Neo4j

NoSQL

A4- S8

ESILV
jihane.mali (at) devinci.fr

Chapter

Contents

Chapter 1

Import your dataset in Neo4j

1.1 Create your database

To launch Neo4j, we can use the *standard community server* or the *desktop community server*.

NB default user : `neo4j`

1.1.1 Neo4j Desktop

The desktop version is the common way to install Neo4j. To create your database you need to:

1.1.1 Launch Neo4j desktop,

1.1.2 **Add** a new graph database in the project,

1.1.3 **Start** the database (if necessary, click on "fix the problems" to change the listening ports),

1.1.4 Click on **Open** the folder in order to get the `import` folder,

1.1.5 Launch the browser to get the Neo4j UI.

1.1.2 Community server on Docker

Download the last `Neo4j` image on Docker desktop and instanciate it with ports 7474 and 7687. Once the server is launched, you can try the Neo4j UI with this link: `http://localhost:7474`.

A password is required, by default use: *neo4j* (you have to change it, and it cannot be *'neo4j'*)

Then, you need to download data on the `import` folder (by default: `/var/lib/neo4j`).

In the CLI environment of your Docker container:

```
cd /var/lib/neo4j/import
wget https://chewbii.com/wp-content/uploads/2019/03/Neo4j-dataset.zip
apt-get update
apt-get install unzip
unzip Neo4j-dataset.zip
```

Files are now available on the 'import folder' of your graph database.

1.1.3 Community server on Linux/Brew...

Once the server is launched, you can try the Neo4j UI with this link: `http://localhost:7474` (if necessary for Docker, redirect the ports for the browser 7474 and bolt 7687).

A password is required, by default use: *neo4j* (you have to change it, and it cannot be *'neo4j'*)

Then, you need to find the `import` folder (by default: `/var/lib/neo4j`), it depends on your distribution.

1.2 Import Dataset

In this practice work we will use the *Airport* dataset (available online[1]). It an archive composed of 3 CSV files which contains:

- Airport information, even with aerodromes and closed ones,
- Airline (companies) that delivers the routes between airports,

[1] `https://chewbii.com/wp-content/uploads/2019/03/Neo4j-dataset.zip`

- Routes that connects airports and the airlines.

In order to import your dataset, you must put the CSV files in the **$Neo4j_folder/import** folder (you can also put http URL or full path to the file). Each file will be identified by *"file:/airports.csv"*

1.2.1 Create indexes

First, we need to build indexes on nodes. In the following it is necessary to set them on properties in order to be more efficient while creating routes and relationships.

*Only **one** instruction (index creation) can be run at a time* :

```
CREATE INDEX FOR (a:Airport) ON (a.id);
```

```
CREATE INDEX FOR (a:Airline) ON (a.id);
```

```
CREATE INDEX FOR (r:Route) ON (r.id);
```

```
CREATE INDEX FOR (a:Airport) ON (a.country);
```

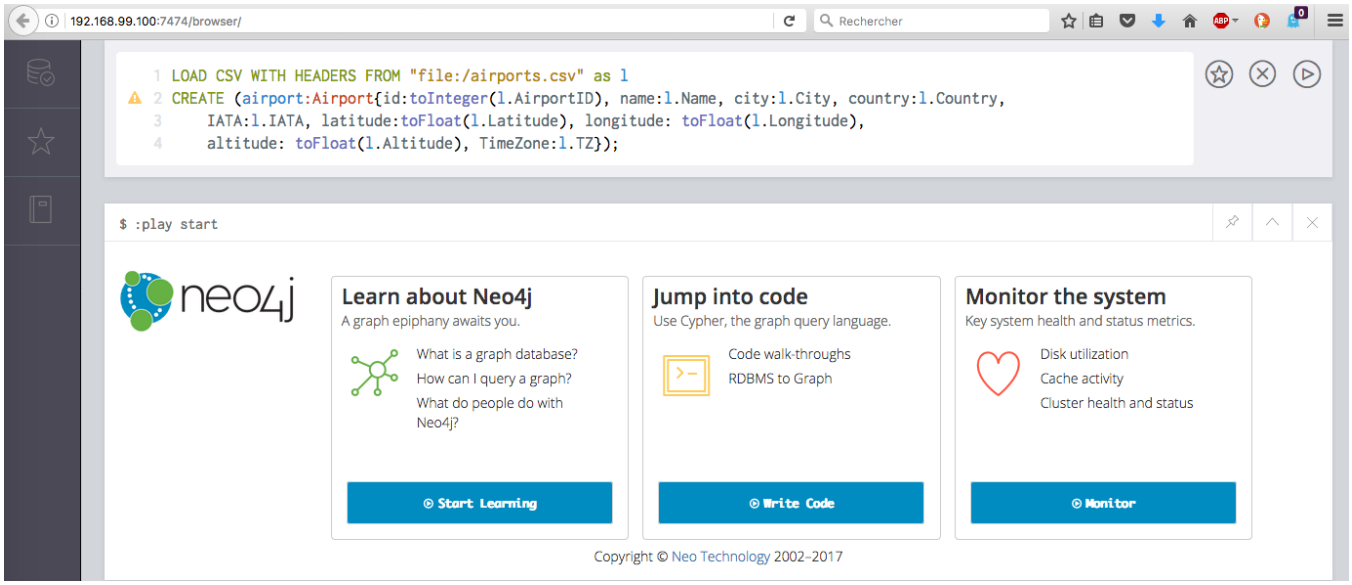
```
CREATE INDEX FOR (a:Airport) ON (a.city);
```

```
CREATE INDEX FOR (a:Airport) ON (a.IATA);
```

```
CREATE INDEX FOR (r:Route) ON (r.name);
```

1.3 Import CSV and build the graph

First, you must connect to the neo4j browser (usually `http://localhost:7474`) where each query must be put in the query field (top of the UI).



To import a CSV file, you must specify for each column in the header the correspondance in your graph

1.3.1 Create Airport nodes

```
CALL {
 LOAD CSV WITH HEADERS FROM "file:/airports.csv" as l
 MERGE (airport:Airport{id:toInteger(l.AirportID), name:l.Name, city:l.City,
   country:l.Country, IATA:l.IATA, latitude:toFloat(l.Latitude),
   longitude: toFloat(l.Longitude), altitude: toFloat(l.Altitude), TimeZone:l.TZ})
};
```

Neo4j has changed the import instruction in the last versions: check your dbms version.

- v5: **CALL {}**
- v4: **:auto**
- v3: **USING PERIODIC COMMIT 200**

The `LOAD CSV` command put each line in main memory, the mapping is set in the `CREATE` clause:

```
MERGE (<var>:<Type>{ <cle>:<colonne du CSV>, ...  })
```

The typed values must be mapped properly with `toInteger` and `toFloat` functions.

1.3.2 Create Airline nodes

Idem for `Airline` nodes:

```
CALL{
 LOAD CSV WITH HEADERS FROM "file:/airlines.csv" as l
 MERGE (airline:Airline{id:toInteger(l.AirlineID), name:l.Name, alias:l.Alias, IATA:l.IATA,
  country:l.Country, active:l.Active})
};
```

1.3.3 Create Route nodes

Now we can create `Route`[2] and *relationships* between `Airports` and `Airlines`. Since we are in a transactional environment (ACID properties) we need to add commits while processing in order to empty the cache and so behing more efficient:

```
CALL {
 LOAD CSV WITH HEADERS FROM "file:/routes.csv" as l
 MERGE (airline:Airline{id:toInteger(l.AirlineID)} )
 MERGE (source:Airport{id:toInteger(l.SourceAirportID)} )
 MERGE (dest:Airport{id:toInteger(l.DestAirportID)} )
 MERGE (route:Route{id:l.AirlineID+"_"+l.SourceAirportID+"_"+l.DestAirportID})
  SET route.equipment = l.Equipment
 MERGE (route) -[:from]-> (source)
 MERGE (route) -[:to]-> (dest)
 MERGE (route) -[:by]-> (airline)
};
```

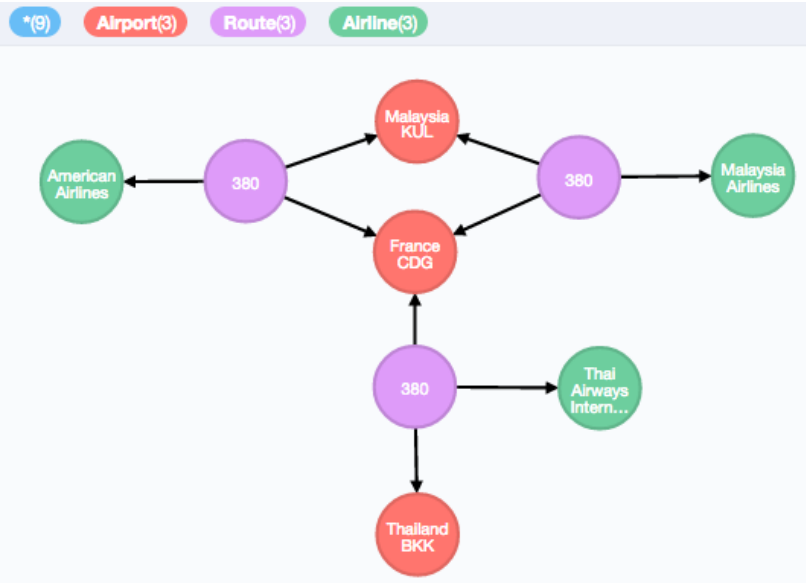
The `MERGE` clause helps to not create nodes a second time but only extract already build ones.

[2] In english, a route is given road between two destinations.

Chapter 2
Cypher: Pattern queries on graphs

Cypher is a query language for graphs and helps to define patterns which express nodes and relationships between nodes. To query your database, you can use the web interface embedded in Neo4j: `http://localhost:7474`

Our dataset contains `Aiport`, `Airline` and `Route` nodes. The `Route` connects `Airport` and `Airline` with directed relationships (*from Route to other nodes*). Every pattern queries must respect this orientations on relationships in order to produce a result. Here a sample of the graph we have in the database:



The queryable properties are:

- Node / `Airport` : id, name, IATA, country, TimeZone, city, latitude, longitude, altitude
- Node / `Airline` : id, name, country, IATA, alias, active
- Node / `Route` : equipment
- Relationship / Route -[:`from`]-> Airport
- Relationship / Route -[:`to`]-> Airport
- Relationship / Route -[:`by`]-> Airline

2.1 Simple queries

2.1.1 Give French Airports' name and IATA code,

2.1.2 Give names and IATA codes of French Airline companies only when the IATA code exists and when it is an active Airline,

2.1.3 Names of French Airlines with at least one existing route,

2.1.4 Graph[1] of routes which departure is Charles de Gaulle (CDG),

2.1.5 Graph of routes from CDG delivered by a A380 (equipment),

2.1.6 Cities and Countries which are the destinations of routes from CDG delivered by an A380,

[1] give the corresponding nodes and the relationships

2.1.7 From the previous result, give the corresponding airlines name,

2.1.8 Graph of routes from CDG to any French airport,

2.1.9 Graph of all routes delivered by an A380,

2.1.10 Graph of all routes coming from a French airport to British airport (United Kingdom),

2.1.11 From previous result, give the distinct list of airline names,

2.1.12 Idem, but only for which they are delivered by an A320.

2.2 Complex queries

2.2.1 To make more complex queries, we need first to create homogeneous relationships between airports in order to make "jumps". We will create new relationships which labels are airlines name. For this, use the following queries:

```
MATCH (FROM:Airport) <-[:from]- (r:Route) -[:to]-> (TO:Airport),
      (r) -[:by]-> (comp)
WHERE FROM <> TO
MERGE (FROM)-[:path{airline:comp.name}]-> (TO)
```

```
CREATE INDEX ON :path(airline);
```

2.2.2 Give the graph of paths delivered by "Air France" between all French airports,

2.2.3 Give per destination country the number of paths delivered by "Air France". Sort the result decreasingly,

2.2.4 Idem, but when the path does not come from a French airport,

2.2.5 Give paths of lengths 2 or 3 (number of relationships) from *Nantes* to *Salt Lake City*,

2.2.6 Give the shortest path from *Nantes* to *Salt Lake City*,

2.3 Hard queries

2.3.1 Give paths of lengths 2 or 3 from *Nantes* to *Salt Lake City*, delivered only by "Air France",

2.3.2 All paths of length 2 from Paris only delivered by "Air France" (without direct flights),

2.3.3 For those destinations, give per country the number of paths sorted decreasingly,

2.3.4 From question 2.3.2, give only those which stops at least once in the United states,

2.4 Execution plan

We wish to extract execution plans from each query in order to see if indexes where properly used. To achieve this, prefix your query with "`EXPLAIN`".

2.4.1 From query 2.3.2 show the corresponding execution plan,

2.4.2 In the following query, a filter is put either on nodes FR and UK which both refer to the index on countries. From which nodes the execution plan begins? When the other side of the path is dealt?

```
EXPLAIN
MATCH (FR:Airport{country:"France"}) <-[:from]- (r:Route) -[:to]->
      (UK:Airport{country:"United Kingdom"}),
      (r) -[:by]-> (comp)
WHERE r.equipment CONTAINS "320"
RETURN DISTINCT comp.name
```

Chapter 3
Bonus/Hard queries/New features

3.1 GDS - Graph Data Science

You can use advanced graph algorithms with Neo4j, for that, you must use the GDS library[1]. Types of algorithms:

- Community detection (Louvain, LabelProp) - graph clustering
- Centrality (PageRank, Betweenness, etc.)
- Similarity (Jaccard, topology)
- Link prediction
- Pathfinding
- node embedding

You first need to create a Cypher Projection (temporary grap):

```
CALL gds.graph.create("tmpGraphName", "inputNodes", {links properties}, {nodes properties})
```

And then, apply a library, here Louvain, on that temporary graph:

```
CALL gds.louvain.stream('tmpGraphName')
YIELD nodeId, communityId, intermediateCommunityIds
RETURN gds.util.asNode(nodeId).name AS name, communityId, intermediateCommunityIds
ORDER BY name ASC
```

3.2 Add an external library

It is possible to add a Java library in Neo4j which can be called in Cypher.

3.2.1 With Neo4j Desktop

If you use Neo4j Desktop, you can add plugins directly from the database administration; click on the db (normaly 'neo4j'), then on the ' plugins' tab, choose 'gds'.

3.2.2 Import library

To import a package/library in Neo4j, it must be put in the "`$NEO4J_FOLDER/plugins/`" folder.[2].

Then, you need to configure your Neo4j server in order to integrate the Java Class to be called. The config file is available here: `$NEO4J_FOLDER/conf/neo4j.conf`.

Add the following line in "neo4j.conf":

```
server.unmanaged_extension_classes=XXXX.extension=/YYYY
```

For which XXXX is the plugin's name and YYYY the running class' name in manifest of the plugin.

3.2.3 Call the library

You can call the library thanks to this command line: `CALL XXXX.<function>(<params>)`.

Here is an example of the RDF reading package.

You can also find:

- APOC: JSON and graph procedures
- Graph algorithms: `https://github.com/neo4j-contrib/neo4j-graph-algorithms/releases`
- NeoSemantics: `https://github.com/jbarrasa/neosemantics`

3.3 Change graph renderer

You can render your graph with your own colors and labels. To achieve this, you need to define a CSS stylesheet, extended with 'grass' information. Use the command line ":`style`" to apply it. Here is an example:

```
:style node {diameter: 50px;color: #A5ABB6;border-color: #9AA1AC;border-width:
↪ 2px;text-color-internal: #FFFFFF;font-size: 10px;} relationship {color:
↪ #A5ABB6;shaft-width: 1px;font-size: 8px;padding: 3px;text-color-external:
↪ #000000;text-color-internal: #FFFFFF;caption: '<type>';} node.Airport {color:
↪ #FF756E;border-color: #E06760;text-color-internal: #FFFFFF;caption: '{country}
↪ {IATA}';} node.Airline {color: #DE9BF9;border-color: #BF85D6;text-color-internal:
↪ #FFFFFF;caption: '{name}';} node.Route {color: #FB95AF;border-color:
↪ #E0849B;text-color-internal: #FFFFFF;caption: '{equipment}';}
```

The order between node styles are important since each of them overloads previous styles.

You can edit:

- `diameter`: nodes size,
- `color`/`border-color`/`text-color-internal`: colors,
- `caption`: Nodes/relationships name. Take properties with "<type>" or "{ATTRIBUTE}".

[1] `https://neo4j.com/product/graph-data-science-library/`
[2] linux: /var/lib/neoj4, windows: C:\Program Files\Neo4j