport_name VARCHAR(20),
   city VARCHAR(20),
   PRIMARY KEY(airport_name)
);

CREATE INDEX AirplaneID_Index ON Airplanes(airplane_id);

CREATE TABLE Flights (
   airline_name VARCHAR(20),
   flight_number CHAR(5),
   airplane_id CHAR(5),
   departure_time DATETIME,
   departure_airport VARCHAR(20),
   arrival_time DATETIME,
   arrival_airport VARCHAR(20),
   ticket_price SMALLINT,
   flight_status VARCHAR(20),
   PRIMARY KEY(airline_name, flight_number),
   FOREIGN KEY(airplane_id) REFERENCES Airplanes(airplane_id) ON DELETE
CASCADE,
   FOREIGN KEY(departure_airport) REFERENCES Airports(airport_name) ON DELETE
CASCADE,
   FOREIGN KEY(arrival_airport) REFERENCES Airports(airport_name) ON DELETE
CASCADE
);

CREATE TABLE Customers (
   customer_email VARCHAR(320),
   customer_name VARCHAR(20),
   customer_password VARCHAR(20),
   building_name VARCHAR(20),
   street VARCHAR(20),
   city VARCHAR(20),
   state VARCHAR(20),
   phone_number CHAR(11),
   passport_number CHAR(9),
   passport_expiration DATE,
   passport_country VARCHAR(20),
   date_of_birth DATE,
   PRIMARY KEY(customer_email)
);

CREATE TABLE BookingAgents (
   agent_email VARCHAR(255),
   agent_password VARCHAR(20),
   booking_agent_id CHAR(5),
   PRIMARY KEY(agent_email)
);

CREATE TABLE Tickets (
   ticket_id CHAR(5),
   airline_name VARCHAR(20),
   flight_number CHAR(5),
   PRIMARY KEY(ticket_id),
   FOREIGN KEY(airline_name, flight_number) REFERENCES Flights(airline_name,
flight_number) ON DELETE CASCADE
```

```
);

CREATE INDEX BookingAgentID_Index ON BookingAgents(booking_agent_id);

CREATE TABLE Purchases (
   ticket_id CHAR(5),
   customer_email VARCHAR(320),
   booking_agent_email VARCHAR(320),
   purchase_date DATETIME,
   booking_agent_id CHAR(5),
   PRIMARY KEY(ticket_id, customer_email),
   FOREIGN KEY(ticket_id) REFERENCES Tickets(ticket_id) ON DELETE CASCADE,
   FOREIGN KEY(customer_email) REFERENCES Customers(customer_email) ON DELETE
CASCADE,
   FOREIGN KEY(booking_agent_email) REFERENCES BookingAgents(agent_email) ON
DELETE CASCADE,
   FOREIGN KEY(booking_agent_id) REFERENCES BookingAgents(booking_agent_id) ON
DELETE CASCADE
);
```

Public page:

```
@app.route('/')
def public_home()
```

Renders public_home.html

```
@app.route('/public_flight_search', methods=['GET', 'POST'])
def public_flight_search()
```

Renders search results after clicking the "Search Flights" button on the public home page

```
SELECT airline_name, flight_number,
      (SELECT city FROM Airports WHERE airport_name = departure_airport) AS
departure_city,
      departure_airport, departure_time,
      (SELECT city FROM Airports WHERE airport_name = arrival_airport) AS arrival_city,
      arrival_airport, arrival_time, ticket_price, airplane_id
FROM Flights
WHERE departure_airport = IF('{}' = '', departure_airport, '{}') AND
     arrival_airport = IF('{}' = '', arrival_airport, '{}') AND
     flight_status = 'upcoming' AND
     (SELECT city FROM Airports WHERE airport_name = departure_airport) = IF('{}' = '',
(SELECT city FROM Airports WHERE airport_name = departure_airport), '{}') AND
     (SELECT city FROM Airports WHERE airport_name = arrival_airport) = IF('{}' = '',
(SELECT city FROM Airports WHERE airport_name = arrival_airport), '{}') AND
```

```
    DATE(departure_time) = IF('{}' = '', DATE(departure_time), '{}') AND
    DATE(arrival_time) = IF('{}' = '', DATE(arrival_time), '{}')
ORDER BY airline_name, flight_number;
```
This query fetches upcoming flights matching the user's input criteria for departure and arrival locations, dates, and status

- Each user input box can be empty
- If all fields are empty, the query returns all upcoming flights

