Name : Rudra Patel
Topic : EU Flight Project Task
Contact No. : 7698251877
Email : 202201193@daiict.ac.in
College : DAIICT

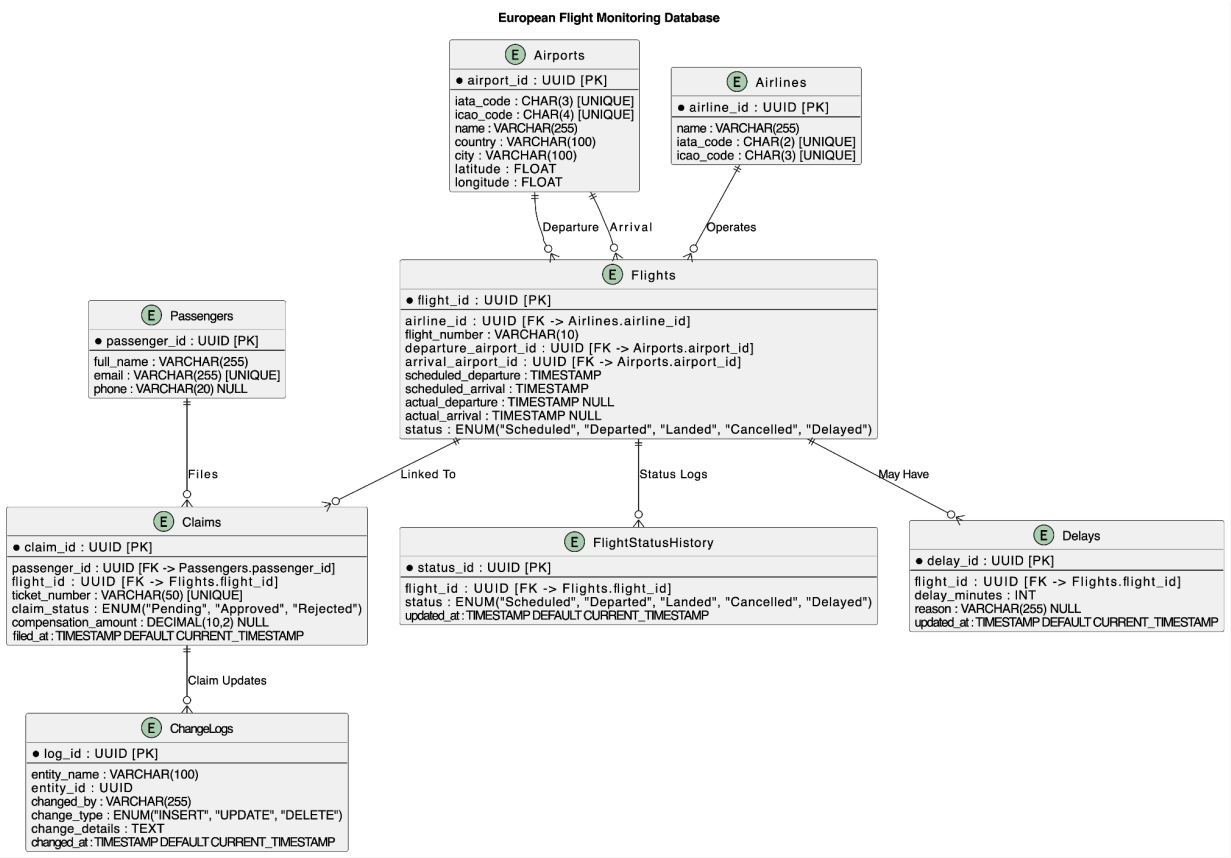
Project Overview:

We plan to develop a comprehensive database of all airports across Europe, along with detailed flight information from all airports. Our primary objective is to monitor flights and identify those delayed by more than 2 hours so we can assist passengers in filing claims for refunds. In the future, we also intend to create our own API that will store and provide daily data of all flights across Europe. Your task is to demonstrate how you would approach this challenge.

Part 1 : Theoretical Tasks

1. Database Design

To efficiently store and manage European airport and flight data, we will use a relational database (e.g., PostgreSQL or MySQL) for structured data storage. The schema will be normalized to avoid redundancy and ensure scalability.



1. Airports Table

- Stores information about all European airports (IATA code, ICAO code, country, city, latitude, longitude).
- Related to Flights as both departure and arrival airports.

🔗 Relationships:

- One airport can have multiple departing and arriving flights (Flights table).

---

2. Airlines Table

- Stores airline details (IATA, ICAO codes, and name).
- Helps track which airline operates a specific flight.

🔗 Relationships:

- One airline operates multiple flights (Flights table).

---

3. Flights Table

- Stores all scheduled and real-time flight information.
- Tracks departure and arrival times, flight status, and related airline.
- Status options: "Scheduled", "Departed", "Landed", "Cancelled", "Delayed".

