Neo4J dans la pratique (TP1 sur 3h) Correction

1. Préalable Neo4J

Neo4J est un système de gestion de bases de données graphe exploité et exploitable pour représenter des réseaux d'entités fortement connectées. Nous en abordons quelques principes de base de mise en œuvre . Pour ce faire, la version community 3.5.21 est disponible dans le cours Moodle (voir aussi `https://neo4j.com/download-center/` et versions plus récentes). Nous utiliserons cette version pour des raisons de compatibilité avec des plugins, que nous mobiliserons ensuite. Décompressez l'archive, et démarrez le serveur :

```
... ./bin/neo4j start
```

Listing 1 – ordre de mise en route du serveur

Une interface web (serveur Jetty) est disponible à l'adresse localhost:7474 et nous exploiterons le langage déclaratif Cypher de Neo4J.

Vous n"oublierez pas d'arrêter le serveur en fin de TP

```
... ./bin/neo4j stop
```

Listing 2 – stopper le serveur

2. CQL (Cypher Query Language)

Nous allons aborder les principales opérations "CRUD" pour Create, Read, Update, Delete) portant sur des objets du graphe. L'ensemble de ces ordres vous sont donnés dans un fichier textuel (extension cql).

2.1 Exercice 1

Testez dans la pratique les ordres donnés ci-dessous :

2.1.1 Explorer les ordres CREATE et MATCH

Vous créerez (ordre CREATE) des objets et des relations entre objets (ici des communes et des départements) puis vous enrichirez la description de ces objets (à l'aide de MATCH).

```
l, m, h, c alias valides le temps de la transaction

CREATE (m:Commune {nom:'MONTPELLIER', latitude:43.610769, longitude:3.876716,
    codeinsee:'34172'}) -[:WITHIN]-> (h:Departement {nom:'HERAULT', numero:'34'}),
```

```
(h) <-[:WITHIN]- (l:Commune {nom:'LUNEL', latitude:43.67445, longitude:4.135366,
    codeinsee:'34145'}), (m) -[:NEARBY]-> (l), (c:Commune {nom:'CLAPIERS',
    latitude:43.6575, longitude:3.88833, codeinsee:'34830', pop_17:5135}) -[:WITHIN]->
    (h), (m) -[:NEARBY {type:'border'}]-> (c)
RETURN l, m, h, c

MATCH (l:Commune {nom:'LUNEL'})
CREATE (m:Commune {nom:'NIMES', latitude:43.836699, longitude:4.360054,
    codeinsee:'30189'}) -[:WITHIN]-> (h:Departement {nom:'GARD', numero:'30'}), (m)
    -[:NEARBY]-> (l)

MATCH (h:Departement {nom:'GARD', numero:'30'})
CREATE (m:Commune {nom:'GARONS', latitude:43.770059, longitude:4.424545,
    codeinsee:'30125'}) -[:WITHIN]-> (h), (m) -[:NEARBY]-> (s:Commune {nom:'SOMMIERES',
    codeinsee:'30321'}), (s) -[:WITHIN]-> (h)

MATCH (g:Commune {nom:'GARONS'})
MATCH (n:Commune {nom:'NIMES'})
CREATE (g) -[:NEARBY]-> (n)
```

Listing 3 – Plusieurs ordres CREATE

```
MATCH (h:Departement {numero:'34'})
MATCH (m:Commune {nom:'MONTPELLIER'})
CREATE (c:Commune {nom:'MONTFERRIER-SUR-LEZ', latitude:43.671824, longitude:3.859265,
    codeinsee:'34169',pop_1975:1682}) -[:WITHIN]-> (h), (c) -[:NEARBY]-> (m),(cr:Commune
    {nom:'CRES', latitude:43.644825, longitude:3.936612,
    codeinsee:'34090',pop_1975:4507}) -[:WITHIN]-> (h), (cr) -[:NEARBY]-> (m),
    (ca:Commune {nom:'CASTELNAU-LE-LEZ', latitude:43.634144, longitude:3.897398,
    codeinsee:'34057',pop_1975:9446}) -[:WITHIN]-> (h), (ca) -[:NEARBY
    {type:'border'}]-> (m),(cas:Commune {nom:'CASTRIES', latitude:43.677589,
    longitude:3.985579, codeinsee:'34058',pop_1975:2494}) -[:WITHIN]-> (h), (cas)
    -[:NEARBY]-> (m), (stc:Commune {nom:'SAINT-CLEMENT-DE-RIVIERE', latitude:43.6844,
    longitude:3.8472, codeinsee:'34247',pop_1975:845,pop_2010:4985}) -[:WITHIN]-> (h),
    (stc) -[:NEARBY]-> (m)
```

