**Conestoga College**

**School of Applied Computer Science & Information Technology**

**SENG8080 - Case Studies**

**Topic Name: Flight Analysis**

|  |  |
| --- | --- |
| **Submitted By:** |  |
| **1.** | **Nilaykumar Patel (8892425)** |
| **2.** | **Rajatkumar Maisuriya (8870363)** |
| **3.** | **Drashti Bhavsar (8890040)** |

**Abstract:**

For both travelers and airlines, effective and dependable flight information management is crucial in the fast-paced world of today. A Python-based method for managing flight details, such as airline details, flight numbers, flight status, operated by, departure time, and arrival time, is presented in this abstract. This solution offers a complete and current source of flight information by integrating data from an API and a dataset.

The following are the main points of emphasis for this Python code project:

Data Retrieval: The code retrieves flight information from two different sources: a dataset containing historical records and an API with real-time data. This combination guarantees that the system meets different analytical and operational needs by providing both historical and current flight information.

Data management: It is simple to query and obtain flight details since the retrieved data is arranged and kept in a structured database. The airline details, flight numbers, flight statuses, operational data, departure and arrival times, and other details are all supported by the database structure.

**Index:**

1. Introduction
2. Data Research and Integration
3. Data Collection
4. Data Storage and Maintenance
5. Images
6. References

* **Introduction:**

First, discuss in more detail about the Python code's data management and retrieval features for managing flight information.

1. **Data Retrieval:**

**API Integration for Real-Time Data:** To collect real-time flight information, the Python code has modules that communicate with a designated flight data API. This API offers up-to-date information on flights, including their status, times of departure and arrival, and operational specifics. By enabling real-time flight tracking, this integration makes sure that the system can meet the immediate demands of both passengers and airline workers.

**Dataset for Historical Records:** The system includes a historical flight records dataset in addition to the API. This dataset could be a compilation of historical flight data that includes information about previous flights including airline names, flight numbers, statuses, departure timings, and arrival times. The dataset enables the system to address other analytical requirements such as historical flight trends and delays.

**Data Synchronization:** To ensure that the real-time data is current, the code is made to regularly synchronize with the API. For historical data, it is possible to either update it on a regular basis or add to it as more data becomes available.

1. **Data Management:**

**Database Structure:** The retrieved data is structured and arranged in a relational database, whether it came from the API or the historical dataset. Information about flights is efficiently stored via the database schema. It has tables or collections for airline information, flight numbers, statuses, operational details, departure times, and arrival times.

**Normalization and Data Consistency:** To reduce redundancy and guarantee data consistency, data is kept in a normalized form. For instance, flight records can relate to airline information that has only been kept once, improving efficiency, and lowering the possibility of data errors.

**Query and Retrieval:** People can query the database to receive flight information based on a variety of parameters. They can look up previous flight delays, search for flights operated by a particular airline, or get operational information on flights run by businesses. These searches are effective because of the database's organization and indexing.

In conclusion, this Python-based system for managing flight details efficiently gets and organizes data from both current sources and archived records. It offers a complete solution for a variety of analytical and operational purposes in the aviation sector by arranging the data in a database and providing strong querying.

* **Data Research and Integration:**

**Data Sources:**

1. **Flight API Real-Time Data:** The Flight API is essential for receiving the most recent knowledge possible on flights, including their statuses, times of departure, and times of arrival. To meet urgent passenger demands, follow flights in real-time, and keep your system current with information, you must have access to this data source.

**API Link:** <https://docs.flightapi.io/track-flights-between-airports>

Key Steps for Utilizing the Flight API:

* Acquire an API key by registering for access to the Flight API.
* To get real-time flight information, use Python libraries like requests or specialized Python SDKs that the API provides.
* To make sure that your system always has the most recent data, use recurring or event-driven data synchronization with the API.
* Put the real-time flight data into your database in a way that follows the schema intended for historical data.

1. **Kaggle Airline Flight Dataset:** The Kaggle dataset includes historical flight information about individual flights, such as the names of the airlines, the flight numbers, the airports of departure and arrival, the times of the departure and arrival, and more. This dataset is useful for gathering historical flight data for trends, delays, and other statistical analysis. It can act as the system's starting point for managing flight details.

**Kaggle Dataset Link:**

<https://www.kaggle.com/datasets/arunjangir245/airline-flight-dataset-schedule-performance-etc>

Key Steps for Utilizing the Kaggle Dataset:

* Utilizing a library like Pandas, download the dataset from Kaggle and import it into your Python environment.
* Make a database table or structure that precisely fits the dataset's structure to hold the historical flight data.
* Add the historical flight information to the database.

