MSC IN BUSINESS ANALYTICS AT AUEB
COURSE: MINING BIG DATASETS
PROFESSOR: Y.KOTIDIS
ASSISTANT: I.FILIPPIDOU

ASSIGNMENT #2

ANALYSING AIRPORTS NETWORK USING NEO4J

// DESPOTIS SPYRIDON P2822111
// PAPAILIOU THANASIS P2822128

2022

Contents

We live in a connected world, and understanding most domains requires processing rich sets of connections to understand what's really happening. Often, we find that the connections between items are as important as the items themselves.[1]

Neo4j is an open-source, NoSQL, native graph database that provides an ACID-compliant transactional backend for your applications that has been publicly available since 2007. Leading telcos like Verizon, Orange, Comcast, and AT&T rely on Neo4j to manage networks, control access, and enable customer 360.

In this assignment we are trying to analyse part OpenFlights Airports network using Neo4J. The given dataset contains data about 698 Airports 6161 Airlines and 65935 flight routes. We start our analysis by designing a graph scheme to have a straightforward view of the nodes and their relationships. Then we proceed by answering the queries in Cypher language.

Figure 0-1 Image Presenting The Mining Big Datasets Project

[1] https://neo4j.com/developer/graph-database/

1. Schema Design & Data Import

For the purpose of this assignment we decided in our schema to represent Airports (:Airport) and Airlines (:Airline) as nodes and flight routes (:Route) as edges between the Airports. Airlines are connected to Airports they fly from (:fliesfrom) and to (:fliesto) respectively. You can see this design in the Figure above.

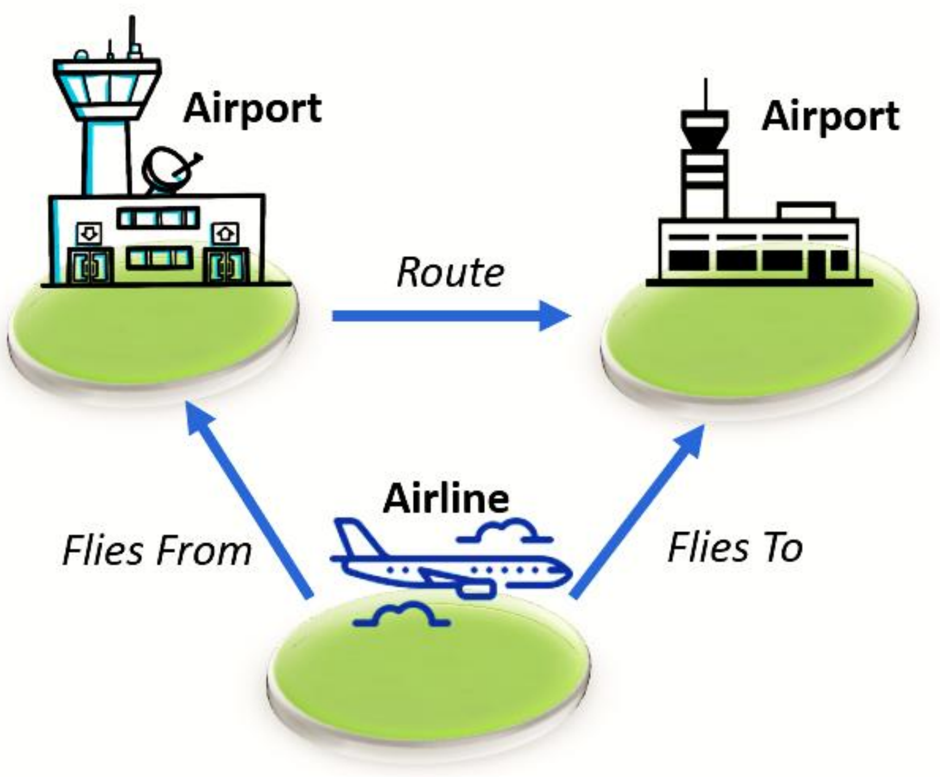


Figure 1-1: Schema design with nodes and edges

The dataset contains additional information about each entity, which we encode as properties in order to facilitate the querying process that will be presented later. Specifically, the properties of our nodes (Airports, Airlines, and Routes) are shown in the tables above:

Property	CSV Column	Description
airportid	AirportID	Unique ID of an airport
airportname	Name	Name of the airport
airportcity	City	City of the airport
airportcountry	Country	Country of the airport
airportlatitude	Latitude	Geolocation of the airport
airportlongitude	Longtitude	Geolocation of the airport

Table 1: Properties of the Airport nodes

Property	CSV Column	Description
airlineid	AirlineID	Unique ID of the airline
airlinename	Name	Name of the airline
airlinecountry	Country	Country of the airline

Table 2 : Properties of the Airline nodes

Property	CSV Column	Description
airlineid	AirlineID	Unique ID of the airline that operates this route
equipment	Equipment	Array containing 3-letter codes that correspond to the kind of planes that are used in this route
stop	Stops	True if this route is no direct flight (Stops <> 0)

Table 3: Properties of Route edges

The edges used to connect Airlines to Airports do not have any properties. To ensure current results and speed up queries, we create indices for *airportid* and *airlineid*.

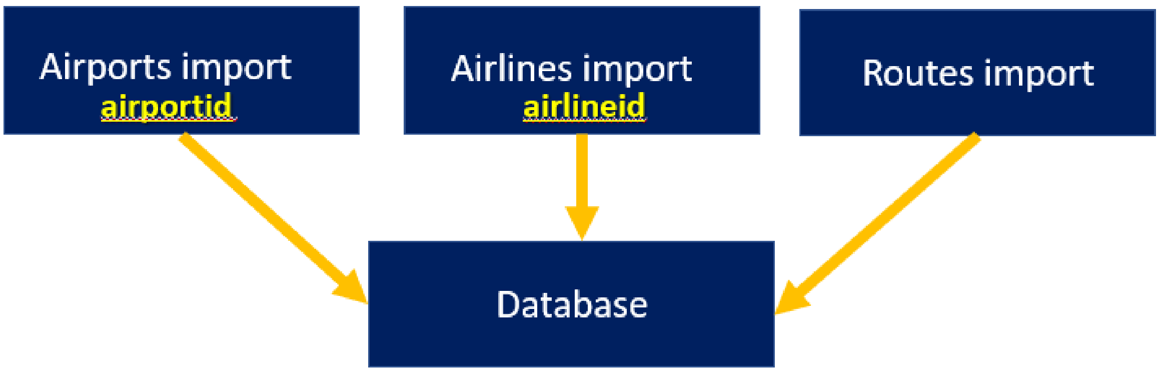


Figure 1-2 Describing the Import Process

To load the dataset in Neo4J, we used the build-in “LOAD CSV WITH HEADERS” command. After placing the CSV files in the *Import* directory, we execute the Cypher code above:

```
// 1. Airports Import
LOAD CSV WITH HEADERS
FROM 'file:///airports.csv' AS row
CREATE (:Airport {
airportid: row.AirportID,
airportname: row.Name,
airportcity: row.City,
airportcountry: row.Country,
airportlatitude: row.Latitude,
airportlongitude: row.Longitude});

// 2. Airlines Import
LOAD CSV WITH HEADERS
FROM 'file:///airlines.csv' AS row
CREATE (:Airline {
airlineid: row.AirlineID,
airlinename: row.Name,
airlinecountry: row.Country});

// 3. Create indices for airportid and airlineid
CREATE CONSTRAINT FOR (c:Airport) REQUIRE c.airportid IS
UNIQUE;
CREATE CONSTRAINT FOR (c:Airline) REQUIRE c.airlineid IS
UNIQUE;

// 4. Routes Import
LOAD CSV WITH HEADERS
FROM 'file:///routes.csv' AS row
MATCH (source:Airport {airportid: row.SourceID})
MATCH (destination:Airport {airportid: row.DestinationID})
MATCH (airline:Airline {airlineid: row.AirlineID})
CREATE (source) -[route:Route {
   airlineid: row.AirlineID,
   equipment: split(row.Equipment, " "),
   stop: row.Stops <> "0"
   }]-> (destination)
MERGE (airline) -[:fliesto]-> (source)
MERGE (airline) -[:fliesfrom]-> (destination)
```

After the creation of the database, in this section we will create the cypher code to answer the following queries.