Relationships:

- departure_airport_id and arrival_airport_id link to the Airports table.
- airline_id links to the Airlines table.
- May have a delay (Delays table) and status updates (FlightStatusHistory table).

---

4. Delays Table

- Stores delay details for flights that are late.
- Tracks reason and updated timestamp.

Relationships:

- A flight can have multiple delay records (Flights table).

---

5. Passengers Table

- Stores passenger information (full name, email, phone).
- Required for processing compensation claims.

Relationships:

- A passenger can file multiple claims (Claims table).

---

6. Claims Table

- Stores compensation claims from passengers for delayed flights.
- Tracks claim status (Pending, Approved, Rejected).
- Stores compensation amount and ticket number.

Relationships:

- A claim is linked to a passenger (Passengers table).
- A claim is linked to a flight (Flights table).
- Updates in claims are tracked in the ChangeLogs table.

---

7. FlightStatusHistory Table

- Stores historical flight status updates.
- Helps track how the status of a flight has changed over time.

Relationships:

- A flight can have multiple status updates (Flights table).

---

8. ChangeLogs Table

- Tracks changes made to claims, flight delays, and other entities.
- Logs who made the change, what was changed, and when.

Relationships:

- A claim can have multiple updates logged (Claims table).

Proof of Normalization:

1st Normal Form (1NF) - Atomicity Ensured

- Flights Table:
  - Each flight has separate columns for scheduled_departure and scheduled_arrival.
  - No repeating groups (e.g., multiple airlines for one flight).
- Claims Table:
  - Each claim is tied to one passenger and one flight (ensures atomicity).
  - No multiple claim statuses in one row; instead, we track changes in ChangeLogs.

2nd Normal Form (2NF) - No Partial Dependencies

- Breaking down composite dependencies:
  - flight_id is the only primary key for Flights, ensuring no partial dependencies.
  - Claims links to Passengers via passenger_id, removing dependency on flight details.
  - Delays table separates delay tracking from Flights, reducing redundancy.

3rd Normal Form (3NF) - No Transitive Dependencies

- Removing indirect dependencies:
  - Passenger information (name, email) is stored separately in Passengers.
  - Flight status history (status, updated_at) is stored in FlightStatusHistory, not in Flights, avoiding redundancy.
  - ChangeLogs table keeps a history of all changes instead of modifying existing records, improving data integrity.

Ensuring Data Accuracy

To maintain data accuracy and integrity, the system enforces strict validation rules, constraints, and regular updates.

A. Data Validation & Constraints

1. Foreign Key Constraints – Ensures valid relationships (e.g., a claim must have a valid flight).
2. Unique Constraints – Prevents duplicate records (e.g., unique ticket numbers in Claims).
3. Check Constraints – Ensures data validity (e.g., delay_minutes ≥ 0).
4. Trigger-Based Validation – Automatically prevents ineligible claims (e.g., only flights delayed > 2 hours qualify).

B. Data Integrity & Consistency

1. Regular Data Syncing – Fetches real-time flight data from APIs like OpenSky & FlightAware.
2. Data Deduplication – Identifies and removes duplicate flight records using stored procedures.
3. Audit Logging – Tracks all data modifications (who, when, and what changed) for accountability.

Ensuring Scalability

To handle large-scale flight data efficiently, the system uses indexing, partitioning, and caching for performance optimization.

A. Indexing for Faster Queries

1. Indexes on Foreign Keys – Speeds up lookups in FlightStatus, Delays, and Claims.
2. Composite Indexes – Optimizes flight searches, arrivals, and claims tracking using multi-column indexes.

B. Partitioning Large Tables

1. Time-Based Partitioning – Splits the Flights table by month/year for faster queries.
2. Geographical Partitioning – Groups flights by region to improve lookup efficiency.

C. Caching for High-Performance Reads

1. Redis Cache – Stores frequently accessed airport, airline, and recent flight data.
2. API Rate Limiting – Uses pagination & throttling to prevent system overload.

2. Data Collection Strategy

To build a comprehensive database of European airports and real-time flight data, we will use multiple data sources:

1. Static airport data (one-time collection, periodically updated).
2. Real-time flight data (continuous updates via APIs).
3. Passenger claim data (processed from delayed flights).

---

1. Open Data Sources
    - OpenStreetMap (OSM) Overpass API → Query airport data programmatically.
    - OurAirports.com (CSV dataset) → Free global airport database with ICAO & IATA codes.
    - Eurocontrol AIP (Aeronautical Information Publication) → Official European airport data.
2. Scraping Airport Data
    - Use ChatGPT API for intelligent web scraping.
    - BeautifulSoup/Scrapy to scrape airport data from Wikipedia or airline websites.

Storing Airport Data

- Store data in a relational database (Airports table) for structured queries.
- Index IATA & ICAO codes for fast lookups.
- Update quarterly using official datasets.

