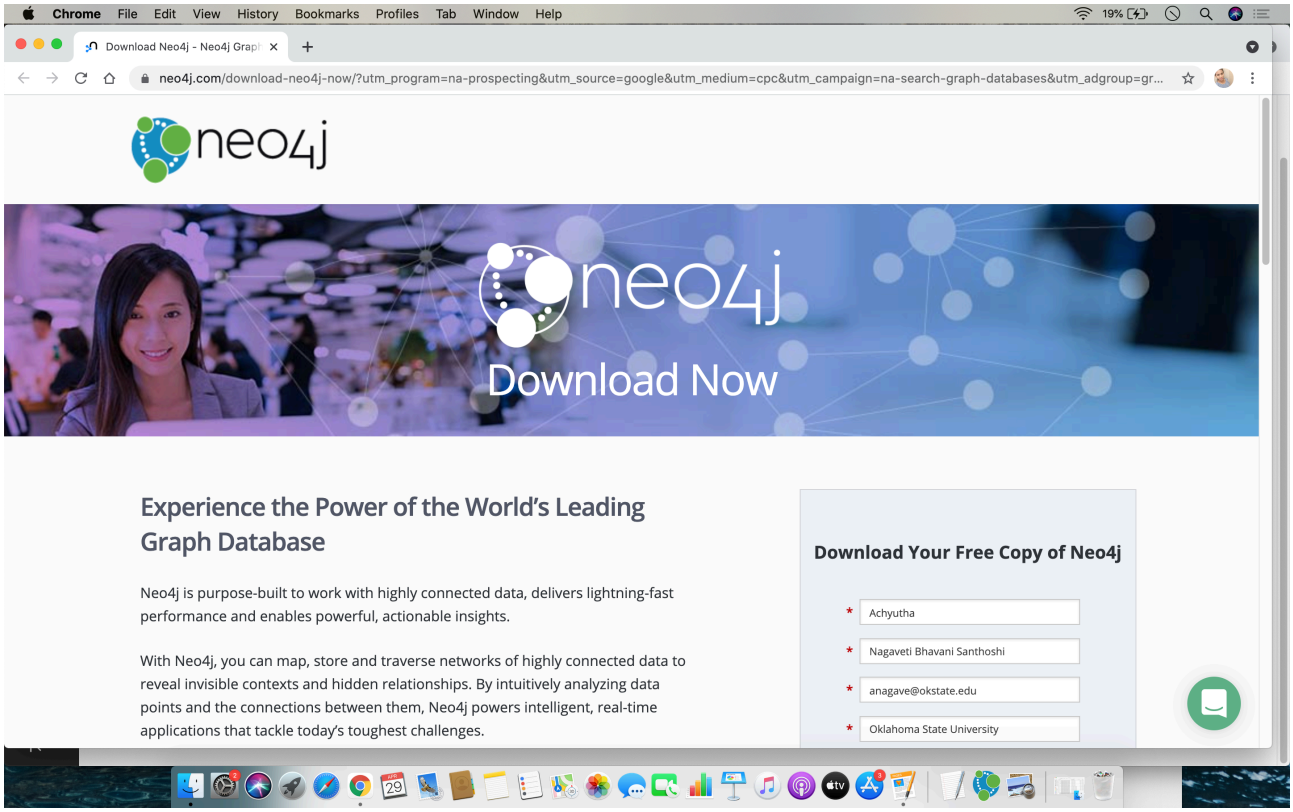
Big Data Management
Programming Assignment #3
Neo4j

Introduction:

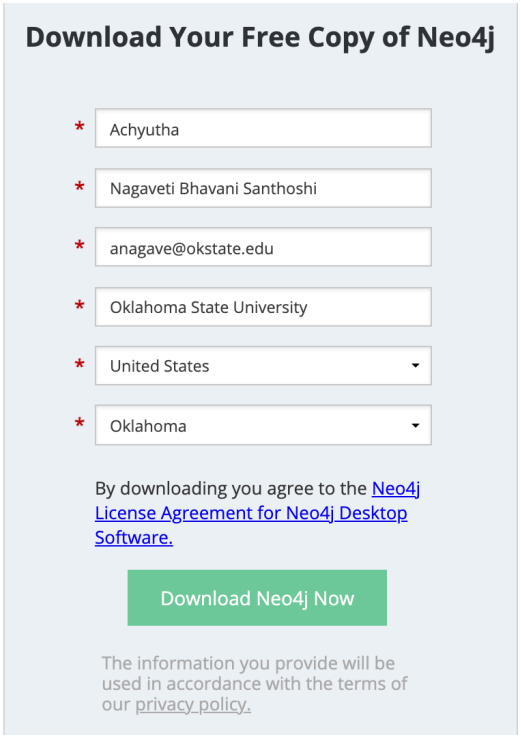
- Neo4j (Network Exploration and Optimization 4 Java) is a graph database management system developed by Neo4j, Inc.
- Described by its developers as an ACID-compliant transactional database with native graph storage and processing, Neo4j is available in a GPL3-licensed open-source "community edition", with online backup and high availability extensions licensed under a closed-source commercial license.
- Neo4j is implemented in Java and accessible from software written in other languages using the Cypher query language through a transactional HTTP endpoint, or through the binary "bolt" protocol.
- In Neo4j, everything is stored in the form of an edge, node, or attribute. Each node and edge can have any number of attributes. Both nodes and edges can be labelled.
- Labels can be used to narrow searches. As of version 2.0, indexing was added to Cypher with the introduction of schemas.[32] Previously, indexes were supported separately from Cypher.

Installation and Setup:

