

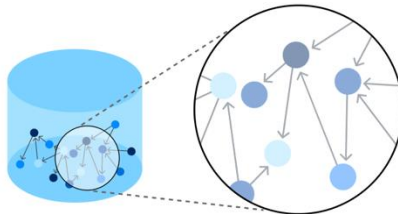


Athens University of Economics and Business
Department of Management Science and Technology

Mining Big Datasets

M.Sc. Business Analytics
Part Time 2021-2023

Neo4j Graph database



Maria Zafeiropoulou
A.M: P2822113
&
Marianna Konstantopoulou
A.M: P2822122

Athens, 03/07/2022

Part I: Importing the datasets and creating the graph

In this assignment we were instructed to represent the data as a property graph by creating the necessary entities and giving the required names, types, and properties. The graph we created for its implementation is fully examined below. We will utilize a diagram to better convey our thoughts:

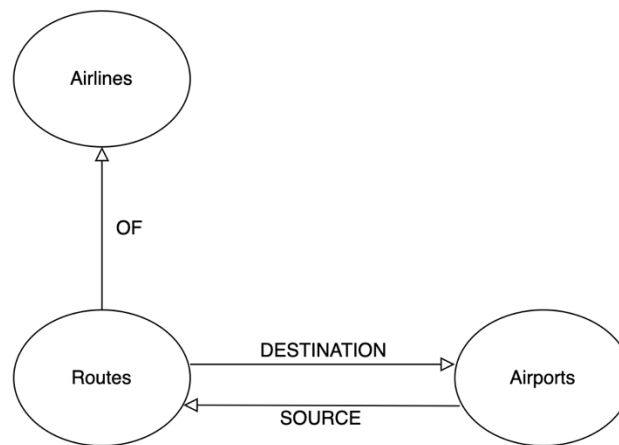


Figure 1: Theoretical graph model chart

After reviewing all the files that define the airline network and represent the necessary properties on nodes and edges of a graph, we decided that it is best to create 3 nodes (Airlines, Airports and Routes) that are connected with the following relationships:

- **Source:** from airport to route
- **Destination:** from route to airport
- **Of:** from route to airline

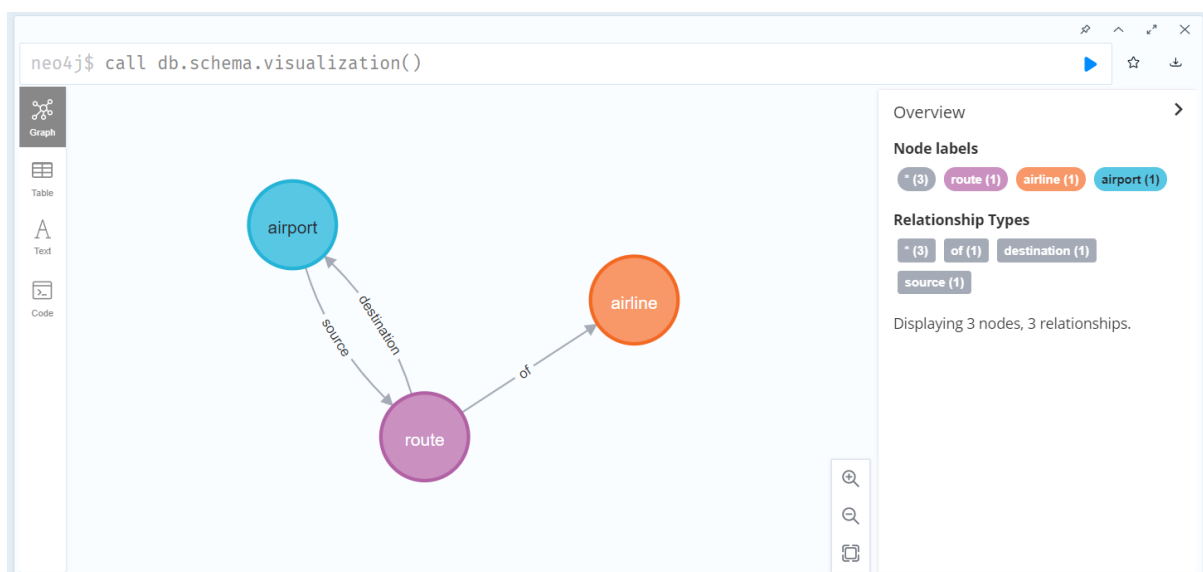


Figure 2: Graph model chart extracted from Neo4j

Firstly, we loaded the CSV files to the Neo4j database.