```
@app.route('/public_status_search', methods=['GET', 'POST'])
def public_status_search()
```
Renders search results after clicking the "Search Status" button on the public home page

```
SELECT *
FROM Flights
WHERE
        flight_number = IF('{}' = '', flight_number, '{}') AND
        DATE(departure_time) = IF('{}' = '', DATE(departure_time), '{}') AND
        DATE(arrival_time) = IF('{}' = '', DATE(arrival_time), '{}') AND
        airline_name = IF('{}' = '', airline_name, '{}')
ORDER BY airline_name, flight_number
```
This query fetches the flight status based on the user's input for airline name, flight number, and dates

- Each user input box can be empty
- If all fields are empty, the query returns all flight statuses

```
@app.route('/logout')
def logout()
```
Clears the session and renders the cuslogin.html page for customer login

---

Customer pages

```
@app.route('/customer_login')
def customer_login()
```
Renders the customer_login.html page for customers to log in to their accounts

```
@app.route('/customer_register')
def customer_register()
```
Renders the customer_register.html page for customers to create a new account

```
@app.route('/customer_login_auth', methods=['GET', 'POST'])
def customer_login_auth()
```
Authenticates a customer using their email and password

- If successful:

        - Retrieves upcoming flights for the customer
        - Redirects to the customer_home.html page
    - If authentication fails, it renders customer_login.html with an error message

SELECT * FROM Customers WHERE customer_email = '{}' AND customer_password = MD5('{}');
This query checks if a customer exists in the Customers table by matching the email and verifying the password using its MD5 hash

```
SELECT Tickets.ticket_id, Flights.airline_name, Flights.airplane_id, Flights.flight_number,
      D.city AS departure_city, Flights.departure_airport, A.city AS arrival_city,
      Flights.arrival_airport, Flights.departure_time, Flights.arrival_time, Flights.flight_status
FROM Purchases
JOIN Tickets ON Purchases.ticket_id = Tickets.ticket_id
JOIN Flights ON Tickets.airline_name = Flights.airline_name AND Tickets.flight_number =
Flights.flight_number
JOIN Airports AS D ON Flights.departure_airport = D.airport_name
JOIN Airports AS A ON Flights.arrival_airport = A.airport_name
WHERE Purchases.customer_email = '{}' AND Flights.flight_status = 'upcoming';
```
Retrieves the list of upcoming flights for the logged-in customer by joining Purchases, Tickets, Flights, and Airports

```
@app.route('/customer_register_auth', methods=['GET', 'POST'])
def customer_register_auth()
```
Registers a new customer with details like name, email, password, address, phone, and passport information

- Checks for an existing user
- If registration succeeds:
    - Inserts the new customer into the database
    - Retrieves the upcoming flights for the customer
    - Redirects to customer_home.html

SELECT * FROM Customers WHERE customer_email = '{}';
Checks if an account with the provided email already exists in the Customers table

```
INSERT INTO Customers (
   customer_email, customer_name, customer_password, building_name, street, city, state,
   phone_number, passport_number, passport_expiration, passport_country, date_of_birth
)
VALUES ('{}', '{}', MD5('{}'), '{}', '{}', '{}', '{}', '{}', '{}', '{}', '{}', '{}');
```
Inserts a new customer record with the provided details into the Customers table

```
@app.route('/customer_home')
def customer_home()
```

Displays the customer's home page, showing their upcoming flights

- Requires an active session.

```
@app.route('/customer_flight_search', methods=['GET', 'POST'])
def customer_flight_search()
```

Allows customers to search for flights based on departure city/airport, arrival city/airport, and dates

- Displays available flights and their ticket details.

