### **Part 1: Theory Questions – Detailed Answers**

## **1. Database Design**

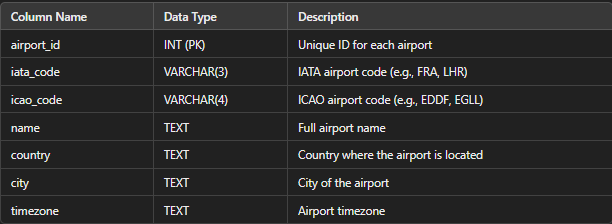
To efficiently store and manage data for **European airports, flight schedules, flight statuses, and delays**, we need a well-structured relational database schema. Below is a proposed design:

### **Tables & Relationships**

1. **Airports Table** – Stores airport details.
2. **Airlines Table** – Stores airline information.
3. **Flights Table** – Stores individual flight details.
4. **FlightStatus Table** – Stores real-time flight status updates.

### **Database Schema Design**

#### **1. Airports Table (Stores airport metadata)**



#### **2. Airlines Table (Stores airline metadata)**

#### **3. Flights Table (Stores flight information)**

#### **4. FlightStatus Table (Stores real-time updates)**

### **Ensuring Data Accuracy & Scalability**

* **Normalization**: The schema follows the third **normal form (3NF)** to remove redundancy and ensure consistency.
* **Indexes**: Adding **indexes on frequently queried columns** (e.g., iata\_code, flight\_number) to improve performance.
* **Partitioning**: Partitioning data based on **date or region** for scalability in high-traffic conditions.
* **Data Validation**: Constraints like **NOT NULL, UNIQUE, and FOREIGN KEYS** ensure data integrity.
* **Caching & Replication**: Using **Redis/Memcached** for fast reads and database replication for reliability.

## **2. Data Collection Strategy**

### **Collecting & Storing European Airport Data**

* **Source**:  
  + Open databases like **OpenFlights.org**, **OurAirports.com**, or **EuroControl ADB**.
  + **Scraping from official aviation authorities** like IATA or ICAO.
  + **APIs**: AviationStack, FlightAware, OpenSky Network.
* **Storage**:  
  + Data is **stored in the Airports table**.
  + Regular **updates every 24 hours** to account for new airports.

### **Real-time Flight Data Collection**

### **Preferred Approach**:

* Use **AviationStack API** for general flight data.
* Use **ADS-B data** for real-time tracking.

### **Handling Missing, Delayed, or Inconsistent Data**

* **Missing Data**: Fill with historical averages or estimate using machine learning.
* **Delayed Data**: Use **moving averages** to smooth real-time estimates.
* **Inconsistent Data**: Cross-validate between **multiple sources** (e.g., API vs. ADS-B data).

## **3. Flight Monitoring & Claim Identification**

**System to Flag Delays Over 2 Hours**

### **Technical Approach**

1️⃣ **Real-time Monitoring**

* **Scheduled Task**: Fetch flight data every 10 minutes.
* **Compare estimated\_arrival vs scheduled\_arrival**.
* **If delay > 120 minutes → Flag flight for alerts**.

2️⃣ **Data Updates**

* Store updates in **FlightStatus table**.
* **Index flight\_number + timestamp** for quick lookups.

3️⃣ **Alerting System**

* **Email/SMS alerts** to notify users of delays.
* **Webhook integration** for airline claim processing.

### **Efficient Data Storage & Management**

* **Use PostgreSQL/MySQL with partitioning** for fast queries.
* **Log-based updates** instead of full database refreshes.
* **Indexing on flight status + timestamps** for fast retrieval.

## **4. Future API Development**

**Scalable API for European Flight Data**

### **API Design**

* **Endpoints**:  
  + /flights?airport=FRA&status=delayed
  + /flights/{flight\_number}
  + /airports/{iata\_code}
* **Response Format (JSON)**:



### **Security, Availability & Reliability**

1️⃣ **Security**

* **Rate Limiting** (e.g., max 1000 requests/hour).
* **API Key Authentication**.
* **HTTPS Encryption**.

2️⃣ **Availability**

* **Load Balancing** (NGINX/Gunicorn).
* **Auto-scaling** with Kubernetes/Docker.

3️⃣ **Reliability**

* **Replication & Backups** to prevent data loss.
* **Monitoring with Prometheus/Grafana**.