

Vinson Lin

Database

Project 3: Airline

Directory:

- resources/ticketgenerating.py: created at the beginning to insert tickets into the database for every flight
- static/css/styles.css: helps create a clean looking user interface
- venv/: set up virtual environment to avoid dependencies issues
- templates/
 - add_airplane: staff (admin): add new airplane
 - add_airport: staff (admin): add new airport
 - add_booking_agent: staff (admin): add new booking agent to the system
 - add_flight: staff (admin) create flights (draft)
 - agent_search_results: booking agent: search for ticket to buy
 - airline_staff_dashboard: booking agent dashboard
 - airplane_list: staff (admin) list all airlines in the admin's airline company
 - airport_list: staff (admin) list all airports
 - base: base template all other templates inherit from (e.g. navbar)
 - booking_agent_dashboard: booking agent dashboard
 - change_flight_status: staff (operator) change status of existing flight
 - create_flight: staff (admin) create flights
 - customer_dashboard: customer dashboard
 - flight_details: view flight detail (test)
 - grant_permissions: staff (admin) grant other staff permission
 - home: guest interface
 - login: login page
 - profile: customer profile page
 - search_results: search for public, customers, and agents
 - signup: signup page

- view_booking_agents: staff view top booking agents based on sales
- view_frequent_customers: staff view top customers based on num of purchases
- view_reports: staff view report on ticket sales
- view_revenue_comparison: staff view indirect vs direct revenue
- view_top_destinations: staff view top destinations
- app.py: main flask application file where all the routes are located
- config.py: for security and easier maintenance
- requirements.txt: used with venv to download all dependencies

Use cases and queries:

General non-user specific functions

Function to get database connection

def get_db_connection():

Decorator to ensure the user is logged in by checking if the 'user_email' is present in the session. Wraps around other functions to ensure they cannot be accessed without a valid session.

def login_required(f):

Homepage route logic

def home():

Get all flight records from the database to display on the homepage as recommendation

cursor.execute("SELECT * FROM flight")

Get distinct departure airports from the flight table to populate the dropdown menu.

cursor.execute("SELECT DISTINCT departure_airport FROM flight")

Get distinct arrival airports from the flight table to populate the dropdown menu.

cursor.execute("SELECT DISTINCT arrival_airport FROM flight")

```
# Test route for database connection check
```

```
def test():
```

```
    # Get all the table names in the database to test if the database connection works.
```

```
    cursor.execute("SHOW TABLES;")
```

```
# Route for displaying flight details
```

```
def flight_details(flight_num):
```

```
    # Get the flight details based on the flight number passed as a parameter.
```

```
    cursor.execute("SELECT * FROM flight WHERE flight_num = %s", (flight_num,))
```

```
# Search Flights route for customer and booking agent logic
```

```
def search_flights():
```

```
    # Check if the booking agent is associated with the user by looking up the airline they work for.
```

```
    cursor.execute("SELECT airline_name FROM booking_agent_work_for WHERE email = %s", (user_email,))
```

```
# Get flights for a booking agent, ensuring the airline matches.
```

```
query = """
```

```
    SELECT * FROM flight
```

```
    WHERE departure_airport = %s
```

```
    AND arrival_airport = %s
```

```
    AND DATE(departure_time) = %s
```

```
    AND airline_name = %s
```

```
"""
```

```
cursor.execute(query, (source, destination, date, airline_name))
```

```
# Get all flights for customers filtered.
```

```
query = """
```

```
    SELECT * FROM flight
```

```
    WHERE departure_airport = %s
```

```

        AND arrival_airport = %s
        AND DATE(departure_time) = %s
    """
    cursor.execute(query, (source, destination, date))

# Signup route logic
def signup():
    # Check if the provided email already exists for customers, booking agents, or airline staff.
    cursor.execute("SELECT * FROM customer WHERE email = %s", (email,))

    # Check if the provided email already exists in the booking agent table.
    cursor.execute("SELECT * FROM booking_agent WHERE email = %s", (email,))

    # Check if the provided username already exists for airline staff.
    cursor.execute("SELECT * FROM airline_staff WHERE username = %s", (email,))
    user = cursor.fetchone()

    # Check if the provided airline name exists in the airline table.
    cursor.execute("SELECT * FROM airline WHERE airline_name = %s", (airline_name,))
    airline = cursor.fetchone()

    # If role is customer, save the data in the customer table
    INSERT INTO customer (email, name, password, building_number, street, city, state,
                          phone_number, passport_number, passport_expiration, passport_country,
                          date_of_birth)
    """VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)
    """, (email, name, password, building_number, street, city, state, phone_number,
    passport_number, passport_expiration, passport_country, date_of_birth)

    # If role is booking agent, save the data in the booking agent table

