Tutoriel Neo4j

Pierre Marrec

18 août 2020

Le début et la création du graphe

Pour commencer, Neo4j utilise des bases de données sous formes de graphes relationnels. On a des noeuds et des relations entre les noeuds. Les noeuds et les relations peuvent avoir des types différents et plusieurs propriétés.
On va commencer par créer la base de données. Pour ça, on va utiliser le début du tutoriel intégré à Neo4j.

```
:play graph-movie
```

Passez à la slide suivant et cliquez sur la partie de code sur le droite. Ça commence par :

```
CREATE (TheMatrix:Movie {title:'The Matrix', released:1999,...
```

Cela va copie le code dans la barre d'exécution. Une fois exécuté, ce code va créer un graphe contenant des acteurs, les films dans lequels, ils ont joué, des réalisateurs et des films qu'ils ont réalisés, des reviewers etc.
Ensuite, on va commencer par regarder à quoi ressemble le graphe :

```
MATCH (n) RETURN n
```

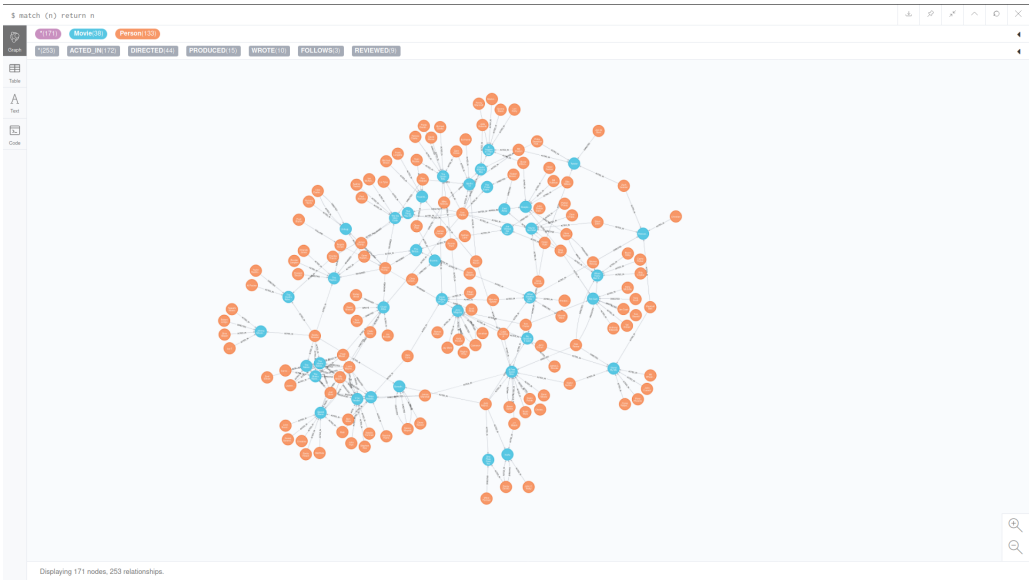

FIGURE 1

Il y a 171 noeuds dont 133 personnes et 38 films. De même, il y a 253 relations de différents types.

Les requêtes

Revenons sur la première requête qu'on a faite.

```
MATCH (n) RETURN n
```

On a simplement dit "Affiche tout". Cela correspond au

```
SELECT * FROM Table
```

du SQL.

Dans une requête Cypher, on commence par "MATCH (Noeud)..." et on fini par "RETURN ce qu'on veut afficher".

Maintenant si je veux afficher les films dans lesquels Tom Cruise a joués, on peut faire.

```
MATCH (a:Person {name:"Tom Cruise"})-[rel:ACTED_IN]->(m) return a,m
```

Dans cette requête, il y a beaucoup de choses mais c'est assez intuitif.
Déjà, on commence par déclarer la variable "a" qui est de type "Person". Ainsi dans l'executeur de la requête ne va chercher que dans les "Person". Ensuite, on filtre en disant qu'on veut que son attribut "name" soit égal à "Tom Cruise".
La commande "-[rel :ACTED_IN]->" signifie qu'on ne regarde que les relations qui sont du type "ACTED_IN". Puis, on cherche tous objets (m) qui sont reliés à (a) par la relation ACTED_IN. On affiche, au final, le noeud (a) ie "Tom Cruise" et toutes les noeuds (m) ie tous les films dans lesquels il a joué.

On peut aussi filter avec la commande "WHERE".

```
MATCH (film:Movie) WHERE film.released >= 2000 AND film.released < 2010
RETURN film.title
```

ou

```
MATCH (film:Movie) WHERE 2000<=film.released<2010 RETURN film.title
```

Ici on a selectionné tous les noeuds de type "Movie" et demandé à ce que ces films a été sortis entre 2000 et 2009 ie leur attribut "realeased" soit entre 2000 et 2009.
Enfin, on affiche le titre de ces films. On remarque que neo4j ne renvoit pas de graphe mais un tableau. En effet, on lui a demandé d'afficher les attributs "title" et non les noeuds du graphe relationnel.
On peut aussi ajouter d'autres mot-clé comme "LIMIT N" à la fin de la commande pour limité le nombre de résultat à N, "ORDER BY variables ASC/DESC" pour trier selon les variables demandé ou encore "DISTINCT" pour n'affiche qu'une seule fois chaque élément renvoyé.
On peut aussi utiliser des fonctions comme "COUNT(variable)" pour compter le nombre de variables renvoyées. Il y a aussi la fonction "collect" qui permet d'aggréger en une liste les valeurs renvoyées. Par exemple ,

```
MATCH (a:Person {name:"Tom Cruise"})-[rel:ACTED_IN]->(m)
RETURN a.name,collect(m.title)
```

qui affiche les films de Tom Cruise ou alors

```
MATCH (a:Person)-[rel:ACTED_IN]->(m) RETURN a.name,collect(m.title)
```

qui affiche les films de tout les acteurs.
Si on veut connaître le nombre de film dans lesquels chaque acteurs à jouer on peut demander à compter le nombre de film dans la liste.

```
MATCH (a:Person)-[rel:ACTED_IN]->(m) WITH a.name AS name, collect(m.title) AS liste
RETURN a.name,size(liste) AS nombre_films
```