Methods for Collecting Real-time Flight Data

1. Third-Party APIs
    - FlightAware - Best for live tracking with delays & cancellations.
    - AviationStack API - Live flight statuses, delays, and schedules.
    - OpenSky Network API - Real-time ADS-B flight data but limited details.
    - Eurocontrol B2B API (Restricted) - Government-approved, high-quality European data.
2. Automated Web Scraping (Backup Method)

    - Use ChatGPT API for intelligent scraping of airline websites (if API access is unavailable).
    - Scrapy/BeautifulSoup to fetch live status from airline sites.
3. Flight Data via ADS-B Receivers (Advanced Option)
    - Set up Raspberry Pi + ADS-B receiver to collect local flight data directly from aircraft signals.
    - Share data with OpenSky to get better access to their network.

1. Handling Missing Data

Prevention at Insertion – Enforce `NOT NULL` & `CHECK` constraints, validate API responses, and retry requests.
Fallback Sources – If primary data is incomplete, fetch from secondary APIs (OurAirports, Wikipedia, Eurocontrol).
Estimations & Reprocessing – Use historical trends for missing values and flag records as "Pending Update" for later reprocessing.

2. Handling Delayed Data (Latency Issues)

Multi-Source Updates – If FlightAware fails, fetch data from OpenSky or ADS-B receivers.
Cache Latest Available Data – Show the last known flight status if real-time data is delayed.
User Notifications – Inform passengers: *"Claim under review. Verifying flight details."*

3. Handling Inconsistent Data

Cross-Validation – Compare flight status across multiple APIs & resolve conflicts using majority consensus.
Automated Rules – Prevent logical errors (e.g., `arrival_time < departure_time` → Reject).
ML-Based Anomaly Detection – Train a model to flag unexpected delays or incorrect statuses.

3. Flight Monitoring and Claim Identification

System Architecture

Data Sources: APIs (AviationStack, FlightAware, OpenSky), Airport Feeds
Processing Engine: Node.js
Database: PostgreSQL
Monitoring & Alerts: Notifications via email/SMS

Steps in the Flight Monitoring System

1. Real-time Flight Data Fetching

- Fetch live flight details using APIs (AviationStack, FlightAware, OpenSky)
- Store scheduled departure time and actual departure time

2. Delay Calculation & Flagging

- Compute delay_minutes = actual_departure - scheduled_departure
- If delay_minutes > 120, mark flight as “Eligible for Claim”

3. Database Storage & Updates

- Update the Flights table with status and delay time
- Maintain a FlightStatus history for tracking

4. Passenger Notification System

- Identify affected passengers (based on ticket data)
- Send email/SMS alerts with refund claim links

5. Analytics Dashboard

- Web dashboard for tracking frequent delays, airlines, routes
- Data visualization for performance insights

Real-Time Monitoring

1. API Fetching (Every 5-10 mins) - Get latest flight data
2. Compare Scheduled vs. Actual Departure Time
3. Identify Delays > 120 mins
4. Mark Flights as ‘Eligible for Claim’
5. Send Notifications to Passengers

We can use Node.js + Cron Jobs (API calls every 5-10 minutes)

Update Workflow

1. Fetch latest flight data from APIs (AviationStack, OpenSky, FlightAware, etc.)
2. Compare the new data with the existing database records
3. Update the database only if changes are detected
4. Trigger alerts for status changes (e.g., flight delay > 2 hours)

Real-Time Alerts & Notifications

When delay_minutes > 120, trigger **email/SMS alert.** We can use **nodemailer** module of Node.js for email alerts.

4. Future API Development

To provide daily flight data across Europe, the API must be:
Scalable – Handle large traffic & real-time updates
Reliable – Ensure data accuracy & uptime
Secure – Protect against abuse & unauthorized access

We can use RESTful API For web & mobile integration

Tech Stack (Since not mentioned a specific language in the task I used node.js for this project, I can also use python)

Backend : Node.js + Express.js → Fast & scalable API development / python
Database :

PostgreSQL → Relational DB for structured flight data
Redis → Caching to improve API response time

API Security & Authentication

JWT (JSON Web Tokens) - Secure user authentication
OAuth 2.0 - For third-party integrations (airlines, travel agencies)
Rate Limiting (Express-rate-limit) - To prevent abuse

Deployment & Monitoring

CI/CD (GitHub Actions, Jenkins) - Automated deployment

API Features & Endpoints

1. Retrieve Airport Data
    - GET /api/airports - Get all airports
    - GET /api/airports/{iata_code} - Get specific airport details
2. Retrieve Flight Data
    - GET /api/flights?date=YYYY-MM-DD - Get all flights for a date
    - GET /api/flights/{flight_number} - Get flight details
    - GET /api/flights/delayed?min_delay=120 - Get all delayed flights (>2 hrs)
3. User Authentication & Claims
    - POST /api/user/register - Passenger signup
    - POST /api/claims/submit - Submit refund claim
4. Admin & Airline Dashboard
    - POST /api/admin/update-flight - Update flight status (Admin only)
    - GET /api/admin/statistics - Get delay trends

And etc etc.

