Graph Gurus 27 demo instructions

This exercise from Graph Gurus 27 demonstrates three centrality algorithms: closeness centrality, betweenness centrality, and PageRank.

These instructions accompany the video recording of Graph Gurus 27.

Install the solution file called guru27_Feb3.tar.gz, which bundles the schema, data, loading jobs, and queries used in this exercise. It also includes the queries for Graph Gurus 26 on Path Algorithms.

(1) From the main page of GraphStudio. Import the solution file export_GG27_feb3.tar.gz.
(2) Go to the Map Data to Graph Page. In the top menu, click on the Add File icon. One at a time, select each of the 4 files and make sure the Has Header box is checked. Click Cancel when done. Now click the Upload (up arrow) button.
(3) Go to the Load Data page. Wait a few seconds for the Load (right arrow) button to become active. Click it and wait for the data to be loaded.
(4) Go to the Write Queries page. Click the Install All Queries button, to the left of the "GSQL Queries" heading.

Dataset:
(Same as Graph Gurus 26).
Global commercial airline routes from https://openflights.org/data.html. Did some minor cleanup (replaced "\n" with "", which had represented NULL, and added a header row.)

- 7,698 airports (airports.dat, cleaned up and renamed airports.csv)
- 67,664 direct flight routes (cleaned up and renamed routes.csv)

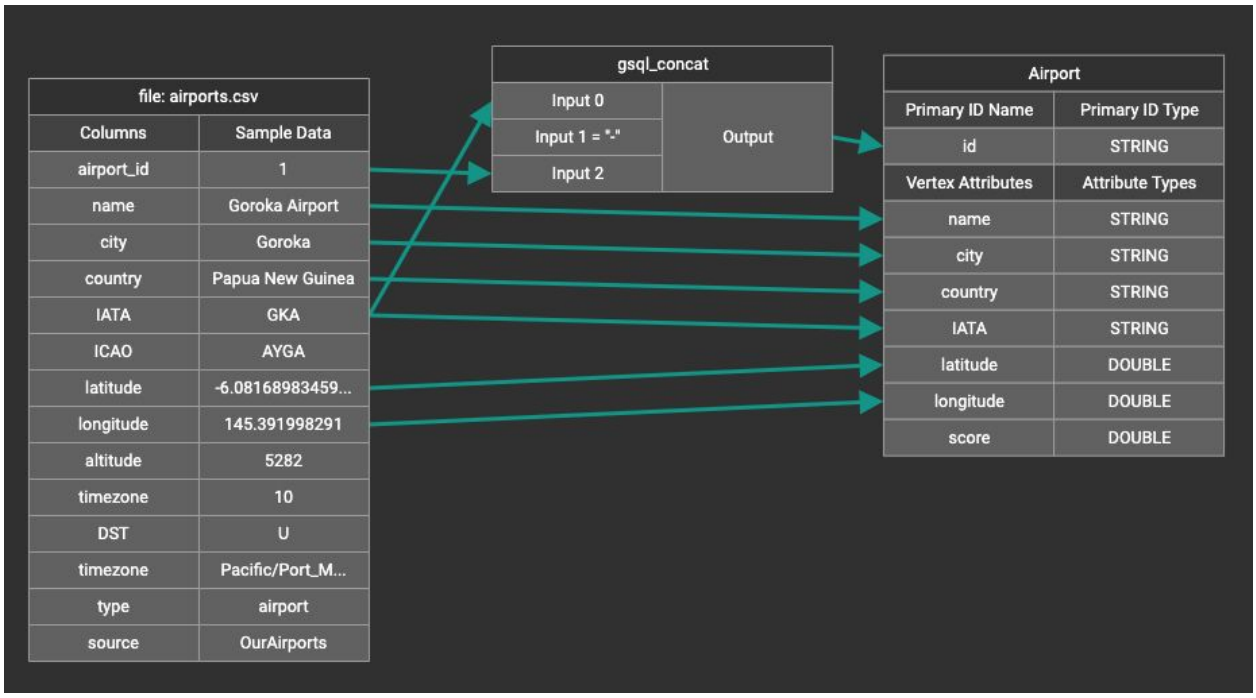
Also manually created a separate small graph of fake airports and routes, for a small-scale demo (airports-path-demo.csv and routes-path-demo.csv)

