# Intégration de Neo4j pour établir des liens entre des concepts

**Grumpy** est un orque bien bête. Il faut donc lui apprendre des concepts et des liens entre des concepts. Éventuellement, il pourrait vous surprendre (ou pas!).



Pour lui apprendre des choses sur la vie, il faut se limiter à 3 mots par phrase. Par exemple, on pourrait lui apprendre : « chat est animal » et il ferait l’association CHAT 🡪 *est* 🡪 ANIMAL. Par la suite, on pourrait lui apprendre que « animal est vivant ».

Ceci donnerait le graphe suivant :

est

est

Lorsqu’il possède suffisamment de concepts, il est possible de lui demander une question similaire à : « chat est vivant? » et il répondrait : « Oui, car CHAT *est* ANIMAL, et ANIMAL *est*  VIVANT ».

## Notes

* IntelliJ a été utilisé pour créer le programme, vous devriez donc l’utiliser pour compléter ce travail.
* Pour réaliser ce projet, je vous recommande FORTEMENT de suivre les étapes en ordre.
* Si vous avez des questions, n’hésitez pas à me demander de l’aide.
* **IMPORTANT** : Ne modifiez en aucun cas la classe « CorClient.java », sous peine d’avoir 0.

## Exemples d’interaction avec Neo4j dans le programme

### Exemple 1 : Faire une requête Cypher et retourner un ou plusieurs éléments

Dans l’exemple suivant, la requête permet de retourner toutes les personnes qui ont comme age : « 23 ». Leur nom est ensuite affiché dans la console.

**Important :** Ce qui est retourné peut être autant un nœud (objet : Node) qu’une relation (objet : Relationship).

try {

Session session = DBConnection.getConnection();

Map<String, Object> params = new HashMap<String, Object>();

params.put("p1", 23);

StatementResult result = session.run("MATCH (a:Personne) WHERE a.age = $p1 RETURN a",

params);

while(result.hasNext()) {

Record record = result.next();

Node node = record.get("a").asNode();

System.out.println(node.get("nom").asString());

}}

catch (Exception e) {

e.printStackTrace();

}

Si l’information retournée est une relation, alors il faut plutôt faire :

Relationship relationship = record.get("a").asRelationship();

### Exemple 2 : Création de deux noeuds ainsi qu’une relation entre eux.

// Dans la méthode voulue

try {

Session session= DBConnection.getConnection();

Map<String, Object> params = new HashMap<String, Object>();

params.put("p1", 23);

session.run("CREATE (a:Personne {age: $p1})", params);

params = new HashMap<String, Object>();

params.put("p1", 32);

session.run("CREATE (a:Personne {age: $p1})", params);

params = new HashMap<String, Object>();

params.put("p1", 23);

params.put("p2", 32);

params.put("p3", "2010");

session.run("MATCH (a:Personne),(b:Personne) WHERE a.age = $p1 AND b.age = $p2

CREATE (a)-[r:TYPE\_FICTIF { annee : $p3 } ]->(b)", params);

}

catch (Exception e) {

e.printStackTrace();

}

### Exemple 3 : Cypher de création d’un index

CREATE INDEX idx\_p\_name FOR (p:Person) ON (p.name)

### Exemple 4 : Cypher permettant de vérifier si un index existe

SHOW indexes where name = "idx\_p\_name"

### Exemple 5 : Pour compter le nombre de noeuds

Session session = DBConnection.getConnection();

StatementResult result = session.run("MATCH (n) RETURN COUNT(n) as nb");

int count = result.next().get("nb").asInt());

### Exemple 6 : Supprimer un index

Pour supprimer un index, il s’agit de faire cette requête cypher :

DROP INDEX idx\_p\_name

## Les tâches à réaliser

### Étape 0 : Vérifier les informations de connexion

Dans la fonction DBConnection.getConnection(), il faut vérifier que votre nom d’usager et le mot de passe sont valides.

Si vous n’avez pas d’erreurs de connexion au chargement du programme, c’est que les informations sont valides.

Évidemment, assurez-vous que neo4j roule (que vous avez démarré la console/neo4j browser)

### Étape 1 : L’intégration de nouveaux concepts

La première étape consiste à enseigner des concepts à Grumpy en écrivant de courtes phrases (de 3 mots!).