Scalability & Performance Optimization

Load Balancer (Nginx / express-rate-limit) - Distributes API traffic
Database Optimization (PostgreSQL + Indexing) - Faster queries
Caching (Redis) → Reduces database calls

Deployment Strategy

AWS or Google Cloud
Containerized (Docker + Kubernetes) - Ensures scalability & fault tolerance
CI/CD (GitHub Actions, Jenkins) - Automated deployment

API Security Measures

Authentication & Authorization: Use JWT for user authentication, OAuth 2.0 for third-party access, and RBAC to control user permissions.
Data Protection: Encrypt data using SSL/TLS (HTTPS) for secure transmission and database encryption for sensitive information.
API Rate Limiting & Protection: Use Express-rate-limit rate limiting to prevent DDoS attacks, and input validation to block SQL injection.

API Availability Strategies

Load Balancing & Auto-Scaling: Deploy Nginx/AWS ALB for traffic distribution and Kubernetes/AWS ECS to handle spikes.

Failover & Disaster Recovery: Ensure database replication, multi-region deployment, and automated backups for reliability.

API Reliability & Performance Optimization

Caching & Query Optimization: Use Redis for caching and indexed PostgreSQL queries to improve response times.
Asynchronous Processing: handling real-time updates and background jobs for heavy processing.
Monitoring & Alerting: Track API health, analyze logs stack.

Part 2 : Theoretical Tasks

In 1st and 2nd exercise I used online PostgreSQL compiler,
Ref : https://onecompiler.com/postgresql

1. Airport Database Creation

```sql
-- Airports Table Creation
CREATE TABLE Airports (
   airport_id SERIAL PRIMARY KEY,
   name VARCHAR(255) NOT NULL,
   iata_code CHAR(3) UNIQUE NOT NULL,
   icao_code CHAR(4) UNIQUE NOT NULL,
   country VARCHAR(100) NOT NULL,
   city VARCHAR(100) NOT NULL,
   latitude DECIMAL(9,6) NOT NULL,
   longitude DECIMAL(9,6) NOT NULL
);
```

```sql
-- Flights Table Creation
CREATE TABLE IF NOT EXISTS Flights (
   flight_id SERIAL PRIMARY KEY,
   flight_number VARCHAR(10) UNIQUE NOT NULL,
   airline VARCHAR(100) NOT NULL,
   departure_airport CHAR(3) NOT NULL,
   arrival_airport CHAR(3) NOT NULL,
   scheduled_departure TIMESTAMP NOT NULL,
   actual_departure TIMESTAMP,
   delay_minutes INT DEFAULT 0 CHECK (delay_minutes >= 0),
   status VARCHAR(20) CHECK (status IN ('On-Time', 'Delayed', 'Cancelled')) NOT NULL,
   FOREIGN KEY (departure_airport) REFERENCES Airports(iata_code) ON DELETE CASCADE,
   FOREIGN KEY (arrival_airport) REFERENCES Airports(iata_code) ON DELETE CASCADE
);
```