```
SELECT Flights.airline_name, Flights.airplane_id, Flights.flight_number,
       D.city AS departure_city, Flights.departure_airport, A.city AS arrival_city,
       Flights.arrival_airport, Flights.departure_time, Flights.arrival_time,
       Flights.ticket_price, Flights.flight_status,
       (Airplanes.total_seats -
        (SELECT COUNT(*) FROM Tickets
         WHERE Tickets.flight_number = Flights.flight_number AND Tickets.airline_name =
Flights.airline_name)) AS num_tickets_left
FROM Flights
JOIN Airports AS D ON Flights.departure_airport = D.airport_name
JOIN Airports AS A ON Flights.arrival_airport = A.airport_name
JOIN Airplanes ON Flights.airline_name = Airplanes.airline_name AND Flights.airplane_id =
Airplanes.airplane_id
WHERE D.city = IF('{}' = '', D.city, '{}') AND Flights.departure_airport = IF('{}' = '',
Flights.departure_airport, '{}') AND
      A.city = IF('{}' = '', A.city, '{}') AND Flights.arrival_airport = IF('{}' = '',
Flights.arrival_airport, '{}') AND
      DATE(Flights.departure_time) = IF('{}' = '', DATE(Flights.departure_time), '{}') AND
      DATE(Flights.arrival_time) = IF('{}' = '', DATE(Flights.arrival_time), '{}')
ORDER BY Flights.airline_name, Flights.flight_number;
```

Searches for flights based on departure/arrival city or airport and dates. It also calculates available tickets by subtracting sold tickets from the airplane's total seats

```
@app.route('/customer_spending', methods=['POST', 'GET'])
def customer_spending()
```

Displays the total spending and monthly spending breakdown for a customer

```
SELECT SUM(f.ticket_price)
FROM Purchases p
JOIN Tickets t ON p.ticket_id = t.ticket_id
JOIN Flights f ON t.airline_name = f.airline_name AND t.flight_number = f.flight_number
WHERE p.customer_email = %s
AND p.purchase_date BETWEEN DATE_ADD(NOW(), INTERVAL -%s DAY) AND NOW();
```

Calculates the total spending of a customer within a specified duration by summing up ticket prices

SELECT YEAR(p.purchase_date) AS year, MONTH(p.purchase_date) AS month,
SUM(f.ticket_price) AS monthly_spending
FROM Purchases p
JOIN Tickets t ON p.ticket_id = t.ticket_id
JOIN Flights f ON t.airline_name = f.airline_name AND t.flight_number = f.flight_number
WHERE p.customer_email = %s AND p.purchase_date >= %s
GROUP BY YEAR(p.purchase_date), MONTH(p.purchase_date);
Calculates the monthly spending of a customer by grouping ticket purchases by year and month

```
@app.route('/customer_buy_tickets', methods=['GET', 'POST'])
def customer_buy_tickets()
```

Allows customers to purchase tickets for a specific flight

- Checks availability of tickets
- Inserts a new ticket into the database if available

```
SELECT *
FROM Flights
INNER JOIN Airplanes ON Flights.airline_name = Airplanes.airline_name
                AND Flights.airplane_id = Airplanes.airplane_id
WHERE Flights.airline_name = '{}'
AND Flights.flight_number = '{}'
AND (Airplanes.total_seats -
   (SELECT COUNT(*) FROM Tickets WHERE Tickets.flight_number = Flights.flight_number
AND Tickets.airline_name = Flights.airline_name)) > 0;
```

Checks if there are available tickets for a specific flight by comparing the airplane's total seats with the number of sold tickets

```
INSERT INTO Tickets (ticket_id, airline_name, flight_number) VALUES ('{}', '{}', '{}');
```

Adds a new ticket record to the Tickets table

```
INSERT INTO Purchases (ticket_id, customer_email, booking_agent_email, purchase_date)
VALUES ('{}', '{}', NULL, CURDATE());
```

Records the purchase of a ticket by the customer in the Purchases table

---

Booking Agent pages

```
@app.route('/agent_login')
def agent_login()
```

Renders the agent_login.html page for booking agents to log in

```
@app.route('/agent_register')
def agent_register()
```

Renders the agent_register.html page for booking agents to create a new account

```
@app.route('/agent_login_auth', methods=['GET', 'POST'])
def agent_login_auth()
```
Authenticates a booking agent using their email and password

- On success, retrieves the agent's ID and associated flight bookings, then redirects to agent_home.html
- On failure, displays an error message

SELECT * FROM BookingAgents WHERE agent_email = '{}' AND agent_password = MD5('{}');
Checks if the agent exists in the BookingAgents table with the provided email and hashed password

```
@app.route('/agent_register_auth', methods=['GET', 'POST'])
def agent_register_auth()
```
Handles booking agent registration by validating the details and inserting a new record into the database

- On success, redirects to agent_home.html
- On failure, displays an error message

INSERT INTO BookingAgents (agent_email, agent_password, booking_agent_id) VALUES ('{}', MD5('{}'), '{}');
Inserts a new booking agent into the BookingAgents table with hashed password