Listing 4 – Un ordre CREATE très complet

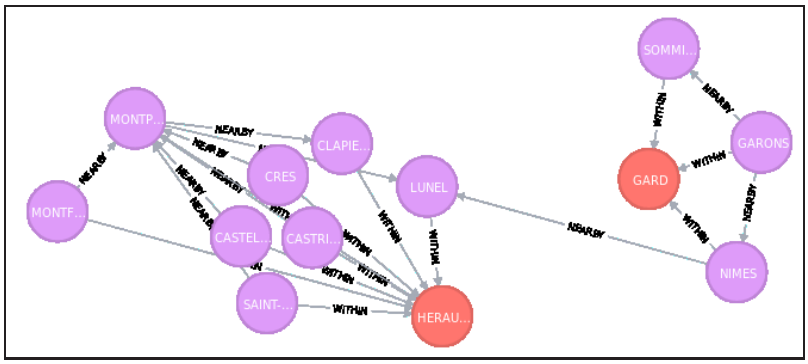

FIGURE 1 – Graphe résultat après création

2.1.2 Explorer la modification d'objets avec SET (UPDATE)

Vous testerez la mise à jour des objets et de leurs relations :

```
MATCH (s:Commune {nom:'SOMMIERES'})
SET s.pop_1975=3072, s.latitude=43.783450, s.longitude=4.089738
RETURN s


MATCH (s:Commune {nom:'SOMMIERES'}) <-[r:NEARBY]- (g:Commune {nom:'GARONS'})
SET r.distance = 27.4
```

Listing 5 – Exemple SET

2.1.3 Comprendre la notion de MERGE

La commande MERGE est une combinaison de CREATE et de MATCH et va permettre d'émailler des nœuds existants (en les créant au besoin) avec de nouvelles relations.

```
MATCH (c1:Commune)-[:NEARBY]->()<-[:NEARBY]-(c2:Commune)
MERGE (c1)-[:NEARBY]-(c2)
```

Listing 6 – exemple de fusion

2.2 Explorez la consultation avec MATCH et RETURN (RETRIEVE)

Consulter des objets et des relations entre objets

```
communes de l'HERAULT
MATCH (d:Departement {nom:'HERAULT'}) <-[p:WITHIN]- (n:Commune)
RETURN d, n, p

communes qui sont proches de MONTPELLIER
MATCH (m:Commune {nom:'MONTPELLIER'}) -[:NEARBY]- (n:Commune)
RETURN m, n
```

Listing 7 – Exemples RETURN

Exploitez la notion de chemin pour renvoyer le réseau de proximité depuis MONTPELLIER

```
MATCH (m:Commune {nom:'MONTPELLIER'}) -[:NEARBY*]- (n:Commune)
RETURN m, n

communes qui sont proches d'une commune qui est proche de Montpellier
MATCH (m:Commune {nom:'MONTPELLIER'}) -[:NEARBY*2]- (n:Commune)
RETURN m, n

tailles des chemins menant de Montpellier a Nimes (ou son inverse)
MATCH p=((c1:Commune {nom:'MONTPELLIER'})-[:NEARBY*]-(c2:Commune {nom:'NIMES'}))
RETURN length(p)
```

Listing 8 – RETURN et chemins

2.2.1 Supprimer des objets et des relations avec DELETE et DETACH (DELETE)

Supprimer tout ou une partie du graphe construit

```
-- Tout supprimer
MATCH (n) OPTIONAL MATCH (n)-[r]-() DELETE n,r

--ou

MATCH (n)
DETACH DELETE n

-- un noeud
MATCH (n:Commune {nom:'SOMMIERES'})
DELETE n

-- une relation
MATCH (n:Commune { nom:'GARONS' })-[r:NEARBY]->()
DELETE r

-- un attribut de noeud
MATCH (n:Commune { nom:'NIMES' })
REMOVE n.codeinsee
return n
```

Listing 9 – Exemple DELETE

2.2.2 Tout supprimer

Vous supprimerez tous les objets de votre graphe avant de passer à la partie suivante (chargement de données)

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

3. Import de données au format tabulé avec Cypher

Un exemple d'import de données dans la base de données Neo4J vous est fourni [1]. Cet exemple exploite la syntaxe cypher et différents ordres de chargement (LOAD CSV) pour insérer respectivement des données sur des communes, départements et régions. Les correspondances entre les communes, départements et régions sont également posées à l'aide de la relation WITHIN. Vous testerez cet exemple qui s'applique aux données commune-département-région. Les fichiers au format csv sont disponibles depuis le répertoire `import`. Vous utiliserez les ordres donnés dans le fichier chargement.cql depuis le serveur Jetty pour charger les données. Un exemple est présenté ici

```
LOAD CSV WITH HEADERS FROM 'file:///Region.csv' AS regions
CREATE (r:Region {id : toInteger(regions.id), name : regions.name});

LOAD CSV WITH HEADERS FROM 'file:///Com_Dep.csv' AS com_dep
MERGE (co : Commune { codeinsee : toInteger(com_dep.codeinsee) })
MERGE (d : Departement { id : com_dep.id })
CREATE (co)-[:WITHIN]->(d);
```

1. voir aussi http ://neo4j.com/docs/stable/cypherdoc-importing-csv-files-with-cypher.html

Listing 10 – ordres de chargement

3.1 Enrichir le graphe

Il s'agit d'indiquer qu'une région possède une capitale administrative nommée chef-lieu de région, qu'un département a un une capitale administrative nommée chef-lieu de département et qu'une commune peut dépendre d'un chef-lieu de région et d'un chef-lieu de département. Vous définirez les éléments supplémentaires (nouvelles relations, propriétés de relations par exemple) qui vous semblent nécessaires à la prise en charge de cette modélisation. Indiquez par exemple que Montpellier est chef lieu de département de l'Hérault et que Nîmes est chef-lieu de département du Gard. Vous pourrez aussi rajouter la ville de Toulouse (TOULOUSE de code insee 31555, latitude 43.604652 et longitude 1.444209) dans le département de la Haute Garonne (31) et dans la région Occitanie. Toulouse est à la fois chef-lieu de département et chef-lieu de région.

```
MATCH (mo:Commune {name:'MONTPELLIER'}), (he:Departement {name:'HERAULT'})
CREATE (mo) -[c:CHEFLIEUDE]-> (he)

MATCH (d:Departement {name:'HAUTE-GARONNE'}), (r:Region {name:'OCCITANIE'})
CREATE (to:Commune {name:'TOULOUSE', codeinsee:31555, latitude:43.604652,
    longitude:1.444209}) -[c1:CHEFLIEUDE]-> (d),
(to) -[c2:CHEFLIEUDE]-> (r)

MATCH (n:Commune) -[:CHEFLIEUDE]-> (:Region) SET n:CapitaleRegion RETURN n
MATCH (n:Commune) -[:CHEFLIEUDE]-> (:Departement) SET n:Prefecture RETURN n
```

Listing 11 – exemples de correction (exercice ouvert)

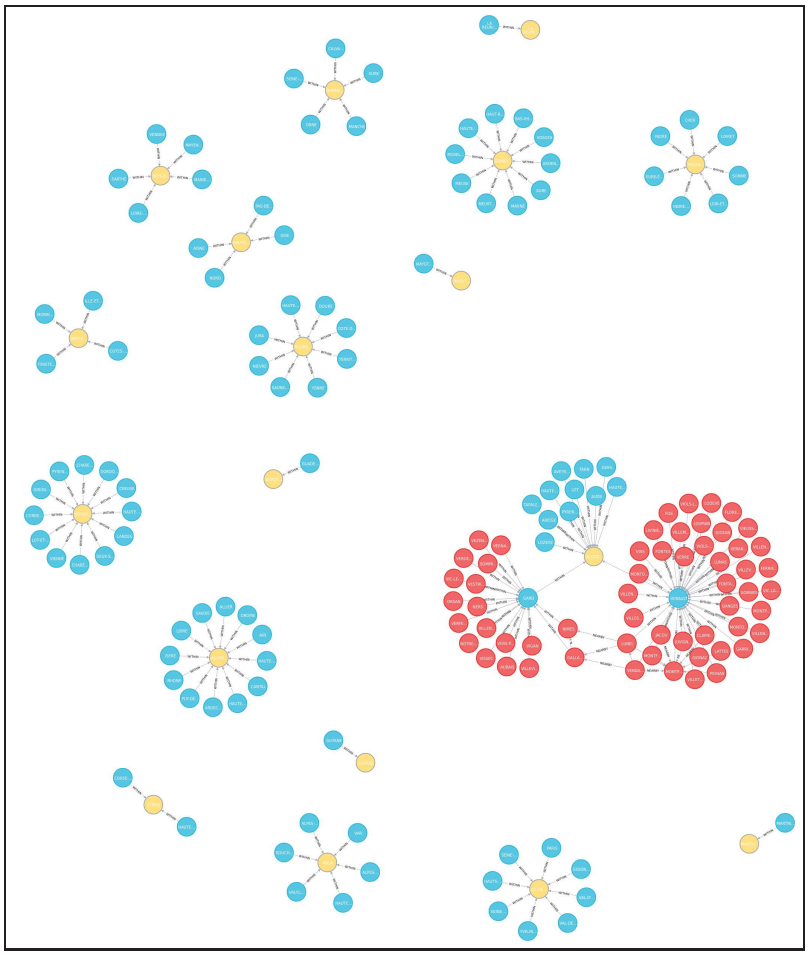

FIGURE 2 – Graphe résultant du chargement

4. Exercices de consultation

Vous répondrez aux questions ci-dessous :

1. compter le nombre de départements de la région OCCITANIE

```
MATCH (d:Departement) -[:WITHIN]-> (r:Region {name:'OCCITANIE'})
RETURN count(d) as nombreDep
```

2. compter le nombre de départements par région et renvoyer le nom de région et le nombre de départements associés à ces régions

```
MATCH (d:Departement) -[:WITHIN]-> (r:Region)
RETURN r.name, count(d) as nombreDep
```

3. compter le nombre de départements par région (quand la région en compte plus de 6) et renvoyer le nom de région et le nombre de départements associés à ces régions

```
MATCH (d:Departement) -[:WITHIN]-> (r:Region)
WITH r, count(d) as nbD
WHERE nbD > 6
```

```
RETURN r.name, nbD
```

4. retourner le nom des communes ainsi que le nom de leur département, quand ces communes sont voisines de MONTPELLIER

```
MATCH (m:Commune) -[:NEARBY]- (v:Commune) -[:WITHIN]-> (d:Departement)
WHERE m.name ='MONTPELLIER'
RETURN v.name, d.name
```

5. retourner les communes (nom et codeinsee) qui sont voisines de communes voisines de MONTPELLIER et qui ne sont pas dans l'HERAULT

```
MATCH (m:Commune) -[:NEARBY]- (v1:Commune)
-[:NEARBY]- (v2:Commune) -[:WITHIN]-> (d:Departement)
WHERE m.name ='MONTPELLIER' AND d.name <> 'HERAULT'
RETURN v2.name, d.name
```

6. retourner le nom de chaque commune ainsi que le nom de sa région et de son département de rattachement

```
MATCH (c:Commune) -[:WITHIN]-> (d:Departement)
-[:WITHIN]-> (r:Region)
RETURN c.name, d.name, r.name
```

