Bases de Données spécialisées - TP 3

Master 2 Informatique et Maths/Info 2023-2024

Cypher

---

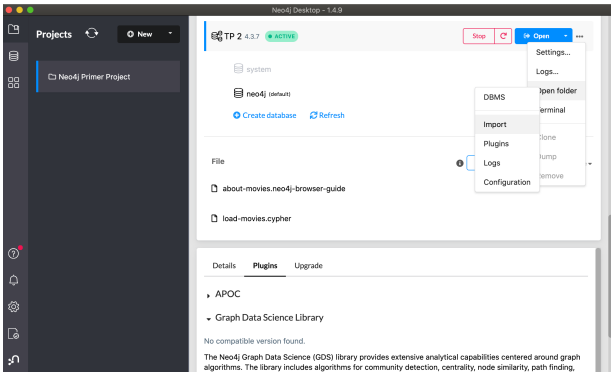
1 Exercice : Loading CSV data and querying

1.1 Import dataset

In this practice work we will use the Airport dataset (available on moodle). It is an archive composed of 3 CSV files which contains :

— Airport information,
— Airlines (companies) that deliver the routes between airports,
— Routes that connect airports and are provided by the airlines.

In order to import your dataset, you must put the CSV files in the `$import` folder of your working dbms. This import folder can be located using the "Open folder" option of your working database as shown in the following picture (you need to click on the three dots) :



To refer to a file 'airports.csv' in the dbms import folder one uses *"file :/airports.csv"*.

Alternatively you can put the CSV files anywhere else, but you will have to specify their full path or their URL to refer to them.

1.2 Import CSV and build the graph

We will use the LOAD query to import CSV data. As any other query, this must be input in the query field at the top of the Neo4J browser user interface. To import a CSV file, you must specify for each column in the CSV header the corresponding property in your graph.

1.3 Create Airport nodes

```
LOAD CSV WITH HEADERS FROM "file:/airports.csv" as l
CREATE (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});
```

The LOAD CSV command scans the CSV file one line at a time ; for each such line the CREATE clause creates nodes and relationships associated to that line. The mapping between the line and the created nodes is set in the CREATE clause :

```
CREATE (<var>:<Type>{ <key>:<CSV column>, ... })
```

Values other than Strings must be casted properly (e.g. with toInteger and toFloat functions) to become key values in the graph.

1.4 Create Airline nodes

Idem for Airline nodes :

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

1.5 Create Indexes

Those two CSV files are pretty small. Thus, we didn't need to put indexes previously. However in the following it is necessary to set them on properties in order to be more efficient while creating routes and relationships.

Note that only one instruction (index creation) can be run at a time :

```
CREATE INDEX airport_id for (n: Airport) ON (n.id);
CREATE INDEX airline_id for (n: Airline) ON (n.id);
CREATE INDEX route_id for (n: Route) ON (n.d);
CREATE INDEX airport_country for (n: Airport) ON (n.country);
CREATE INDEX airport_city for (n: Airport) ON (n.city);
CREATE INDEX airport_IATA for (n: Airport) ON (n.IATA);
CREATE INDEX route_name for (n: Route) ON (n.name);
```

Don't forget to add constraints as well.

1.6 Create Route nodes

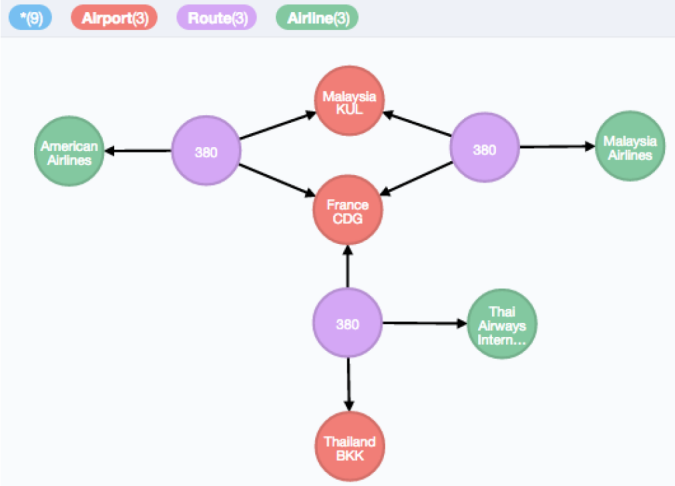
Now we can create Route and relationships between Airports and Airlines. Since we are in a transactional environment (ACID properties) we need to add commits while processing in order to empty the cache and be more efficient :

```
:auto USING PERIODIC COMMIT 200
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)} )
CREATE (route:Route{equipment:l.Equipment})
CREATE (route) -[:from]-> (source)
CREATE (route) -[:to]-> (dest)
CREATE (route) -[:by]-> (airline)
```