Une phrase prend la forme suivante : « *Concept1* lien *Concept2 »*. Chaque *concept* est donc un nœud et le *lien* correspond à la relation. Voici quelques exemples :

- Table contient chaise

- Soleil est chaud

- Mer possède poisson

Pour ce faire, modifiez la méthode : « ConceptDAO.addKnowledge() ». Ceci est une première version, qui sera ensuite raffinée à l’étape 4.

**Concernant les noeuds**

**TOUS** Les nœuds doivent avoir le libellé (label) : « **Concept** » (c’est le type de nœud, comme :Personne).

Un nœud doit posséder l’attribut « **nom** », qui correspond au concept (chat, animat, chien, argent, etc.)

**Concernant les relations**

**TOUTES** les relations doivent être de type « **LIEN** ». Une relation doit avoir comme propriété « **desc** », qui correspond au verbe utilisé (ex : est, aime, contient, possède, etc.)

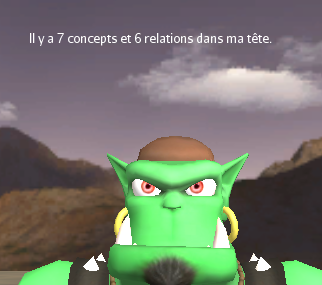
public static boolean addKnowledge(String concept1, String link, String concept2) {  
 boolean alreadyKnow = true;  
  
 try {  
 Session session= DBConnection.*getConnection*();  
  
 Map<String, Object> params = new HashMap<String, Object>();  
 params.put("p1", concept1);  
 session.run("CREATE (a:Concept {nom: $p1})", params);  
  
 params = new HashMap<String, Object>();  
 params.put("p3", concept2);  
 session.run("CREATE (a:Concept {nom: $p3})", params);  
  
 params = new HashMap<String, Object>();  
 params.put("p1", concept1);  
 params.put("p2", link);  
 params.put("p3", concept2);  
 session.run("MATCH (a:Concept),(b:Concept) WHERE a.nom = $p1 AND b.nom = $p3 CREATE (a)-[r:LIEN { desc : $p2 }]->(b)", params);  
 }  
 catch (Exception e) {  
 e.printStackTrace();  
 }  
  
  
 return alreadyKnow;  
}

### Étape 2 : « stats »

Lorsque la phrase «grumpy stats please?» est envoyée, Grumpy doit afficher le nombre de nœuds et le nombre de relations présent dans sa mémoire.

Pour ce faire, modifiez la méthode « ConceptDAO.getStats() »

Par exemple :



Si vous avez ajouté 1 seule connaissance « chat est noir », alors il devrait y avoir 2 concepts et 1 relation.

#### Notes/Requête/Énoncé:

public static Integer[] getStats() {  
 int nbRelations = 0;  
 int nbNodes = 0;  
  
 Session session = DBConnection.*getConnection*();  
  
 // Compte les relations  
 StatementResult result = session.run("MATCH (n:Concept)-[r:LIEN]->() RETURN COUNT(r) as nb");  
 nbRelations = result.next().get("nb").asInt();  
  
// Compte les nœuds  
 StatementResult result2 = session.run("MATCH (n:Concept) RETURN COUNT(n) as nb");  
 nbNodes = result2.next().get("nb").asInt();  
  
 return new Integer[]{nbNodes, nbRelations};  
 }

### Étape 3 : « Flash Grumpy! »

Lorsque la phrase «flash your memory» est envoyée, Grumpy doit **supprimer** toute sa mémoire, c’est-à-dire les nœuds et leurs relations.

Pour ce faire, modifiez la méthode : ConceptDAO.flash();

#### Notes/Requête/Énoncé:

public static void flash() {  
 Session session = DBConnection.*getConnection*();  
 StatementResult result = session.run("MATCH (N) DETACH DELETE N");  
}

### Étape 4 : Raffinement de l’étape 1

Avant d’ajouter les nœuds et la relation entre ceux-ci, il faudra d’abord vérifier si les concepts existent, et si la relation existe également!

