TP5 : Neo4j*

Pierre Monnin, Guillaume Méroué, Célian Ringwald, Nicolas Robert
{pierre.monnin, guillaume.meroue, celian.ringwald, nicolas.robert}@inria.fr

2024 – 2025

Objectif

Dans ce TP, vous apprendrez à utiliser les commandes de base, la visualisation et les plans d'exécution de requêtes sur une base graphes Neo4j.

Software

Pour utiliser Neo4j, nous allons utiliser l'image Docker suivante :

```
1 docker run --publish=7474:7474 --publish=7687:7687 --volume=$PWD/neo4j-data:/data -v $PWD
    /import:/var/lib/neo4j/import --env=NEO4J_AUTH=none neo4j
```

Vous pourrez accéder à Neo4j via votre navigateur sur http://localhost:7474. Le répertoire `neo4j-data` dans votre répertoire courant est mappé sur `/data` dans le conteneur pour assurer la persistance de la base. Le dossier `import` de votre dossier courant est utilisé comme répertoire d'import pour Neo4j. L'authentification est désactivée.

Documentation

- Neo4j : https://neo4j.com/
- Neo4j documentation : https://neo4j.com/docs/
- Docker Neo4j : https://hub.docker.com/_/neo4j

Importation des données

Nous allons créer une base de données sur l'ensemble des aéroports du monde, les routes aériennes entre eux, ainsi que les compagnies associées. Vous pourrez trouver les fichiers nécessaires sur LMS. Pour charger un fichier CSV (avec entêtes), il faut d'abord définir les correspondances de chaque colonne pour créer les noeuds "Aeroports". Attention ! Le fichier `file:///airports.csv` doit être dans le répertoire d'import.

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

La commande LOAD CSV ne fait que charger les données en mémoire. La correspondance (ou mapping) est spécifiée par la commande CREATE :

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

*D'après le TP proposé par Sabeur Aridhi

On peut remarquer que pour les nombres (entiers ou flottants), une fonction de parsing est nécessaire pour le typage (toInt, toFloat). Il est également possible d'importer des données directement d'une URL, en remplaçant `file://...` par `http://...`. On peut ensuite charger les compagnies d'aviation :

```
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.ITA
    , country:l.Country, active:l.Active});
```

Ces deux fichiers étaient relativement petits. Nous allons nous attaquer maintenant aux routes et aux relations dans la base. Mais pour que cela soit plus efficace (lors de l'importation et de l'interrogation), il nous faut poser des indexes sur les clés qui seront fréquemment utilisées :

```
CREATE INDEX airport_id_index FOR (a:Airport) ON (a.id);
CREATE INDEX airline_id_index FOR (al:Airline) ON (al.id);
CREATE INDEX route_id_index FOR (r:Route) ON (r.id);
CREATE INDEX airport_country_index FOR (a:Airport) ON (a.country);
CREATE INDEX airport_city_index FOR (a:Airport) ON (a.city);
CREATE INDEX airport_iata_index FOR (a:Airport) ON (a.IATA);
CREATE TEXT INDEX route_name_index FOR (r:Route) ON (r.name);
```

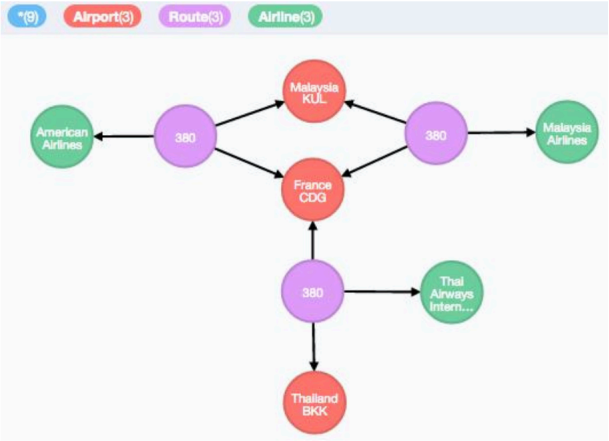
L'importation des routes aériennes se fera avec des confirmations régulières (commit) pour plus d'efficacité :

```
:auto LOAD CSV WITH HEADERS FROM "file:///routes.csv" AS l
CALL {
    WITH l
    // Vérifier que les identifiants ne sont pas NULL avant de les utiliser
    WITH l,
         toIntegerOrNull(l.AirlineID) AS airlineID,
         toIntegerOrNull(l.SourceAirportID) AS sourceID,
         toIntegerOrNull(l.DestAirportID) AS destID

    // Ignorer les lignes où un ID obligatoire est NULL
    WHERE airlineID IS NOT NULL AND sourceID IS NOT NULL AND destID IS NOT NULL

    MERGE (airline:Airline {id: airlineID})
    MERGE (source:Airport {id: sourceID})
    MERGE (dest:Airport {id: destID})
    CREATE (route:Route {equipment: l.Equipment})
    MERGE (route)-[:FROM]->(source)
    MERGE (route)-[:TO]->(dest)
    MERGE (route)-[:BY]->(airline)
} IN TRANSACTIONS OF 200 ROWS;
```