```
@app.route('/agent_home')
def agent_home()
```
Displays the booking agent's home page, showing their booked flights and agent ID

SELECT Tickets.ticket_id, Purchases.customer_email, Purchases.purchase_date, Flights.airline_name, Flights.flight_number, D.city, Flights.departure_airport, Flights.departure_time, A.city, Flights.arrival_airport, Flights.arrival_time, Flights.ticket_price
FROM Purchases
JOIN Tickets ON Purchases.ticket_id = Tickets.ticket_id
JOIN Flights ON Tickets.airline_name = Flights.airline_name AND Tickets.flight_number = Flights.flight_number
JOIN Airports AS D ON Flights.departure_airport = D.airport_name
JOIN Airports AS A ON Flights.arrival_airport = A.airport_name
WHERE Purchases.booking_agent_email = '{}';
Retrieves details of all flights booked through the agent

```
@app.route('/agent_purchase_search')
def agent_purchase_search()
```

Renders the agent_purchase_search.html page for agents to search for flights and manage purchases

```
@app.route('/agent_commission', methods=['POST', 'GET'])
def agent_commission()
```

Calculates and displays the agent's total commission, average commission, and ticket sales count over a specified time frame

SELECT SUM(Flights.ticket_price * 0.1) AS total_commission, AVG(Flights.ticket_price * 0.1) AS average_commission, COUNT(Tickets.ticket_id) AS ticket_count
FROM Purchases
JOIN Tickets ON Purchases.ticket_id = Tickets.ticket_id
JOIN Flights ON Tickets.airline_name = Flights.airline_name AND Tickets.flight_number = Flights.flight_number
WHERE Purchases.booking_agent_email = '{}' AND (Purchases.purchase_date BETWEEN DATE_ADD(NOW(), INTERVAL -{} DAY) AND NOW());
Calculates the total and average commission earned by the agent and the total tickets sold

```
@app.route('/agent_customers_ranking')
def agent_customers_ranking()
```

Displays the top customers for a booking agent, ranked by tickets sold and commission earned

SELECT Purchases.customer_email, COUNT(Tickets.ticket_id) AS ticket_count
FROM Purchases
JOIN Tickets ON Purchases.ticket_id = Tickets.ticket_id
WHERE Purchases.booking_agent_email = '{}' AND DATEDIFF(CURDATE(), DATE(Purchases.purchase_date)) < 183
GROUP BY Purchases.customer_email
ORDER BY ticket_count DESC;
Ranks customers by the number of tickets purchased

SELECT Purchases.customer_email, SUM(Flights.ticket_price) * 0.1 AS total_commission
FROM Purchases
JOIN Tickets ON Purchases.ticket_id = Tickets.ticket_id
JOIN Flights ON Tickets.airline_name = Flights.airline_name AND Tickets.flight_number = Flights.flight_number
WHERE Purchases.booking_agent_email = '{}' AND DATEDIFF(CURDATE(), DATE(Purchases.purchase_date)) < 365
GROUP BY Purchases.customer_email
ORDER BY total_commission DESC;
Ranks customers by the total commission earned

```
@app.route('/agent_search_flight', methods=['GET', 'POST'])
def agent_search_flight()
```

Allows agents to search for flights based on various criteria like departure/arrival city or airport and dates

```
SELECT Flights.airplane_id, Flights.flight_number, D.city AS departure_city,
Flights.departure_airport, A.city AS arrival_city, Flights.arrival_airport, Flights.departure_time,
Flights.arrival_time, Flights.flight_status, Flights.ticket_price, Flights.airline_name,
    (Airplanes.total_seats - (SELECT COUNT(*) FROM Tickets WHERE
Tickets.flight_number = Flights.flight_number AND Tickets.airline_name =
Flights.airline_name)) AS num_tickets_left
FROM Flights
JOIN Airports AS D ON Flights.departure_airport = D.airport_name
JOIN Airports AS A ON Flights.arrival_airport = A.airport_name
JOIN Airplanes ON Flights.airline_name = Airplanes.airline_name AND Flights.airplane_id =
Airplanes.airplane_id
WHERE D.city = IF('{}' = '', D.city, '{}') AND Flights.departure_airport = IF('{}' = '',
Flights.departure_airport, '{}') AND
   A.city = IF('{}' = '', A.city, '{}') AND Flights.arrival_airport = IF('{}' = '',
Flights.arrival_airport, '{}') AND
   DATE(Flights.departure_time) = IF('{}' = '', DATE(Flights.departure_time), '{}') AND
   DATE(Flights.arrival_time) = IF('{}' = '', DATE(Flights.arrival_time), '{}')
ORDER BY Flights.airline_name, Flights.flight_number;
```
Searches for flights and calculates available tickets