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

2 Exercise : Cypher pattern queries on graphs

Our dataset contains Airport, 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 is 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
Using Neo4J Browser run the following to retrieve schema information aboute the graph :

```
CALL db.labels
CALL db.propertyKeys
CALL db.labels
CALL db.schema.visualization
CALL db.indexes
```

2.1 Simple queries

1. Give French Airports' name and IATA code,
2. Give names and IATA codes of French Airline companies only when the IATA code exists and when it is an active Airline,
3. Names of French Airlines with at least one existing route,
4. Graph of routes whose departure is Charles de Gaulle (CDG),
5. Graph of routes from CDG delivered by a A380 (equipment),
6. Cities and Countries which are the destinations of routes from CDG delivered by an A380,
7. From the previous result, give the corresponding airline names,
8. Graph of routes from CDG to any French airport,
9. Graph of all routes delivered by an A380,
10. Graph of all routes coming from a French airport to British airport (United Kingdom),
11. From the previous result, give the list of distinct airline names,
12. Idem, but only for routes delivered by an A320.

2.2 Complex queries

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 whose labels are airline names. 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 path_airline FOR (p:path) on (p.airline)
```

2. Give the graph of paths delivered by "Air France" between all French airports,
3. Give per destination country the number of paths delivered by "Air France". Sort the result decreasingly,
4. Idem, but when the path does not come from a French airport,
5. Give paths of lengths 2 or 3 (number of relationships) from Nantes to Salt Lake City,
6. Give the shortest path from Nantes to Salt Lake City

2.3 Hard queries

1. Give paths of lengths 2 or 3 from Nantes to Salt Lake City, delivered only by "Air France",
2. All paths of length 2 from Paris only delivered by "Air France" (without direct flights),
3. For those destinations, give per country the number of paths sorted decreasingly,
4. From question 2.2.2, give only those which stop 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 were properly used. To achieve this, you can prefix your query with `EXPLAIN` or `PROFILE`.

1. Prefix some of your previous queries with `EXPLAIN` or `PROFILE`. Compare execution plans with and without index (just drop and then recreate indexes).
2. In the following query, a filter added to both origin and destination countries. These filters can exploit the index on countries. Which nodes does the execution plan begin from ? When is the other side of the path dealt with ?

```
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
```

3 Exercise : more Cypher queries

Come back to Section 4 of TP2. Complete all the exercices proposed in that guide :

```
:play https://guides.neo4j.com/4.0-intro-neo4j-exercises/
```

4 Explore available Neo4J datasets and demos

4.1 Browser guides

In the same spirit as the the movie Browser guide explored in TP2, many other Browser guides are available via the Neo4J Browser. You can access and run them in the following way : open the Neo4J Browser on an empty database, then type

```
:play <guide name>
```

This will create locally data in your database and guide you to query and/or manipulate this data with Cypher.

Available guide names can be found here : `https://neo4j.com/docs/getting-started/appendix/example-data/#built-in-examples`

More guides can be found by selecting the play button in the left part of your Neo4J Browser (this includes a selection of Neo4J gists).

4.2 Neo4J Sandbox

To explore a wide variety of datasets in an online setup without a local installation, you can use the Neo4j Sandbox : `https://neo4j.com/sandbox/?ref=developer-ex-data`

4.3 Neo4J dataset demo server

A set of demo datasets are available for querying online `https://neo4j.com/docs/getting-started/appendix/example-data/#demo-server`. This Web interface only provides you with access to example datasets, it is not intended to provide you with exercices and solutions ; you will have to come up with interesting queries yourself.