INFO834 – TP – BD NoSQL Orienté Graphe Neo4j

Préparation

1. Il faut avoir installé :
   - *Neo4j community edition* (Java devrai être en version 11 au minimum) (cf. TD3)
   - Pour *Python* : la bibliothèque *neo4j.* Nous en utiliserons le driver *GraphDatabase* (https://neo4j.com/docs/api/python-driver/current/api.html#graph).

2. Depuis le dossier [dossier de neo4j]/bin :
   - Ouvrir le fichier *neo4j.conf* et dé-commenter la ligne
     dbms.security.auth_enabled=false
   - Dans une fenêtre de commande, exécuter la commande suivante pour lancer *Neo4j*
     [chemin]/neo4j start
     ou bien lancer Neo4J Desktop et lancer le serveur

3. Accéder à l'interface utilisateur via votre navigateur web préféré *http://localhost:7474/* ou bien sous Neo4j Desktop via *Open > Neo4j Browser*

4. Pour déboguer, des logs sont stockés dans le dossier *`…/neo4j-community-4.2.3/logs`*

Travail à faire

L'objectif du TP est de construire une API Python permettant d'effectuer les opérations de base sur un graphe :

- Récupérer l'ensemble des *Node* d'un graphe
- Récupérer un *Node* particulier
- Récupérer l'ensemble des *Relationship* (ou *Path*) d'un graphe
- Récupérer un *Relationship* (ou *Path*) spécifique
- Créer un *Node* (nous considérons que le graphe existe déjà sur Neo4j)
- Créer un *Relationship* (en créant les *Node* correspondant ou non)

1. Importer le driver (*from neo4j import GraphDatabase*) et utiliser les fonctions disponibles comme dans l'exemple des universités qui se trouve ici : https://pythontic.com/database/neo4j/create%20nodes%20and%20relationships. Des informations sur le langage Cypher se trouvent ici https://sql-nosql.org/en/cypher-tutorial.
Tester au fur et à mesure sur l'exemple des universités. Il vous est fortement conseillé de tester vos requêtes sur le navigateur Neo4j avant de les inclure dans votre code Python. Il vous faudra veiller à ce que votre API ne soit pas liée à un exemple particulier et qu'elle pourra être utilisée sur d'autres graphes.

2. Travailler sur de grands graphes : il s'agit à partir de données récupérées depuis *data.gouv.fr*, de créer un graphe des communes françaises (plus de 36000 nœuds), faire apparaître les communes nouvelles et ajouter d'autres informations, comme les équipements, les bilans comptables, les maires, les habitants (à choisir).
Une solution intéressante serait de prendre la France comme un ensemble de Régions, constituées de Départements, constitués de Communes. Chaque nœud est donc une Région, un Département, ou une Commune (nouvelle ou pas). A partir de là, il faut ajouter d'autres propriétés ou nœuds et tenter d'augmenter le nombre de relations entre les nœuds.

Pour aller plus loin

Lorsque votre graphe est créé, il apparaît clairement qu'il est trop volumineux pour la visualisation dans Neo4j. Nous allons dans ce cas utiliser l'outil de visualisation *Gephi*. La solution la plus élégante mais la plus difficile est de considérer importer les données *Neo4j* dans *Gephi* grâce à une extension dédiée (plugin *APOC* de *neo4j*). Une fois les données importées dans *Gephi*, il vous sera possible d'avoir différentes vues sur votre graphe et d'exécuter des fonctions statistiques (composantes connexes et autres métriques).
Gephi est disponible sur https://gephi.org ; documentation https://gephi.org/tutorials/gephi-tutorial-quick_start.pdf.

Pour exporter au format csv depuis Neo4j

CALL apoc.export.csv.all("products.csv", {})

NB. Dans Gephi, activer le plugin pour importer du csv.

Si `ERROR Failed to start Neo4j on dbms.connector.http.listen_address,` vérifier la compatibilité de la version APOC avec celle de neo4j ici.

Une autre solution est d'utiliser un DataFrame de Pandas pour le faire efficacement.

ATTENTION : insérer toutes les données va rendre *Gephi* et *Neo4j* inutilisables sur nos machines pas très puissantes, utiliser en conséquence une petite partie des communes (env. 1000).