```
@app.route('/agent_buy_tickets', methods=['GET', 'POST'])
def agent_buy_tickets()
```
Enables booking agents to buy tickets on behalf of customers after validating credentials and ticket availability

```
SELECT Flights.*,
    (Airplanes.total_seats - (SELECT COUNT(*) FROM Tickets WHERE
Tickets.flight_number = Flights.flight_number AND Tickets.airline_name =
Flights.airline_name)) AS available_seats
FROM Flights
JOIN Airplanes ON Flights.airline_name = Airplanes.airline_name AND Flights.airplane_id =
Airplanes.airplane_id
WHERE Flights.airline_name = '{}' AND Flights.flight_number = '{}'
HAVING available_seats > 0;
```
Checks if tickets are available for a flight

```
INSERT INTO Tickets (ticket_id, airline_name, flight_number) VALUES ('{}', '{}', '{}');
```
Inserts a new ticket into the Tickets table

```
INSERT INTO Purchases (ticket_id, customer_email, booking_agent_email, purchase_date)
VALUES ('{}', '{}', '{}', CURDATE());
```

Records the purchase of the ticket in the Purchases table

Airline Staff pages

```
@app.route('/staff_login')
def staff_login()
```
Renders the staff_login.html page for airline staff to log in to their accounts

```
@app.route('/staff_register')
def staff_register()
```
Renders the staff_register.html page for airline staff to create a new account

```
@app.route('/staff_login_auth', methods=['GET', 'POST'])
def staff_login_auth()
```
Authenticates an airline staff member using their username and password

- On success: Fetches and displays upcoming flights managed by the logged-in staff
- On failure: Displays an error message

SELECT *
FROM AirlineStaff
WHERE staff_username = '{}' AND staff_password = MD5('{}');
Checks if the username and password match an existing airline staff record

SELECT AirlineStaff.staff_username, Flights.airline_name, Flights.airplane_id, Flights.flight_number,
    Flights.departure_airport, Flights.arrival_airport, Flights.departure_time, Flights.arrival_time
FROM Flights
JOIN AirlineStaff ON Flights.airline_name = AirlineStaff.airline_name
WHERE AirlineStaff.staff_username = '{}' AND Flights.flight_status = 'upcoming' AND DATEDIFF(CURDATE(), DATE(Flights.departure_time)) < 30;
Fetches upcoming flights managed by the logged-in staff within the last 30 days

```
@app.route('/staff_register_auth', methods=['GET', 'POST'])
def staff_register_auth()
```
Handles registration for new airline staff members

- Verifies the airline exists and the username is unique
- On success: Inserts the staff member into the database and redirects to their home page
- On failure: Displays an error message

INSERT INTO AirlineStaff (staff_username, staff_password, first_name, last_name, date_of_birth, airline_name)
VALUES ('{}', '{}', '{}', '{}', '{}', '{}');
Inserts a new staff member record into the AirlineStaff table

```
@app.route('/staff_home')
def staff_home()
```

Displays the home page for logged-in airline staff, showing upcoming flights associated with their airline

SELECT AirlineStaff.staff_username, Flights.airline_name, Flights.airplane_id, Flights.flight_number,
    Flights.departure_airport, Flights.arrival_airport, Flights.departure_time, Flights.arrival_time
FROM Flights
JOIN AirlineStaff ON Flights.airline_name = AirlineStaff.airline_name
WHERE AirlineStaff.staff_username = '{}' AND Flights.flight_status = 'upcoming' AND DATEDIFF(CURDATE(), DATE(Flights.departure_time)) < 30;
Fetches upcoming flights for the airline staff member