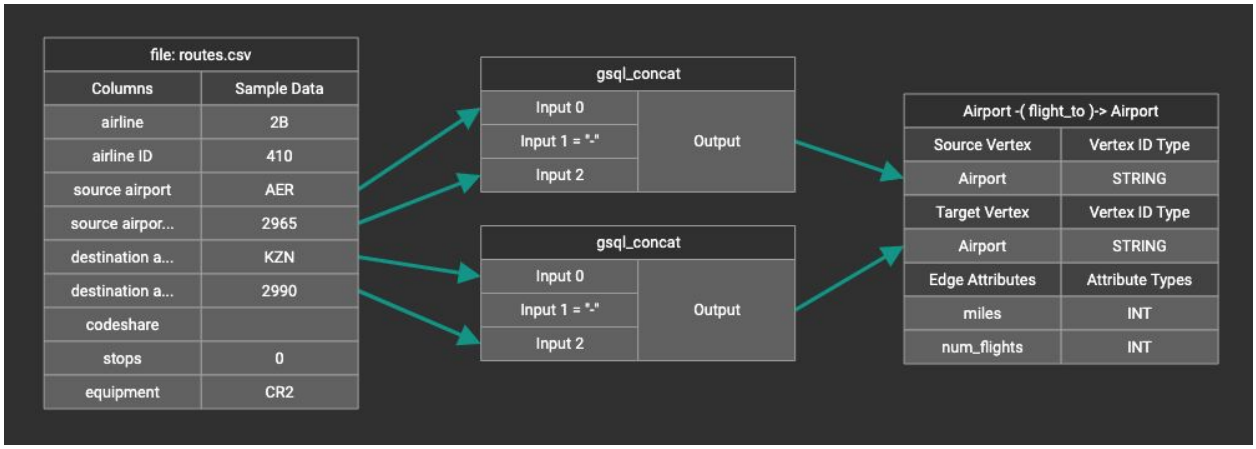
Schema:
We extended the Graph Gurus 26 schema to include a directed edge for airline flight routes. This is needed for PageRank which works on directed graphs.

```
VERTEX Airport(PRIMARY_ID id STRING, name STRING, city STRING, country STRING, IATA
STRING, latitude DOUBLE, longitude DOUBLE, score DOUBLE)

UNDIRECTED EDGE flight_route (FROM Airport, TO Airport, miles INT)

DIRECTED EDGE   flight_to    (FROM Airport, TO Airport, miles INT, num_flights INT)
             WITH REVERSE_EDGE reverse_flight_to
```

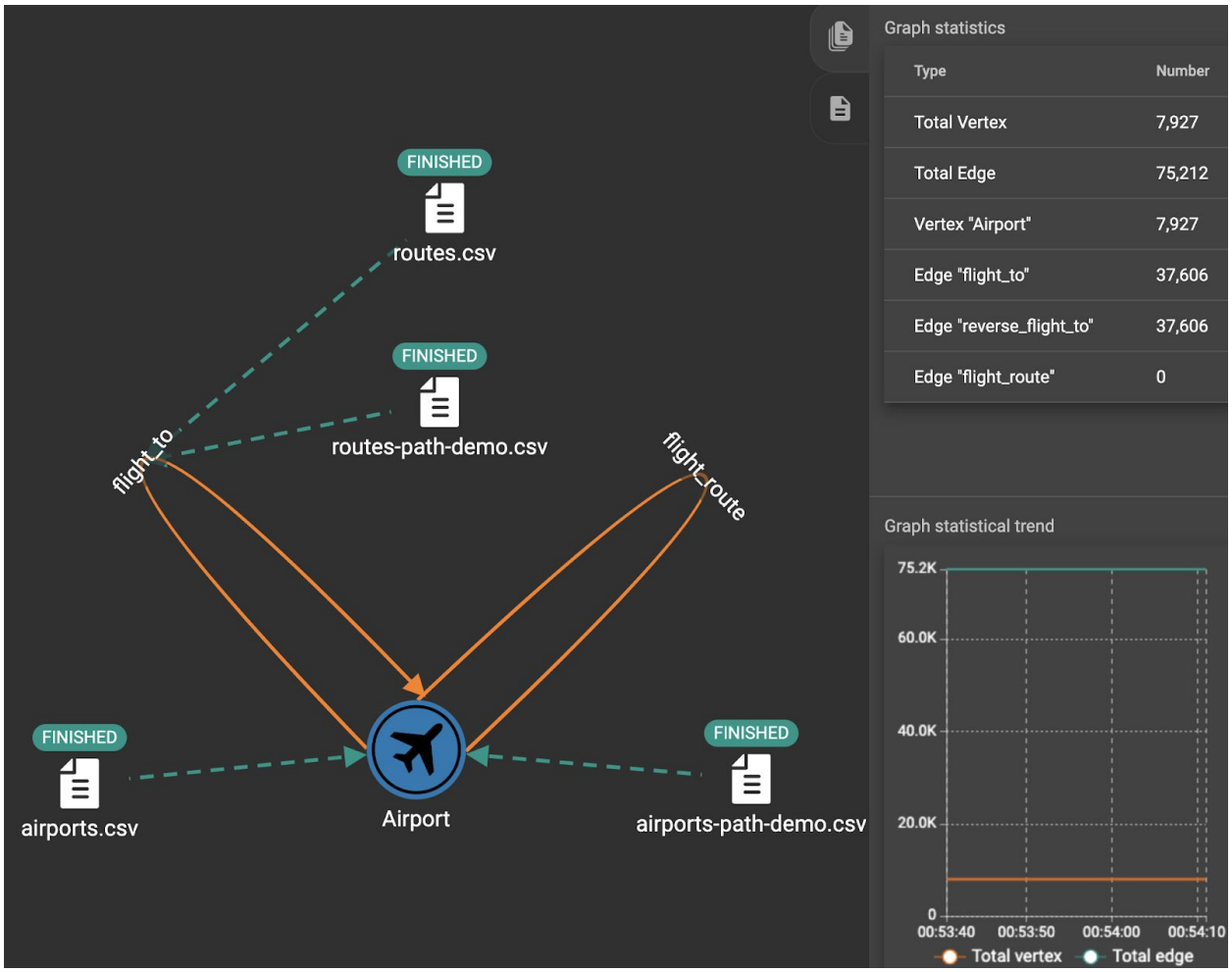
Data Mapping and Loading:
The data mapping is the same as for Graph Gurus 26, except that we map the input file routes.csv and routes-path-demo.csv to the directed edge flight_to.
These figures show how we mapped the CSV data to the vertex and edge types. For the Airports, notice how we constructed the ID by concatenating the IATA code with the airport_id