2. Data Insertion and Querying

```
-- Sample Data Insertion in Airports Table
INSERT INTO Airports (name, iata_code, icao_code, country, city, latitude, longitude)
VALUES
('London Heathrow Airport', 'LHR', 'EGLL', 'United Kingdom', 'London', 51.4700,
-0.4543),
('Charles de Gaulle Airport', 'CDG', 'LFPG', 'France', 'Paris', 49.0097, 2.5479),
('Frankfurt Airport', 'FRA', 'EDDF', 'Germany', 'Frankfurt', 50.0379, 8.5622),
('Amsterdam Schiphol Airport', 'AMS', 'EHAM', 'Netherlands', 'Amsterdam', 52.3086,
4.7639),
('Madrid Barajas Airport', 'MAD', 'LEMD', 'Spain', 'Madrid', 40.4719, -3.5626);


-- Sample Data Insertion in Flights Table
INSERT INTO Flights (flight_number, airline, departure_airport, arrival_airport,
scheduled_departure, actual_departure, delay_minutes, status) VALUES
-- On-Time Flights
('LH100', 'Lufthansa', 'FRA', 'LHR', '2025-03-20 08:00:00', '2025-03-20 08:00:00', 0,
'On-Time'),
('AF200', 'Air France', 'MAD', 'CDG', '2025-03-20 09:30:00', '2025-03-20 09:30:00', 0,
'On-Time'),
('BA300', 'British Airways', 'AMS', 'LHR', '2025-03-20 10:15:00', '2025-03-20
10:15:00', 0, 'On-Time'),
('KL400', 'KLM', 'CDG', 'AMS', '2025-03-20 12:45:00', '2025-03-20 12:45:00', 0,
'On-Time'),
('IB500', 'Iberia', 'FRA', 'MAD', '2025-03-20 14:00:00', '2025-03-20 14:00:00', 0,
'On-Time'),

-- Delayed Flights
('LH600', 'Lufthansa', 'AMS', 'FRA', '2025-03-20 15:30:00', '2025-03-20 18:00:00',
150, 'Delayed'),
('AF700', 'Air France', 'LHR', 'AMS', '2025-03-20 17:00:00', '2025-03-20 19:30:00',
150, 'Delayed'),
('BA800', 'British Airways', 'MAD', 'FRA', '2025-03-20 20:00:00', '2025-03-20
22:45:00', 165, 'Delayed'),
```

```
('KL900', 'KLM', 'CDG', 'AMS', '2025-03-20 22:00:00', '2025-03-21 01:30:00', 210,
'Delayed'),
('IB1000', 'Iberia', 'FRA', 'MAD', '2025-03-21 06:00:00', '2025-03-21 09:00:00', 180,
'Delayed');
```

```
 airport_id |           name            | iata_code | icao_code |    country     |   city    | latitude  | longitude
------------+---------------------------+-----------+-----------+----------------+-----------+-----------+-----------
          1 | London Heathrow Airport   | LHR       | EGLL      | United Kingdom | London    | 51.470000 | -0.454300
          2 | Charles de Gaulle Airport | CDG       | LFPG      | France         | Paris     | 49.009700 |  2.547900
          3 | Frankfurt Airport         | FRA       | EDDF      | Germany        | Frankfurt | 50.037900 |  8.562200
          4 | Amsterdam Schiphol Airport| AMS       | EHAM      | Netherlands    | Amsterdam | 52.308600 |  4.763900
          5 | Madrid Barajas Airport    | MAD       | LEMD      | Spain          | Madrid    | 40.471900 | -3.562600
(5 rows)
```

```
 flight_id | flight_number |     airline     | departure_airport | arrival_airport | scheduled_departure |  actual_departure   | delay_minutes | status
-----------+---------------+-----------------+-------------------+-----------------+---------------------+---------------------+---------------+---------
         1 | LH100         | Lufthansa       | FRA               | LHR             | 2025-03-20 08:00:00 | 2025-03-20 08:00:00 |             0 | On-Time
         2 | AF200         | Air France      | MAD               | CDG             | 2025-03-20 09:30:00 | 2025-03-20 09:30:00 |             0 | On-Time
         3 | BA300         | British Airways | AMS               | LHR             | 2025-03-20 10:15:00 | 2025-03-20 10:15:00 |             0 | On-Time
         4 | KL400         | KLM             | CDG               | AMS             | 2025-03-20 12:45:00 | 2025-03-20 12:45:00 |             0 | On-Time
         5 | IB500         | Iberia          | FRA               | MAD             | 2025-03-20 14:00:00 | 2025-03-20 14:00:00 |             0 | On-Time
         6 | LH600         | Lufthansa       | AMS               | FRA             | 2025-03-20 15:30:00 | 2025-03-20 18:00:00 |           150 | Delayed
         7 | AF700         | Air France      | LHR               | AMS             | 2025-03-20 17:00:00 | 2025-03-20 19:30:00 |           150 | Delayed
         8 | BA800         | British Airways | MAD               | FRA             | 2025-03-20 20:00:00 | 2025-03-20 22:45:00 |           165 | Delayed
         9 | KL900         | KLM             | CDG               | AMS             | 2025-03-20 22:00:00 | 2025-03-21 01:30:00 |           210 | Delayed
        10 | IB1000        | Iberia          | FRA               | MAD             | 2025-03-21 06:00:00 | 2025-03-21 09:00:00 |           180 | Delayed
(10 rows)
```

1. **Retrieve all flights from a specific airport.**

```
-- q-1
SELECT f.flight_id, f.flight_number, f.airline,
       f.departure_airport, f.arrival_airport,
       f.scheduled_departure, f.actual_departure,
       f.delay_minutes, f.status
FROM Flights f
WHERE f.departure_airport = 'FRA' OR f.arrival_airport = 'FRA';
```

```
 flight_id | flight_number |     airline     | departure_airport | arrival_airport | scheduled_departure |  actual_departure   | delay_minutes | status
-----------+---------------+-----------------+-------------------+-----------------+---------------------+---------------------+---------------+---------
         1 | LH100         | Lufthansa       | FRA               | LHR             | 2025-03-20 08:00:00 | 2025-03-20 08:00:00 |             0 | On-Time
         5 | IB500         | Iberia          | FRA               | MAD             | 2025-03-20 14:00:00 | 2025-03-20 14:00:00 |             0 | On-Time
         6 | LH600         | Lufthansa       | AMS               | FRA             | 2025-03-20 15:30:00 | 2025-03-20 18:00:00 |           150 | Delayed
         8 | BA800         | British Airways | MAD               | FRA             | 2025-03-20 20:00:00 | 2025-03-20 22:45:00 |           165 | Delayed
        10 | IB1000        | Iberia          | FRA               | MAD             | 2025-03-21 06:00:00 | 2025-03-21 09:00:00 |           180 | Delayed
(5 rows)
```