The image shows the Neo4j Cypher query editor interface. The query entered is: `neo4j$ LOAD CSV WITH HEADERS FROM "file:///airlines.csv" AS LINE RETURN LINE limit 1`. The result is displayed in a table view with the column header "LINE". The first record is a JSON object: `{ "AirlineID": "-1", "Active": "Y", "Callsign": "\N", "Alias": "\N", "IATA": "-", "Country": "\N", "ICAO": "N/A", "Name": "Unknown" }`. The status bar at the bottom indicates: "Started streaming 1 records after 2 ms and completed after 98 ms."

```
neo4j$ LOAD CSV WITH HEADERS FROM "file:///airlines.csv" AS LINE
RETURN LINE limit 1
```

LINE
<pre>{ "AirlineID": "-1", "Active": "Y", "Callsign": "\N", "Alias": "\N", "IATA": "-", "Country": "\N", "ICAO": "N/A", "Name": "Unknown" }</pre>

Started streaming 1 records after 2 ms and completed after 98 ms.

Figure 3: Loading CSV file for Airlines

The image shows the Neo4j Cypher query editor interface. The query entered is: `neo4j$ LOAD CSV WITH HEADERS FROM "file:///airports.csv" AS LINE RETURN LINE limit 1`. The result is displayed in a table view with the column header "LINE". The first record is a JSON object: `{ "Timezone": "10", "DST": "U", "Tz": "Pacific/Port_Moresby", "IATA": "GKA", "Latitude": "-6.081689834590001", "ICAO": "AYGA", "City": "Goroka", "Longitude": "145.391998291", "Source": "OurAirports", "Name": "Goroka Airport", "Type": "airport", "AirportID": "1", "Country": "Papua New Guinea", "Altitude": "5282" }`. The status bar at the bottom indicates: "Started streaming 1 records after 2 ms and completed after 29 ms."

```
neo4j$ LOAD CSV WITH HEADERS FROM "file:///airports.csv" AS LINE
RETURN LINE limit 1
```

LINE
<pre>{ "Timezone": "10", "DST": "U", "Tz": "Pacific/Port_Moresby", "IATA": "GKA", "Latitude": "-6.081689834590001", "ICAO": "AYGA", "City": "Goroka", "Longitude": "145.391998291", "Source": "OurAirports", "Name": "Goroka Airport", "Type": "airport", "AirportID": "1", "Country": "Papua New Guinea", "Altitude": "5282" }</pre>

Started streaming 1 records after 2 ms and completed after 29 ms.

Figure 4: Loading CSV file for Airports

The screenshot shows the Neo4j Cypher Shell interface. The command entered is `neo4j$ LOAD CSV WITH HEADERS FROM "file:///routes.csv" AS LINE RETURN LINE limit 1`. The result is displayed in a table view with one record. The record is a JSON object representing a route entry.

LINE
<pre>{ "AirlineID": "410", "Destination": "KZN", "Equipment": "CR2", "Airline": "2B", "SourceID": "2965", "Stops": "0", "Codeshare": null, "Routeid": "1", "DestinationID": "2990", "Source": "AER" }</pre>

Started streaming 1 records after 6 ms and completed after 71 ms.

Figure 5: Loading CSV file for Routes

Then, we created all nodes indexes (unique constraint).

The screenshot shows the Neo4j Cypher Shell interface with three commands entered to create unique constraints on the nodes:

```
1 CREATE CONSTRAINT ON (a:airport) ASSERT a.airportid IS UNIQUE;
2 CREATE CONSTRAINT ON (a:airline) ASSERT a.airlineid IS UNIQUE;
3 CREATE CONSTRAINT ON (r:route) ASSERT r.routeid IS UNIQUE;
```

Below the commands, the execution results are shown, each with a green checkmark indicating success:

```
neo4j$ CREATE CONSTRAINT ON (a:airport) ASSERT a.airportid IS UNIQU... ✓
neo4j$ CREATE CONSTRAINT ON (a:airline) ASSERT a.airlineid IS UNIQU... ✓
neo4j$ CREATE CONSTRAINT ON (r:route) ASSERT r.routeid IS UNIQUE ✓
```

Figure 6: Creating all nodes indexes

And finally, we had to create the nodes with the selected attributes.
For Airports we chose:

Attributes

AirportID

Name

Country

City

Latitude

Longitude

IATA

```
1 LOAD CSV WITH HEADERS FROM "file:///airports.csv" AS line
2 CREATE (a:airport{airportID: ToInteger(line.AirportID)})
3 SET a.name = line.Name,
4 a.country = line.Country,
5 a.city = line.City,
6 a.latitude = ToFloat(line.Latitude),
7 a.longitude = ToFloat(line.Longitude),
8 a.iata = line.IATA
```

Added 7698 labels, created 7698 nodes, set 53886 properties, completed after 646 ms.

Figure 7: Creating Airports nodes

For Airlines we chose:

Attributes

AirlineID

Name

Country



Figure 8: Creating Airlines nodes

For Routes we chose:

Attributes

Routeid

SourceID

DestinationID

Equipment

Destination

Source

AirlineID

Stops

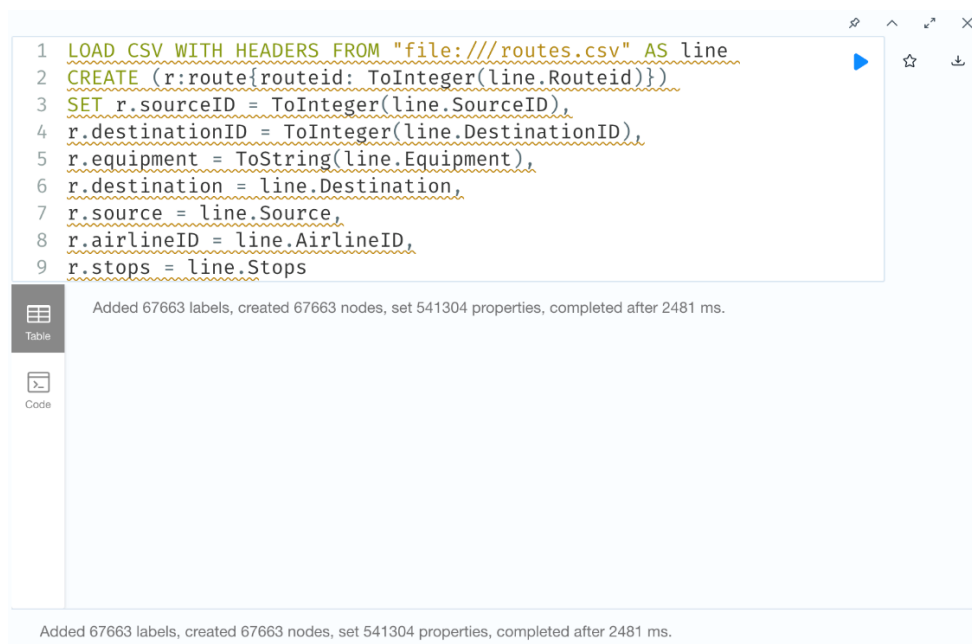


Figure 9: Creating Routes nodes

The screenshot shows a Cypher query editor with a light blue background. At the top right, there are icons for search, undo, redo, and close. The query is as follows:

```
1 LOAD CSV WITH HEADERS
2 FROM "file:///airports.csv" AS line
3 MATCH (airport:airport {iata: line.IATA })
4 MATCH (route:route {source: line.IATA })
5 CREATE (airport)-[:source]→(route)
```

Below the query, there is a status bar that says "Created 67257 relationships, completed after 463832 ms." On the left side, there is a sidebar with two icons: a table icon labeled "Table" and a code icon labeled "Code".

Figure 10: Creating “source” relationship from airport to route

The screenshot shows a Cypher query editor with a light blue background. At the top right, there are icons for search, undo, redo, and close. The query is as follows:

```
1 LOAD CSV WITH HEADERS
2 FROM "file:///airports.csv" AS line
3 MATCH (airport:airport {iata: line.IATA })
4 MATCH (route:route {destination: line.IATA })
5 CREATE (route)-[:destination]→(airport)
```

Below the query, there is a status bar that says "Created 67247 relationships, completed after 397101 ms." On the left side, there is a sidebar with two icons: a table icon labeled "Table" and a code icon labeled "Code".

Figure 11: Creating “destination” relationship from route to airport

The screenshot shows a Neo4j Cypher query editor interface. The query is as follows:

```
1 LOAD CSV WITH HEADERS
2 FROM "file:///routes.csv" AS line
3 MATCH (airline:airline {airlineID: ToInteger(line.AirlineID)
4   })
5 MATCH (route:route {routeid: ToInteger(line.Routeid) })
6 CREATE (route)-[:of]-(airline)
```

Below the query editor, a status bar indicates: "Created 67184 relationships, completed after 196626 ms." The interface includes a sidebar with "Table" and "Code" views, and a top toolbar with icons for undo, redo, and other editor functions.

Figure 12: Creating “of” relationship from route to airline

Part II: Querying the database

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

```
1 MATCH (airport1: airport)-[:source]→(route1: route)
2 MATCH (route2: route)-[:destination]→(airport1: airport)
3 WITH count(distinct(route1.routeid)) AS In, count(distinct(route2.routeid)) AS Out, airport1.name
   AS Airport_Name
4 RETURN Airport_Name, (In+Out) AS Number_of_routes
5 ORDER BY Number_of_routes DESC
6 LIMIT 5;
```

	Airport_Name	Number_of_routes
1	"Hartsfield Jackson Atlanta International Airport"	1826
2	"Chicago O'Hare International Airport"	1108
3	"Beijing Capital International Airport"	1069
4	"London Heathrow Airport"	1051
5	"Charles de Gaulle International Airport"	1041

Started streaming 5 records after 21 ms and completed after 3052 ms.

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

```
1 MATCH (airport)
2 RETURN airport.country AS Country, COUNT(distinct airport.airportID) AS Number_Of_Airports
3 ORDER BY Number_Of_Airports DESC
4 LIMIT 5;
```

	Country	Number_Of_Airports
1	"United States"	1512
2	"Canada"	430
3	"Australia"	334
4	"Brazil"	264
5	"Russia"	264

Started streaming 5 records after 12 ms and completed after 154 ms.

3) Which are the top 5 airlines with international flights from/to 'Greece'. Return airline name and number of flights.

```

1 MATCH (airport1: airport)
2 WHERE NOT airport1.country = 'Greece'
3 MATCH (airport1:airport)-[:source]→(route1: route)
4 MATCH (route1: route)-[:destination]→(airport2: airport {country: 'Greece'})
5 MATCH (route1: route)-[:of]→(airline)
6 MATCH (airport3: airport)
7 WHERE NOT airport3.country = 'Greece'
8 MATCH (a4:airport {country: 'Greece'})-[:source]→(route2: route)
9 MATCH (route2: route)-[:destination]→(airport3: airport)
10 MATCH (route2: route)-[:of]→(airline)
11 WITH count(distinct(route1.routeid)) AS to_Greece, count(distinct(route2.routeid)) AS from_Greece,
    airline.name AS airline_Name

```

	airline_Name	number_of_routes
1	"Ryanair"	150
2	"Aegean Airlines"	136
3	"Air Berlin"	120
4	"easyJet"	80
5	"TUIfly"	64

Started streaming 5 records after 63 ms and completed after 2032 ms.

4) Which are the top 5 airlines with local flights inside 'Germany'. Return airline name and number of flights.

```

1 MATCH (airport1:airport {country: 'Germany'})-[:source]→(route1: route)
2 MATCH (route1: route)-[:destination]→(airport2:airport {country: 'Germany'})
3 MATCH (route1: route)-[:of]→(airline)
4 RETURN airline.name AS Airline, COUNT(*) AS Flights
5 ORDER BY Flights DESC
6 LIMIT 5;

```

	Airline	Flights
1	"Lufthansa"	64
2	"Germanwings"	54
3	"Air Berlin"	44
4	"Hainan Airlines"	16
5	"Ethiopian Airlines"	6

Started streaming 5 records after 19 ms and completed after 46 ms.

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

```

1 MATCH (airport1: airport)
2 WHERE NOT airport1.country = 'Greece'
3 MATCH (airport1:airport)-[:source]→(route1: route)
4 MATCH (route1: route)-[:destination]→(a2: airport {country: 'Greece'})
5 MATCH (route1: route)-[:of]→(airline)
6 RETURN airport1.country as Country, COUNT(*) AS Flights
7 ORDER BY Flights DESC
8 LIMIT 10;
9

```

	Country	Flights
1	"Germany"	176
2	"United Kingdom"	105
3	"Austria"	36
4	"France"	32
5	"Russia"	28
6	"Italy"	25
7		

