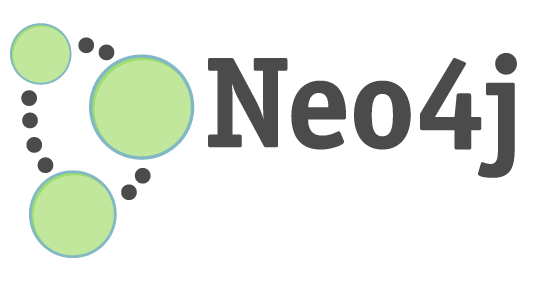
Introduction aux BDD Orientées Graphe (Neo4j)

**2015**

Bernard Dousset

IRIT

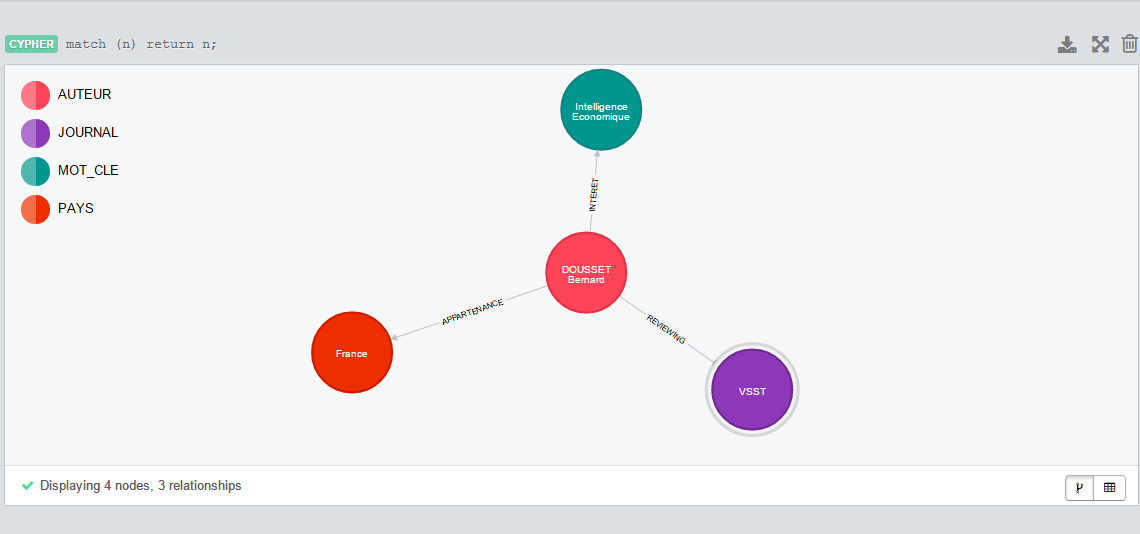
10/03/2015



Les bases de données relationnelles ont pu résoudre la plupart des problématiques de stockage pour les données de nos applications. Mais de temps en temps Il se peut que cela devienne très compliqué de représenter de façon relationnelle certains types de données. Par exemple un graphe dynamique d’objet du même type. Je vous invite donc, à découvrir une autre face du monde des bases de données.

**Présentation de Neo4j**

Imaginez un chercheur qu’on voudrait avoir une idée sur son environnment (son pays, les domaines sur lesquels il travaille, les journaux qu’il aurait pu reviewés etc…). En voici un exemple basique :



L’implémentation du schéma ci-dessus dans une base de données relationnelle n’est pas facile. Il existe plusieurs façons de le faire, comme le Pattern Query pour faire des résolutions, mais cela reste compliqué à utiliser. Pour résoudre ces problèmes plusieurs bases de données de type graphe existent, dont la célèbre base Neo4j ainsi que HyperGraphDB et InfoGrid.

Dans un graphe nous allons pouvoir stocker deux types d’informations : des nœuds et des liens. En anglais des nodes et des edges. Chaque nœud peut posséder plusieurs liens qui pointent sur d’autres nœuds. C’est grâce à cela que les relations peuvent se faire entre les nœuds. Elles vont ainsi nous permettre de les organiser. De plus chaque nœud peut avoir plusieurs propriétés ou attributs pour stoker sous forme de clé/valeur nos données.

**En bref** : Un graphe stocke des données dans des nœuds qui ont des propriétés.

Les relations organisent les nœuds entre eux. Nous allons pouvoir trier nos données sous forme de liste, d’arbre ou d’une façon plus libre en fonction de nos besoins. Elles peuvent avoir plusieurs directions, soit sortante, soit entrante ou les deux à la fois. De même que les nœuds, elles peuvent aussi avoir des propriétés pour nous faciliter l’organisation.