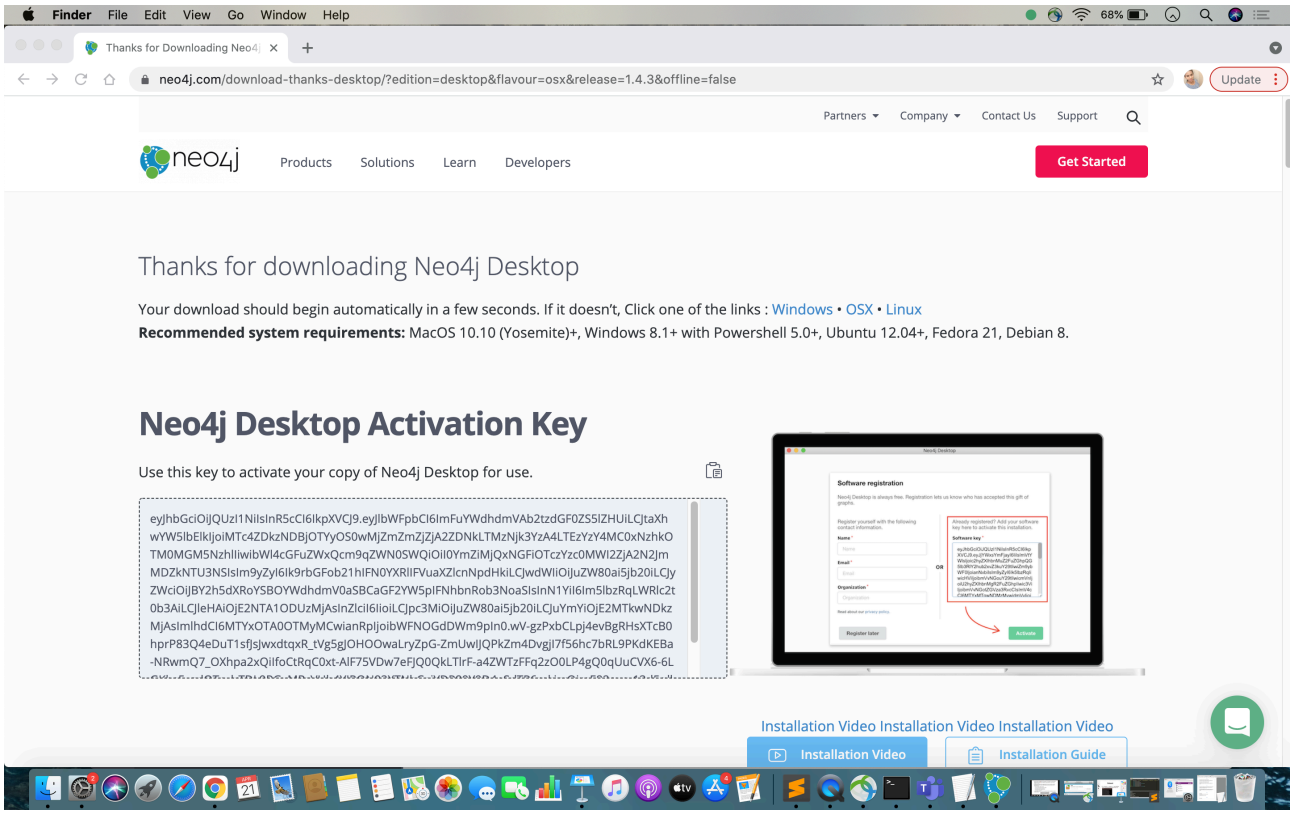
1.Navigate to https://neo4j.com/download-neo4j-now to Install the Software.



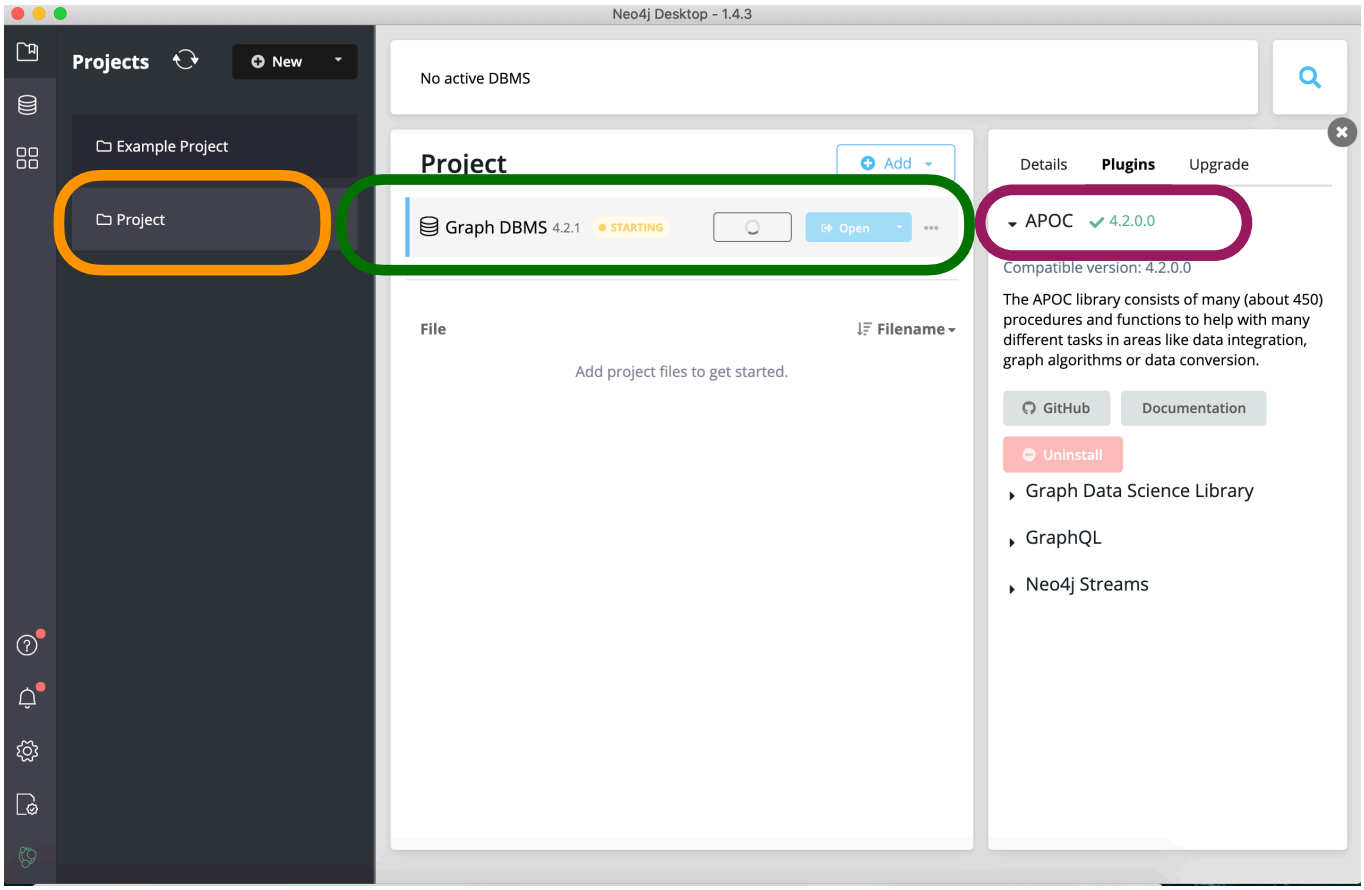
2.Enter details like First Name, Last Name, Email, Organization, City, and State.



3. We get an Activation key that should be applied on downloaded software to Activate it



4.