J'en profite pour utiliser les mot-clés "WITH" et "AS" qui permettent de nommer des types de variables.
On peut aussi utiliser une autre syntaxe pour trouver les noeuds qui sont à une certaine distance dans le graphe.

```
MATCH (a:Person {name:"Tom Cruise"})-[*1..4]-(hollywood)
RETURN DISTINCT hollywood
```

Ici on affiche les personnes qui sont à une distance d'aux plus 4 de Tom Cruise dans le graphe. (on remarquera qu'on autorise les relations dans les deux sens.)
On n'est pas obligé de définir la distance ou peut simplement mettre "-[*]-" comme par exemple pour trouver le plus court chemin entre Tom Cruise et Carrie Fisher.

```
MATCH p=shortestPath(
(bacon:Person {name:"Tom Cruise"})-[*]-(meg:Person {name:"Carrie Fisher"})
)
RETURN p
```

Modification des graphes

On a vu précédemment la fonction CREATE. On va maintenant plus rentrer dans les détails. Dans un Graphe, on peut créer soit un noeud soit une arête. Pour un noeud :

```
CREATE (a:Person {name : "Leonardo Di Caprio", born:1974})
CREATE (b:Movie {title : "Inception", released:2010})
```

Pour une relation :

```
MATCH (a:Person {name:"Leonardo Di Caprio"})
MATCH (b:Movie {title:"Inception"})
CREATE (a)-[rel:ACTED_IN]->(b)
```

On remarque ici qu'on doit faire une recherche des deux noeuds avant de créer la relation. Prenons un exemple :

```
CREATE (a:Person {name : "Leonardo Di Caprio", born:1974})-[rel:ACTED_IN]->(b:Movie {ti
```

Si on avait fait directement cette commande avant de créer les noeuds, on aurait créer les noeuds et la relation. Mais si on l'avait fait après la création des noeuds, il y aurait eu 2 Leonardi Di Caprio et 2 Inception dans le graphe.

On peut aussi modifier une propriété avec la "SET". Par exemple,

```
MATCH (a:Movie{ title:"Inception"})
SET a.tagline = "Dreaming"
return a
```

Le "return a" n'est pas nécessaire. Il sert juste à vérifier ce qu'on à modifier.

Pour la suppression, c'est assez similaire.

```
MATCH (a:Person {name:"Leonardo Di Caprio"})-[r:ACTED_IN]->(b:Movie {title:"Inception"}
DELETE r
```

Pour les noeuds :

```
MATCH (a:Person {name:"Leonardo Di Caprio"})
MATCH (b:Movie {title:"Inception"})
DELETE a,b
```

Ou alors tout-en-1 :

```
MATCH (a:Person {name:"Leonardo Di Caprio"})-[r:ACTED_IN]->(b:Movie {title:"Inception"}
DELETE a,b,r
```

Pour supprimer une propriété, on utilise à la place "REMOVE".

Parler du Merge ?.

Convertir depuis le SQL

Si on veut convertir une base de données SQL en base de données sous forme de graphes, il faut créer tous les noeuds et créer les relations à partir des tables SQL.
Chaque table SQL correspond en général à un type de noeud. Utilisons un exemple de Neo4j pour illustrer.

```
LOAD CSV WITH HEADERS FROM "http://data.neo4j.com/northwind/products.csv" AS row
CREATE (n:Product)
SET n = row,
n.unitPrice = toFloat(row.unitPrice),
n.unitsInStock = toInteger(row.unitsInStock), n.unitsOnOrder = toInteger(row.unitsOnOrd
n.reorderLevel = toInteger(row.reorderLevel), n.discontinued = (row.discontinued <> "0"

LOAD CSV WITH HEADERS FROM "http://data.neo4j.com/northwind/categories.csv" AS row
CREATE (n:Category)
SET n = row

LOAD CSV WITH HEADERS FROM "http://data.neo4j.com/northwind/suppliers.csv" AS row
CREATE (n:Supplier)
SET n = row
```

Neo4j récupère les attributs des tables SQL et les transformer en propriétés du type de noeuds. Par contre, si on veut avoir des entiers ou flottant, il faut convertir.

```
CREATE INDEX ON :Product(productID)
CREATE INDEX ON :Category(categoryID)
CREATE INDEX ON :Supplier(supplierID)
```

Cela permet d'indexer les noeuds.

Pour les relations, on fait des "MATCH" et on créer les relations.

```
MATCH (p:Product),(c:Category)
WHERE p.categoryID = c.categoryID
CREATE (p)-[:PART_OF]->(c)
```

```
MATCH (p:Product),(s:Supplier)
WHERE p.supplierID = s.supplierID
CREATE (s)-[:SUPPLIES]->(p)
```

On peut ensuite faire des requêtes.