Méthodes d'accès aux données 2021–2022
Professeur : Nastaran Fatemi
Assistants : Christopher Meier
Edward Ransome

Labo 2 : Neo4j

1 Introduction

1.1 Objectif

L'objectif de ce laboratoire est d'exercer la base de données graphe, *Neo4j*, et son langage de requête *Cypher*.

1.2 Organization

Ce laboratoire est à effectuer **par groupe de 2** étudiants. Tout plagiat sera sanctionnée par la note de 1.

Ce laboratoire est à rendre pour le **28.03.2022** à 23h59 sur Cyberlearn, une archive zip contenant les sources de votre projet doit y être déposée.

1.3 Mise en place

Le code source du labo vous est fourni sur Cyberlearn. Celui-ci contient:

- Un fichier `pom.xml` qui configure les dépendances du projet Maven.
- Un fichier `docker-compose.yml` qui permet le démarrage simplifié d'une instance de *Neo4j*.
- Un fichier `import.sh` qui permet d'importer le dataset pour le container docker sur Linux ou MacOS.
- Un fichier `import.ps1` qui permet d'importer le dataset pour le container docker sur Windows.
- Un dossier `source` qui contient le dataset à utiliser pour ce labo.
- Un dossier `plugins` qui contient des extensions pour *Neo4j*.
- Un fichier `Main.java` qui se connecte à la base de donnée *Neo4j* et exécute les requêtes de `Requests.java`.
- Un fichier `QueryOutputFormatTest.java` qui permet de vérifier que vous avez le bon format d'output pour les requêtes. (Lancer les tests avant le rendu pour simplifier les tests automatiques de la correction)

Ainsi que le fichier à compléter:

- Un fichier `Requests.java`.

Le driver pour *Neo4j* nécessite au minimum Java 11.

1.4 Déploiement

Vous avez trois possibilité pour déployer *Neo4j*:

- Notre recommendation: Utiliser le fichier `docker-compose.yml` et le script `import.sh` fourni pour déployer une instance avec docker (c.f. appendix A.1).
- Utiliser *Neo4j* aura pour déployer une base de donnée gratuite sur le cloud (c.f. appendix A.2). Cette méthode nécessite la création d'un compte.
- Installer nativement *Neo4j Desktop*. Cette méthode est à utiliser à vos risques et périls.

1.5 Dataset

Ce laboratoire utilise un dataset synthétique appelé *contact-tracing*.