```

```
"INSERT INTO booking_agent (email, password, booking_agent_id) VALUES (%s, %s, %s)", (email, password, booking_agent_id)
```

```
# If role is staff, save the data in the staff table
```

```
""""INSERT INTO airline_staff (username, password, first_name, last_name, airline_name, date_of_birth) VALUES (%s, %s, %s, %s, %s, %s)""", (email, password, first_name, last_name, airline_name, date_of_birth)
```

```
# Login route logic
```

```
def login():
```

```
# Check if the provided email exists for a customer.
```

```
cursor.execute("SELECT * FROM customer WHERE email = %s", (email,))
```

```
# Check if the provided email exists for a booking agent.
```

```
cursor.execute("SELECT * FROM booking_agent WHERE email = %s", (email,))
```

```
# Check if the provided username exists for airline staff.
```

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

```
# Logout by clearing session
```

```
def logout()
```

Customer

```
# Customer Dashboard route logic
```

```
def customer_dashboard():
```

```
# Get distinct departure airports for dropdown in customer dashboard
```

```
cursor.execute("SELECT DISTINCT departure_airport FROM flight")
```

```
# Get distinct arrival airports for dropdown in customer dashboard
```

```
cursor.execute("SELECT DISTINCT arrival_airport FROM flight")
```

```
# Get customer information from the customer table using email
```

```
cursor.execute("SELECT * FROM customer WHERE email = %s", (user_email,))
```

```
# Get total spending data for the last 6 months
```

```
cursor.execute("""
```

```
    SELECT DATE_FORMAT(p.purchase_date, '%Y-%m') AS month,
```

```
           SUM(f.price) AS total_spent
```

```
    FROM purchases p
```

```
    JOIN ticket t ON p.ticket_id = t.ticket_id
```

```
    JOIN flight f ON t.flight_num = f.flight_num
```

```
    WHERE p.customer_email = %s AND p.purchase_date >= CURDATE() - INTERVAL 6
```

```
MONTH
```

```
    GROUP BY month
```

```
    ORDER BY month;
```

```
""", (user_email,))
```

```
# Get total spending data for the last 1 year
```

```
cursor.execute("""
```

```
    SELECT SUM(f.price) AS total_spent
```

```
    FROM purchases p
```

```
    JOIN ticket t ON p.ticket_id = t.ticket_id
```

```
    JOIN flight f ON t.flight_num = f.flight_num
```

```
    WHERE p.customer_email = %s AND p.purchase_date >= CURDATE() - INTERVAL 1
```

```
YEAR;
```

```
""", (user_email,))
```

```
# Get upcoming flights for the customer
```

```
cursor.execute("""
```

```
    SELECT f.airline_name, f.flight_num, f.departure_time, f.arrival_time, f.departure_airport,
```

```

        f.arrival_airport, f.status, p.purchase_date, f.price
    FROM flight f
    JOIN ticket t ON f.flight_num = t.flight_num AND f.airline_name = t.airline_name
    JOIN purchases p ON t.ticket_id = p.ticket_id
    WHERE p.customer_email = %s AND f.status = 'upcoming'
    """ , (user_email,))

```

Get booking history for the customer

```

cursor.execute("""
    SELECT f.airline_name, f.flight_num, f.departure_time, f.arrival_time, f.departure_airport,
           f.arrival_airport, f.status, p.purchase_date, f.price
    FROM flight f
    JOIN ticket t ON f.flight_num = t.flight_num AND f.airline_name = t.airline_name
    JOIN purchases p ON t.ticket_id = p.ticket_id
    WHERE p.customer_email = %s AND f.status != 'upcoming'
    """ , (user_email,))