Pour ce faire, modifiez les méthodes suivantes (voir la Javadoc pour détails):

- ConceptDAO.getConceptNode()

- ConceptDAO.getRelationship()

- ConceptDAO.addKnowledge() (en verifiant si les noeuds/relation existent, via les deux autres méthodes)

Pseudocode de l’algorithme :

…

Node concept1 = getConceptNode(…)

Node concept2 = getConceptNode(…)

if (concept1 == null) {

concept1 = createNode…

}

if (concept2 == null) {

concept2 = createNode…

}

Relationship r = getRelationShip(…)

if (r == null) {

// créer la relation

}

…

Notes:

- La méthode doit retourner « true » si Grumpy savait déjà cette connaissance. Autrement, retourner « false »

#### Notes/Requête/Énoncé:

public static boolean addKnowledge(String concept1, String link, String concept2) {  
 boolean alreadyKnow = false;  
 try {  
 Session session= DBConnection.*getConnection*();  
  
 Map<String, Object> params = new HashMap<String, Object>();  
  
 Node node1 = *getConceptNode*(concept1);  
  
 Node node2 = *getConceptNode*(concept2);  
  
 if (node1 == null){  
 params.put("p1", concept1);  
 session.run("CREATE (a:Concept {nom: $p1})", params);  
 node1 = *getConceptNode*(concept1);  
 }  
  
  
 if (node2 == null){  
 params = new HashMap<String, Object>();  
 params.put("p3", concept2);  
 session.run("CREATE (a:Concept {nom: $p3})", params);  
 node2 = *getConceptNode*(concept2);  
 }  
  
 Relationship r = *getRelationship*(node1, node2, link);  
  
  
  
 if (r == null){  
 params = new HashMap<String, Object>();  
 params.put("p1", concept1);  
 params.put("p2", link);  
 params.put("p3", concept2);  
 session.run("MATCH (a:Concept),(b:Concept) WHERE a.nom = $p1 AND b.nom = $p3 CREATE (a)-[r:LIEN { desc : $p2 }]->(b)", params);  
 alreadyKnow = true;  
 }  
 }  
 catch (Exception e) {  
 e.printStackTrace();  
 }  
  
 return alreadyKnow;  
}

### Étape 5 : Tests

Pour tester le fonctionnement de votre base de données (et les interrogations), demandez-le à Grumpy! Par exemple, si vous avez insérer : « lion est animal » et « animal doit manger », écrivez : « lion doit manger**?** ». Si le chemin dans le graphe existe, il vous le dira.

Il faut cependant que vos nœuds possèdent bien la propriété « nom » et que les relations possèdent « desc ».

### Étape 6 : « Cogite »

Si un index n’est pas encore créé sur les nœuds/concepts, la phrase « Cogite » devrait indiquer à Grumpy d’en créer un. Dans le cas où l’index existe déjà, il ne doit pas en refaire un.

Pour ce faire, modifiez la méthode « ConceptDAO.indexExists() » et « ConceptDAO.addIndex() »

#### Notes/Requête/Énoncé:

public static boolean indexExists() {  
  
 boolean exists = false;  
  
 Session session= DBConnection.*getConnection*();  
  
 StatementResult result = session.run("SHOW indexes where name = idx\_p\_name");  
  
 if(result.hasNext()) {  
 exists =true;  
  
 }  
  
  
 return exists;  
}

public static boolean addIndex() {  
 boolean added = false;  
  
 Session session= DBConnection.*getConnection*();  
  
 if (!*indexExists*()){  
 session.run("CREATE INDEX idx\_p\_name FOR (p:Concept) ON (p.nom)");  
 added = true;  
 }  
 return added;  
}

### Étape 7 : Raffinement de l’étape 3

Lorsque Grumpy vide sa mémoire, il doit supprimer l’index créé à l’étape 6 également.

public static void flash() {  
 Session session = DBConnection.*getConnection*();  
 StatementResult result = session.run("MATCH (N) DETACH DELETE N");  
 StatementResult result2 = session.run("DROP INDEX idx\_p\_name");  
}

### Étape 8 : Correction

La dernière étape consiste à activer la correction de votre projet. Pour ce faire, il faut écrire la phrase suivante dans le logiciel Grumpy :

correction *IP* *motDePasse* *nomDeOuDesÉtudiants*