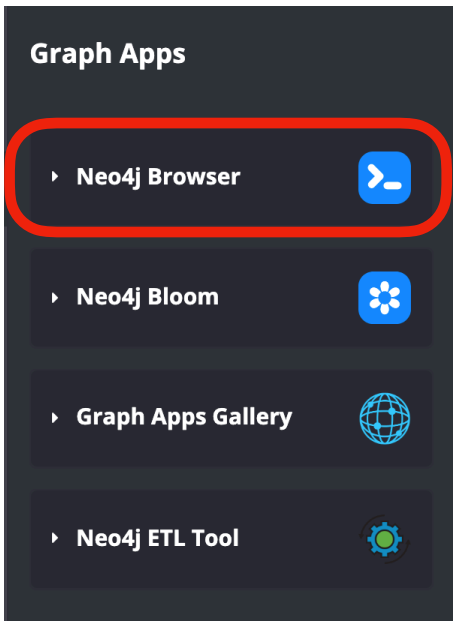


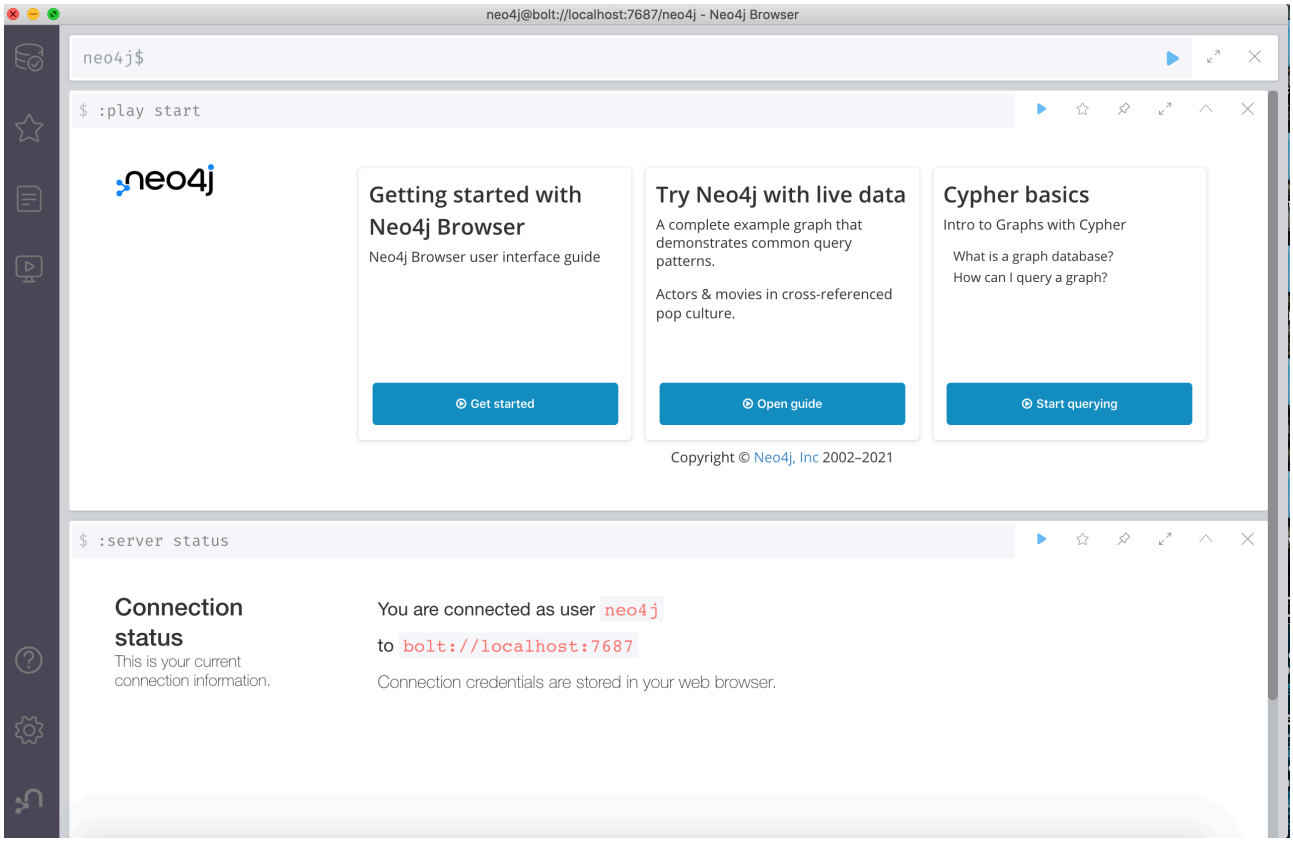
Create a project by clicking +New button,
A new project will be created

A new DBMS is created, now I have just imported Airport.csv file,
Click start the Database.

Start the Plugin named Apoc which is necessary for Loading the data

5. Open the Neo4j Browser





The Above Picture shows Neo4j Browser. Where we can Query using Cypher Language.

Dataset:

Used the " Airport.csv" dataset posted on canvas for this assignment.

Dataset Fields

- Origin_airport: Three letter airport code of the origin airport
- Destination_airport: Three letter airport code of the destination airport
- Origin_city: Origin city name
- Destination_city: Destination city name
- Distance: Distance (to nearest mile) flown between origin and destination
- Fly_date: The date (yyyymm) of flight
- Origin_population: Origin city's population as reported by US Census
- Destination_population: Destination city's population as reported by US Census

Query for Task1:

Task-1: Load the airport data (Airport.csv) into Neo4J. Before you load it, clearly identify the nodes, edges, properties etc. of your graph

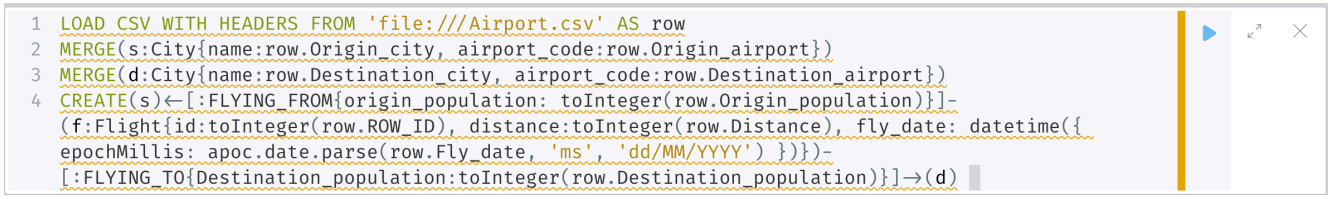
Source Code:

```
LOAD CSV WITH HEADERS FROM 'file:///Airport.csv' AS row //file is imported to the DBMS

MERGE(s:City{name:row.Origin_city, airport_code:row.Origin_airport}) //Source Columns

MERGE(d:City{name:row.Destination_city, airport_code:row.Destination_airport}) // Destination Columns

CREATE(s)<-[:FLYING_FROM{origin_population: toInteger(row.Origin_population)}]-(f:Flight{id:toInteger(row.ROW_ID), distance:toInteger(row.Distance), fly_date: datetime({ epochMillis: apoc.date.parse(row.Fly_date, 'ms', 'dd/MM/YYYY') })})-[:FLYING_TO{Destination_population:toInteger(row.Destination_population)}]->(d)

//Using “CREATE” to map the nodes (or) attributes and edges
```

Nodes, Edges and Properties :