```

Customer Profile route logic

def profile():

Get customer profile details based on email

```

query = """
    SELECT name, email, date_of_birth, passport_number,
           passport_expiration, phone_number,
           CONCAT(building_number, ' ', street, ' ', city, ' ', state) AS address,
           passport_country
    FROM Customer
    WHERE email = %s
    """

```

```

cursor.execute(query, (user_email,))

```

Customer Purchase Ticket route logic

```

def purchase_ticket():
    # Find an available ticket for the selected flight
    query = """
        SELECT t.ticket_id
        FROM ticket t
        LEFT JOIN purchases p ON t.ticket_id = p.ticket_id
        WHERE t.flight_num = %s AND p.ticket_id IS NULL
        LIMIT 1;
    """
    cursor.execute(query, (flight_num,))

    # Insert the purchase record for the customer
    purchase_query = """
        INSERT INTO purchases (ticket_id, customer_email, purchase_date)
        VALUES (%s, %s, CURDATE());
    """
    cursor.execute(purchase_query, (ticket_id, user_email))

# Customer Track Spending route logic
def track_spending():
    # Get total spending over the last year by customer
    query = """
        SELECT DATE_FORMAT(purchase_date, '%Y-%m') AS month, SUM(price) AS total
        FROM purchases
        JOIN ticket ON purchases.ticket_id = ticket.ticket_id
        WHERE purchases.customer_email = %s AND purchase_date >= DATE_SUB(NOW(),
INTERVAL 1 YEAR)
        GROUP BY month
        ORDER BY month
    """
    cursor.execute(query, (user_email,))

```

Booking Agent

Booking Agent Dashboard route logic

def booking_agent_dashboard():

Get the booking agent ID based on the logged-in agent's email

cursor.execute("SELECT booking_agent_id FROM booking_agent WHERE email = %s",
(user_email,))

Get the distinct departure airports for the booking agent's dashboard

cursor.execute("SELECT DISTINCT departure_airport FROM flight")

Get the distinct arrival airports for the booking agent's dashboard

cursor.execute("SELECT DISTINCT arrival_airport FROM flight")

Get the airline associated with the booking agent from booking_agent_work_for

cursor.execute("""

SELECT airline_name FROM booking_agent_work_for

WHERE email = %s

""", (user_email,))

Get upcoming flights booked for customers by this booking agent's airline

cursor.execute("""SELECT f.airline_name, f.flight_num, f.departure_time, f.arrival_time,
 f.departure_airport, f.arrival_airport, f.price, f.status, p.purchase_date, c.email AS
customer_email

FROM flight f

JOIN ticket t ON f.flight_num = t.flight_num AND f.airline_name = t.airline_name

JOIN purchases p ON t.ticket_id = p.ticket_id

JOIN customer c ON p.customer_email = c.email

JOIN booking_agent b ON p.booking_agent_id = b.booking_agent_id

```
WHERE f.status = 'upcoming' AND f.airline_name = %s AND p.booking_agent_id =  
b.booking_agent_id;  
""", (airline_name,))
```

Query to calculate commission, number of tickets sold, and average commission per ticket

```
cursor.execute("""  
SELECT  
    SUM(f.price * 0.05) AS total_commission,  
    COUNT(p.ticket_id) AS total_tickets_sold,  
    AVG(f.price * 0.05) AS avg_commission_per_ticket  
FROM purchases p  
JOIN ticket t ON p.ticket_id = t.ticket_id  
JOIN flight f ON t.flight_num = f.flight_num AND t.airline_name = f.airline_name  
WHERE p.booking_agent_id = %s  
    AND p.purchase_date >= CURDATE() - INTERVAL 30 DAY;  
""", (booking_agent_id,))
```

Query to get top 5 customers by number of tickets bought in the last 6 months

```
SELECT p.customer_email, COUNT(p.ticket_id) AS tickets_bought  
FROM purchases p  
JOIN ticket t ON p.ticket_id = t.ticket_id  
WHERE p.purchase_date >= CURDATE() - INTERVAL 6 MONTH  
GROUP BY p.customer_email  
ORDER BY tickets_bought DESC  
LIMIT 5;
```

Query to get top 5 customers by commission received in the last year

```
SELECT p.customer_email, SUM(f.price * 0.05) AS commission_received  
FROM purchases p  
JOIN ticket t ON p.ticket_id = t.ticket_id  
JOIN flight f ON t.flight_num = f.flight_num
```

```
WHERE p.purchase_date >= CURDATE() - INTERVAL 1 YEAR
GROUP BY p.customer_email
ORDER BY commission_received DESC
LIMIT 5;
```

```
# Booking Agent Search Flights route logic
```

```
def agent_search_flights():
```

```
    # Get the airline associated with the booking agent
```

```
    cursor.execute("SELECT airline_name FROM booking_agent_work_for WHERE email = %s", (user_email,))
```

```
    # Search for flights based on the source, destination, and date
```

```
    query = ""
```

```
        SELECT * FROM flight
```

```
        WHERE departure_airport = %s
```

```
            AND arrival_airport = %s
```

```
            AND DATE(departure_time) = %s
```

```
            AND airline_name = %s
```

```
    ""
```

```
    cursor.execute(query, (source, destination, date, airline_name))
```

```
# Booking Agent Purchase Ticket route logic
```

```
def agent_purchase_ticket():
```

```
    # Get the booking agent ID based on the logged-in agent's email
```

```
    cursor.execute("SELECT booking_agent_id FROM booking_agent WHERE email = %s", (user_email,))
```

```
    # Get the airline associated with the booking agent
```

```
    cursor.execute("SELECT airline_name FROM booking_agent_work_for WHERE email = %s", (user_email,))
```

Find an available ticket for the selected flight

query = """

SELECT t.ticket_id

FROM ticket t

LEFT JOIN purchases p ON t.ticket_id = p.ticket_id

WHERE t.flight_num = %s AND p.ticket_id IS NULL

LIMIT 1;

"""

cursor.execute(query, (flight_num,))

Ensure the customer exists by checking their email in the customer table

cursor.execute("SELECT * FROM customer WHERE email = %s", (customer_email,))

Insert the purchase record for the selected ticket, customer, and booking agent

cursor.execute("""

INSERT INTO purchases (ticket_id, customer_email, booking_agent_id, purchase_date)

VALUES (%s, %s, %s, CURDATE());

""", (ticket_id, customer_email, booking_agent_id))

Airline Staff

Airline Staff Dashboard route logic

def airline_staff_dashboard():

Fetch the airline name for the logged-in airline staff

cursor.execute("SELECT airline_name FROM airline_staff WHERE username = %s",
(user_email,))

Query to get upcoming flights for the airline in the next 30 days

query = """

```

SELECT f.flight_num, f.departure_time, f.arrival_time, f.departure_airport,
f.arrival_airport,
    f.airline_name, COUNT(p.ticket_id) AS num_customers
FROM flight f
LEFT JOIN ticket t ON f.flight_num = t.flight_num AND f.airline_name = t.airline_name
LEFT JOIN purchases p ON t.ticket_id = p.ticket_id
WHERE f.airline_name = %s AND f.departure_time >= CURDATE()
GROUP BY f.flight_num
HAVING f.departure_time <= CURDATE() + INTERVAL 30 DAY
ORDER BY f.departure_time;

```

"""

```

cursor.execute(query, (airline_name,))

```

Query for custom filtering based on date range, airports/cities

```

query = """

```

```

SELECT f.flight_num, f.departure_time, f.arrival_time, f.departure_airport,
f.arrival_airport,
    COUNT(p.ticket_id) AS num_customers
FROM flight f
LEFT JOIN ticket t ON f.flight_num = t.flight_num AND f.airline_name = t.airline_name
LEFT JOIN purchases p ON t.ticket_id = p.ticket_id
WHERE f.airline_name = %s
AND f.departure_time BETWEEN %s AND %s
AND f.departure_airport LIKE %s
AND f.arrival_airport LIKE %s
GROUP BY f.flight_num
ORDER BY f.departure_time;