Le `MERGE` permet d'associer une colonne à un élément déjà existant (via jointure avec la clé précisée), ensuite l'élément "Route" est créé. Les relations sont ensuite instanciées dans Neo4j lorsque la commande `CREATE (type1) -[:xxx]-> (type2)` est définie. Nous pourrons constater le typage et l'orientation de cette relation `[:xxx]`.

Interrogation de graphes

Nous allons maintenant nous intéresser à la manière de stocker et manipuler les éléments et les relations dans une base Neo4j. Pour ce faire, nous utiliserons l'interface Web fournie par Neo4j lors de son démarrage : http://localhost:7474. Le graphe stocké dans la base de données contient des informations ressemblant à ce graphe orienté :



Les nœuds et relations contiennent les informations suivantes :

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

Interrogation

1. Donner la liste des noms et IATA des aéroports français.
2. Donner la liste des noms et IATA de compagnies françaises avec un code IATA existant et encore en activité (active "Y").
3. Donner la liste des noms de compagnies françaises avec au moins une route existante.
4. Donner le graphe des routes partant de l'aéroport Charles de Gaulle (CDG).
5. Donner le graphe des routes partant de l'aéroport Charles de Gaulle (CDG) avec un A380 (l'équipement doit contenir 380).
6. Donner le pays et et la ville de destination des routes provenant de CDG avec un A380.
7. Donner en plus les noms des compagnies effectuant ces trajets.
8. Donner le graphe des routes entre CDG et les aéroports français.
9. Donner le graphe de toutes les routes effectuées par un A380.
10. Donner le graphe des routes partant de France vers le Royaume-Uni (United Kingdom).
11. Donner la liste distincte des compagnies effectuant ces trajets.
12. Donner la liste distincte de ces compagnies effectuant ces trajets avec un A320.

Création de liens

1. Créer des relations (path) entre les aéroports à partir des routes, l'étiquette contiendra le nom de la compagnie. Utiliser pour cela la requête suivante :
2. Donner le graphe des chemins de la compagnie Air France en France (source et destination).
3. Donner les pays et le nombre de chemins associés des destinations desservies par Air France. Trier le résultat par résultat décroissant.
4. Donner les pays et le nombre de chemins associés des destinations desservies par Air France, MAIS ne partant pas de France. Trier le résultat par résultat décroissant.
5. Donner le chemin le plus court (nombre d'escales) entre Nantes et Salt Lake City en utilisant la relation "path".
6. Donner la liste des chemins de longueur 1 à 2 (nombre d'escales) entre Nantes et Salt Lake City en utilisant la relation "path".
7. Vérifier s'il existe un chemin qui n'a que la compagnie Air France.
8. Donner les chemins de longueur 2 au départ de Paris uniquement desservi par Air France.
9. Donner les chemins de longueur 2 au départ de Paris uniquement desservi par Air France, mais sans chemins directs.
10. Donner la liste des pays desservi par ces chemins et le nombre de chemins associés.
11. Donner les chemins de longueur 2 au départ de Paris uniquement desservi par Air France, mais sans chemins directs, et passant par au moins une fois par les Etats-Unis (par forcément la destination).

Plan d'exécution

Pour consulter le plan d'exécution utilisé pour une requête, il suffit de préfixer la requête par le mot-clé `EXPLAIN`.

1. Consulter le plan d'exécution de la requête précédente
2. Consulter le plan de la requête suivante. Quel index est utilisé ?

```
1 MATCH (FR:Airport{country:"France"}) <-[:FROM]- (r:Route) -[:TO]-> (UK:Airport{
      country:"United Kingdom"}), (r) -[:BY]-> (comp)
2 WHERE r.equipment CONTAINS "320"
3 RETURN DISTINCT comp.name
```