Use Cases & Executed SQL

This document enumerates the application's use cases, the SQL executed for each (with %s placeholders for parameter binding via PyMySQL), and the server-side validations/edge cases implemented in app.py. It maps directly to Part 3 requirements for the Airline Ticket Reservation System.

Note: Table/column names follow the Part 2 schema used in this project (Customer, Airline_Staff, Flight, Airplane, Airport, Ticket, Rating/Review).

0) Session & Authentication

0.1 Login (Customer)

Route/Page: POST /login (branching by role), template: login.html Purpose: Authenticate a customer and create a session (session["role"]="customer", session["email"]=...).

SQL:

```
SELECT email, name, password
FROM Customer
WHERE email=%s AND password=MD5(%s);
```

Validation & Notes

- Credentials are checked with MD5 (per starter policy).
- On success: set session and redirect to customer_home.
- On failure: flash("Invalid credentials") and redisplay the form.

0.2 Login (Airline Staff)

Route/Page: POST /login (branching by role), template: login.html Purpose: Authenticate staff by username or email and create a session (session["role"]="staff", session["username"]/["email"], session["airline"]).

SQL (username or email)

```
SELECT username, email, airline_name, password
FROM Airline_Staff
WHERE (username=%s OR email=%s) AND password=MD5(%s);
```

Validation & Notes

- Uses prepared statements with %s.
- On success: set session["airline"] to scope all staff actions.
- On failure: flash("Invalid credentials").

0.3 Logout (Both Roles)

Route: GET /logout Purpose: Clear session and redirect to home or login.

No SQL

1) Public / Customer Use Cases

1.1 Public Search

Route/Page: GET/POST /customer/search, template: customer_search.html Purpose: Allow users (including not logged in) to search future and non-cancelled flights by optional From/To IATA and departure date.

SQL:

```
SELECT airline_name, flight_number, departure_date_time,
arrival_date_time,
       base_price, departure_airport, arrival_airport, status
FROM Flight
WHERE departure_date_time > NOW()
  AND status <> 'CANCELLED'
  /* + AND departure_airport=%s        -- if provided */
  /* + AND arrival_airport=%s          -- if provided */
  /* + AND DATE(departure_date_time)=%s -- if provided */
ORDER BY departure_date_time;
```

Validation & Notes

- IATA codes are normalized to uppercase.
- Results hide past or cancelled flights by default.
- The Buy form is rendered per row (for logged-in customers).

1.2 Purchase Ticket

Route: POST /customer/purchase Purpose: Create a ticket for the selected flight, with strict server-side rules.

Validations

- Flight exists.
- Flight is not CANCELLED and departure_date_time > NOW().
- name_on_card matches the account name (case/space-insensitive exact match).
- card_number is digits only.
- Required fields present; friendly flash() messages on failure.

SQL:

```
-- Check flight status & time
SELECT status, departure_date_time
FROM Flight
WHERE airline_name=%s AND flight_number=%s AND departure_date_time=%s
LIMIT 1;

-- Customer’s account name
SELECT name FROM Customer WHERE email=%s;

-- Next ticket ID (if not AUTO_INCREMENT)
SELECT COALESCE(MAX(ticket_ID),0)+1 AS next_id FROM Ticket;

-- Insert ticket
INSERT INTO Ticket(
  ticket_ID, customer_email, airline_name, flight_number,
departure_date_time,
  card_type, card_number, name_on_card, expiration_date,
purchase_date_time
) VALUES (%s,%s,%s,%s,%s,%s,%s,%s,%s,NOW());
```

Notes

- All values bound via %s (prepared statements).
- On success: flash("Ticket purchased (#...)") then redirect to customer_home.

1.3 View “My Flights”

Route/Page: GET /customer/myflights, template: customer_myflights.html Purpose: Show all purchased flights for the logged-in customer; optional filters for period/range.

SQL:

```
SELECT t.ticket_ID, f.airline_name, f.flight_number,
       f.departure_date_time, f.arrival_date_time,
       f.departure_airport, f.arrival_airport, f.status
FROM Ticket t
JOIN Flight f ON (f.airline_name=t.airline_name
              AND f.flight_number=t.flight_number
              AND f.departure_date_time=t.departure_date_time)
WHERE t.customer_email=%s
/* Optional:
   AND f.departure_date_time >= NOW()      -- future
   AND f.departure_date_time <  NOW()      -- past
   AND DATE(f.departure_date_time) BETWEEN %s AND %s  -- custom range */
ORDER BY f.departure_date_time DESC;
```

1.4 Rate / Comment (Completed Flights Only)

Route/Page: POST /customer/review, template: customer_reviews.html Purpose: Allow a customer to rate/comment a flight they actually took and that has already arrived.

SQL:

```
-- Verify ownership and completion
SELECT 1
FROM Ticket t
JOIN Flight f ON (f.airline_name=t.airline_name
              AND f.flight_number=t.flight_number
              AND f.departure_date_time=t.departure_date_time)
WHERE t.customer_email=%s
  AND t.airline_name=%s
  AND t.flight_number=%s
  AND t.departure_date_time=%s
  AND f.arrival_date_time < NOW();

-- Insert or upsert rating/comment
INSERT INTO Rating(customer_email, airline_name, flight_number,
departure_date_time, score, comment, created_at)
VALUES (%s,%s,%s,%s,%s,%s,NOW())
ON DUPLICATE KEY UPDATE score=VALUES(score), comment=VALUES(comment);
```

Notes

- Comments are listed newest first; average score is computed for staff view.

2) Airline Staff Use Cases

2.1 View Flights (Default = Next 30 Days, with Filters)

Route/Page: GET/POST /staff (or /staff/view), template: staff_home.html Purpose: Staff can browse their airline's flights. Filters include:

- Period: Default (next 30 days) / Current / Future / Past
- Explicit Date Range (start, end)
- From/To IATA and From/To City

Base SQL (built dynamically per filter)

```
SELECT airline_name, flight_number, departure_date_time,
arrival_date_time,
       base_price, departure_airport, arrival_airport, status
FROM Flight
WHERE airline_name=%s
  /* Default (next 30 days) */
  AND departure_date_time BETWEEN NOW() AND (NOW() + INTERVAL 30 DAY)
  /* Alternate WHERE blocks for CURRENT/FUTURE/PAST or explicit range:
     AND DATE(departure_date_time) BETWEEN %s AND %s
```

```
      AND departure_airport=%s
      AND arrival_airport=%s
      AND (SELECT city FROM Airport WHERE code=departure_airport) LIKE
CONCAT('%',%s,'%')
      AND (SELECT city FROM Airport WHERE code=arrival_airport) LIKE
CONCAT('%',%s,'%') */
ORDER BY departure_date_time;
```

Notes

- “Create Flight” page also shows the next 30 days list beneath the form.

2.2 Create New Flight

Route/Page: GET/POST /staff/create-flight, template: staff_create_flight.html Purpose: Insert a flight for the staff’s airline; validate airplane ownership; after creation, re-display the form with a Next 30 Days list.

###SQL

```
-- Ownership check (airplane belongs to staff airline)
SELECT 1
FROM Airplane
WHERE airline_name=%s AND id_number=%s;

-- Insert flight
INSERT INTO Flight(
  airline_name, flight_number, departure_date_time, arrival_date_time,
  base_price, departure_airport, arrival_airport, airplane_id_number,
status
) VALUES (%(airline_name)s, %(flight_number)s, %(departure_date_time)s, %
(arrival_date_time)s,
          %(base_price)s, %(departure_airport)s, %(arrival_airport)s, %
(airplane_id_number)s, %(status)s);
```

Notes

- airline_name is taken from the session; IATA codes normalized to uppercase.
- Next-30-days list query:

```
SELECT flight_number, departure_date_time, arrival_date_time,
       departure_airport, arrival_airport, status
FROM Flight
WHERE airline_name=%s
  AND departure_date_time >= NOW()
  AND departure_date_time < DATE_ADD(NOW(), INTERVAL 30 DAY)
ORDER BY departure_date_time;
```

2.3 Change Flight Status

Route/Page: GET/POST /staff/change-status, template: staff_change_status.html Purpose: Update a flight's status (e.g., ON_TIME, DELAYED, CANCELLED).

SQL

```
-- Verify flight belongs to staff airline
SELECT 1
FROM Flight
WHERE airline_name=%s AND flight_number=%s AND departure_date_time=%s;

-- Update status
UPDATE Flight
SET status=%s
WHERE airline_name=%s AND flight_number=%s AND departure_date_time=%s;
```

2.4 Add Airplane

Route/Page: GET/POST /staff/add-airplane, template: staff_add_airplane.html Purpose: Add a new airplane to the airline fleet.

SQL

```
INSERT INTO Airplane(id_number, airline_name, seats, manufacturer, age)
VALUES(%s,%s,%s,%s,%s);
```

Notes

- airline_name is enforced from the session, not read from the form.

2.5 View Flight Ratings & Comments

Route/Page: GET /staff/ratings, template: staff_ratings.html Purpose: View average score and list of comments per flight.

SQL

```
-- Aggregate
SELECT AVG(score) AS avg_score, COUNT(*) AS num_reviews
FROM Rating
WHERE airline_name=%s AND flight_number=%s AND departure_date_time=%s;

-- Comments list
SELECT customer_email, score, comment, created_at
FROM Rating
```

```sql
WHERE airline_name=%s AND flight_number=%s AND departure_date_time=%s
ORDER BY created_at DESC;
```

2.6 Ticket Sales Reports

Route/Page: GET/POST /staff/reports, template: staff_reports.html Purpose: Display sales totals (by day or month) for a date range, last month, or last year.

SQL

```sql
-- Date range: per-day
SELECT DATE(t.purchase_date_time) AS day, COUNT(*) AS tickets
FROM Ticket t
WHERE t.airline_name=%s
  AND DATE(t.purchase_date_time) BETWEEN %s AND %s
GROUP BY day
ORDER BY day;

-- Last month: per-month (YYYY-MM)
SELECT DATE_FORMAT(t.purchase_date_time, '%Y-%m') AS ym, COUNT(*) AS
tickets
FROM Ticket t
WHERE t.airline_name=%s
  AND t.purchase_date_time >= DATE_SUB(CURDATE(), INTERVAL 1 MONTH)
GROUP BY ym
ORDER BY ym;

-- Last year: per-month (YYYY-MM)
SELECT DATE_FORMAT(t.purchase_date_time, '%Y-%m') AS ym, COUNT(*) AS
tickets
FROM Ticket t
WHERE t.airline_name=%s
  AND t.purchase_date_time >= DATE_SUB(CURDATE(), INTERVAL 1 YEAR)
GROUP BY ym
ORDER BY ym;
```

Notes

- Results displayed in a table; optionally chartable.

3) Security & Validation (Global)

- Prepared statements everywhere (%s) to prevent SQL injection.
- Role checks at route level (as_customer(), as_staff()).
- Business rules enforced server-side:
  - Purchase allowed only for future flights, not CANCELLED.
  - Staff operations restricted to the airline in session["airline"].
  - Airplane ownership validated before creating flights.

- Data validations:
  - Normalize IATA codes to uppercase.
  - name_on_card must match account name (normalized); card_number must be digits only.
- XSS mitigation: Jinja2 auto-escapes variables; we do not inject raw user input into HTML or SQL strings.
- Friendly errors: Invalid actions flash a message and redirect back to the relevant page.

4) Error Handling

- Missing/invalid inputs → flash() and redirect to the originating page.
- Record not found (e.g., unknown flight) → flash("Flight not found") and return to search.
- DB errors during development shown in console; in production they should be logged and a generic message returned.