```

"""

```

cursor.execute(query, (airline_name, start_date, end_date, f'{source_airport}%',
f'{destination_airport}%'))

```

```
# Query to get customers for each flight
cursor.execute("""
    SELECT c.name, c.email
    FROM purchases p
    JOIN ticket t ON p.ticket_id = t.ticket_id
    JOIN customer c ON p.customer_email = c.email
    WHERE t.flight_num = %s AND t.airline_name = %s;
""", (flight['flight_num'], airline_name))
```

Grant Permissions route logic

```
def grant_permissions():
```

```
    # Check if the staff username exists in the airline_staff table
```

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

```
    # Check if the staff member already has the requested permission
```

```
    cursor.execute("SELECT * FROM permission WHERE username = %s AND
permission_type = %s", (staff_username, new_permission))
```

```
    # Insert the new permission for the staff member
```

```
    cursor.execute("INSERT INTO permission (username, permission_type) VALUES (%s, %s)",
(staff_username, new_permission))
```

Check Admin Permissions helper function

```
def check_admin_permissions(user_email):
```

```
    # Query to check if the logged-in staff has "Admin" permission
```

```
    cursor.execute("SELECT * FROM permission WHERE username = %s AND
permission_type = 'Admin'", (user_email,))
```

Check Operator Permissions helper function

```
def check_operator_permission(user_email):
```

```
    # Query to check if the logged-in staff has "Operator" permission
```

```
cursor.execute("SELECT * FROM permission WHERE username = %s AND  
permission_type = 'Operator'", (user_email,))
```

```
# Add Booking Agent route logic
```

```
def add_booking_agent():
```

```
    # Check if the email already exists in the booking_agent table
```

```
    cursor.execute("SELECT * FROM booking_agent WHERE email = %s",  
(booking_agent_email,))
```

```
    # Insert the new booking agent into the booking_agent table
```

```
    cursor.execute("INSERT INTO booking_agent (email, password, booking_agent_id) VALUES  
(%s, %s, %s)", (booking_agent_email, hashed_password, generate_booking_agent_id()))
```

```
    # Link the booking agent to the airline in booking_agent_work_for table
```

```
    cursor.execute("INSERT INTO booking_agent_work_for (email, airline_name) VALUES (%s,  
%s)", (booking_agent_email, airline_name))
```

```
# Add Flight route logic
```

```
def create_flight():
```

```
    # Ensure flight number is unique for the airline
```

```
    cursor.execute("SELECT * FROM flight WHERE airline_name = %s AND flight_num =  
%s", (airline_name, flight_num))
```

```
    # Insert the new flight into the flight table
```

```
    cursor.execute("""INSERT INTO flight (airline_name, flight_num, departure_airport,  
departure_time, arrival_airport, arrival_time, price, status, airplane_id)
```

```
VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s)
```

```
""", (airline_name, flight_num, departure_airport, departure_time, arrival_airport,  
arrival_time, price, status, airplane_id))
```

```
# Create tickets based on the number of seats on the airplane
```

```
        cursor.execute("SELECT seats FROM airplane WHERE airline_name = %s AND airplane_id  
= %s", (airline_name, airplane_id))
```

```
# Change Flight Status route logic
```

```
def change_flight_status(airline_name, flight_num):
```

```
    # Update the status of the flight
```

```
    cursor.execute("""
```

```
        UPDATE flight
```

```
        SET status = %s
```

```
        WHERE airline_name = %s AND flight_num = %s
```

```
    """, (new_status, airline_name, flight_num))
```

```
# Add Airplane route logic
```

```
def add_airplane():
```

```
    # Ensure airplane ID is unique for the airline
```

```
    cursor.execute("SELECT * FROM airplane WHERE airline_name = %s AND airplane_id =  
%s", (airline_name, airplane_id))
```

```
    # Insert the new airplane into the airplane table
```

```
    cursor.execute("INSERT INTO airplane (airline_name, airplane_id, seats) VALUES (%s, %s,  
%s)", (airline_name, airplane_id, seats))
```

```
# Airplane List route logic
```

```
def airplane_list():
```

```
    # Fetch all airplanes for the logged-in airline
```

```
    cursor.execute("SELECT airplane_id, seats FROM airplane WHERE airline_name = %s",  
(airline_name,))
```

```
# Add Airport route logic
```

```
def add_airport():
```

```
    # Ensure airport name is unique
```



```

cursor.execute("SELECT * FROM airport WHERE airport_name = %s", (airport_name,))

# Insert the new airport into the airport table
cursor.execute("INSERT INTO airport (airport_name, airport_city) VALUES (%s, %s)",
(airport_name, airport_city))

# Airport List route logic
def airport_list():
    # Fetch all airports
    cursor.execute("SELECT airport_name, airport_city FROM airport")

# View Booking Agents route logic
def view_booking_agents():
    # Fetch airline name for the logged-in staff member
    cursor.execute("SELECT airline_name FROM airline_staff WHERE username = %s",
(user_email,))

# Get top 5 booking agents based on the number of tickets sold in the past month
cursor.execute("""
    SELECT ba.email, COUNT(p.ticket_id) AS tickets_sold
    FROM booking_agent ba
    LEFT JOIN purchases p ON ba.booking_agent_id = p.booking_agent_id
    WHERE p.purchase_date >= CURDATE() - INTERVAL 1 MONTH OR p.purchase_date IS
NULL
    GROUP BY ba.email
    ORDER BY tickets_sold DESC
    LIMIT 5
""")

# Get top 5 booking agents based on the number of tickets sold in the past year
cursor.execute("""

