TP Neo4j

Ce TP a pour objectif de vous familiariser avec Neo4j et Cypher, son langage d'interrogation le plus répandu. Vous allez dans la suite charger un jeu de données et écrire les requêtes Cypher permettant chacune de répondre à un besoin en information précis.

Installer et lancer Neo4j

Windows

1. Téléchargez la version *Community Edition* de Neo4j à cet url : https://neo4j.com/download/community-edition/
2. Exécutez le .msi téléchargé (les droits d'administration sont requis).
3. Lancez le programme *Neo4j Community Edition* installé.
4. Changez éventuellement le dossier dans lequel la base de données sera sauvegardée.
5. Cliquez sur Start pour lancer le serveur.

Linux

Cherchez et suivez les instructions adaptées à votre distribution ; par exemple, pour Debian : http://debian.neo4j.org/

Cloud

Vous pouvez également pour ce TP utiliser une instance de Neo4j hébergée sur un serveur. Il est par exemple possible de créer sur *GrapheneDB* un plan gratuit (limité à 1000 nœuds et 10000 relations, ce qui est suffisant pour ce TP).

1. Créez un compte utilisateur sur https://www.graphenedb.com/
2. Créez une base de données.
   Sélectionner la région "*US East (Northern Virginia) Region*" afin de se voir proposer le plan gratuit (*Sandbox*).
3. Inutile de créer un utilisateur pour la base de données, étant donné que l'ensemble des Manipulations de ce TP sont réalisés via le *Browser Neo4j*.

Browser Neo4j

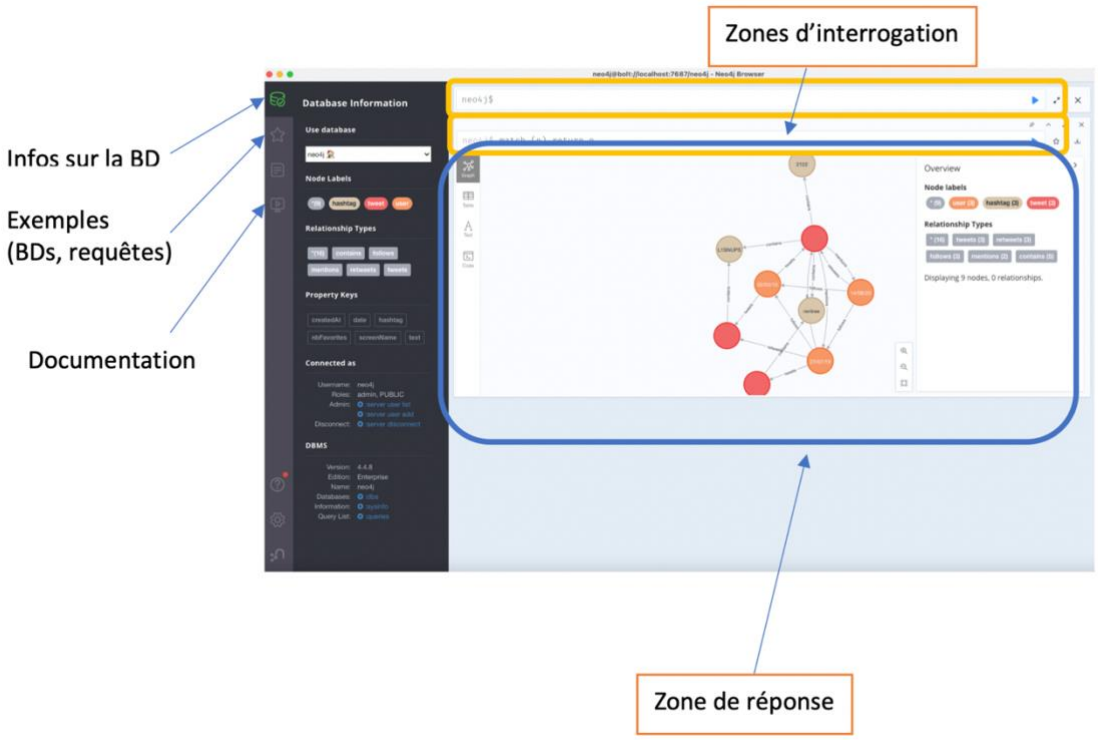
Sur http://localhost:7474/ (ou en suivant le lien depuis l'onglet *Overview* de votre base de données sur *GrapheneDB*) vous accédez au browser Neo4j, une interface qui vous permet de parcourir vos données en exécutant des requêtes et visualisant les résultats. Le browser peut être vu comme une console avec des possibilités d'interactions riches. Il s'agit d'une interface d'exploration, pratique pour se familiariser avec Cypher (un langage d'interrogation de Neo4j), qui peut être utilisée pour des besoins de développement mais pas en production.