2. **Identify flights delayed by more than 2 hours.**

```
-- q-2
SELECT f.flight_id, f.flight_number, f.airline,
       f.departure_airport, f.arrival_airport,
       f.scheduled_departure, f.actual_departure,
```

```sql
      f.delay_minutes, f.status
FROM Flights f
WHERE f.delay_minutes > 120
ORDER BY f.delay_minutes DESC;
```

```
 flight_id | flight_number |     airline      | departure_airport | arrival_airport | scheduled_departure |  actual_departure   | delay_minutes | status
-----------+---------------+------------------+-------------------+-----------------+---------------------+---------------------+---------------+---------
         9 | KL900         | KLM              | CDG               | AMS             | 2025-03-20 22:00:00 | 2025-03-21 01:30:00 |           210 | Delayed
        10 | IB1000        | Iberia           | FRA               | MAD             | 2025-03-21 06:00:00 | 2025-03-21 09:00:00 |           180 | Delayed
         8 | BA800         | British Airways  | MAD               | FRA             | 2025-03-20 20:00:00 | 2025-03-20 22:45:00 |           165 | Delayed
         6 | LH600         | Lufthansa        | AMS               | FRA             | 2025-03-20 15:30:00 | 2025-03-20 18:00:00 |           150 | Delayed
         7 | AF700         | Air France       | LHR               | AMS             | 2025-03-20 17:00:00 | 2025-03-20 19:30:00 |           150 | Delayed
(5 rows)
```

3. Fetch flight details using the flight number.

```sql
-- q-3
SELECT
   f.flight_id,
   f.flight_number,
   f.airline,

   -- Departure Airport Details
   a1.name AS departure_airport,
   a1.city AS departure_city,
   a1.country AS departure_country,
   a1.iata_code AS departure_iata,
   a1.icao_code AS departure_icao,
   a1.latitude AS departure_latitude,
   a1.longitude AS departure_longitude,

   -- Arrival Airport Details
   a2.name AS arrival_airport,
   a2.city AS arrival_city,
   a2.country AS arrival_country,
   a2.iata_code AS arrival_iata,
   a2.icao_code AS arrival_icao,
   a2.latitude AS arrival_latitude,
   a2.longitude AS arrival_longitude,

   -- Flight Details
   f.scheduled_departure,
   f.actual_departure,
   f.delay_minutes,
   f.status
FROM Flights f
JOIN Airports a1 ON f.departure_airport = a1.iata_code
```

```
JOIN Airports a2 ON f.arrival_airport = a2.iata_code
WHERE f.flight_number = 'LH600';
```

```
 flight_id | flight_number |  airline  |     departure_airport      | departure_city | departure_country | departure_iata | departure_icao | departure_latitude | departure_longitude |  arrival_airport  | arrival_city | arrival_country | arrival_iata | arrival_icao | arrival_latitude | arrival_longitude | scheduled_departure |  actual_departure   | delay_minutes | status
-----------+---------------+-----------+----------------------------+----------------+-------------------+----------------+----------------+--------------------+---------------------+-------------------+--------------+-----------------+--------------+--------------+------------------+-------------------+---------------------+---------------------+---------------+---------
         6 | LH600         | Lufthansa | Amsterdam Schiphol Airport | Amsterdam      | Netherlands       | AMS            | EHAM           |          52.308600 |           4.763900 | Frankfurt Airport | Frankfurt    | Germany         | FRA          | EDDF         |         50.037600 |          8.562200 | 2025-03-20 15:30:00 | 2025-03-20 18:00:00 |           150 | Delayed
(1 row)
```

3. Data Collection Simulation

For this exercise i Used pgAdmin4 for postgreSQL database.
And used node.js for API as there is no mention of specific language in project description. Other option can be python too.

1. Require necessary Packages

Firstly, we need some packages listed below:

axios : Used for making API requests.
pg : PostgreSQL client for inserting flight data into the database.
dotenv : Loads Environment vars like API key, Database url from an environment file.

We have to run command on terminal:

npm install axios pg dotenv

2. `.env` File (For Storing API Keys, Database URL)

```
AVIATIONSTACK_API_KEY=your_api_key_here
DATABASE_URL=postgresql://username:password@localhost:5432/your_database
```

I have replaced `your_api_key_here` with your AviationStack API Key and `DATABASE_URL` with PostgreSQL database credentials.