1. [:FLYING_FROM{origin_population: toInteger(row.Origin_population)}}
2. (f:Flight{id:toInteger(row.ROW_ID)
3. distance:toInteger(row.Distance)
4. fly_date: datetime({ epochMillis: apoc.date.parse(row.Fly_date, 'ms', 'dd/MM/YYYY')
5. [:FLYING_TO{Destination_population:toInteger(row.Destination_population)}]->(d)

Screenshots:



```
1 LOAD CSV WITH HEADERS FROM 'file:///Airport.csv' AS row
2 MERGE(s:City{name:row.Origin_city, airport_code:row.Origin_airport})
3 MERGE(d:City{name:row.Destination_city, airport_code:row.Destination_airport})
4 CREATE(s)<-[:FLYING_FROM{origin_population: toInteger(row.Origin_population)}]-
  (f:Flight{id:toInteger(row.ROW_ID), distance:toInteger(row.Distance), fly_date: datetime({
  epochMillis: apoc.date.parse(row.Fly_date, 'ms', 'dd/MM/YYYY') })})-
  [:FLYING_TO{Destination_population:toInteger(row.Destination_population)}]→(d)
```

```
neo4j$ LOAD CSV WITH HEADERS FROM 'file:///Airport.csv' AS row MERGE(s:City{name:row.Origin_
```

Table

Added 1048575 labels, created 1048575 nodes, set 5242875 properties, created 2097150 relationships, completed after 971630 ms.

Query for Task2:

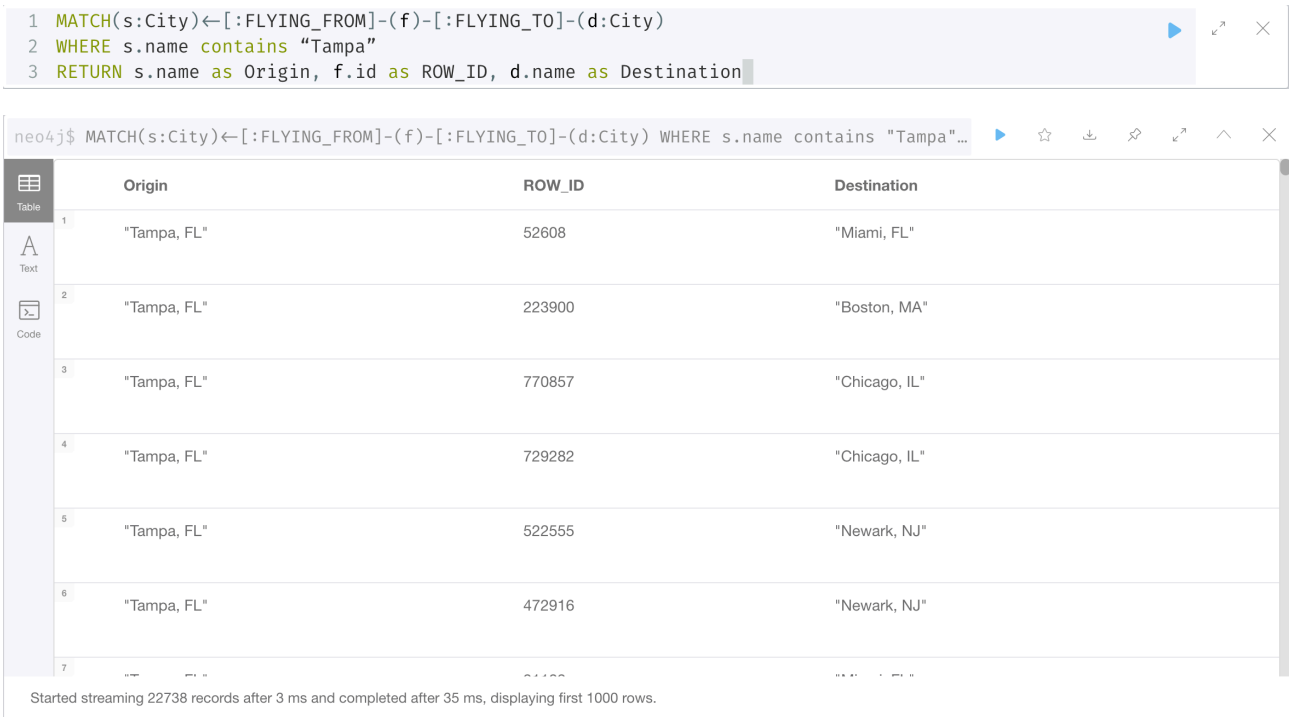
Task-2: Implement the following queries using the cypher query language.

2.1: Show all direct flights departing from a particular city. You may choose any city. The output should the origin city and all the destination cities and the ROW_IDs. For example, there are flights from Amarillo, TX to Elko, NV, Reno, NV,Waco, TX ...

Source Code:

```
MATCH(s:City)<-[:FLYING_FROM]-(f)-[:FLYING_TO]-(d:City)

WHERE s.name contains “Tampa” //Checking for “Tampa” city

RETURN s.name as Origin, f.id as ROW_ID, d.name as Destination //To show Origin, id, Destination
```

Screenshots:



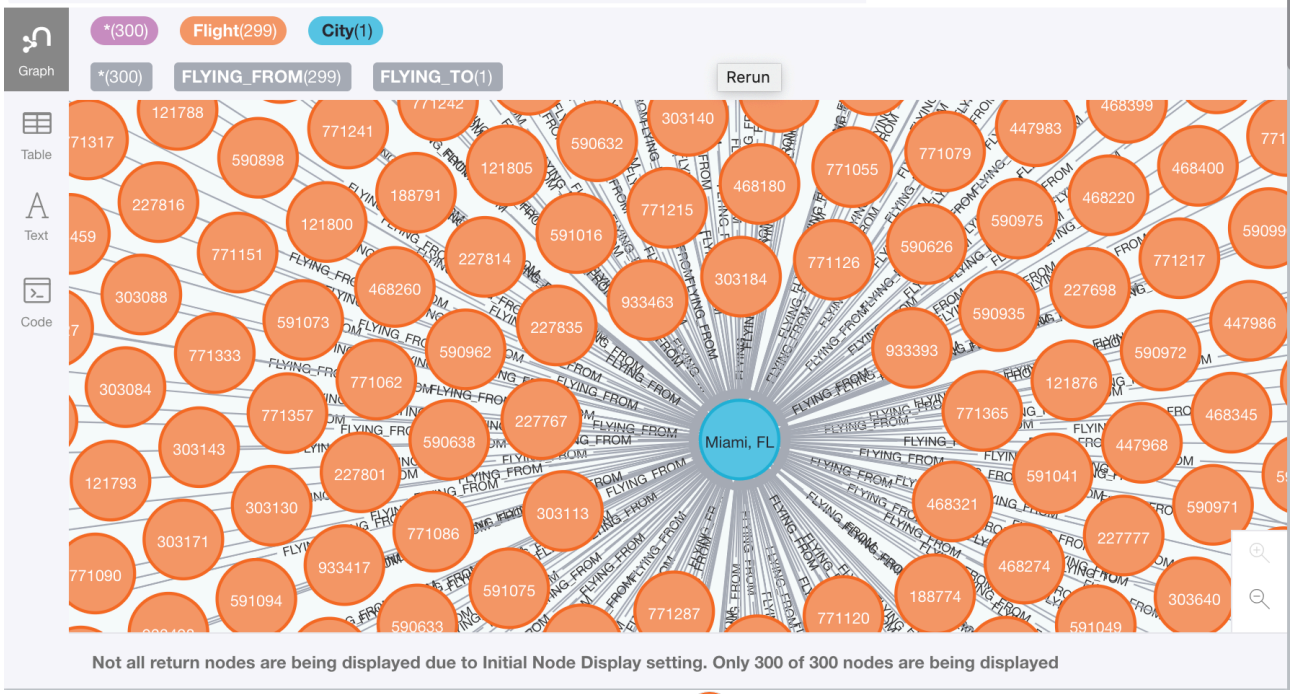
2.2: Show all flights that flew on a certain date from a particular city. You may choose any date and any city. Output City, date and ROW_ID in
-tabular format
-graph format