**En bref** : Les nœuds sont organisés par des relations qui ont-elles même des propriétés.

La base de données permet de gérer tous ces types d’objet, nœuds, relations et index. Et grâce à des algorithmes, des outils internes et des modules externes comme Apache Lucuene, Cypher, ou Gremlin la récupération de nos données est plus facile.

**Mise en pratique**

Tout d’abord il faut installer le SGBD Neo4j :

1. Téléchargement de l’installation depuis <http://neo4j.com/download/>
2. Suivre les étapes de l’installeur

Après il faut récupérer les librairies de Neo4j :

1. Téléchargement du fichier zip de Neo4j depuis <http://neo4j.com/download/other-releases/>
2. Ouverture du dossier lib et récupération des librairies à ajouter dans votre projet.

Là on peut créer notre projet.

Dans cet exemple, nous allons mettre au point un projet Java qui va créer des nœuds et les connecter avec des liens qui von nous permettre d’avoir un graphe comme celui sur l’exemple précité.

Avec Java, il y a 3 méthodes pour créer, stocker et récupérer des graphes sur la BDD Neo4j :

1. L’API Java Native : qui fera objet de notre exemple.
2. L’API Cypher : elle fonctionne de la même manière que JDBC mais nécessite une connaissance préalable du langage de requêtes CQL (Cypher Query Language).
3. Spring Data : la même qu’en relationnel, sauf qu’il y a des configurations à revoir.

Vous pouvez voir des exemples des méthodes 2 et 3 sur le site : <http://www.tutorialspoint.com/neo4j/>

**Exemple (API Java Native)**

On commence tout d’abord par définir les types de nœuds auxquels nous aurons affaire. Par exemple, on peut avoir des nœuds de type Auteur, Pays, etc…

Pour cela on crée un « enum » de type « Label » dans lequel nous allons stocker tous les types de nœuds qu’on veut.

**import** org.neo4j.graphdb.Label;

**public** **enum** NodeLabels **implements** Label {

***AUTEUR***, ***JOURNAL***, ***MOT\_CLE***, ***PAYS***;

}

Ensuite, nous ferons la même chose pour définir les types de relations qu’on veut.

**import** org.neo4j.graphdb.RelationshipType;

**public** **enum** TypeRelation **implements** RelationshipType {

***PUBLICATION***, ***REVIEWING***, ***INTERET***, ***APPARTENANCE***;

}

Maintenant on peut mettre notre programme au point.

Pour créer et démarrer la base de données Neo4j en mode embarqué, il suffit de déclarer un objet de type « GraphDatabaseService » et de l’instancier à partir d’un objet « GraphDatabaseFactory » en lui passant comme paramètre le chemin de la base :

GraphDatabaseFactory dbFactory = **new** GraphDatabaseFactory();

GraphDatabaseService db = dbFactory.newEmbeddedDatabase("C:\\Zakaria\\NeoTests\\IntroNeo4j");

Maintenant on peut commencer :

Pour créer un nœud et lui affecter des propriétés on fait :

Node bernard = db.createNode(NodeLabels.***AUTEUR***); // Création d'un nouveau noeud

bernard.setProperty("NomLong", "DOUSSET Bernard"); // Ajout d'une propriété à ce noeud, vous pouvez en ajouter autant que vous voulez, toujours en respectant le couple (nom de propriété, valeur)

bernard.setProperty("NomCourt", "DOUSSE1");

**Nota Bene** : On pourrait penser à ajouter un index à notre nœud mais il faut savoir qu’à partir de la version 2 de Neo4j, l’indexation de nœuds se fait automatiquement lors de son enregistrement au niveau de la BDD.

De la même manière, nous allons créer un autre nœud pour un Journal (VSST par exemple) :

Node vsst = db.createNode(NodeLabels.***JOURNAL***);

vsst.setProperty("NomJounal", "VSST");

vsst.setProperty("Edition", 2015);

vsst.setProperty("Editeur", "Elsevier");

Maintenant, nous allons créer une relation entre ces deux nœuds. Pour cela, une méthode « createRelationship » est disponible dans chaque objet de type Node. Elle prend en paramètre le nœud à lier et le type de relation. Elle peut aussi avoir des propriétés qui lui sont spécifiques. Voici un exemple qui lie Bernard avec le journal VSST :

Relationship reviewer = bernard.createRelationshipTo(vsst, TypeRelation.***REVIEWING***);

reviewer.setProperty("Nbre Avis Positifs", 16);

Puisque nous allons créer plusieurs nœuds et relations, nous devons les encapsuler dans une transaction. Car si une erreur se produit pendant une insertion il faut invalider les présentes. Voici toutes les insertions de l’exemple :

/\* Début des transactions (Dans notre exemple on va se concentrer sur l'écriture dans la BDDG) \*/