1. ***Which are the top 5 airports with the most flights? Return airport name and number of flights.***

✓ **Query**

```
MATCH (src:Airport) -[route:Route]-> (:Airport)
RETURN src.airportname as Airport, count(route) as Flights
ORDER BY Flights DESC LIMIT 5;
```

✓ **Output**

	Airport	Flights
1.	"Hartsfield Jackson Atlanta International Airport"	915
2.	"Chicago O'Hare International Airport"	558
3.	"Beijing Capital International Airport"	531
4.	"London Heathrow Airport"	525
5.	"Charles de Gaulle International Airport"	524

2. ***Which are the top 5 countries with the most airports? Return country name and number of airports.***

✓ **Query**

```
MATCH (src:Airport)
RETURN src.airportcountry as CountryName,
count(src.airportname) as AirportCount
ORDER BY AirportCount DESC
LIMIT 5;
```

✓ **Output**

	CountryName	AirportCount
1.	"United States"	1512
2.	"Canada"	430
3.	"Australia"	334
4.	"Brazil"	264
5.	"Russia"	264

3. ***<u>Which are the top 5 airlines with international flights from/to 'Greece'. Return airline name and number of flights ?</u>***

✓ **<u>Query</u>**

```
MATCH (airline:Airline) -[route]-> (airport:Airport)
WHERE airport.airportcountry = "Greece"
RETURN airline.airlinename as Airline, count(route) as Flights
ORDER BY Flights DESC LIMIT 5;
```

✓ **<u>Output</u>**

	Airline	Flights
1.	"Aegean Airlines"	68
2.	"Olympic Airlines"	66
3.	"Big Sky Airlines"	38
4.	"Transavia Holland"	36
5.	"Air Berlin"	31

4. ***<u>Which are the top 5 airlines with local flights inside 'Germany'. Return airline name and number of flights.</u>***

✓ **<u>Query</u>**

```
MATCH (a:Airline) - [ffrom:fliesfrom] -> (airport:Airport) <-
     [fto:fliesto] - (a:Airline)

WHERE airport.airportcountry="Germany"

RETURN a.airlinename, count(airport) as Flights

ORDER BY Flights DESC LIMIT 5;
```

✓ **<u>Output</u>**

	a.airlinename	Flights
1.	"Air Berlin"	21
2.	"Lufthansa"	19
3.	"Germanwings"	18
4.	"Germania"	16
5.	"SunExpress"	13

5. ***Which are the top 10 countries with flights to Greece. Return country name and number of flights.***

✓ **Query**

```
MATCH (src:Airport) -[route:Route]-> (dest:Airport)
WHERE dest.airportcountry  = "Greece" AND src.airportcountry
<> "Greece"
RETURN src.airportcountry  as Country, count(route) as Flights
ORDER BY Flights DESC LIMIT 10;
```

✓ **Output**

Country	Flights
1. "Germany"	176
2. "United Kingdom"	105
3. "Austria"	36
4. "France"	32
5. "Russia"	28
6. "Italy"	25
7. "Netherlands"	21
8. "Belgium"	20
9. "Cyprus"	12
10. "Norway"	11

6. ***Find the percentage of air traffic (inbound and outbound) for every city in Greece? Return city name and the corresponding traffic percentage in descending order.***