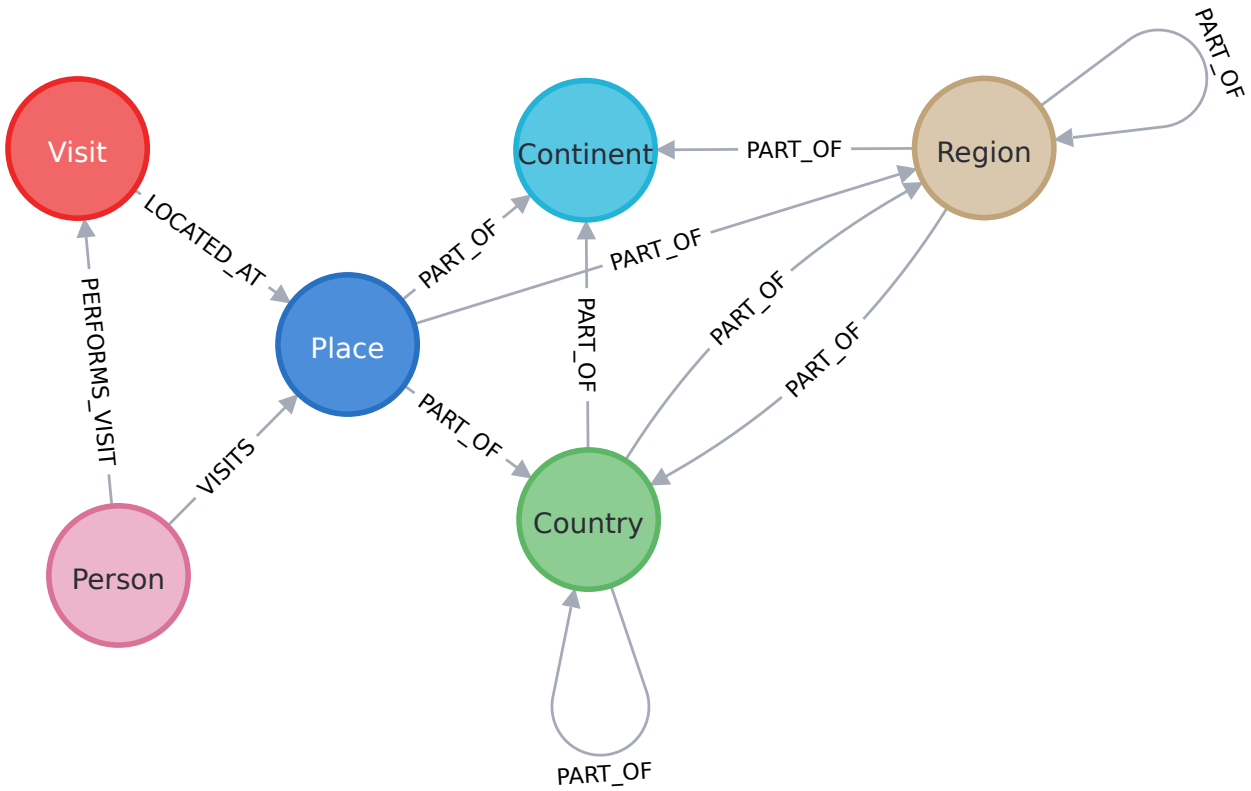

Figure 1 Modèle du dataset *contact-tracing*
Généré avec la requête `CALL db.schema.visualization()`

Ce dataset contient 6 types de noeuds. Les plus importants étant `Person` (avec 501 noeuds), `Visit` (avec 5009 noeuds) et `Place` (avec 101 noeuds). Les types de noeud `Region`, `Country` et `Continent` ont chacun seulement une valeur. La figure 1 montre les types de noeuds ainsi que les relations possibles.

Il existe également un noeud appelé `_Bloom_Perspective_` qu'il faut ignorer. C'est un vestige d'une interface Bloom de *Neo4j*.

Les tables 1 et 2 énumèrent les propriétés des noeuds et des relations. On y voit que la relation `VISITS` est un duplicat du noeud `Visit`. Ils sont interchangeable, selon les besoin de la requête.

2 Connexion

Vérifiez que vous pouvez vous connecter en exécutant la classe `Main`. Les noms des types de noeuds de `contact-tracing` devraient s'afficher dans votre console. Si besoin modifier la méthode `openConnection`.

3 Indications

Certaines requêtes demande l'utilisation de paramètres. La documentation contient quelques exemples pour vous aider. Il existe également d'autres manières d'executer une requête parametrizée.

Afin de simplifier la correction veuillez renommer les champs de retours de toutes vos requêtes pour correspondre aux noms entre parenthèse. Le nom des champs de retours est également vérifié par la suite de test `QueryOutputFormatTest`.

Type de noeud	Nom de la propriété	Type de la propriété
:Person	id	String
:Person	confirmedtime	DateTime
:Person	healthstatus	String
:Person	name	String
:Person	addresslocation	Point
:Place	id	String
:Place	name	String
:Place	type	String
:Place	homelocation	Point
:Visit	id	String
:Visit	endtime	DateTime
:Visit	starttime	DateTime
:Visit	duration	Duration
:Region	name	String
:Country	name	String
:Continent	name	String

Table 1 Propriétés des noeuds du dataset *contact-tracing*
Généré avec la requête `CALL db.schema.nodeTypeProperties`

Type de relation	Nom de la propriété	Type de la propriété
:PERFORMS_VISIT	null	null
:LOCATED_AT	null	null
:VISITS	id	String
:VISITS	endtime	DateTime
:VISITS	starttime	DateTime
:VISITS	duration	Duration
:PART_OF	null	null

Table 2 Propriétés des relations du dataset *contact-tracing*
Les relations `PERFORMS_VISIT`, `LOCATED_AT` et `PART_OF` n'ont pas de propriétés
Généré avec la requête `CALL db.schema.relTypeProperties`

Placer correctement les `DISTINCT` pour avoir le résultat souhaité.

Toutes les requêtes peuvent et doivent être exécutées en une seul fois. Il n'y a donc pas de logique coté Java.