**try** (Transaction tx = db.beginTx()) {

Node bernard = db.createNode(NodeLabels.***AUTEUR***); // Création d'un nouveau noeud

bernard.setProperty("NomLong", "DOUSSET Bernard"); // Ajout d'une propriété à ce noeud, vous pouvez en ajouter autant que vous voulez

// Toujours en respectant le couple (nom de propriété, valeur)

bernard.setProperty("NomCourt", "DOUSSE1");

Node vsst = db.createNode(NodeLabels.***JOURNAL***);

vsst.setProperty("NomJounal", "VSST");

vsst.setProperty("Edition", 2015);

vsst.setProperty("Editeur", "Elsevier");

Node ie = db.createNode(NodeLabels.***MOT\_CLE***);

ie.setProperty("Mot-Cle", "Intelligence Economique");

Node fr = db.createNode(NodeLabels.***PAYS***);

fr.setProperty("NomPays", "France");

Relationship appartenir = bernard.createRelationshipTo(fr, TypeRelation.***APPARTENANCE***); // Création d'une nouvelle relation noeud\_source.createRelationshipTo(noeud\_destination, type\_relation);

appartenir.setProperty("Annee de Naissance", 1947); // Ajout d'une propriété à cette relation, vous pouvez en ajouter autant que vous voulez

Relationship reviewer = bernard.createRelationshipTo(vsst, TypeRelation.***REVIEWING***);

reviewer.setProperty("Nbre Avis Positifs", 16);

Relationship interet = bernard.createRelationshipTo(ie, TypeRelation.***INTERET***);

interet.setProperty("Nbre Articles Publies", 20);

tx.success();

}

Voici donc le code complet :

* **Fichier « NodeLabels.java » :**

**package** com.intro.neo4j.nodeLabels;

**import** org.neo4j.graphdb.Label;

**public** **enum** NodeLabels **implements** Label{

***AUTEUR***, ***JOURNAL***, ***MOT\_CLE***, ***PAYS***;

}

* **Fichier « TypeRelation.java » :**

**package** com.intro.neo4j.typeRelation;

**import** org.neo4j.graphdb.RelationshipType;

**public** **enum** TypeRelation **implements** RelationshipType {

***PUBLICATION***, ***REVIEWING***, ***INTERET***, ***APPARTENANCE***;

}

* **Fichier « Main.java » :**

**package** com.intro.neo4j;

**import** org.neo4j.graphdb.GraphDatabaseService;

**import** org.neo4j.graphdb.Node;

**import** org.neo4j.graphdb.Relationship;

**import** org.neo4j.graphdb.Transaction;

**import** org.neo4j.graphdb.factory.GraphDatabaseFactory;

**import** com.intro.neo4j.nodeLabels.NodeLabels;

**import** com.intro.neo4j.typeRelation.TypeRelation;

**public** **class** Main {

**public** **static** **void** main(String[] args) {

/\* Création d'une nouvelle BDD Orientée Graphe \*/

GraphDatabaseFactory dbFactory = **new** GraphDatabaseFactory();

GraphDatabaseService db = dbFactory.newEmbeddedDatabase("C:\\Zakaria\\NeoTests\\IntroNeo4j"); // Vous pouvez définir votre path vers un dossier vide

/\* Début des transactions (Dans notre exemple on va se concentrer sur l'écriture dans la BDDG) \*/

**try** (Transaction tx = db.beginTx()) {

Node bernard = db.createNode(NodeLabels.***AUTEUR***); // Création d'un nouveau noeud

bernard.setProperty("NomLong", "DOUSSET Bernard"); // Ajout d'une propriété à ce noeud, vous pouvez en ajouter autant que vous voulez

// Toujours en respectant le couple (nom de propriété, valeur)

bernard.setProperty("NomCourt", "DOUSSE1");

Node vsst = db.createNode(NodeLabels.***JOURNAL***);

vsst.setProperty("NomJounal", "VSST");

vsst.setProperty("Edition", 2015);

vsst.setProperty("Editeur", "Elsevier");

Node ie = db.createNode(NodeLabels.***MOT\_CLE***);

ie.setProperty("Mot-Cle", "Intelligence Economique");

Node fr = db.createNode(NodeLabels.***PAYS***);

fr.setProperty("NomPays", "France");

Relationship appartenir = bernard.createRelationshipTo(fr, TypeRelation.***APPARTENANCE***); // Création d'une nouvelle relation

// noeud\_source.createRelationshipTo(noeud\_destination, type\_relation);

appartenir.setProperty("Annee de Naissance", 1947); // Ajout d'une propriété à cette relation, vous pouvez en ajouter autant que vous voulez

Relationship reviewer = bernard.createRelationshipTo(vsst, TypeRelation.***REVIEWING***);

reviewer.setProperty("Nbre Avis Positifs", 16);

