Comprehensive Technical Test for Data Engineer at Samay

Lung Sound Data Pipeline and ETL Challenge

Objective:

Design and implement a robust data pipeline for processing, organizing, and storing lung sound recordings and associated metadata from multiple sources. You will work with disorganized medical sensor data to create a production-ready data infrastructure.

Background:

The respiratory health department has collected lung sound recordings from various hospitals and clinics. The data is currently scattered across different formats, naming conventions, and storage systems. Your task is to create a unified, scalable data pipeline that can handle this heterogeneous data.

Dataset Description:

You will receive:

- **Raw audio files** with inconsistent naming conventions
- **CSV files** with patient metadata (some with missing columns)
- **JSON files** from different stethoscope devices with varying schemas
- **Text files** with physician notes in an unstructured format

- **Excel sheets** with diagnostic information

1. First Part: Data Discovery and Assessment

A. Data Audit

a. Connect to the provided data sources
b. Identify all data formats and schemas present
c. Document data quality issues:
   i. Missing values
   ii. Inconsistent formats
   iii. Duplicate records
   iv. Naming convention violations

B. Create Data Catalog

Build a comprehensive data catalog that includes:

```
{
        "data_source": "source_name",
        "file_type": "wav/csv/json",
        "schema": {},
        "quality_metrics": {
                "completeness": 0.0,
                "consistency": 0.0,
                "validity": 0.0
        },
        "volume": "size_in_mb",
        "update_frequency": "daily/weekly"
}
```

2. Second part: Data Pipeline Design

A. Schema Standardization

Design and implement a unified schema for the lung sound data:

- **Audio Metadata Schema:**

```
CREATE TABLE audio_recordings (
        recording_id VARCHAR(50) PRIMARY KEY,
        patient_id VARCHAR(50),
```

```
    timestamp TIMESTAMP,
    duration_seconds FLOAT,
    sample_rate INTEGER,
    bit_depth INTEGER,
    filter_mode VARCHAR(20), -- Bell/Diaphragm/Extended
    recording_location VARCHAR(50),
    file_path VARCHAR(255),
    checksum VARCHAR(64),
    processing_status VARCHAR(20)
);
```

- **Patient Information Schema:**

```
CREATE TABLE patient_demographics (
    patient_id VARCHAR(50) PRIMARY KEY,
    age INTEGER,
    gender VARCHAR(10),
    diagnosis_codes TEXT[],
    recording_date DATE,
    hospital_id VARCHAR(50)
);
```

B. Create a Python-based ETL pipeline that

a. Extraction Layer:

i. Connects to multiple data sources
ii. Handles different file formats
iii. Implements retry logic for failed connections

b. Transformation Layer:

i. Standardizes audio file naming: {patient_id}_{timestamp}_{location}_{filter}.wav
ii. Normalizes patient demographics
iii. Converts physician notes to a structured format
iv. Validates data quality rules

c. Loading Layer:

i. Implements batch and streaming ingestion

ii. Creates data partitions by date and diagnosis
iii. Maintains data lineage

3. Third Part: Data Quality and Validation

A. Data Quality Framework

Implement data quality checks:

```
class DataQualityValidator:
    def validate_audio_file(self, file_path):
        """
        Checks:
        - File corruption
        - Sample rate consistency
        - Duration within expected range (5-30 seconds)
        - Signal-to-noise ratio
        """
        pass

    def validate_metadata(self, metadata_dict):
        """
        Checks:
        - Required fields present
        - Data type consistency
        - Value ranges (age: 0-120, etc.)
        - Referential integrity
        """
        pass
```

B. Error Handling and Logging

Design a comprehensive error handling system:

a. Log all data quality issues
b. Create quarantine tables for problematic records

c. Generate daily quality reports
d. Implement alerting for critical issues

4. Fourth Part: Performance Optimization

A. Optimize for Scale

a. Implement parallel processing for audio file conversion
b. Design an indexing strategy for quick retrieval
c. Create a data compression strategy for storage optimization
d. Implement caching for frequently accessed data

B. Monitoring and Metrics

Create a monitoring dashboard tracking:

a. Pipeline throughput (files/hour)
b. Error rates by data source
c. Storage utilization
d. Processing latency

Deliverables

1. **Python Code:**
   - Complete ETL pipeline implementation
   - Data quality validation framework
   - Unit tests with >80% coverage
   - A folder called “cleaned data”, which contains:
     i. patient_demographics.csv - *Standardized patient information*
     ii. audio_recordings.csv - *Standardized recording metadata*
     iii. data_quality_report.csv - *Processing statistics*

2. **SQL Scripts:**
   - Database schema creation
   - Optimization indexes

- Sample queries for common use cases

3. **Documentation:**
    - Data flow diagram
    - API documentation for pipeline components
    - **PDF Report** with executive summary
4. **Performance Report:**
    - Benchmark results for different data volumes
    - Optimization recommendations
    - Scalability analysis

Evaluation Criteria

- **Code Quality (30%):** Clean, modular, well-documented code
- **Data Quality Implementation (25%):** Comprehensive validation and error handling
- **Scalability (20%):** Ability to handle large volumes efficiently
- **Schema Design (15%):** Normalized, efficient database design
- **Documentation (10%):** Clear technical documentation

Sample Disorganized Data Structure (Will Be Provided)

```
/raw_data/
├── hospital_A/
│   ├── recordings_2024_jan.wav
│   ├── patient_list.xlsx
│   └── notes.txt
├── clinic_B_data/
│   ├── JSON_exports/
│   │   └── device_readings_*.json
│   └── AUDIO FILES/
└── research_center/
    ├── subj001_to_050/
    └── metadata_incomplete.csv
```