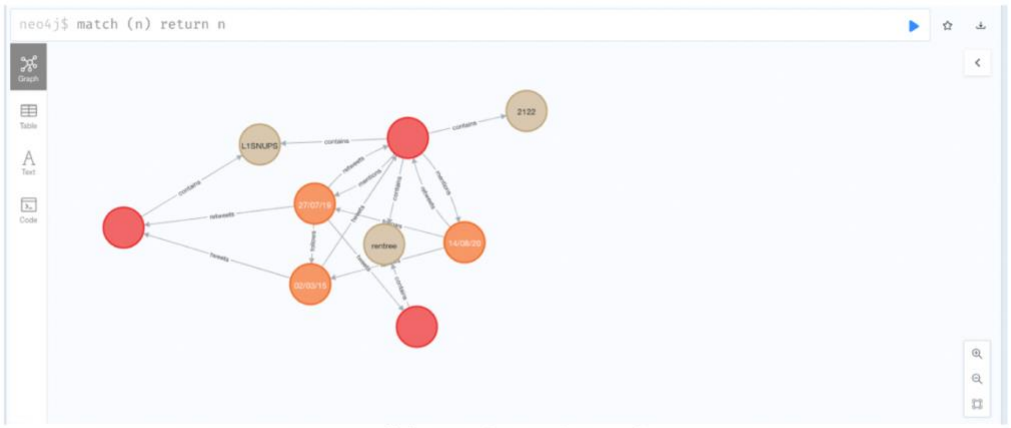
C'est l'interface qu'il est recommandé d'utiliser au cours de ce TP. Vous pouvez cependant, si vous Préférez, exécuter vos requêtes via l'API REST ou Java.

Prise en main de Cypher

Avant de commencer à écrire vos propres requêtes, il est conseillé de parcourir le GraphGist suivant afin de voir quelques exemples d'utilisation de Cypher : https://neo4j.com/graphgist/34b3cf4c-da1e-4650-81e4-3a7107336ac9

https://memgraph.com/blog/cypher-cheat-sheet





(a) Sous forme de graphe

(b) Sous forme de tableau

(c) Sous forme textuelle

Effacer la base de données

Dans l'étape précédente, vous avez ajouté quelques éléments de la BD. Avant de charger le jeu de données pour la suite du TP, vous avez besoin de partir d'une BD vierge. Pour cela, vous avez plusieurs options :

- Arrêter le serveur, supprimer le contenu du dossier cible de la BD et relancer le serveur ;
- Arrêter le serveur et le relancer en spécifiant un nouveau dossier ;
- Supprimer directement les données avec la requête Cypher suivante :

```
MATCH (n)
DETACH DELETE n
```

Jeu de données *MovieLens*

Charger le jeu de données

Exécutez successivement les instructions Cypher suivantes dans le browser Neo4j :

```
USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM 'users.csv' AS users
CREATE (u:User {id : toInteger(users.id), sex : users.sex, age : toInteger(users.age),
occupation : users.occupation, zip_code : users.zip_code});
```

```
USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM 'movies.csv' AS movies
CREATE (m:Movie {id : toInteger(movies.id), title : movies.title, date :
toInteger(movies.date)});
```

```
USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM
'genres.csv' AS genres
CREATE (u:Genre {id : toInteger(genres.id), name : genres.name});
```

```
USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM
'mov_genre.csv' AS mov_genreMERGE (m : Movie { id : toInteger(mov_genre.mov_id) })
MERGE (g : Genre { id : toInteger(mov_genre.genre) })
CREATE (m)-[:CATEGORIZED_AS]->(g);
```

```
USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM
'ratings.csv' AS ratingsMERGE (u : User { id : toInteger(ratings.user_id) })
MERGE (m : Movie { id : toInteger(ratings.mov_id) })
CREATE (u)-[:RATED { note : toInteger(ratings.rating), timestamp :
toInteger(ratings.timestamp) } ]->(m);
```

```
USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM
'friends.csv' AS friends
```

```
MERGE (u1:User { id : toInteger(friends.user1_id) })
MERGE (u2:User { id : toInteger(friends.user2_id) })
CREATE (u1)-[:FRIEND_OF]->(u2);
```

Présentation du jeu de données

Les données utilisées dans ce TP sont issues de l'export d'un sous-ensemble de la base de données *MovieLens* (https://movielens.org/). Les instructions exécutées dans l'étape précédente permettent de transformer des données csv en format graphe exploitable par Neo4j.

Ces données suivent le schéma suivant :

Il y a trois types de nœuds :

- User : décrit un utilisateur de la base de données MovieLens ; contient les attributs suivants :
  - id : int, sex : string, age : int, occupation : string, zip code : int
- Movie : représente un film de la base de données MovieLens ; contient les attributs suivants :
  - id : int, title : string, date : int
- Genre : fait référence au genre d'un film ; contient les attributs suivants :
  - id : int, name : string

Le jeu de données exploite également trois types de relations :

- RATED : spécifie la note qu'a laissée un utilisateur à propos d'un film ; contient les attributs suivants :
  - note : int, timestamp : int
- FRIEND_OF: représente un lien d'amitié entre deux utilisateurs ; notez que, comme toutes les relations dans Neo4j, celle-ci est orientée mais elle est également symétrique dans ce jeu de données (c'est-à-dire que si A est ami avec B, alors B est ami avec A).
- CATEGORIZED_AS : permet d'assigner un genre à un film.

La requête Cypher suivante permet d'extraire un sous-graphe du jeu de données (l'ensemble des éléments liés à l'utilisateur qui a pour id la valeur 1. Exécutez-là dans le Browser Neo4j.

```
MATCH (u1:User {id:1})-[:RATED]->(m:Movie)-[:CATEGORIZED_AS]->(g:Genre),
(u1)-[:FRIEND_OF]->(un:User)
RETURN *
LIMIT 50
```

Le résultat s'affiche sous forme de graphe en dessous de la zone de requête. Explorez les interactions possible avec l'interface :

- Mode d'affichage des résultats (boutons sur la gauche),
- Complétion des relations entre les nœuds, existantes dans la base de données mais non référencées dans la requête (switch en bas en droite),

- Consultation des valeurs d'attributs de chaque nœud et relation,
- …

Questions portant sur des nœuds

Question 1. Combien y a-t-il d'utilisateurs dans la base de données ?

- http://neo4j.com/docs/developer-manual/current/cypher/#_basic_node_finding
- http://neo4j.com/docs/developer-manual/current/cypher/#_count

Question 2. Quel est l'âge et le sexe de l'utilisateur qui a pour id la valeur 1?

- http://neo4j.com/docs/developer-manual/current/cypher/#return-return-property

Question 3.1. Quelles sont les femmes qui sont artistes (*occupation='artist'*) ?

Question 3.2. Quelles sont les femmes qui sont artistes ou auteurs (*occupation='writer'*) ?

- http://neo4j.com/docs/developer-manual/current/cypher/#query-where

Question 4. Quel est l'âge moyen des étudiants de la base de données ?

- http://neo4j.com/docs/developer-manual/current/cypher/#query-aggregation

Question 5. Quel est l'âge moyen des utilisateurs pour chaque occupation?

Question 6.1. Quel est le nombre d'utilisateurs pour chaque occupation ?

Question 6.2. Quelles sont les 5 occupations les plus populaires ?

- http://neo4j.com/docs/developer-manual/current/cypher/#query-order
- http://neo4j.com/docs/developer-manual/current/cypher/#query-limit

Question 7. Combien y a-t-il d'occupations différentes dans le jeu de données ?

- http://neo4j.com/docs/developer-manual/current/cypher/#query-aggregation
- http://neo4j.com/docs/developer-manual/current/cypher/#aggregation-distinct

Question 8. Combien y a-t-il d'occupations pour lesquelles l'âge moyen des utilisateurs est supérieur à 30 ?

- http://neo4j.com/docs/developer-manual/current/cypher/#query-with

Question 9. Quels sont les films produits en 1995? Notez que la date de production d'un film n'apparait pas comme propriété des nœuds Movie, mais entre parenthèses à la fin de chaque titre de film ; vous pouvez exploiter cette information avec une expression régulière.

- http://neo4j.com/docs/developer-manual/current/cypher/#query-where-regex

Questions portant sur des relations

Question 10. Combien de notes contient la base de données ? (Comptez les relation de type RATED)

- http://neo4j.com/docs/developer-manual/current/cypher/#_relationship_basics

La façon la plus simple d'obtenir tous les types de relations de la base de données et leur cardinalité (nombre de fois où ils sont utilisés) est d'utiliser la requête suivante :

```
MATCH ()-[r]->()
RETURN TYPE(r) AS rel_type, count(*) AS rel_cardinality
```

Adaptez cette requette pour n'avoir en résultat que la cardinalité de la relation RATED.

Question 11. Quelles sont toutes les notes laissées par l'utilisateur qui a pour id la valeur 1 ? Affichez le résultat sous forme de graphe reliant chaque note au film concerné.

Question 12. Combien d'utilisateurs ont note le film intitulé "Braveheart (1995)" ?

Question 13. Quels sont les cinq genres les plus représentés parmi les films de la base de données ?

Question 14. Quels sont les dix films qui ont reçu le plus de notes ?

Question 15. Combien de films ont reçu au moins une fois la note 1 ?

Question 16. Quels sont les films qui ont reçu au moins une fois la note 1, combien de fois ont-ils-obtenu cette note, et qui la leur a attribué ? Cette requête doit retourner une ligne par film (donc autant de lignes que la réponse à la question précédente) avec trois colonnes : le titre du film, le nombre de fois où il a reçu la note 1, et la liste des identifiants de chaque utilisateur qui lui a donné cette note.

- http://neo4j.com/docs/developer-manual/current/cypher/#aggregation-collect

Question 17. Quels sont les films dont la note moyenne dépasse 4.

- http://neo4j.com/docs/developer-manual/current/cypher/#query-with

Question 18. Quels sont les films qui ont la note moyenne maximale (note = 5) ?

Requêtes avancées.

Question 19. Qui sont les amis et les amis des amis de l'utilisateur qui a pour id la valeur 1 ? La requête ne doit pas retourner l'utilisateur 1 dans la liste de résultats.

Question 20. Qui sont les amis et les amis des amis de l'utilisateur qui a pour id la valeur 1, et leur degrés de séparation (nombre de relations FRIEND_OF qui les séparent) ?

- http://neo4j.com/docs/developer-manual/current/cypher/#query-shortest-path

Question 21. Quel est l'utilisateur qui a le plus d'amis en commun avec l'utilisateur 1 ?

Question 22. Quel est l'utilisateur qui a le plus d'amis en commun avec l'utilisateur 1 mais qui n'est pas déjà son ami ?