(openflight.org's numbering). We did this because while the IATA is the most recognizable (e.g., SFO, LAX, JFK), some smaller airports in the dataset do not have an IATA code.





Here are the vertex and edge counts after loading. We have fewer airports than expected. Some of the datalines must have had formatting or data value problems, but since this is just a demo, we didn't bother to trace the cause. There are significantly fewer routes than lines in the routes.csv file. This is because the data file sometimes has multiple routes between two cities (e.g. LAX to JFK), but we counted each route only once.



Data Preparation before Querying

The data from openflights.org does not tell us the distance between airports, but we can compute this from the latitude and longitude information, using the Haversine formula. We wrote a query to apply the Haversine formula for each edge and then store the result in the the edge's *miles* property. The addWeights query in Graph Gurus 26 hardcoded the names of the vertex type (Airport) and edge type (flight_route). We now have an additional edge type, so we have changed the edge type to be an input parameter. Run the query with e_type = flight_to. Notice how similar the GSQL code in addWeights looks to an implementation in Python.

Centrality Queries

We used three algorithms which are in our GSQL Graph Algorithm Library: https://github.com/tigergraph/gsql-graph-algorithms

closeness_cent.gsql

This file actually contains two queries, a main query called *closeness_cent* and a subquery called *cc_subquery*. GraphStudio does not allow you to define two queries in one file, so we needed to split this into two separate files.

Edits we made:

1. Created a GraphStudio query called cc.
    a. Copied just the main query here. In the header line, changed the query's name to cc, and changed the graph's name to MyGraph.
    b. Changed `Start = {ANY}` to `Start = {Airport.*}`
    c. Changed `FROM Start:s -(:e)-> :t`
       to `FROM Start:s -(flight_to:e)-> :t`
    d. Changed maxHops to be in input parameter
2. Created a GraphStudio query called cc_subquery.
    a. Changed `FROM Start:s -(:e)-> :t`
       to `FROM Start:s -(flight_to:e)-> :t`

Changes to improve the filtering and the output:

1. Filter the graph by country:
    a. Add an input parameter STRING country.
    b. After the line `Start = {Airport.*};`
    add the following block:

```
IF country != "" THEN
    Start = SELECT v
        FROM Start:v
        WHERE v.country == country;
END;
```

2. In the output, include each vertex's name attribute.
    a. Change the definition of the vertexScore tuple from
    `TYPEDEF TUPLE<VERTEX Vertex_ID, FLOAT score> vertexScore;`
    to
    `TYPEDEF TUPLE<VERTEX Vertex_ID, STRING name, FLOAT score> vertexScore;`
    b. Change the line
    `@@topScores += vertexScore(s, cc_subquery(s,numVert,maxHops));`
    to
    `@@topScores += vertexScore(s, s.name, cc_subquery(s,numVert,maxHops));`

betweenness_cent.gsql

Similar to closeness_cent.gsql, this file actually contains two queries, a main query called *betweenness_cent* and a subquery called *bc_subquery*. We made similar changes (splitting, renaming, maxHops parameter, specifying vertex types and edge types)

Additional change: sort and limit the results, so we see the top scores first.

1. Add a parameter INT maxItems.
2. Define a local accumulator:
   `SumAccum<float> @cent;`
3. Replace the final PRINT statement (`PRINT @@BC;`) with the following block:

```
Start = SELECT s FROM Start:s
              POST-ACCUM s.@cent += @@BC.get(s)
              ORDER BY s.@cent DESC
              LIMIT maxItems;
PRINT Start[Start.name, Start.@cent];
```

pageRank.gsql

We just set the vertex type to be **Airport** and the edge type to **flight_to**.
We made a variant called pageRank_by_country.gsql so that we could focus just on one country and therefore have a more intuitive sense of what *should* be the most "authoritative" airport. Since pageRank focuses in in-edges, therefore it means which airport you are mostly likely to be found at, if you travel all routes randomly. (Who does that?)