**Data Management and Integration:**

With data from both sources now available in your system, you can integrate them as follows:

**Database Schema:** Design a database schema that can handle both past and present flight information. Tables for airline information, flight numbers, flight statuses, operational data, departure times, and arrival times are all possible.

**Synchronization:** Use the Flight API for scheduled synchronization to keep your real-time data current. To guarantee the most recent information, this could be done on a frequent basis (e.g., every few minutes or every hour).

* **Data Collection:**

**Step 1: Data Retrieval from Kaggle (Historical Data):**

**Data Retrieval:** The first step is to obtain the Kaggle dataset with historical flight data from the link that is provided above. This dataset may be accessible in CSV file format. We can directly retrieve it using Python's requests library, or you can manually download and save it to your project directory.

**Pandas Data Loading:** The historical flight data from the Kaggle dataset can be loaded into a Pandas DataFrame using the Pandas package.

**Pyodbc database connection:** Using the pyodbc package, establish a connection to your SQL Server database so that you may save the historical flight information.

**Database Table Creation:** Create the database tables in your SQL Server database using SQL commands, defining their structure to correspond to the Kaggle dataset.

**Data Insertion:** Embed the Kaggle Data Frame’s historical flight data in your SQL Server database. Insert records into the database that are consistent with the Data Frame’s table structure.

**Step 2: Real-Time Data Retrieval from Flight API:**

**Real-time data fetching:** We used API key to retrieve current flight information from the Flight API using Python's requests module.

**Database Connection and Table**: Establish a connection to your SQL Server database and we ensure that you have a table to store the real-time flight data.

**Data Insertion:** Like how historical data is inserted, insert the real-time flight data into your SQL Server database.

* **Data Storage and Maintenance with SQL Server:**

**Database Structure:**

**Historical Data Structure:** We created a table in SQL Server first to hold historical flight information received from the Kaggle dataset. Define the columns in the dataset, such as airline, flight\_number, departure\_airport, arrival\_airport, departure\_time, and arrival\_time, to be included in the structure of this table.

**Real-time Data Table:** Created a table for holding real-time flight data that has been acquired from the Flight API in a similar manner. This table should have columns for airline, flight\_number, departure\_airport, arrival\_airport, departure\_time, arrival\_time, and status that correspond to the schema of the real-time data obtained from the API.

**Data Insertion:**

**Historical Data Insertion:** Insert the historical flight information into the database table that is made specifically for historical information. To populate the table, SQL INSERT commands can be executed using Python and pyodbc.

**SQL Query:**

**CREATE TABLE HistoricalData(**

**FlightID INT IDENTITY(1,1) PRIMARY KEY,**

**Date DATE,**

**DepTime TIME,**

**ArrTime TIME,**

**UniqueCarrier VARCHAR(255),**

**FlightNum VARCHAR(10)**

**);**

**INSERT INTO FlightData (Date, DepTime, ArrTime, UniqueCarrier, FlightNum)**

**VALUES (?, ?, ?, ?, ?);**

**Real-Time Data Insertion:** Insert real-time flight data into the real-time data table in a manner like that described above. Python may be used to carry out SQL INSERT statements.

**SQL Query:**

**CREATE TABLE Schedules(**

**Airline VARCHAR(255),**

**FlightNumber VARCHAR(10),**

**Status VARCHAR(50),**

**OperatedBy VARCHAR(255),**

**DepartureTime DATETIME,**

**ArrivalTime DATETIME**

**);**

**INSERT INTO YourTableName (Airline, FlightNumber, Status, OperatedBy, DepartureTime, ArrivalTime)**

**VALUES (?, ?, ?, ?, ?, ?, ?);**

**Data Maintenance:**

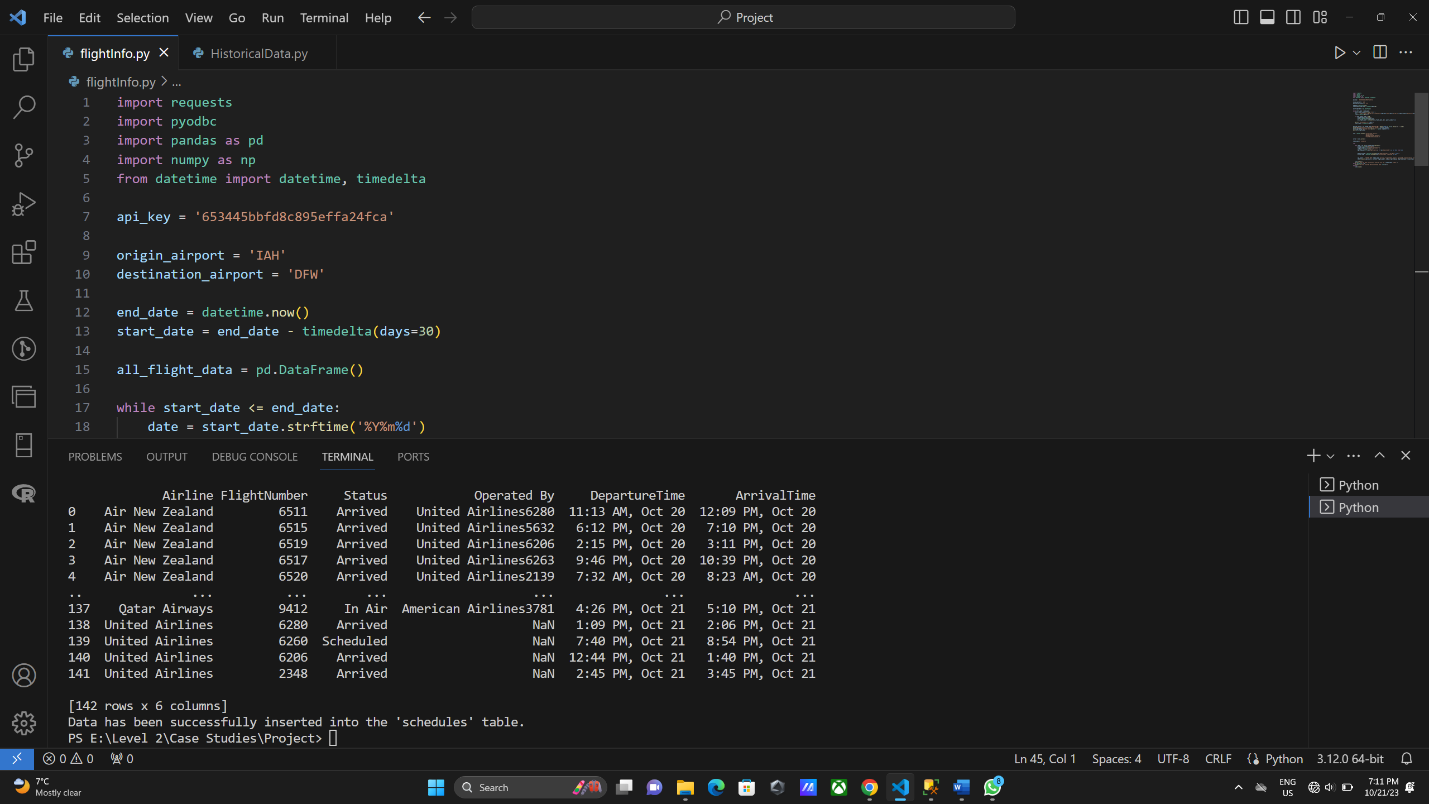
**Data Cleanup:** Over time, databases may gather unnecessary or out-of-date data. Consider putting in place data cleanup procedures to preserve data quality and improve database performance. For instance, you can set up data retention policies to have records that are automatically deleted when they reach a particular age be automatically deleted records older than a certain threshold.

* **Project Timeline:**

|  |  |  |
| --- | --- | --- |
| **Date** | **Deliverable** | **Responsible** |
| Oct 05 | Data Collected and planned | Nilay |
| Oct 18 | 1st Draft Circulated to Team | Rajat, Drashti |
| Oct 21 | 1st Draft of Presentation Circulated | Drashti, Nilay, Rajat |

* **Images:**

**FlightInfo.py**



**Database**

A screenshot of a computer

Description automatically generated

**HistoricalData.py**

A screenshot of a computer

Description automatically generated

**Database**

A screenshot of a computer

Description automatically generated

**References:**

# By Arun Jangir. (Updated 3 months ago). Airline Flight Dataset: Schedule, Performance etc.

# <https://www.kaggle.com/datasets/arunjangir245/airline-flight-dataset-schedule-performance-etc>

# By FlightAPI. Track Flights between Airports.

# <https://docs.flightapi.io/track-flights-between-airports>