✓ **Query**

```
MATCH (src:Airport) -[outRoute:Route]-> (dest:Airport)
WHERE src.airportcountry = "Greece" AND dest.airportcountry <>
"Greece"
WITH collect(outRoute) as outRoutes
WITH reduce(a = 0, n in outRoutes | a + 1) as totaloutRoutes
MATCH (src:Airport) -[inRoute:Route]-> (dest:Airport)
WHERE dest.airportcountry = "Greece" AND src.airportcountry <>
"Greece"
WITH totaloutRoutes,
collect(inRoute) as inRoutes
```

```
WITH totaloutRoutes,
reduce(a = 0, n in inRoutes | a + 1) as totalinRoutes
MATCH
(overseas:Airport) -[inRoute:Route]-> (domestic:Airport),
(domestic:Airport) -[outRoute:Route]-> (overseas:Airport)
WHERE domestic.airportcountry = "Greece" AND
overseas.airportcountry <> "Greece"
RETURN domestic.airportcity as City,
round(toFloat(count(outRoute))/totaloutRoutes * 100, 2) as
OutboundPercentage,
round(toFloat(count(inRoute))/totalinRoutes * 100, 2) as
InboundPercentage
ORDER BY OutboundPercentage DESC
```

✓ **Output**

City	OutboundPercentage	InboundPercentage
1. "Athens"	58.33	55.61
2. "Heraklion"	45.65	43.52
3. "Rhodos"	25.36	24.18
4. "Thessaloniki"	25.0	23.83
5. "Kerkyra/corfu"	13.95	13.3
6. "Kos"	12.5	11.92
7. "Chania"	8.33	7.94
8. "Thira"	3.8	3.63
9. "Zakynthos"	3.62	3.45
10. "Kalamata"	2.54	2.42
11. "Mykonos"	2.36	2.25
12. "Preveza"	2.17	2.07
13. "Nea Anghialos"	1.09	1.04
14. "Kavala"	0.72	0.69
15. "Samos"	0.72	0.69
16. "Patras"	0.54	0.52
17. "Keffallinia"	0.36	0.35
18. "Mytilini"	0.36	0.35

7. ***Find the number of international flights to Greece with plane types '738' and '320'. Return for each plane type the number of flights***

✓ **Query**

```
MATCH (:Airport) -[route:Route]-> (dest:Airport)
WHERE dest.airportcountry = "Greece"
UNWIND route.equipment as PlaneType
WITH PlaneType, route
WHERE PlaneType = "738" OR PlaneType = "320"
RETURN PlaneType, count(route) as Flights
ORDER BY Flights;
```

✓ **Output**

	PlaneType	Flights
1.	"738"	134
2.	"320"	274

8. ***Which are the top 5 flights that cover the biggest distance between two airports (use function point({ longitude: s1.longitude, latitude: s1.latitude }) and function distance(point1, point2)). Return From (airport), To (airport) and distance in km.***

✓ **Query**

```
MATCH (src:Airport) - [r:Route] -> (sss:Airport)
SET src.airportlatitude = toFloat(src.airportlatitude),
src.airportlongitude=toFloat(src.airportlongitude),
sss.airportlatitude= toFloat(sss.airportlatitude),
sss.airportlongitude = toFloat(sss.airportlongitude)
WITH point({ longitude: src.airportlongitude,latitude: src.air
portlatitude }) AS
p1, point({ longitude: sss.airportlongitude, latitude: sss.air
portlatitude
}) AS p2, src, sss
RETURN src.airportname AS From, sss.airportname AS To, point.d
istance(p1, p2) AS Distance
ORDER BY Distance DESC LIMIT 10;
```

✓ **Output**

From	To	Distance
1. "Los Alamitos Army Air Field"	**"Kenneth Kaunda International Airport Lusaka"**	16100278.634125533
2. "Los Alamitos Army Air Field"	**"Simon Mwansa Kapwepwe International Airport"**	15954975.648145452
3. "Sydney Kingsford Smith International Airport"	**"Dallas Fort Worth International Airport"**	13823653.123181123
4. "OR Tambo International Airport"	**"Hartsfield Jackson Atlanta International Airport"**	13597809.92728571
5. "Dubai International Airport"	**"Los Angeles International Airport"**	13415094.537073221

9. ***Find 5 cities that are not connected with direct flights to 'Berlin'. Score the cities in descending order with the total number of flights to other destinations. Return city name and score.***

