Airline Search Engine

BMW-415

By Mitchell Kolb, Brian Joo, and Noah Waxman

Problem Statement

- To develop a program that can query, analyze, and display data from the OpenFlights.org Airline dataset.
- This tool is supposed to help users to find out facts/trips with requested information/constraints
- Use effective MapReduce, SQL/SPARQL/PYSPARK, and/or graph algorithms

Project Details

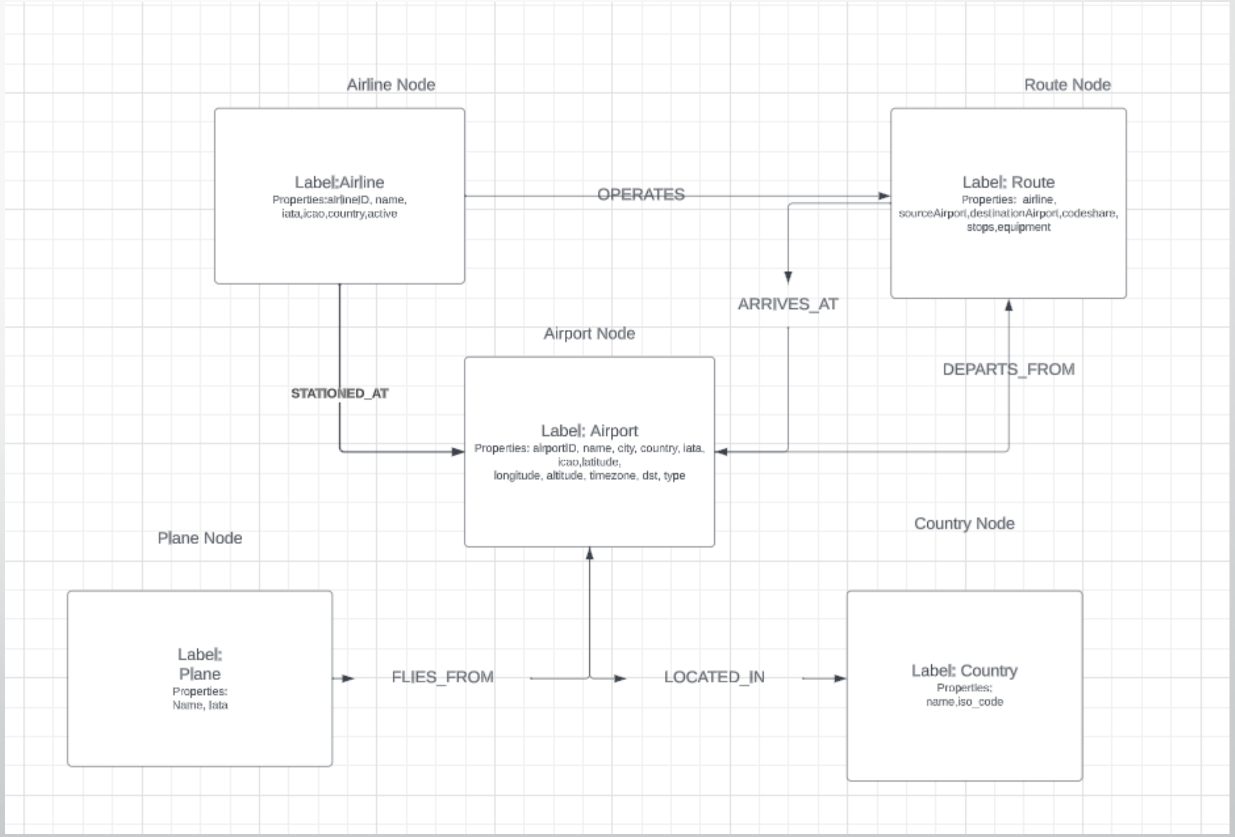
- Database connection
  - Neo4j
- Search screen / User Interface (UI)
  - Displays various search options for users to select
- Functionality
  - Search operations are implemented using graph algorithms and SQL queries
- GUI
  - PySimpleGUI

Dataset

- Our original dataset contains 5 .csv files
- airlines (400KB)
- airports (1.3MB)
- routes (2.27MB)
- planes (5KB)
- country (5KB)

Architecture

- We have five sets of nodes created from the five .csv files and four relationships that connect them together
- Relationships: OPERATES, DEPARTS_FROM, ARRIVES_AT, DISTANCE


Airline Node
Label:Airline
Properties:airlineID, name, iata,icao,country,active
OPERATES
Route Node
Label: Route
Properties: airline, sourceAirport,destinationAirport,codeshare, stops,equipment
ARRIVES_AT
DEPARTS_FROM
STATIONED_AT
Airport Node
Label: Airport
Properties: airportID, name, city, country, iata, icao,latitude, longitude, altitude, timezone, dst, type
Plane Node
Label: Plane
Properties: Name, Iata
FLIES_FROM
LOCATED_IN
Country Node
Label: Country
Properties: name,iso_code

Search options

- Find airline details.
- Find active airlines in the US.
- Find the top k cities with the most incoming and outgoing airlines.
- Find all cities reachable within a specified number of hops from a city.
- Find cities within a specified number of hops from a city.
- Determining which city has the highest number of airports.
- Find a trip that connects two cities X and Y (reachability).

Scalability

- Adding new data is easy
- Sharding may not be beneficial because the dataset is already small.
- Parallel processing is technically possible but may not result in a significant performance improvement.
- Horizontal Scaling will increase user/query quantity

Data visualization

- Operations Focus:
  - Complex tasks on structured airline data in Neo4j.
  - Retrieval and manipulation, not visualization.
- Not all search options can be visualized
- Search options *with* visualization use python libraries:
  - Matplotlib
  - Numpy

UI / UX Design

- Initial UI: Terminal based with only text being returned
- Demonstration UI: PySimpleGUI which is a lightweight library built on top of the integrated UI python library tkinter.
  - Includes a connection screen to enter credentials to connect the user to the local neo4j database
  - Includes a search screen that has a similar look and feel to other popular search engines

Testing / Experiments

- Insertion: In the data from the airlines.csv file there are 6161 line items. To insert it into neo4j it used about 584 bytes of memory. Inserting our full dataset won’t even reach a 20th of that maximum size limit.
- Query Scalability: We used two VM's with different hardware configurations and compared the times to test how that effects speed on query delivery

Demonstration