Pour les requêtes 1 et 2, il n'est pas nécessaire que les visites se chevauchent.

4 Requêtes

⋆ 1. Implementer la méthode `possibleSpreaders` qui retourne les nom de toutes les personnes malades (`sickName`) qui ont visité, après la confirmation de leur maladie, un lieu qu'une autre personne en bonne santé à fréquenté. Les deux visites (personne malade et personne en bonne santé) ont lieu après la confirmation de la maladie.

⋆ 2. Implementer la méthode `possibleSpreadCounts` qui retourne, pour chaque personne malade, son nom (`sickName`) et le nombre des personnes en bonne santé (`nbHealthy`) qui ont fréquenté le même lieu qu'elle après la confirmation de sa maladie. Les deux visites (personne malade et personne en bonne santé) ont lieu après la confirmation de la maladie.

⋆ 3. Implementer la méthode `carelessPeople` qui retourne le nom des personnes malades (`sickName`) qui ont fréquenté plus de 10 lieux différents après la confirmation de la maladie, ainsi que le nombre de lieux visités (`nbPlaces`). Trier par ordre décroissant du nombre de lieux.

⋆⋆ 4. Implementer la méthode `sociallyCareful` qui retourne le nom des personnes malades (`sickName`) qui n'ont jamais fréquenté un "Bar" après la confirmation de leur maladie.

⋆⋆⋆ 5. Implementer la méthode `peopleToInform` qui retourne, pour chaque persone malade, son nom (`sickName`), ainsi que la liste de toutes les personnes en bonne santé qu'elle risque d'avoir infecté (`peopleToInform`) avec la condition suivante: la personne malade a visité l'autre personne dans un endroit avec un chevauchement d'au moins 2 heures.

- Chevauchement: La durée de chevauchement correspond à la différence entre le maximum des temps de début et le minimum des temps de fin des visites.
- Astuce: Utiliser les fonctions `apoc.coll.min` et `apoc.coll.max`.
- Voir la documentation sur les types temporels. Particulièrement le type duration.
- Selon le hardware, cette requête peut prendre du temps à s'exécuter (37 secondes sur Aura, 4 secondes sur la machine de l'assistant)

⋆⋆⋆ 6. Implementer la méthode `setHighRisk` qui modifie la requête précèdente (5) pour ajouter un attribut `risk = "high"` à toutes les personnes `peopleToInform` et qui retourne le nom des personne (`highRiskName`) à haut risque.

⋆⋆ 7. Implementer la méthode `healthyCompanionsOf` qui retourne le nom des personnes en bonne santé (`healthyName`) à une distance de maximum 3 visites d'une personne donnée. (Ex: Si A visite le même endroit que B et B visite le même endroit que C alors A est à une distance de 2 visites de C).

⋆⋆ 8. Implementer la méthode `topSickSite` qui retourne le type de lieu (`placeType`) où il y a eu le plus de fréquentation des personnes malades (`nbOfSickVisits`) après leur confirmation. Retourner une seul valeur même s'il y a égalité.

⋆ 9. Implementer la méthode `sickFrom` qui retourne le nom des personnes malades (`sickName`) parmi une liste donnée.

Professeur : Nastaran Fatemi
Assistants : Christopher Meier
Edward Ransome

A Instructions de déploiement

A.1 Docker compose

1. Sur Linux ou MacOS:
   (a) Ouvrir un terminal dans le dossier ou vous avez extrait les fichiers du labo.
   (b) Exécuter le script `import.sh`. Il utilise un container éphémère pour importer le dataset dans un volume nommé. Le container *Neo4j* ne doit pas être démarré lors de l'importation.
2. Sur Windows:
   (a) Lancer Powershell en tant qu'administrateur
   (b) Naviguer à la racine du labo
   (c) Autoriser l'exécution du script avec:
   `> Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope Process`
   (d) Exécuter le script `import.ps1` avec `> .\import.ps1`. Le container *Neo4j* ne doit pas être démarré lors de l'importation.
3. Exécuter la commande `> docker-compose up` pour démarrer *Neo4j*.
4. L'interface web est disponible à `http://localhost:7474`
5. Pour vous connecter, laisser le password vide.