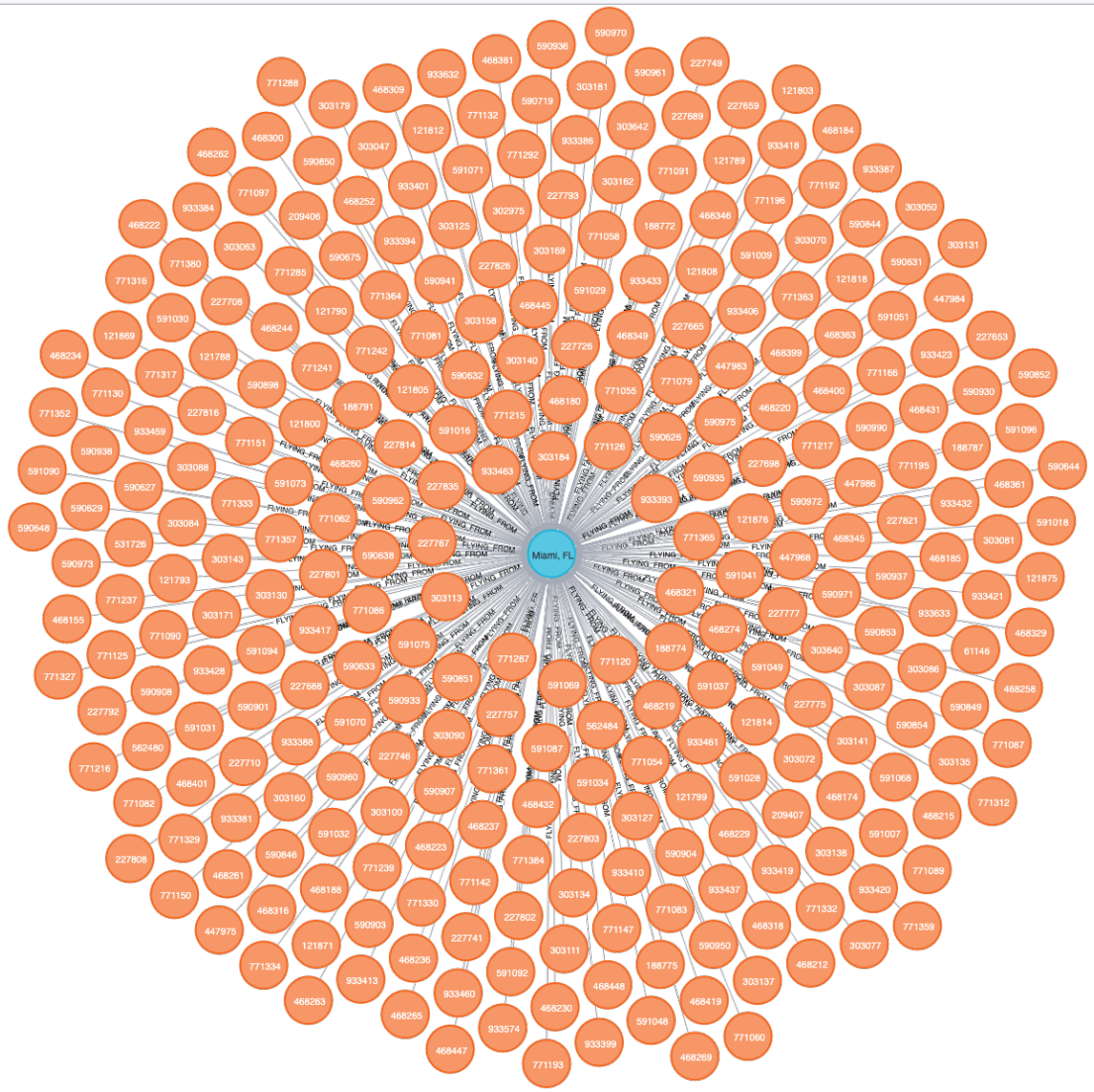
Source Code:

```
MATCH(s:City)<-[:FLYING_FROM]-(f)
where s.name contains "Miami" and date(f.fly_date) = date("1995-01-02")
RETURN  s.name as city, date(f.fly_date), f.id as ROW_ID //for Tabular view

MATCH(s:City)<-[:FLYING_FROM]-(f)
where s.name contains "Miami" and date(f.fly_date) = date("1995-01-02")
RETURN * //for Graph view
```

Screenshots:

```
1 MATCH(s:City)←[:FLYING_FROM]-(f)
2 where s.name contains "Miami" and date(f.fly_date)
  = date("1995-01-02")
3 RETURN *
```





2.3: Show all flights to a city that has a population above a certain threshold on a particular date. You may choose the city, the population threshold and the date. The output should show the city, the population of the city, the date and all the flights.Output city, population threshold, date and ROW_ID in
-tabular format
-graph format

Source Code:

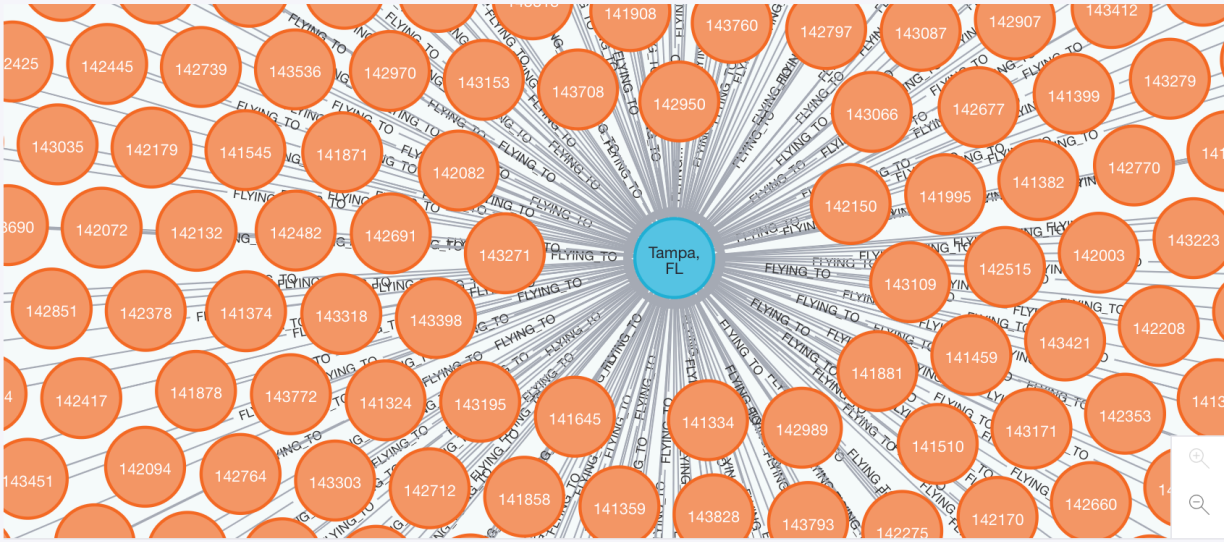
```
MATCH(f:Flight)-[r:FLYING_TO]->(c:City{name:"Tampa, FL"})
WHERE r.Destination_population> 150000 and date(f.fly_date) = date("2005-01-03")
RETURN c.name, r.Destination_population, date(f.fly_date), f.id //for Tabular view

MATCH(f:Flight)-[r:FLYING_TO]->(c:City{name:"Tampa, FL"})
WHERE r.Destination_population> 150000 and date(f.fly_date) = date("2005-01-03")
RETURN * //for Graph view
```

Screenshots:

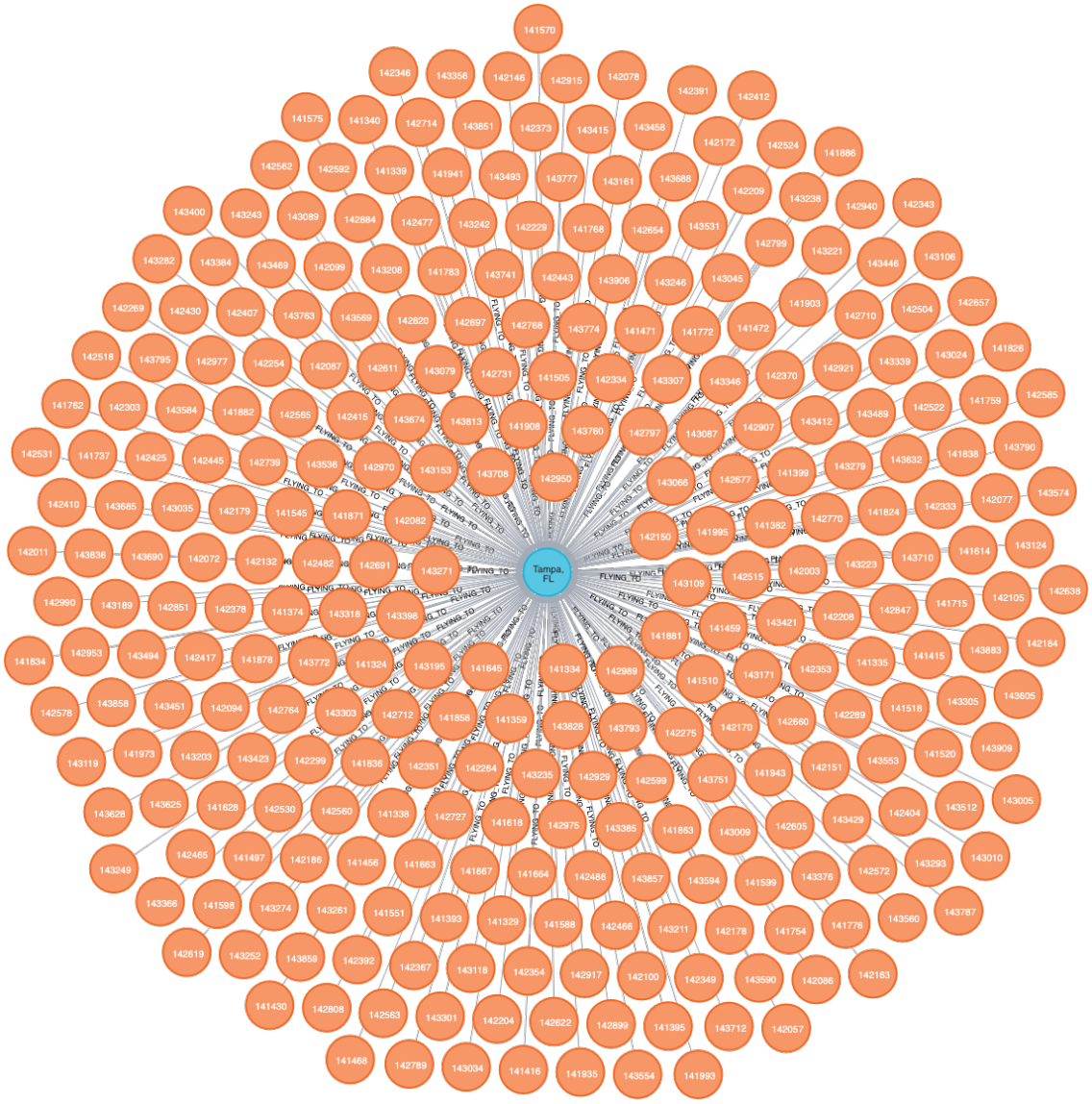
```
1 MATCH(f:Flight)-[r:FLYING_TO]→
  (c:City{name:"Tampa, FL"})
2 WHERE r.Destination_population> 150000 and
  date(f.fly_date) = date("2005-01-03")
3 RETURN *
```

Graph | Table | Text | Code

*(300) City(1) Flight(299) *(299) FLYING_TO(299)



Not all return nodes are being displayed due to Initial Node Display setting. Only 300 of 300 nodes are being displayed