```
@app.route('/staff_search_flight', methods=['GET', 'POST'])
def staff_search_flight()
```

Allows staff to search for flights based on departure/arrival details and dates. Displays matching flights or an error if no matches are found

SELECT Flights.airline_name, Flights.airplane_id, Flights.flight_number, D.city AS departure_city, Flights.departure_airport,
    A.city AS arrival_city, Flights.arrival_airport, Flights.departure_time, Flights.arrival_time, Flights.flight_status, Flights.ticket_price
FROM Flights
JOIN Airports AS D ON Flights.departure_airport = D.airport_name
JOIN Airports AS A ON Flights.arrival_airport = A.airport_name
JOIN AirlineStaff ON Flights.airline_name = AirlineStaff.airline_name
WHERE D.city = IF('{}' = '', D.city, '{}') AND Flights.departure_airport = IF('{}' = '', Flights.departure_airport, '{}') AND
    A.city = IF('{}' = '', A.city, '{}') AND Flights.arrival_airport = IF('{}' = '', Flights.arrival_airport, '{}') AND
    DATE(Flights.departure_time) = IF('{}' = '', DATE(Flights.departure_time), '{}') AND
    DATE(Flights.arrival_time) = IF('{}' = '', DATE(Flights.arrival_time), '{}') AND AirlineStaff.staff_username = '{}'
ORDER BY Flights.airline_name, Flights.flight_number;
Searches for flights that match the specified criteria

```
@app.route('/staff_flight')
def staff_flight()
```

Renders the staff_flight_info.html page to allow the airline staff to view and manage flights associated with their airline

SELECT staff_username, airline_name
FROM AirlineStaff
WHERE staff_username = '{}';
Allows airline staff to view flight details and perform related management tasks while ensuring the displayed data is tied to their airline

```
@app.route('/staff_insert_data')
def staff_insert_data()
```
Renders a form for staff to add flight or airplane information

SELECT Airplanes.airplane_id, Airplanes.total_seats
FROM Airplanes
JOIN AirlineStaff ON Airplanes.airline_name = AirlineStaff.airline_name
WHERE AirlineStaff.staff_username = '{}';
Fetches all airplanes managed by the logged-in staff's airline

```
@app.route('/change_status_flight', methods=['GET', 'POST'])
def change_status_flight()
```
Allows airline staff to update the status of a specific flight

UPDATE Flights
SET flight_status = '{}'
WHERE flight_number = '{}';
Updates the status of a specific flight identified by its flight_number in the Flights table

SELECT staff_username, airline_name
FROM AirlineStaff
WHERE staff_username = '{}';
Retrieves the staff_username and their associated airline_name from the AirlineStaff table to confirm the user's role and airline

```
@app.route('/add_flight', methods=['GET', 'POST'])
def add_flight()
```
Allows staff to add a new flight. Validates departure/arrival airports, airplane availability, and seat limits before inserting the flight

INSERT INTO Flights (airline_name, flight_number, departure_airport, departure_time, arrival_airport, arrival_time, ticket_price, flight_status, airplane_id)
VALUES ('{}', '{}', '{}', '{} {}', '{}', '{} {}', '{}', '{}', '{}');
Inserts a new flight record into the Flights table

```
@app.route('/insert_airplane', methods=['GET', 'POST'])
def insert_airplane()
```
Allows staff to add a new airplane to their airline's fleet. Ensures the airplane ID is unique within the airline

INSERT INTO Airplanes (airline_name, airplane_id, total_seats)
VALUES ('{}', '{}', '{}');
Adds a new airplane record to the Airplanes table

```
@app.route('/insert_airport', methods=['GET', 'POST'])
def insert_airport()
```
Allows staff to add a new airport. Verifies the airport name is unique before adding

INSERT INTO Airports (airport_name, city)
VALUES ('{}', '{}');
Adds a new airport record to the Airports table

```
@app.route('/staff_agent')
def staff_agent()
```
Displays booking agents ranked by commission earned and tickets sold for the logged-in staff's airline. Includes both monthly and yearly rankings

SELECT BookingAgents.agent_email, BookingAgents.booking_agent_id,
SUM(Flights.ticket_price) * 0.1 AS commission
FROM BookingAgents
NATURAL JOIN Purchases
NATURAL JOIN Tickets AS T
JOIN Flights ON T.airline_name = Flights.airline_name AND T.flight_number =
Flights.flight_number
JOIN AirlineStaff ON AirlineStaff.airline_name = Flights.airline_name
WHERE AirlineStaff.staff_username = '{}' AND DATEDIFF(CURDATE(),
DATE(Purchases.purchase_date)) < 365
GROUP BY BookingAgents.agent_email, BookingAgents.booking_agent_id
ORDER BY commission DESC
LIMIT 5;
Ranks agents by commission earned in the last year

```
@app.route('/staff_customer')
def staff_customer()
```
Displays the airline's most frequent customer based on ticket purchases

SELECT Customers.customer_email, Customers.customer_name, COUNT(Tickets.ticket_id) AS
ticket_count

```
FROM Customers
JOIN Purchases ON Customers.customer_email = Purchases.customer_email
NATURAL JOIN Tickets
NATURAL JOIN Flights
JOIN AirlineStaff ON AirlineStaff.airline_name = Flights.airline_name
WHERE AirlineStaff.staff_username = '{}' AND DATEDIFF(CURDATE(),
DATE(Purchases.purchase_date)) < 365
GROUP BY Customers.customer_email, Customers.customer_name
ORDER BY ticket_count DESC
LIMIT 1;
```
Fetches the airline's most frequent customer within the past year

```
@app.route('/staff_customer_flight', methods=['GET', 'POST'])
def staff_customer_flight()
```
Enables airline staff to search for flights taken by a specific customer

```
SELECT staff_username, airline_name
FROM AirlineStaff
WHERE staff_username = '{}';
```
Retrieves the logged-in staff member's username and associated airline

```
SELECT DISTINCT Flights.airplane_id, Flights.flight_number, Flights.departure_airport,
        Flights.arrival_airport, Flights.departure_time, Flights.arrival_time,
        Flights.flight_status
FROM Customers
JOIN Purchases ON Customers.customer_email = Purchases.customer_email
NATURAL JOIN Tickets
NATURAL JOIN Flights
JOIN AirlineStaff ON AirlineStaff.airline_name = Flights.airline_name
WHERE Customers.customer_email = '{}'
  AND AirlineStaff.staff_username = '{}';
```
Fetches flights taken by a customer associated with the airline

```
SELECT customer_email
FROM Customers
WHERE customer_email = '{}';
```
Checks if a customer with the given email exists

```
SELECT DISTINCT Customers.customer_email, Customers.customer_name
FROM Customers
JOIN Purchases ON Customers.customer_email = Purchases.customer_email
NATURAL JOIN Tickets
NATURAL JOIN Flights
JOIN AirlineStaff ON AirlineStaff.airline_name = Flights.airline_name
```

```
WHERE Flights.flight_number = '{}'
  AND AirlineStaff.staff_username = '{}';
```
Retrieves all customers who booked a specific flight managed by the airline

```
SELECT Customers.customer_email, Customers.customer_name, COUNT(Tickets.ticket_id) AS
ticket_count
FROM Customers
JOIN Purchases ON Customers.customer_email = Purchases.customer_email
NATURAL JOIN Tickets
NATURAL JOIN Flights
JOIN AirlineStaff ON AirlineStaff.airline_name = Flights.airline_name
WHERE AirlineStaff.staff_username = '{}'
  AND DATEDIFF(CURDATE(), DATE(Purchases.purchase_date)) < 365
GROUP BY Customers.customer_email, Customers.customer_name
ORDER BY ticket_count DESC
LIMIT 1;
```
Identifies the most frequent customer by counting tickets purchased over the past year

```
@app.route('/staff_flight_customer', methods=['GET', 'POST'])
def staff_flight_customer()
```
Allows airline staff to retrieve customers for a specific flight and display the most frequent customer over the past year

```
SELECT staff_username, airline_name FROM AirlineStaff WHERE staff_username = '{}'
```
Gets the logged-in staff's username and airline

```
SELECT DISTINCT
            Customers.customer_email,
            Customers.customer_name
        FROM
            Customers
        JOIN Purchases ON Customers.customer_email = Purchases.customer_email
        NATURAL JOIN Tickets
        NATURAL JOIN Flights
        JOIN AirlineStaff ON AirlineStaff.airline_name = Flights.airline_name
        WHERE
            Flights.flight_number = '{}'
            AND AirlineStaff.staff_username = '{}'
```
Lists customers who took the specified flight

```
SELECT
            Customers.customer_email,
            Customers.customer_name,
            COUNT(Tickets.ticket_id) AS ticket_count
```

```
          FROM
             Customers
          JOIN Purchases ON Customers.customer_email = Purchases.customer_email
          NATURAL JOIN Tickets
          NATURAL JOIN Flights
          JOIN AirlineStaff ON AirlineStaff.airline_name = Flights.airline_name
          WHERE
             AirlineStaff.staff_username = '{}'
             AND DATEDIFF(CURDATE(), DATE(Purchases.purchase_date)) < 365
          GROUP BY
             Customers.customer_email, Customers.customer_name
          ORDER BY
             ticket_count DESC
          LIMIT 1
```

Identifies the airline's top customer based on ticket purchases

```
SELECT flight_number
             FROM Flights
             NATURAL JOIN AirlineStaff
             WHERE flight_number = '{}'
               AND AirlineStaff.staff_username = '{}'
```

Checks if the flight exists to handle errors

```
@app.route('/staff_destination')
def staff_destination()
```

Displays top destination cities for the airline based on ticket sales over the past 3 months and the past year

```
SELECT Airports.city AS airport_city, COUNT(Tickets.ticket_id) AS ticket_count
FROM Purchases
NATURAL JOIN Tickets
NATURAL JOIN Flights
JOIN Airports ON Airports.airport_name = Flights.arrival_airport
WHERE DATEDIFF(CURDATE(), DATE(Purchases.purchase_date)) < 90
GROUP BY Airports.city
ORDER BY ticket_count DESC
LIMIT 3;
```

Identifies the top 3 destinations for the airline in the last 3 months

```
@app.route('/staff_revenue')
def staff_revenue()
```

Displays the airline's revenue breakdown by direct and indirect bookings for both monthly and yearly timeframes

```
SELECT SUM(Flights.ticket_price)
FROM Purchases
```

NATURAL JOIN Tickets
NATURAL JOIN Flights
JOIN AirlineStaff ON AirlineStaff.airline_name = Flights.airline_name
WHERE AirlineStaff.staff_username = '{}' AND Purchases.booking_agent_id IS NULL AND DATEDIFF(CURDATE(), DATE(Purchases.purchase_date)) < 30;
Calculates revenue from direct bookings in the past month

```
@app.route('/staff_tickets')
def staff_tickets()
```
Displays the airline staff's ticket management page, requiring the user to be logged in

```
SELECT staff_username, airline_name
        FROM AirlineStaff
        WHERE staff_username = '{}'
```
Fetches the logged-in staff member's username and associated airline details

```
@app.route('/staff_fix_ticket', methods=['GET', 'POST'])
def staff_fix_ticket()
```
Displays ticket sales trends for the past month or year for the logged-in airline staff, grouped by month

```
SELECT staff_username, airline_name
        FROM AirlineStaff
        WHERE staff_username = '{}'
```
Fetches the airline name and username associated with the logged-in staff

```
SELECT
            YEAR(Purchases.purchase_date) AS year,
            MONTH(Purchases.purchase_date) AS month,
            COUNT(Tickets.ticket_id) AS ticket_count
        FROM
            Purchases
        NATURAL JOIN Tickets
        NATURAL JOIN Flights
        JOIN AirlineStaff ON AirlineStaff.airline_name = Flights.airline_name
        WHERE
            DATEDIFF(CURDATE(), DATE(Purchases.purchase_date)) < 30
            AND AirlineStaff.staff_username = '{}'
        GROUP BY
            year, month
        ORDER BY
            year, month
```
Retrieves the count of tickets sold within the past 30 days, grouped by year and month

```
SELECT
        YEAR(Purchases.purchase_date) AS year,
        MONTH(Purchases.purchase_date) AS month,
        COUNT(Tickets.ticket_id) AS ticket_count
      FROM
        Purchases
      NATURAL JOIN Tickets
      NATURAL JOIN Flights
      JOIN AirlineStaff ON AirlineStaff.airline_name = Flights.airline_name
      WHERE
        DATEDIFF(CURDATE(), DATE(Purchases.purchase_date)) < 365
        AND AirlineStaff.staff_username = '{}'
      GROUP BY
        year, month
      ORDER BY
        year, month
```

Retrieves the count of tickets sold within the past 365 days, grouped by year and month

```
@app.route('/staff_ticket', methods=['GET', 'POST'])
def staff_ticket()
```

Fetches and displays ticket sales for a custom date range, summarizing data by month

```
SELECT YEAR(Purchases.purchase_date) AS year, MONTH(Purchases.purchase_date) AS
month, COUNT(Tickets.ticket_id) AS ticket_count
FROM Purchases
NATURAL JOIN Tickets
NATURAL JOIN Flights
JOIN AirlineStaff ON AirlineStaff.airline_name = Flights.airline_name
WHERE Purchases.purchase_date > '{}' AND Purchases.purchase_date < '{}' AND
AirlineStaff.staff_username = '{}'
GROUP BY year, month
ORDER BY year, month;
```

Summarizes tickets sold by month for a specified date range