3. Node.js Script to Fetch and Store Real-Time Flight Data

```
require('dotenv').config();
const axios = require('axios');
const { Client } = require('pg');
```

```
// PostgreSQL connection setup
const client = new Client({
   connectionString: process.env.DATABASE_URL,
});
client.connect();


// Function to fetch real-time flight data
async function fetchFlightData() {
   try {


       // Fetch flight data from the AviationStack API
       const apiKey = process.env.AVIATIONSTACK_API_KEY;
       const url
=`http://api.aviationstack.com/v1/flights?access_key=${apiKey}&limit=5`;

       // Get flight data from the API
       const response = await axios.get(url);
       const flights = response.data.data;

       for (const flight of flights) {
           if (!flight.departure || !flight.arrival) continue; // Skip missing data

           // Insert or update flight data in the database
           const query = `
               INSERT INTO flights (flight_number, departure_airport, arrival_airport,
scheduled_departure, actual_departure, status, delay_minutes)
               VALUES ($1, $2, $3, $4, $5, $6, $7)
               ON CONFLICT (flight_number) DO UPDATE
               SET actual_departure = EXCLUDED.actual_departure, status =
EXCLUDED.status, delay_minutes = EXCLUDED.delay_minutes;
           `;

           // Flight data values
           const values = [
               flight.flight.iata || flight.flight.icao,
               flight.departure.iata,
               flight.arrival.iata,
               flight.departure.estimated || flight.departure.scheduled,
               flight.departure.actual || null,
```

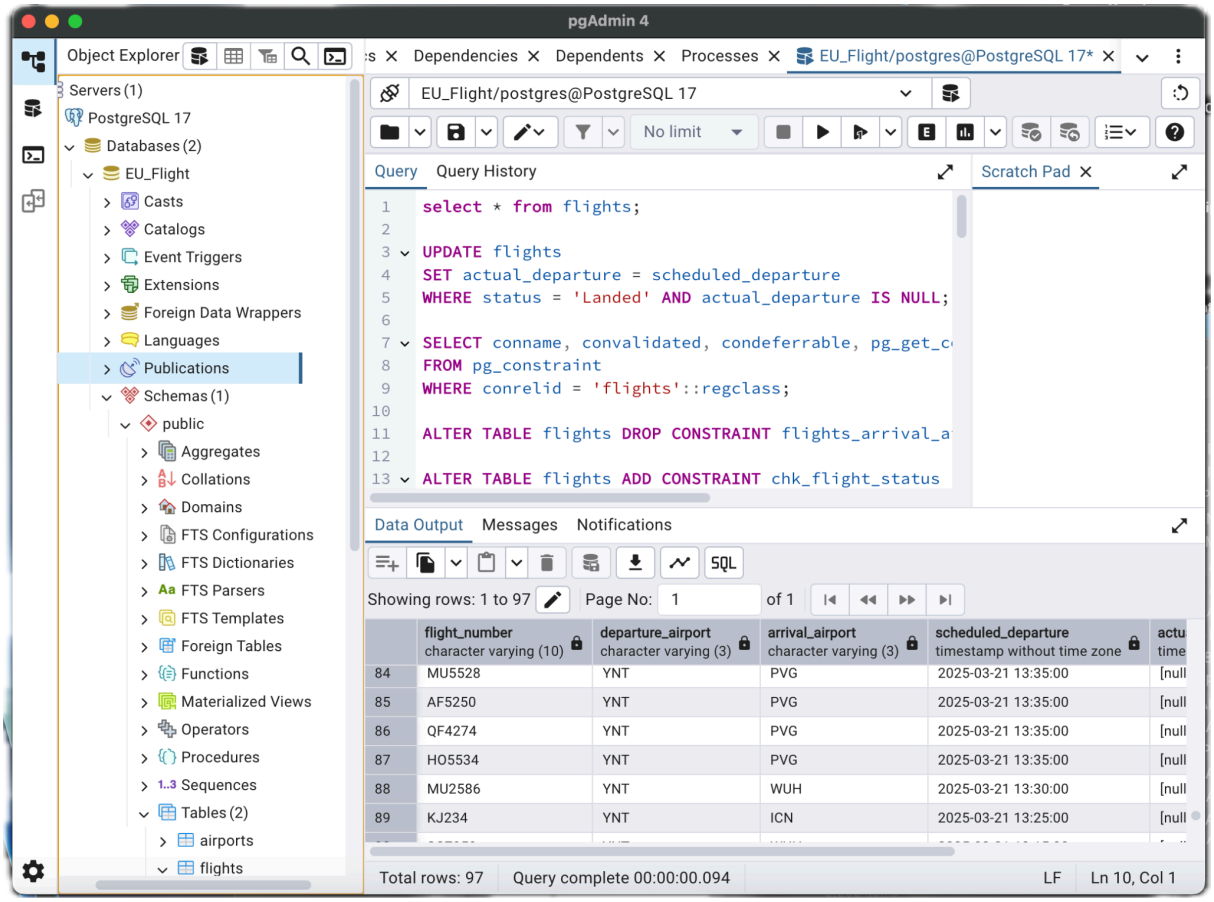
```
                flight.flight_status,
                flight.departure.delay || 0,
            ];

            // Store flight data in the database
            await client.query(query, values);
            console.log("Flight data successfully stored in database.");
        }
    } catch (error) {
        console.error("Error fetching flight data:", error.message);
    }
}

// Fetch data every 10 minutes
setInterval(fetchFlightData, 600000);
fetchFlightData();
```

```
RP235@Rudras-Mac EU_Flight % node fetchData.js
Fetching real-time flight data...
Flight data successfully stored in database.
RP235@Rudras-Mac EU_Flight % node fetchData.js
Fetching real-time flight data...
Flight data successfully stored in database.
RP235@Rudras-Mac EU_Flight %
```