```

```

SELECT ba.email, COUNT(p.ticket_id) AS tickets_sold
FROM booking_agent ba
LEFT JOIN purchases p ON ba.booking_agent_id = p.booking_agent_id
WHERE p.purchase_date >= CURDATE() - INTERVAL 1 YEAR OR p.purchase_date IS
NULL
GROUP BY ba.email
ORDER BY tickets_sold DESC
LIMIT 5
""")

```

```

# Get top 5 booking agents based on the commission earned in the past year
cursor.execute("""
SELECT ba.email, COALESCE(SUM(f.price * 0.05), 0) AS commission_received
FROM booking_agent ba
LEFT JOIN purchases p ON ba.booking_agent_id = p.booking_agent_id
LEFT JOIN ticket t ON p.ticket_id = t.ticket_id
LEFT JOIN flight f ON t.flight_num = f.flight_num AND t.airline_name = f.airline_name
WHERE p.purchase_date >= CURDATE() - INTERVAL 1 YEAR OR p.purchase_date IS
NULL
GROUP BY ba.email
ORDER BY commission_received DESC
LIMIT 5
""")

```

```

# View Frequent Customers route logic
def view_frequent_customers():
    # Get the airline name for the logged-in airline staff
    cursor.execute("SELECT airline_name FROM airline_staff WHERE username = %s",
(user_email,))

    # Get the top frequent customers based on ticket purchases in the last year

```

```

cursor.execute("""
    SELECT c.email, c.name, COUNT(p.ticket_id) AS num_tickets
    FROM customer c
    JOIN purchases p ON c.email = p.customer_email
    JOIN ticket t ON p.ticket_id = t.ticket_id
    JOIN flight f ON t.flight_num = f.flight_num AND f.airline_name = %s
    WHERE p.purchase_date >= CURDATE() - INTERVAL 1 YEAR
    GROUP BY c.email
    ORDER BY num_tickets DESC
    LIMIT 5
    """, (airline_name,))

```

Query for flights of the selected customer

```

cursor.execute("""
    SELECT f.flight_num, f.departure_time, f.arrival_time, f.departure_airport, f.arrival_airport
    FROM purchases p
    JOIN ticket t ON p.ticket_id = t.ticket_id
    JOIN flight f ON t.flight_num = f.flight_num AND f.airline_name = %s
    WHERE p.customer_email = %s
    """, (airline_name, selected_customer_email))