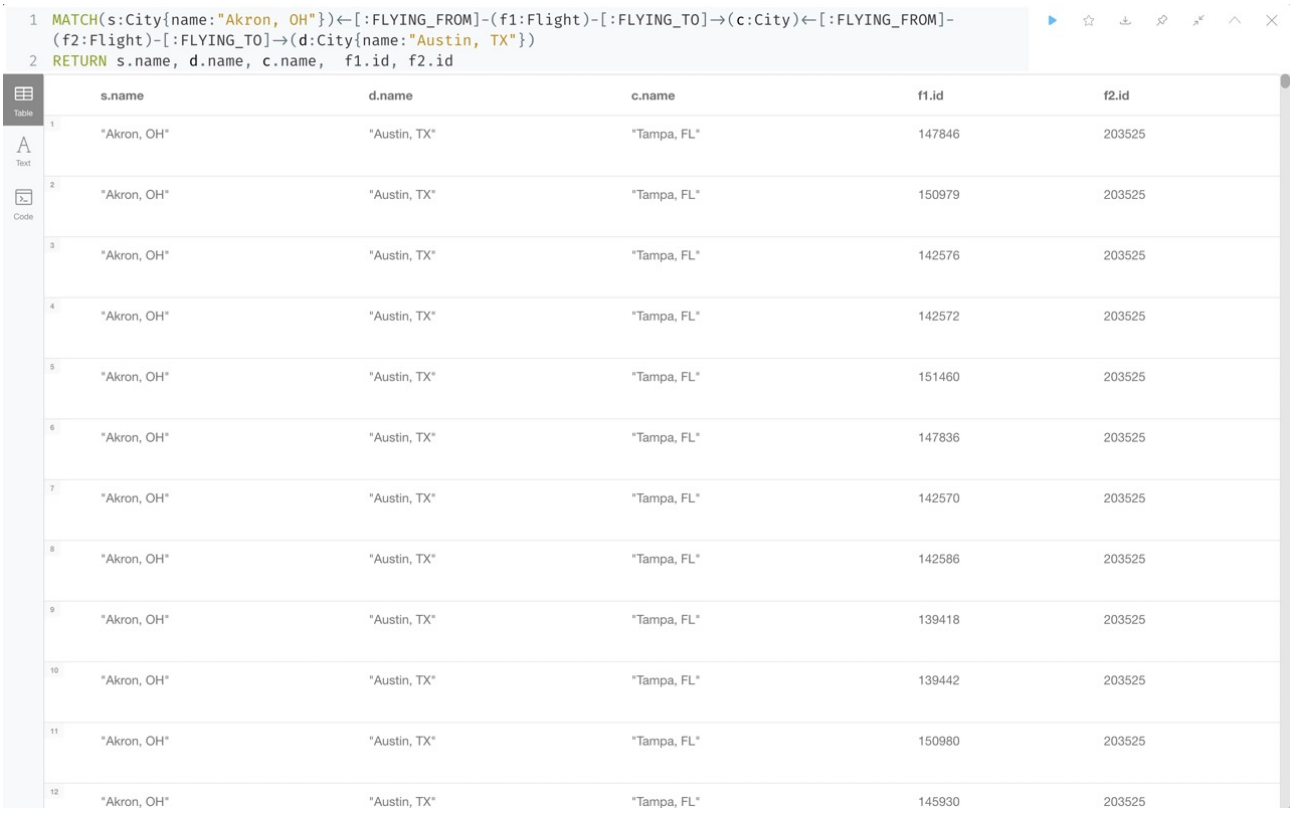
2.4 : Show all flights that that go from city A to city B with a changeover in city C, in other words, there is no direct flight from A to B. Only one changeover is required. You may choose cities A and B. Output cities A, B, C and ROW_ID in

- tabular format
- graph format

Source Code:

```
MATCH(s:City{name:"Akron, OH"})<-[:FLYING_FROM]-(f1:Flight)-[:FLYING_TO]->(c:City)<-
[:FLYING_FROM]-(f2:Flight)-[:FLYING_TO]->(d:City{name:"Austin, TX"})

RETURN s.name, d.name, c.name,  f1.id, f2.id //for Tabular view


MATCH(s:City{name:"Akron, OH"})<-[:FLYING_FROM]-(f1:Flight)-[:FLYING_TO]->(c:City)<-
[:FLYING_FROM]-(f2:Flight)-[:FLYING_TO]->(d:City{name:"Austin, TX"})

RETURN * //for Graph view
```

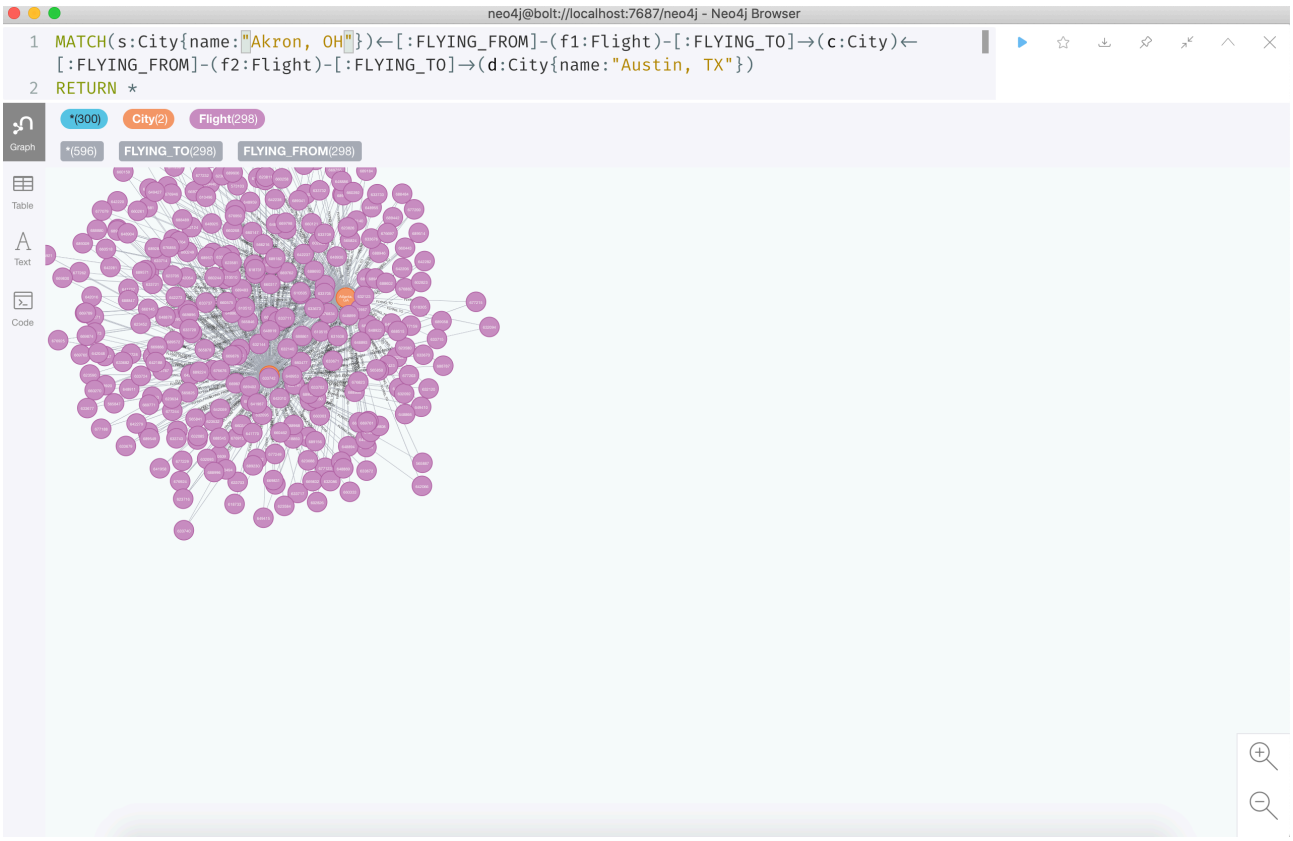
Screenshots:

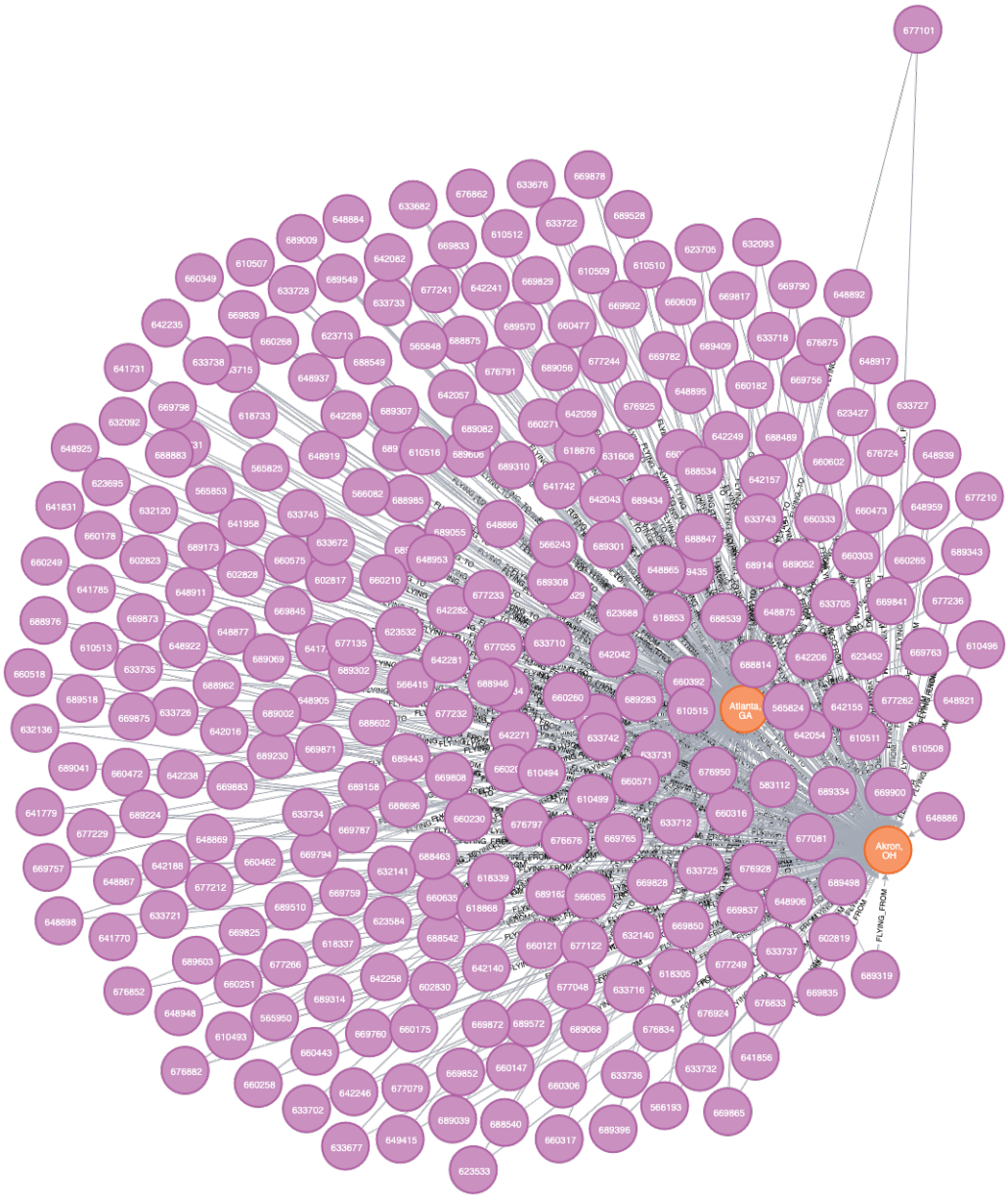


```
1 MATCH(s:City{name:"Akron, OH"})←[:FLYING_FROM]-(f1:Flight)-[:FLYING_TO]→(c:City)←[:FLYING_FROM]-
  (f2:Flight)-[:FLYING_TO]→(d:City{name:"Austin, TX"})
2 RETURN s.name, d.name, c.name,  f1.id, f2.id
```

Table | Text | Code

	s.name	d.name	c.name	f1.id	f2.id
1	"Akron, OH"	"Austin, TX"	"Tampa, FL"	147846	203525
2	"Akron, OH"	"Austin, TX"	"Tampa, FL"	150979	203525
3	"Akron, OH"	"Austin, TX"	"Tampa, FL"	142576	203525
4	"Akron, OH"	"Austin, TX"	"Tampa, FL"	142572	203525
5	"Akron, OH"	"Austin, TX"	"Tampa, FL"	151460	203525
6	"Akron, OH"	"Austin, TX"	"Tampa, FL"	147836	203525
7	"Akron, OH"	"Austin, TX"	"Tampa, FL"	142570	203525
8	"Akron, OH"	"Austin, TX"	"Tampa, FL"	142586	203525
9	"Akron, OH"	"Austin, TX"	"Tampa, FL"	139418	203525
10	"Akron, OH"	"Austin, TX"	"Tampa, FL"	139442	203525
11	"Akron, OH"	"Austin, TX"	"Tampa, FL"	150980	203525
12	"Akron, OH"	"Austin, TX"	"Tampa, FL"	145930	203525


neo4j@bolt://localhost:7687/neo4j - Neo4j Browser
1 MATCH(s:City{name:"Akron, OH"})←[:FLYING_FROM]-(f1:Flight)-[:FLYING_TO]→(c:City)←
[:FLYING_FROM]-(f2:Flight)-[:FLYING_TO]→(d:City{name:"Austin, TX"})
2 RETURN *
Graph
Table
Text
Code
*(300)
City(2)
Flight(298)
*(596)
FLYING_TO(298)
FLYING_FROM(298)


Atlanta, GA
Akron, OH
FLYING_FROM

2.5 : Count the number of cities with more than one airport. Output the number of cities

Source Code:

```
MATCH(c:City)
WITH c.name as city, count(c.airport_code) as airports
WHERE airports >1
RETURN count(city) as number_of_cities //Using count keyword
```

Screenshots:

2.6 : Find the longest flight in terms of distance. Output the distance.

Source Code:

```
MATCH(f:Flight)
RETURN f.distance as distance
ORDER BY distance DESC
LIMIT 1 //Using ORDER BY
```

Screenshot: