**Part 1: Theory Questions**

# 1. Database Design

**Database Schema Design:**

The database schema consists of the following tables:

1. **Airports Table**:
    - AirportID (Primary Key)
    - AirportName
    - IATACode
    - ICAOCode
    - Country
    - City
    - Latitude
    - Longitude
2. **Airlines Table**:
    - AirlineID (Primary Key)
    - AirlineName
    - IATACode
    - ICAOCode
3. **Flights Table**:
    - FlightID (Primary Key)
    - FlightNumber
    - AirlineID (Foreign Key referencing Airlines)
    - DepartureAirportID (Foreign Key referencing Airports)
    - ArrivalAirportID (Foreign Key referencing Airports)
    - ScheduledDepartureTime
    - ScheduledArrivalTime
    - ActualDepartureTime
    - ActualArrivalTime
    - Status (e.g., On-Time, Delayed, Cancelled)
4. **FlightStatus Table**:
    - StatusID (Primary Key)
    - FlightID (Foreign Key referencing Flights)
    - DelayDuration (in minutes)
    - StatusUpdateTime

**Relationships:**

- Airports and Flights are related through DepartureAirportID and ArrivalAirportID.
- Airlines and Flights are related through AirlineID.
- Flights and FlightStatus are related through FlightID.

**Ensuring Data Accuracy and Scalability:**

- Data Accuracy: Implement constraints (e.g., NOT NULL, UNIQUE) and foreign key relationships to maintain referential integrity. Use validation checks during data insertion.
- Scalability: Use indexing on frequently queried columns (e.g., FlightNumber, AirportID). Partition large tables (e.g., Flights) by date or region to improve query performance.

## 2. Data Collection Strategy

**Collecting Airport Information:**

- Use publicly available datasets such as OpenFlights or OurAirports to gather information on European airports, including IATA/ICAO codes, country, city, and coordinates.
- Alternatively, scrape data from aviation websites using tools like BeautifulSoup or Scrapy.

**Collecting Real-Time Flight Data:**

- Use third-party APIs like AviationStack, FlightAware, or FlightRadar24 to fetch real-time flight data.
- Alternatively, use ADB (AeroDataBox) or ChatGPT API for scraping and extracting flight data from aviation websites.

**Handling Missing or Inconsistent Data:**

- Implement data validation rules to ensure that only complete and accurate data is stored.
- Use data cleaning techniques (e.g., removing duplicates, filling missing values with defaults) before inserting data into the database.
- For delayed or inconsistent data, implement retry mechanisms when fetching data from APIs.

## 3. Flight Monitoring and Claim Identification

**Proposed System:**

- A real-time monitoring system that fetches flight data from APIs at regular intervals (e.g., every 5 minutes).
- The system compares the ScheduledDepartureTime and ActualDepartureTime to calculate delays.
- Flights delayed by more than 2 hours are flagged and stored in a separate table for further processing (e.g., notifying passengers).

**Technical Approach:**

- Use a message queue (e.g., RabbitMQ, Kafka) to handle real-time data updates.

- Implement cron jobs or scheduled tasks to periodically fetch and update flight data.
- Use push notifications or email alerts to notify passengers about delays.

**Data Storage and Management:**

- Store historical flight data in a partitioned database to manage large volumes efficiently.
- Use cloud storage (e.g., AWS S3, Google Cloud Storage) for archiving old data.

## 4. Future API Development

**API Design:**

- Develop a RESTful API with endpoints for retrieving flight data, airport information, and delay notifications.
- Use Swagger or OpenAPI for API documentation.

**Ensuring API Security, Availability, and Reliability:**

- Implement authentication (e.g., API keys, OAuth) to secure the API.
- Use rate limiting to prevent abuse.
- Deploy the API on a scalable cloud platform (e.g., AWS, Azure) with load balancing and auto-scaling to ensure high availability.
- Monitor API performance using tools like Prometheus or New Relic.