7. retourner tous les chemins qui relient MONTPELLIER à NIMES

```
MATCH p=((m:Commune {name:'MONTPELLIER'}) -[:NEARBY*]-
(n:Commune {name:'NIMES'}))
RETURN p
```

8. retourner le plus court chemin (ou un des plus courts chemins) entre MONTPELLIER et NIMES

```
MATCH (c1:Commune {nom:'MONTPELLIER'}), (c2:Commune {nom:'NIMES'}),
path = shortestpath((c1)-[:NEARBY*]-(c2))
RETURN path
```

5. Exercices sur le mode embarqué

Un exemple de classe Java vous est donné. Vous vous l'approprierez pour créer quelques nœuds supplémentaires à savoir les nœuds des communes de Saint Gaudens (SAINT-GAUDENS, code insee 31483, latitude 43.106895 et longitude .723763) et de Balma (BALMA, code insee 31044, latitude 43.606098, et longitude 1.500032) en Haute Garonne (31). Vous manipulerez également une requête Cypher de type "RETRIEVE" qui va permettre de renvoyer le nom et le code insee des communes qui sont plus à l'est que Montpellier (en d'autres termes dont la longitude est supérieure à celle de Montpellier).

```
MATCH (c1:Commune {name:'MONTPELLIER'}), (c2:Commune)
WHERE c1.longitude < c2.longitude
RETURN c2.name, c2.codeinsee
```

6. Exercices de consultation sur le modèle

1. renvoyer le nombre de catégories (labels) venant typer les nœuds du graphe

   ```
   MATCH (n) RETURN distinct labels(n)
   ```

2. renvoyer le nombre de catégories (type) venant typer les relations du graphe

   ```
   MATCH ()-[r]-() RETURN distinct type(r)
   ```

3. renvoyer les catégories de nœuds et les types de relations nouées à partir de ces nœuds

   ```
   MATCH (n)-[r]-() RETURN distinct labels(n), type(r)
   ```

4. retourner le type et le nombre d'instances de relation

   ```
   MATCH ()-[r]->() RETURN type(r), count(*) as cardinality_relation
   ```

7. Modèle de travail

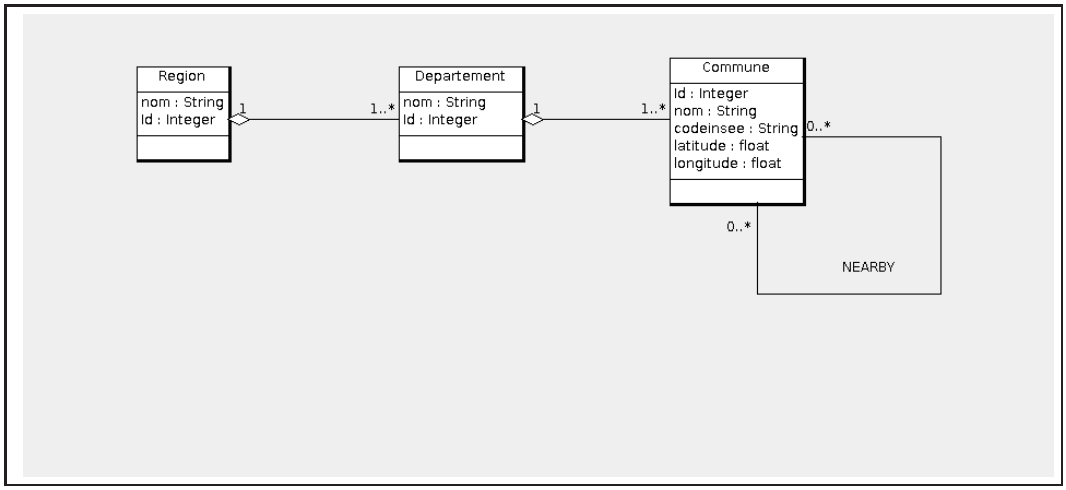

FIGURE 3 – Diagramme de classes UML de l'exercice