Question 23. Quel est l'utilisateur qui a noté le plus grand nombre de films parmi ceux que l'utilisateur 1 a notés, et quel est ce nombre ?

Question 24 : Comment porter le modèle données, utilisé sous neo4J, Dans MongoDB ?

Question 25 : Quelles sont les questions qui ne peuvent pas être exécutées sous MongoDB

"Cypher" le language de requêtage

- **Exemple d'une requête Cypher**

```
MATCH (p:Person { name:"Steave" })
RETURN p
```

Retourne le noeud "Person" qui a le propriété "name" est "Steave"

Equivalent en SQL:

```
SELECT * FROM Person
WHERE name = "Steave";
```

- **Insertion avec Cypher**

```
CREATE (a:Artist { Name : "Strapping Young Lad" })
```

Créer un noeud avec l'étiquette "Artist" avec une propriété "name" qui à la valeur "Strapping young lad"

Le préfixe **a** est un nom de variable fournie. Cette variable peut être utilisée si nous avons besoin de s'y référer plus tard.

- **Afficher le noeud créé**

L'instruction CREATE crée le noeud, mais il ne l'affiche pas

```
CREATE (b:Album { Name : "Heavy as a Really Heavy Thing", Released : "1995" })
RETURN b
```

Retourner le noeud en utilisant le nom de variable (**b** dans ce cas) pour l'afficher

- **Créer des noeuds multiples**

```
CREATE (a:Album { Name: "Killers"}), (b:Album { Name: "Fear of the Dark"})
RETURN a,b
```

Creation de plusieurs noeuds en séparant chaque noeud par une virgule

Ou on peut utiliser plusieurs instructions CREATE

```
CREATE (a:Album { Name: "Piece of Mind"})
CREATE (b:Album { Name: "Somewhere in Time"})
RETURN a,b
```

- **Mise à jour d’un noeud**

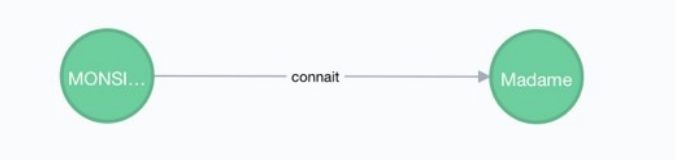
```
MATCH (b:Business {  name: “GraphStory” })
SET b.description = "The leading Graph Database"
RETURN b
```

1. On sélectionne la noeud à mettre a jour avec MATCH
2. On change la valeur de la/les propriétés á changer avec SET

- **Création d’une relation avec Cypher**

Pour créer une relation entre deux nœuds, il faut préalablement rechercher les deux nœuds, puis créer la ou les relations.

```
match (toto:Personne {nom: 'MONSIEUR'}), (tata:Personne {nom: 'MADAME'})
create (toto)-[:connait]->(tata)
```



- **Suppression d’un noeud**

```
MATCH (u:User {username: “greg”})
DELETE u
```

1 On sélectionne la noeud à supprimer MATCH
2 On supprime le noeud avec l’instruction DELETE

- **Suppression un noeud avec ses dépendances**

```
MATCH (u:User {username: “greg”})-[r]-()
DELETE u
```

Transaction sur Neo4j

• Quand une modification commence, ceci va être avec une nouvelle transaction ou avec une transaction déja existante

1 Start transaction
2 Execution des requêtes Cypher
3- Commit transaction

Les index sur Neo4j

Un index est une structure de données qui permet d'améliorer la rapidité de récupération des données dans une base de données.

Les index sur Neo4j

On crée un index avec l'instruction **CREATE INDEX ON**

```
CREATE INDEX ON :Album(Name)
```

Dans l'exemple ci-dessus, nous créons un index sur la propriété "**Name**" de tous les index avec l'etiquette "**Album**"

Les index sur Neo4j

Pour consulter les Index sur Neo4j, il suffit de tapper la commande suivante:

```
:schema
```

Résultat de la commande =>