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.

```

1 CALL{MATCH (airport1: airport{country: 'Greece'})-[:source]→(route1: route)
2 MATCH (route2: route)-[:destination]→(airport1)
3 WITH count(distinct(route1.routeid)) AS Inb, count(distinct(route2.routeid)) AS Out
4 RETURN (Inb+Out) AS Total}
5 MATCH (airport1: airport{country: 'Greece'})-[:source]→(route1: route)
6 MATCH (route2: route)-[:destination]→(airport1)
7 WITH count(distinct(route1.routeid)) AS Inb1, count(distinct(route2.routeid)) AS Out1,
8 Total,airport1.city AS City
9 RETURN City, ROUND(((Inb1+Out1)*100.00)/Total,2) AS Traffic
10 ORDER BY Traffic DESC;

```

	City	Traffic
1	"Athens"	25.19
2	"Heraklion"	13.69
3	"Thessaloniki"	10.88
4	"Rhodos"	10.5
5	"Kerkyra/corfu"	6.44
6	"Kos"	5.63

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

```

1 MATCH (airport1: airport)
2 WHERE NOT airport1.country = 'Greece'
3 MATCH (route1 :route)
4 WHERE route1.equipment = '738' OR route1.equipment = '320'
5 MATCH (airport1: airport)-[:source]→(route1:route)
6 MATCH (route1:route)-[:destination]→(airport2: airport {country: 'Greece'})
7 MATCH (route1:route)-[:of]→(airline1: airline)
8 RETURN route1.equipment AS Airplane_Type, COUNT(*) AS Flights
9 ORDER BY Flights DESC;

```

	Airplane_Type	Flights
1	"320"	147
2	"738"	110

Started streaming 2 records after 20 ms and completed after 40 ms.

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.

```

1 MATCH (airport1: airport)-[:source]→(route1: route)
2 MATCH (route1)-[:destination]→(airport2: airport)
3 WITH airport1.name AS From, airport2.name AS To, ROUND(distance(point({ longitude:
airport1.longitude, latitude: airport1.latitude }),point({ longitude: airport2.longitude, latitude:
airport2.latitude })),4) AS Dist
4 WHERE From < To
5 RETURN distinct From, To, Dist
6 ORDER BY Dist DESC
7 LIMIT 5;

```

	From	To	Dist
1	"Hartsfield Jackson Atlanta International Airport"	"OR Tambo International Airport"	13597809.9273
2	"Dubai International Airport"	"Los Angeles International Airport"	13415094.5371
3	"King Abdulaziz International Airport"	"Los Angeles International Airport"	13404826.0739
4	"Dubai International Airport"	"George Bush Intercontinental Houston Airport"	13139523.5681
5	"Dubai International Airport"	"San Francisco International Airport"	13034662.8697

Started streaming 5 records after 26 ms and completed after 836 ms.

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.

```

1 CALL{
2 MATCH (airport1: airport)-[:source]→(route1: route)
3 MATCH (route1: route)-[:destination]→(airport2: airport{city:'Berlin'})
4 WITH airport1.name AS air_name
5 RETURN COLLECT(air_name) AS non_wanted_airports}
6 MATCH (airport3: airport)
7 WHERE NOT airport3.city = 'Berlin'
8 MATCH (airport4:airport)
9 WHERE NOT airport4.city = 'Berlin'
10 MATCH (airport3)-[:source]→(route2:route)
11 MATCH (route2)-[:destination]→(airport4)
12 WHERE NOT airport3.name in non_wanted_airports

```

	City	Flights
1	"Atlanta"	915
2	"Shanghai"	606
3	"Los Angeles"	489
4	"Dallas-Fort Worth"	469
5	"Tokyo"	443

Started streaming 5 records after 266 ms and completed after 636 ms.

```

neo4j$
5 RETURN COLLECT(air_name) AS non_wanted_airports}
6 MATCH (airport3: airport)
7 WHERE NOT airport3.city = 'Berlin'
8 MATCH (airport4:airport)
9 WHERE NOT airport4.city = 'Berlin'
10 MATCH (airport3)-[:source]→(route2:route)
11 MATCH (route2)-[:destination]→(airport4)
12 WHERE NOT airport3.name in non_wanted_airports
13 WITH airport3.city as City, count(distinct(route2.routeid)) as Flights
14 RETURN distinct City, Flights
15 ORDER BY Flights DESC
16 LIMIT 5;

```

	City	Flights
1	"Atlanta"	915
2	"Shanghai"	606
3	"Los Angeles"	489
4	"Dallas-Fort Worth"	469
5	"Tokyo"	443

Started streaming 5 records in less than 1 ms and completed after 193 ms.

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

