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 Organisation

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

Ce laboratoire est à rendre sur Github Classroom à la date qui y est indiquée, 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 Github Classroom. 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 dossier `source` qui contient le jeu de données 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 17.

1.4 Déploiement

Nous vous recommandons d'utiliser le fichier `docker-compose.yml` fournit pour déployer une instance Neo4j avec le bon dataset :

1. Ouvrir un terminal dans le dossier ou vous avez extrait les fichiers du labo.
2. Exécuter la commande `> docker compose up` pour importer le jeu de données et démarrer *Neo4j*.
3. L'interface web est disponible à http://localhost:7474
4. Pour vous connecter, laisser le password vide.
5. Chaque fois que vous faites `> docker compose up` le dataset est réimporté et les modifications éventuelles sont perdues. Si, après la première importation, vous souhaitez garder vos modifications utilisez la commande `> docker compose up graphdb`.

Les autres possibilités (à utiliser à vos risques et périls) :

— Utiliser *Neo4j aura* pour déployer une base de donnée gratuite sur le cloud.
— Installer nativement *Neo4j Desktop*.

1.5 Dataset

Ce laboratoire utilise un jeu de données synthétique appelé *contact-tracing*.

Ce jeu de données 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 duplicata du noeud `Visit`. Ils sont interchangeable, selon les besoin de la requête.

Seul le dernier état de santé connu à la date du 2020-05-09 est enregistré pour les `Person`. Dans le cadre de ce labo, on suppose qu'une personne est malade tant qu'elle n'a pas effectué un test pour prouver son rétablissement.

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`.

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 jeu de données *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 jeu de données *contact-tracing*

Généré avec la requête `CALL db.schema.relTypeProperties`
Les relations `PERFORMS_VISIT`, `LOCATED_AT` et `PART_OF` n'ont pas de propriétés.

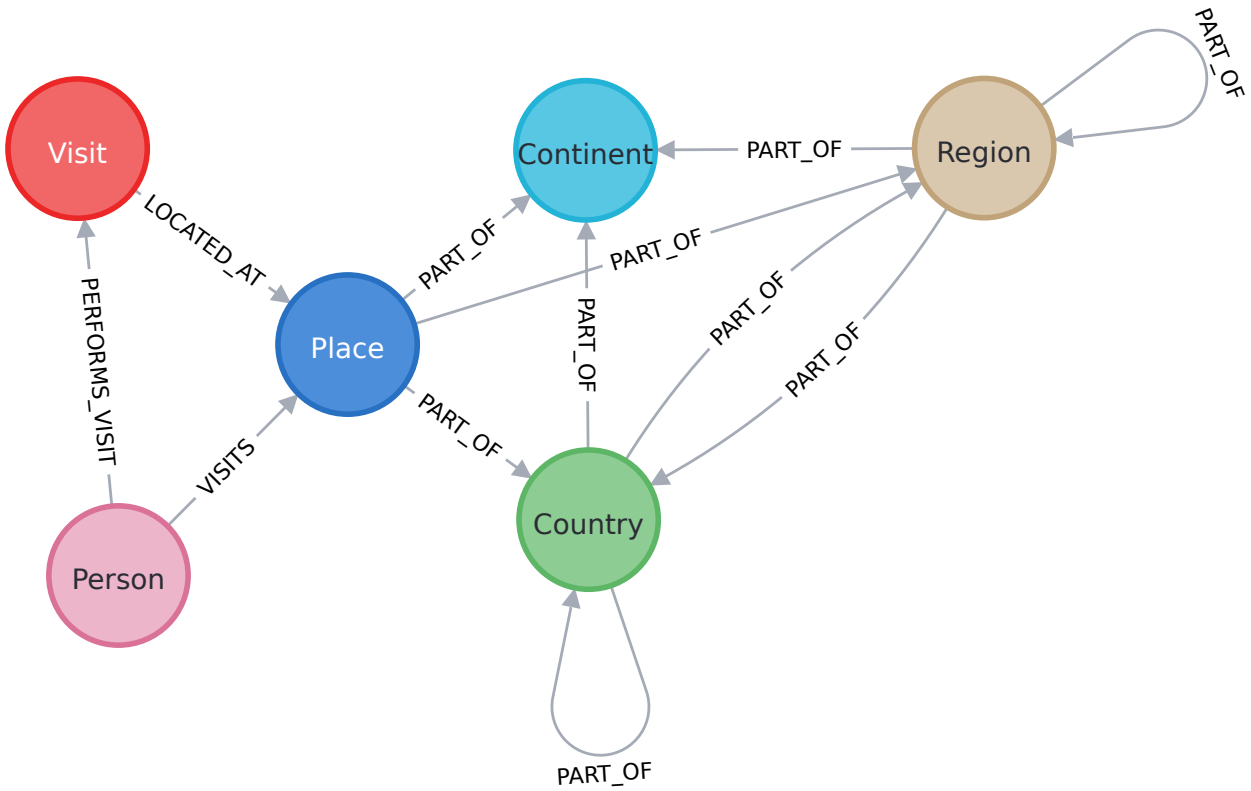

FIGURE 1 – Modèle du jeu de données *contact-tracing*

Généré avec la requête `CALL db.schema.visualization()`

3 Indications

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

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

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. Implémenter 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. La visite de la personne en bonne santé a commencé après le début de la visite de la personne malade et après la confirmation de son état de santé.

⋆ 2. Implémenter 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. La visite de la personne en bonne santé a commencé après le début de la visite de la personne malade et après la confirmation de son état de santé.

⋆ 3. Implémenter 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. Implémenter la méthode `sociallyCareful` qui retourne le nom des personnes malades (`sickName`) qui n'ont jamais fréquenté un "Bar" après la confirmation de leur maladie. Les personne qui ne fréquente de toute façon jamais de "Bar" sont aussi incluses.

⋆⋆⋆ 5. Implémenter la méthode `peopleToInform` qui retourne, pour chaque personne 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 (après chacune de leur confirmations).

- 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.

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

⋆⋆ 7. Implémenter 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. Implémenter 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. Implémenter la méthode `sickFrom` qui retourne le nom des personnes malades (`sickName`) parmi une liste donnée.