✓ **Query**

```
MATCH (source:Airport) -[route:Route]-> (dest:Airport)
WHERE dest.airportcity = "Berlin" AND NOT(route.stop)

WITH collect(source) as AirportsDirectlyConnectedWithBerlin
MATCH (source:Airport) -[route:Route]-> (dest:Airport)
WHERE NOT source in AirportsDirectlyConnectedWithBerlin
RETURN source.airportcity as City, count(route) as Flights
ORDER BY Flights DESC LIMIT 5;
```

✓ **Output**

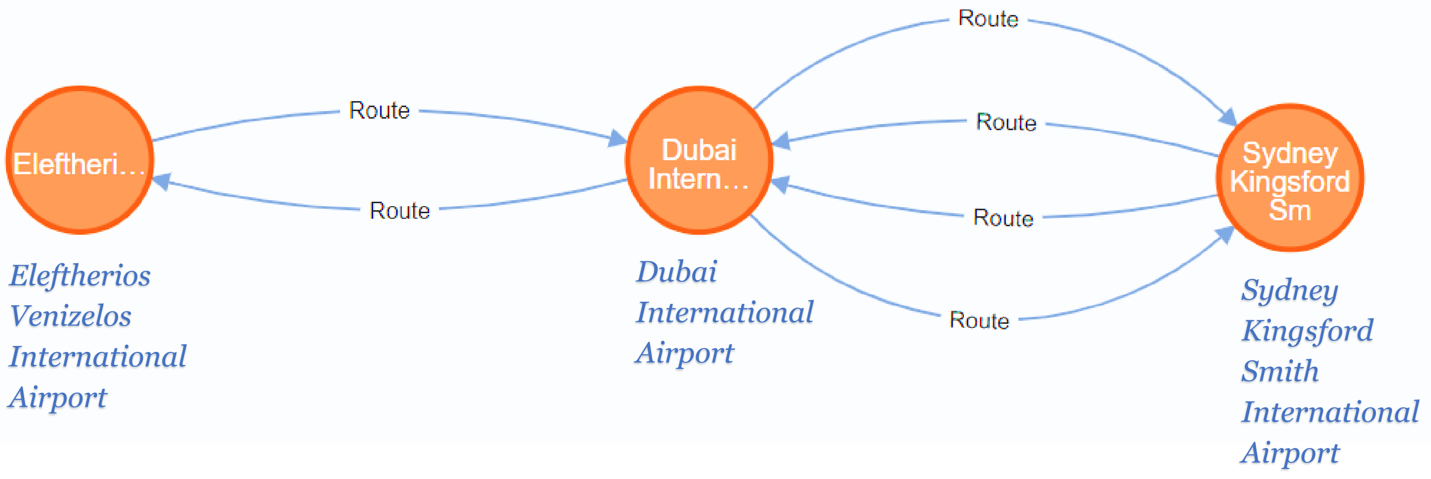
City	Flights
1. "Atlanta"	915
2. "Shanghai"	610
3. "Los Angeles"	489
4. "Dallas-Fort Worth"	469
5. "Tokyo"	443

10. ***Find all shortest paths from 'Athens' to 'Sydney'. Use only relations between flights and city airports.***

✓ **Query**

```
MATCH (source:Airport {
airportname: "Eleftherios Venizelos International Airport"}),
(dest:Airport {
airportname: "Sydney Kingsford Smith International Airport"})
RETURN source, dest, shortestPath((source)-[:Route*]-(dest))
```

✓ **Output**



Eleftheri…
Route
Route
Dubai Intern…
Route
Route
Route
Route
Sydney Kingsford Sm
Eleftherios Venizelos International Airport
Dubai International Airport
Sydney Kingsford Smith International Airport

Figure 0-1 Image Describing the Mining Big Datasets project

This assignment was a great opportunity to have a deep dive into NEO4J database and Cypher language. Through our analysis, we found out that NEO4J was very easy to learn and a great tool to present connected data. However, it requires special attention when importing data to database because this step is crucial for the later analysis. Moreover, NEO4J has a great performance and low memory footprints and the ouputs from our queries were presented fast. Also, our approach was to create efficient queries to decrease the response time, prevent the excessive consumption of resources, and identify poor query performance. Finally, a great feature is that NEO4J, was free to download and experiment.