Run Testcases ⊗ 0 ⚠ 0


pgAdmin 4
Object Explorer
Servers (1)
PostgreSQL 17
Databases (2)
EU_Flight
Casts
Catalogs
Event Triggers
Extensions
Foreign Data Wrappers
Languages
Publications
Schemas (1)
public
Aggregates
Collations
Domains
FTS Configurations
FTS Dictionaries
FTS Parsers
FTS Templates
Foreign Tables
Functions
Materialized Views
Operators
Procedures
Sequences
Tables (2)
airports
flights
Dependencies
Dependents
Processes
EU_Flight/postgres@PostgreSQL 17*
EU_Flight/postgres@PostgreSQL 17
No limit
Query
Query History
Scratch Pad
select * from flights;
UPDATE flights
SET actual_departure = scheduled_departure
WHERE status = 'Landed' AND actual_departure IS NULL;
SELECT conname, convalidated, condeferrable, pg_get_c
FROM pg_constraint
WHERE conrelid = 'flights'::regclass;
ALTER TABLE flights DROP CONSTRAINT flights_arrival_a
ALTER TABLE flights ADD CONSTRAINT chk_flight_status
Data Output
Messages
Notifications
SQL
Showing rows: 1 to 97
Page No: 1
of 1
flight_number character varying (10)
departure_airport character varying (3)
arrival_airport character varying (3)
scheduled_departure timestamp without time zone
84 MU5528 YNT PVG 2025-03-21 13:35:00
85 AF5250 YNT PVG 2025-03-21 13:35:00
86 QF4274 YNT PVG 2025-03-21 13:35:00
87 HO5534 YNT PVG 2025-03-21 13:35:00
88 MU2586 YNT WUH 2025-03-21 13:30:00
89 KJ234 YNT ICN 2025-03-21 13:25:00
Total rows: 97
Query complete 00:00:00.094
LF
Ln 10, Col 1

```sql
-- Create the "flights" table
CREATE TABLE flights (
    flight_id SERIAL PRIMARY KEY,
    flight_number VARCHAR(10) UNIQUE NOT NULL,
    departure_airport VARCHAR(3),
    arrival_airport VARCHAR(3),
    scheduled_departure TIMESTAMP NOT NULL,
    actual_departure TIMESTAMP,
    status VARCHAR(20) CHECK (status IN ('On Time', 'Delayed', 'Cancelled')),
    delay_minutes INT DEFAULT 0
);
```

Data Output | Messages | Notifications

Showing rows

	flight_number character varying (10)	departure_airport character varying (3)	arrival_airport character varying (3)	scheduled_departure timestamp without time zone	actua times
84	MU5528	YNT	PVG	2025-03-21 13:35:00	[null]
85	AF5250	YNT	PVG	2025-03-21 13:35:00	[null]
86	QF4274	YNT	PVG	2025-03-21 13:35:00	[null]
87	HO5534	YNT	PVG	2025-03-21 13:35:00	[null]
88	MU2586	YNT	WUH	2025-03-21 13:30:00	[null]
89	KJ234	YNT	ICN	2025-03-21 13:25:00	[null]
90	SC7959	YNT	WUH	2025-03-21 13:15:00	[null]
91	ZH2781	YNT	WUH	2025-03-21 13:15:00	[null]
92	TV3325	YNT	WUH	2025-03-21 13:15:00	[null]
93	G56595	YNT	WUH	2025-03-21 13:15:00	[null]
94	FR5174	FMM	BNX	2025-03-21 07:05:00	[null]
95	FR8552	FMM	PMI	2025-03-21 06:10:00	[null]

The End