This is the file that lists usecases and the queries executed by them. Python codes are also included when in need. By default **%s** means the corresponding input flask grabs from the frontend.

1 View Public Info

1.1 View Public Info

Searching for upcoming flights based on source city/airport name, destination city/airport name, date is done by this:

```
#Showing all upcoming flights by default
SELECT flight_number, airline_name, departure_airport, arrival_airport,
        departure_time, arrival_time
FROM Flight
WHERE departure_time > NOW()
#If source city/airport provided
AND(
    departure_airport IN (SELECT airport_name FROM Airport WHERE city LIKE %s)
    OR departure_airport LIKE %s
    )
#If destination provided
AND (
        arrival_airport IN (SELECT airport_name FROM Airport WHERE city LIKE %s)
        OR arrival_airport LIKE %s
    )
#If date provided
AND DATE(departure_time) = %s
```

Seeing flight status is done by this query:

```
SELECT flight_number, airline_name, departure_airport, arrival_airport,
        departure_time, arrival_time, flight_status
FROM Flight
WHERE 1=1
#If flight number provided
AND flight_number = %s
#If departure date provided
AND DATE(departure_time) = %s
#If arrival date provided
AND DATE(arrival_time) = %s
```

2 register

The type of registration is controlled by flask backend.
If registering as customer:

```
INSERT INTO Customer (email, customer_password, name_customer, phone_number, passport_nu
VALUES (%s, MD5(%s), %s, %s, %s)
```

If registering as booking agent, a new transaction will be started by flask, and the following query is executed to find the current maximum booking_agent_id:

```
SELECT MAX(booking_agent_id) AS max_id
FROM Booking_agent
FOR UPDATE
```

We add 1 on it and assign this number as the new booking_agent_id for the newly registered agent. The information is inserted into the table using this query:

```
INSERT INTO Booking_agent
(email, Name_agent, agent_password, booking_agent_id)
VALUES (%s, %s, MD5(%s), %s)
```

If there's no error this will be committed. If an error occurred we rollback and return a simply error message.
If registering as airline staff:

```
INSERT INTO Airline_staff (username, password_stuff, airline_name, date_of_birth, first_
VALUES (%s, MD5(%s), %s, %s, %s, %s, %s)
```

3 login

Again, the type of login is controlled by flask backend. Flask run these three queries to determine which usercase the login will redirect to corresponding homepage, assuming unique email:

```
#Customer
SELECT email, name_customer FROM Customer WHERE email = %s AND customer_password = MD5(%
#Booking agent
SELECT email, Name_agent FROM Booking_agent WHERE email = %s AND agent_password = MD5(%s
#Airline_staff
SELECT staff_email, username FROM Airline_staff WHERE staff_email = %s AND password_stuf
```

If email/password combination found in non of these three tables, we return an error saying invalid username or password. Or in case any error happens, we return error message saying an error occurred.

4 Customer use cases

4.1 View My Flights

Customer view his upcoming flights in My Upcoming Flights using the following query:

```
SELECT F.flight_number, F.airline_name, F.departure_airport, F.arrival_airport,
       F.departure_time, F.arrival_time
FROM Flight F
JOIN Ticket T ON (T.airline_name = F.airline_name AND T.flight_number = F.flight_number)
WHERE T.customer_email = %s
AND F.departure_time > NOW()
#If departure_date:
AND DATE(F.departure_time) >= %s
#If arrival_date:
AND DATE(F.departure_time) <= %s
#If departure airport/city:
AND (F.departure_airport IN
        (SELECT airport_name FROM Airport WHERE city LIKE %s)
    OR F.departure_airport LIKE %s)
#If destination city/airport:
AND (F.arrival_airport IN
        (SELECT airport_name FROM Airport WHERE city LIKE %s)
    OR F.arrival_airport LIKE %s)
```

4.2 Search & Purchase Tickets

Search for flights and purchase tickets are combined into the same page. Customer search for flights by executing this query:

```
SELECT flight_number, airline_name, departure_airport, arrival_airport,
        departure_time, arrival_time, price, flight_number AS flight_id
FROM Flight
WHERE departure_time > NOW()
#If source city/airport specified:
AND (departure_airport IN
        (SELECT airport_name FROM Airport WHERE city LIKE %s)
    OR departure_airport LIKE %s)
#If destination city/airport specified:
AND (arrival_airport IN
        (SELECT airport_name FROM Airport WHERE city LIKE %s)
    OR arrival_airport LIKE %s)
```

```
#If departure date:
AND DATE(departure_time) = %s
```

During ticket purchase, an transaction is started by flask. The flight is specified using the previous query and info including flight and airplane_id is fetched by this query:

```
SELECT airline_name,
       flight_number,
       departure_airport,
       arrival_airport,
       departure_time,
       arrival_time,
       price,
       airplane_id
FROM Flight
WHERE flight_number = %s
```

The backend return an error if no flight was found. Then flask executes this query to count sold tickets:

```
SELECT COUNT(*) AS sold_count
FROM Ticket
WHERE airline_name = %s
AND flight_number = %s
```

And executes this query to lookup plane capacity:

```
SELECT seats
FROM Airplane
WHERE airline_name = %s
AND airplane_id = %s
```

the transaction rolls back if sold tickets >= plane capacity. If it didn't rollback, These two queries are executed to insert a ticket into ticket and purchases:

```
INSERT INTO Ticket
(ticket_id, airline_name, flight_number, customer_email)
VALUES (%s, %s, %s, %s);
INSERT INTO purchases
(ticket_id, customer_email, booking_agent_id, purchase_time)
VALUES (%s, %s, NULL, NOW())
```

4.3 Check My Spending

Track my spending is achieved using this query:

```
SELECT SUM(F.price) as total_spent
FROM Ticket T
JOIN Flight F ON (T.airline_name = F.airline_name AND T.flight_number = F.flight_number)
JOIN Purchases P ON (T.ticket_id = P.ticket_id)
WHERE P.purchase_time >= DATE_SUB(CURDATE(), INTERVAL 1 YEAR)
AND T.customer_email = %s
#If departure_date or arrival_date
SELECT SUM(F.price) as total_spent
FROM Ticket T
JOIN Flight F ON (T.airline_name = F.airline_name AND T.flight_number = F.flight_number)
JOIN Purchases P ON (T.ticket)
WHERE T.customer_email = %s
#If start_date:
AND DATE(F.departure_time) >= %s
#If end_date:
AND DATE(F.departure_time) <= %s
```

Check monthly spending is achieved using this query:

```
SELECT DATE_FORMAT(P.purchase_time, '%Y-%m') AS month_key,
           SUM(F.price)                 AS monthly_sum
FROM Ticket T
JOIN Flight F     ON T.airline_name = F.airline_name AND T.flight_number = F.flight_numbe
JOIN Purchases P ON T.ticket_id = P.ticket_id
WHERE T.customer_email = %s
#If start_date:
AND DATE(P.purchase_time) >= %s
#Else:
AND P.purchase_time >= DATE_SUB(CURDATE(), INTERVAL 6 MONTH)
#If end_date:
AND DATE(P.purchase_time) <= %s
GROUP BY month_key ORDER BY month_key
```

5 Booking Agent Use Cases

5.1 View My Flights

Viewing Agent flights is done using this query:

```
SELECT F.flight_number, F.airline_name, F.departure_airport,
       F.arrival_airport, F.departure_time, F.arrival_time
FROM Flight F
```

```
JOIN Ticket  T USING (airline_name, flight_number)
JOIN Booking_agent BA ON T.booking_agent_id = BA.booking_agent_id
WHERE BA.email = %s
AND F.departure_time > NOW()
#If start_date:
AND DATE(F.departure_time) >= %s
#If end_date:
AND DATE(F.departure_time) <= %s
#If source:
AND (F.departure_airport LIKE %s OR F.departure_airport IN
(SELECT airport_name FROM Airport WHERE city LIKE %s))
#If destination
AND (F.arrival_airport LIKE %s OR F.arrival_airport IN
(SELECT airport_name FROM Airport WHERE city LIKE %s))
```

5.2 Agent Search & Purchase

Booking agent search and purchase is integrated into the same page. Search is done by this query:

```
SELECT flight_number, airline_name, departure_airport, arrival_airport,
   departure_time, arrival_time, price, flight_number AS flight_id
FROM Flight
WHERE departure_time > NOW()
#If source city/airport:
AND (departure_airport IN
    (SELECT airport_name FROM Airport WHERE city LIKE %s)
    OR departure_airport LIKE %s)
#If destination city/airport:
AND (arrival_airport IN
    (SELECT airport_name FROM Airport WHERE city LIKE %s)
     OR arrival_airport LIKE %s)
#If departure_date:
AND DATE(departure_time) = %s
```

Then a transaction is initialized by flask, mostly the same as the one we see in customer search/-purchase ticket:

```
SELECT airline_name, flight_number, airplane_id, price
FROM Flight
WHERE flight_number = %s
```

Flask uses this query to determine if the booking agent works for the airline:

```
SELECT 1
    FROM Agent_status
    WHERE agent_email = %s
    AND airline_name = %s
```

If there's record the backend allows that booking agent to purchase ticket of that airline, else prohibited.
flight_number is fetched by flask via the last query. Flask executes this query to lookup plane capacity and sold tickets:

```
#plane capacity
SELECT COUNT(*) AS sold
FROM Ticket
WHERE airline_name = %s
AND flight_number = %s
#sold tickets
SELECT seats
FROM Airplane
WHERE airline_name = %s
AND airplane_id  = %s
```

If sold equals or greater than capacity, flask return error saying this flight is fully booked and rollback. Else flask execute these two queries to create a new ticket and a new purchase record:

```
INSERT INTO Ticket
(ticket_id, airline_name, flight_number,
customer_email, booking_agent_id)
VALUES (%s, %s, %s, %s,
(SELECT booking_agent_id
  FROM Booking_agent
 WHERE email = %s));
 INSERT INTO purchases
(ticket_id, customer_email, booking_agent_id, purchase_time)
VALUES (%s, %s,
(SELECT booking_agent_id FROM Booking_agent WHERE email=%s),
NOW())
```

And flask commits the transaction.

5.3 View My Commission

Agents view commission by executing this query if no date is specified:

```
SELECT SUM(0.1*F.price) AS total_comm,
          COUNT(*)         AS tickets_sold,
          CASE WHEN COUNT(*)=0 THEN 0
               ELSE SUM(0.1*F.price)/COUNT(*) END AS avg_comm
      FROM purchases   P
      JOIN Ticket      T USING(ticket_id)
      JOIN Flight      F USING(airline_name, flight_number)
      JOIN Booking_agent BA ON P.booking_agent_id = BA.booking_agent_id
     WHERE BA.email = %s
     AND P.purchase_time >= DATE_SUB(CURDATE(), INTERVAL 30 DAY)
```

If the range of dates is specified, this query will be executed:

```
SELECT SUM(0.1*F.price) AS total_comm,
          COUNT(*)         AS tickets_sold,
          CASE WHEN COUNT(*)=0 THEN 0
               ELSE SUM(0.1*F.price)/COUNT(*) END AS avg_comm
FROM purchases   P
JOIN Ticket      T USING(ticket_id)
JOIN Flight      F USING(airline_name, flight_number)
JOIN Booking_agent BA ON P.booking_agent_id = BA.booking_agent_id
WHERE BA.email = %s
#If start date:
AND DATE(P.purchase_time) >= %s
#If end date:
AND DATE(P.purchase_time) <= %s
```

5.4 View Top Customer

Agents view top customer by executing this query:

```
#Top 5 by tickets in the last 6 months
SELECT
    t.customer_email,
    COUNT(*) AS num_tickets
FROM Ticket t
JOIN Booking_agent b
    ON b.booking_agent_id = t.booking_agent_id
JOIN purchases p
    ON p.ticket_id = t.ticket_id
WHERE
    b.email = %s
    AND p.purchase_time >= DATE_SUB(CURDATE(), INTERVAL 6 MONTH)
GROUP BY
    t.customer_email
ORDER BY
    num_tickets DESC
LIMIT 5;
#Top 5 by Commission last year
SELECT
    p.customer_email,
    SUM(f.price * 0.10) AS total_commission
FROM purchases p
JOIN Ticket t
    ON p.ticket_id = t.ticket_id
JOIN Flight f
    ON f.airline_name = t.airline_name
    AND f.flight_number = t.flight_number
JOIN Booking_agent b
    ON b.booking_agent_id = p.booking_agent_id
WHERE
    p.booking_agent_id IS NOT NULL
    AND p.purchase_time >= DATE_SUB(CURDATE(), INTERVAL 1 YEAR)
    AND b.email = %s
GROUP BY
    p.customer_email
ORDER BY
    total_commission DESC
LIMIT 5
```

6 Airline Staff Usecase

6.1 Handling Permissions

Those without permission Ädminör Öperator"can only view instead of modify. In Flask backend I defined a permission_required method whenever we need to know what type of staff can get access to the page:

```
def permission_required(*allowed):
    """
    Decorator:  ensure logged-in staff member has at least one of the
    given permissions.  If not, return 403.
    """
    def decorator(f):
        @wraps(f)
        def inner(*args, **kwargs):
            if not is_staff_logged_in():
                return redirect(url_for('login'))
            perms = session.get('permissions', [])
            if not any(p in perms for p in allowed):
                return "Unauthorized - missing permission", 403
            return f(*args, **kwargs)
        return inner
    return decorator
```

The permission a logged in staff have is accessed using this query:

```
SELECT permission_type FROM Permission_status WHERE staff_email=%s
```

And we store it during his login session

6.2 View My Flights

Agents view his or her flights use this query:

```
SELECT flight_number, departure_airport, arrival_airport,
     departure_time, arrival_time, flight_status
FROM Flight
WHERE airline_name=%s
AND departure_time BETWEEN NOW() AND DATE_ADD(NOW(), INTERVAL 30 DAY)
#If date / source / destination specified:
SELECT flight_number, departure_airport, arrival_airport,
    departure_time, arrival_time, flight_status
FROM Flight
WHERE airline_name=%s
#If start date:
AND departure_time >= %s
#If end date:
AND departure_time <= %s
#If source city/airport:
AND departure_airport LIKE %s"
#If destination city/airport:
AND arrival_airport LIKE %s
ORDER BY departure_time
```

6.3 Create New Flights

If the agent has Admin permission, he can create new flights using this query:

```
INSERT INTO Flight
(airline_name, flight_number, price, flight_status,
departure_time, arrival_time,
departure_airport, arrival_airport, airplane_id)
VALUES (%s,%s,%s,%s,%s,%s,%s,%s,%s)
```

6.4 Change Status of Flights

If the agent has operator or admin permission, he can change flight status using this query:

```
#Select flight to update
SELECT flight_number, flight_status
FROM Flight
WHERE airline_name=%s
ORDER BY departure_time;
#User can select from the above flights
UPDATE Flight
SET flight_status=%s
WHERE airline_name=%s AND flight_number=%s
```

Only status within a list can be used to change the status.

6.5 Add Airplane

If the staff has admin permission, he can add airplane in the system:

```
INSERT INTO Airplane (airplane_id, airline_name, seats)
VALUES (%s, %s, %s)
```

airline_name is fetched by flask.

6.6 Add Airport

If the staff has admin permission, he can add airport to the system:

```
INSERT INTO Airport (airport_name, city)
VALUES (%s, %s)
```

6.7 View all booking agents

A staff view all the booking agents using this query:

```
#All agents
SELECT ba.email AS agent_email
FROM Booking_agent ba
JOIN Agent_status a ON ba.email = a.agent_email
WHERE a.airline_name = %s;
#Top 5 agents last month
SELECT ba.email AS agent_email,
     COUNT(*) AS tickets_sold
FROM Ticket t
JOIN purchases p ON t.ticket_id=p.ticket_id
JOIN Booking_agent ba ON t.booking_agent_id=ba.booking_agent_id
WHERE t.airline_name=%s AND p.purchase_time>=DATE_SUB(NOW(), INTERVAL 1 MONTH)
GROUP BY ba.email
ORDER BY tickets_sold DESC
LIMIT 5;
#Top 5 agents last year
SELECT ba.email AS agent_email,
    COUNT(*) AS tickets_sold
FROM Ticket t
JOIN purchases p ON t.ticket_id=p.ticket_id
JOIN Booking_agent ba ON t.booking_agent_id=ba.booking_agent_id
WHERE t.airline_name=%s AND p.purchase_time>=DATE_SUB(NOW(), INTERVAL 1 YEAR)
GROUP BY ba.email
ORDER BY tickets_sold DESC
LIMIT 5;
#Top 5 booking agents based on commissions received last year:
SELECT ba.email AS agent_email,
     SUM(f.price*0.1) AS total_commission
FROM Ticket t
JOIN purchases p ON t.ticket_id=p.ticket_id
JOIN Booking_agent ba ON t.booking_agent_id=ba.booking_agent_id
JOIN Flight f ON t.airline_name=f.airline_name AND t.flight_number=f.flight_number
WHERE t.airline_name=%s AND p.purchase_time>=DATE_SUB(NOW(), INTERVAL 1 YEAR)
GROUP BY ba.email
ORDER BY total_commission DESC
LIMIT 5
```

6.8 View Frequent Customers

Agents view frequent customers by this query:

```
#Top 1 customer
SELECT p.customer_email, COUNT(*) AS freq
FROM Ticket t
JOIN purchases p ON t.ticket_id = p.ticket_id
WHERE t.airline_name = %s
AND p.purchase_time >= DATE_SUB(NOW(), INTERVAL 1 YEAR)
GROUP BY p.customer_email
ORDER BY freq DESC
LIMIT 1
# Fetch all customers who have purchased tickets for this airline
SELECT DISTINCT p.customer_email
FROM Ticket t
JOIN purchases p ON t.ticket_id = p.ticket_id
WHERE t.airline_name = %s;
#Fetch their flights if a specific customer is selected
SELECT f.flight_number, f.departure_time, f.arrival_time, f.departure_airport, f.arrival_
FROM Ticket t
JOIN purchases p ON t.ticket_id = p.ticket_id
JOIN Flight f ON t.airline_name = f.airline_name
         AND t.flight_number = f.flight_number
WHERE t.airline_name = %s
AND p.customer_email = %s
```

6.9 View Reports

Staff view report by running this query:

```
#Total sold between a specific time
SELECT COUNT(*) AS total_sold
FROM Ticket t
JOIN purchases p ON t.ticket_id=p.ticket_id
WHERE t.airline_name=%s
AND p.purchase_time BETWEEN %s AND %s;
#Monthly breakdown:
SELECT MONTH(p.purchase_time) AS month, COUNT(*) AS tickets
FROM Ticket t
JOIN purchases p ON t.ticket_id = p.ticket_id
WHERE t.airline_name = %s
AND p.purchase_time BETWEEN %s AND %s
```

```
GROUP BY MONTH(p.purchase_time)
ORDER BY month
```

last month / last year usecase have time specified as one month or one year by flask.

6.10 Comparison of Revenue Earned

Revenue comparison is handled by this query:

```
#direct sell
SELECT
SUM(f.price)
FROM Ticket t
JOIN purchases p ON t.ticket_id=p.ticket_id
JOIN Flight f ON t.airline_name=f.airline_name
    AND t.flight_number=f.flight_number
WHERE t.airline_name=%s
AND t.booking_agent_id IS NULL
AND p.purchase_time>=DATE_SUB(NOW(), INTERVAL 1 MONTH); #INTERVAL 1 YEAR if viewing last
#indirect sell
SELECT
SUM(f.price)
FROM Ticket t
JOIN purchases p ON t.ticket_id=p.ticket_id
JOIN Flight f ON t.airline_name=f.airline_name
    AND t.flight_number=f.flight_number
WHERE t.airline_name=%s
AND t.booking_agent_id IS NOT NULL
AND p.purchase_time>=DATE_SUB(NOW(), INTERVAL 1 MONTH) #INTERVAL 1 YEAR if viewing last
```

6.11 View Top Destinations

View Top destinations is handles by this query:

```
SELECT f.arrival_airport   AS dest,
    COUNT(*)             AS cnt
FROM Ticket t
JOIN purchases p
    ON t.ticket_id = p.ticket_id
JOIN Flight f
    ON t.airline_name = f.airline_name
AND t.flight_number = f.flight_number
```

```
WHERE t.airline_name = %s
AND p.purchase_time >= DATE_SUB(NOW(), INTERVAL 3 MONTH) #INTERVAL 1 YEAR if viewing las
GROUP BY f.arrival_airport
ORDER BY cnt DESC
LIMIT 3
```

6.12 Grant New Permissions

Grant new permission is done by this query for staff with Admin permission:

```
INSERT INTO Permission_status (username, permission_type)
VALUES (%s, %s)
```

6.13 Add booking agent

Adding booking agents is done by this query for staff with Admin permission:

```
INSERT INTO Agent_status (agent_email, airline_name)
VALUES (%s, %s)
```