A.2 Neo4j Aura

1. Créer un compte Neo4j Aura.
2. Une fois votre adresse email confirmée. Vous arrivez sur une page pour créer votre base de données. Utiliser la configuration suivante (voir figure 2):
   Database type *Aura Free*
   Database name Au choix (par exemple *MAC*)
   GCP Region De préférence *Belgium (europe-west1)*
   Starting Dataset *Load or create your own data in a blank database*
3. Enregistrer les informations de login qui sont affiché (voir figure 3)
4. Attendre quelques minutes pour que la base de données soit prête.
5. Depuis la liste des base de données, cliquer sur le nom de la base nouvellement créer (voir figure 4)
6. Importer le fichier `contact-tracing-43.dump` (voir figure 5)
7. Attendre quelques minutes pour que l'importation termine.
8. Ouvrir l'interface Neo4j de la base de donnée (*Open with* > *Neo4j Browser*)

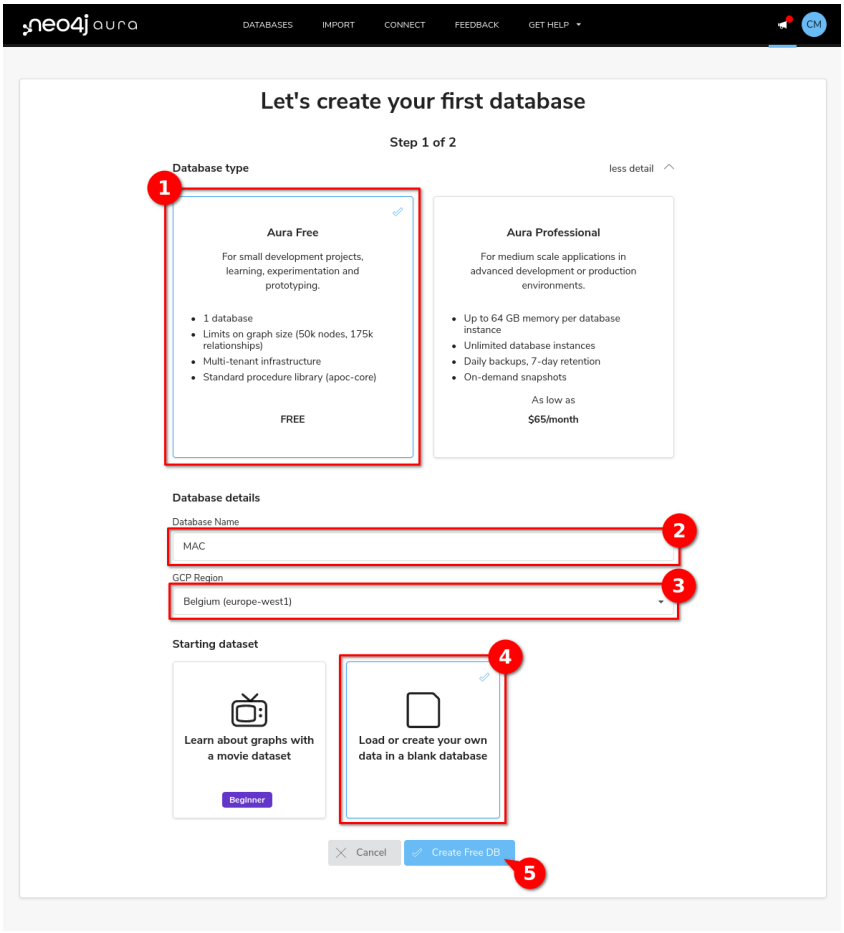

Figure 2 Capture d'écran de la création d'une base de donnée sur Neo4j Aura

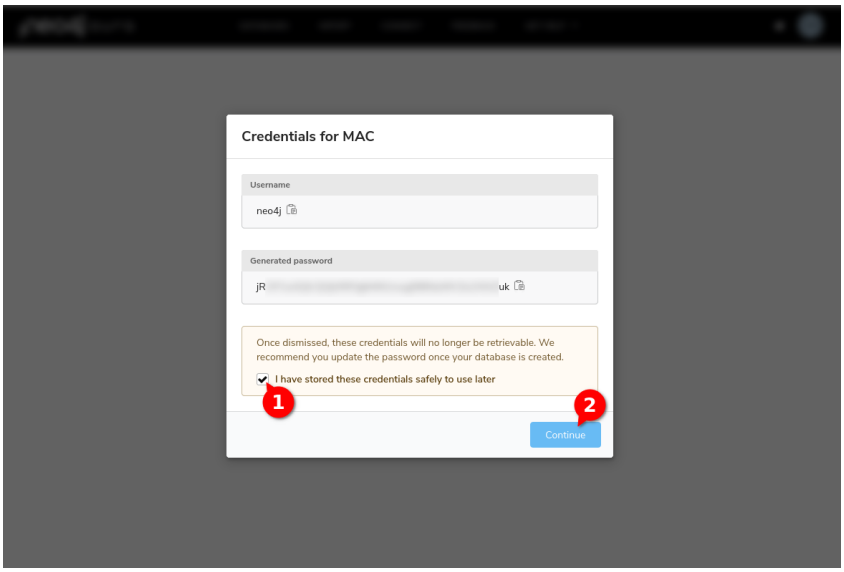

Figure 3 Capture d'écran de la récupération des informations de connexion sur Neo4j Aura

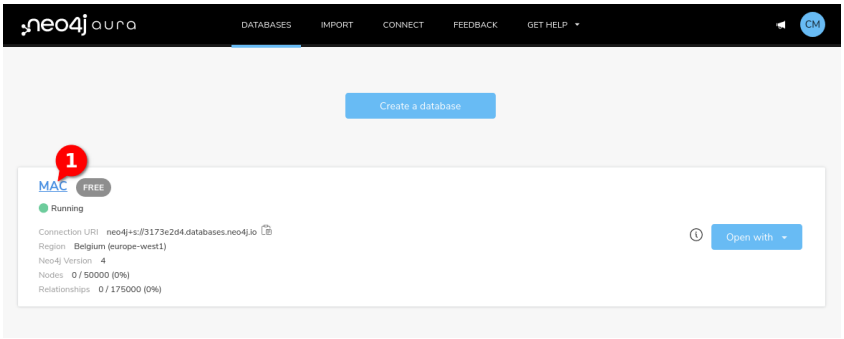

Figure 4 Capture d'écran des base de données disponible sur Neo4j Aura

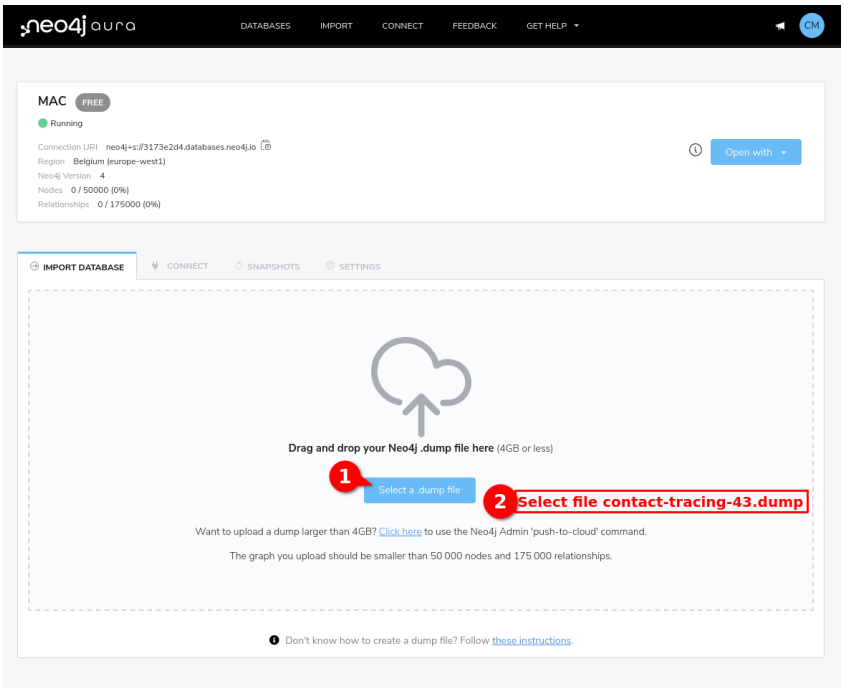

Figure 5 Capture d'écran de l'importation d'une base de donnée sur Neo4j Aura