```

View Reports route logic

def view_reports():

Get the airline name for the logged-in staff

```

    cursor.execute("SELECT airline_name FROM airline_staff WHERE username = %s",
    (user_email,))

```

Query for total tickets sold in the custom date range

```

cursor.execute("""
    SELECT COUNT(p.ticket_id) AS total_sales
    FROM purchases p

```

```

JOIN ticket t ON p.ticket_id = t.ticket_id
JOIN flight f ON t.flight_num = f.flight_num AND t.airline_name = %s
WHERE p.purchase_date BETWEEN %s AND %s
""", (airline_name, start_date, end_date))

```

Query for month-wise sales for the custom date range

```

cursor.execute("""

```

```

    SELECT DATE_FORMAT(p.purchase_date, '%Y-%m') AS month, COUNT(p.ticket_id)

```

```

AS tickets_sold

```

```

    FROM purchases p

```

```

    JOIN ticket t ON p.ticket_id = t.ticket_id

```

```

    JOIN flight f ON t.flight_num = f.flight_num AND t.airline_name = %s

```

```

    WHERE p.purchase_date BETWEEN %s AND %s

```

```

    GROUP BY month

```

```

    ORDER BY month

```

```

""", (airline_name, start_date, end_date))

```

View Revenue Comparison route logic

```

def view_revenue_comparison():

```

```

    # Get the airline name for the logged-in staff

```

```

    cursor.execute("SELECT airline_name FROM airline_staff WHERE username = %s",

```

```

    (user_email,))

```

Query for direct and indirect revenue for last month and last year

```

cursor.execute("""

```

```

    SELECT SUM(f.price) AS direct_revenue

```

```

    FROM purchases p

```

```

    JOIN ticket t ON p.ticket_id = t.ticket_id

```

```

    JOIN flight f ON t.flight_num = f.flight_num AND t.airline_name = %s

```

```

    WHERE p.booking_agent_id IS NULL AND p.purchase_date >= CURDATE() -

```

```

INTERVAL 1 MONTH

```

```
""", (airline_name,))
```

```
# Query for indirect revenue for last month
```

```
cursor.execute("""
```

```
    SELECT SUM(f.price) AS indirect_revenue
```

```
    FROM purchases p
```

```
    JOIN ticket t ON p.ticket_id = t.ticket_id
```

```
    JOIN flight f ON t.flight_num = f.flight_num AND t.airline_name = %s
```

```
    WHERE p.booking_agent_id IS NOT NULL AND p.purchase_date >= CURDATE() -  
INTERVAL 1 MONTH
```

```
""", (airline_name,))
```

```
# View Top Destinations route logic
```

```
def view_top_destinations():
```

```
    # Get the airline name for the logged-in staff
```

```
    cursor.execute("SELECT airline_name FROM airline_staff WHERE username = %s",  
(user_email,))
```

```
# Query for the top 3 destinations for the last 3 months
```

```
cursor.execute("""
```

```
    SELECT f.arrival_airport, a.airport_city, COUNT(f.flight_num) AS num_flights
```

```
    FROM flight f
```

```
    JOIN airport a ON f.arrival_airport = a.airport_name
```

```
    WHERE f.airline_name = %s AND f.departure_time >= CURDATE() - INTERVAL 3  
MONTH
```

```
    GROUP BY f.arrival_airport
```

```
    ORDER BY num_flights DESC
```

```
    LIMIT 3
```

```
""", (airline_name,))
```

```
# Query for the top 3 destinations for the last year
```

```
cursor.execute("""
    SELECT f.arrival_airport, a.airport_city, COUNT(f.flight_num) AS num_flights
    FROM flight f
    JOIN airport a ON f.arrival_airport = a.airport_name
    WHERE f.airline_name = %s AND f.departure_time >= CURDATE() - INTERVAL 1
YEAR
    GROUP BY f.arrival_airport
    ORDER BY num_flights DESC
    LIMIT 3
""", (airline_name,))
```