Relationship interet = bernard.createRelationshipTo(ie, TypeRelation.***INTERET***);

interet.setProperty("Nbre Articles Publies", 20);

tx.success();

}

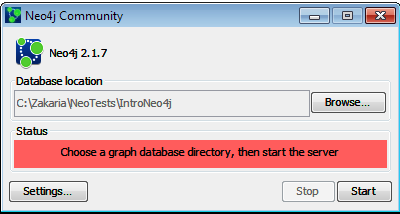
System.***out***.println("Done!");

}

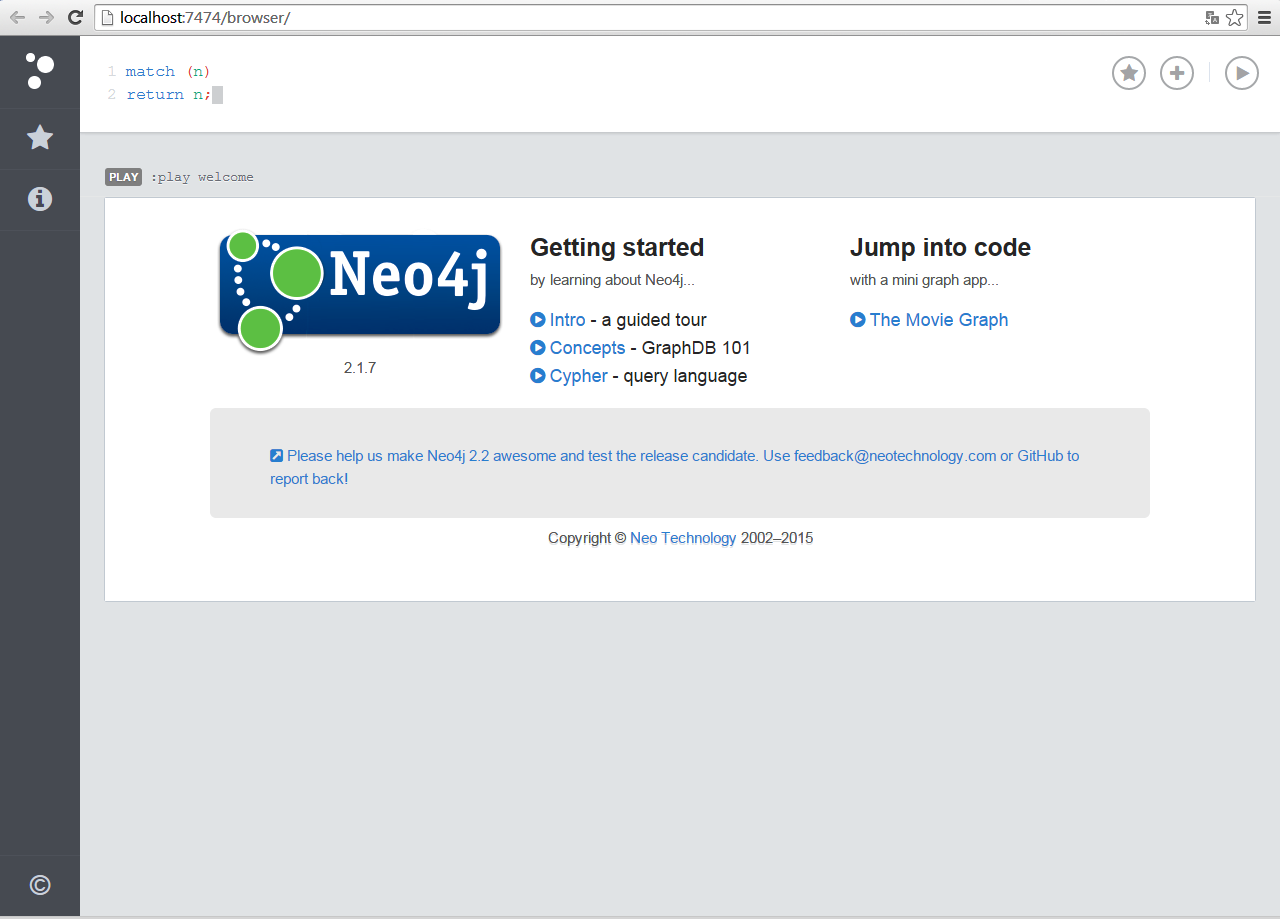
}

Maintenant on peut lancer le SGBD Neo4j :

1. Depuis le menu « Démarrer », choisir « Neo4j »
2. Spécifier l’endroit où se trouve la BDD avec le bouton « Browse »
3. Appuyer sur « Start »



On peut accéder au SGBD de Neo4j depuis l’adresse <http://localhost:7474>



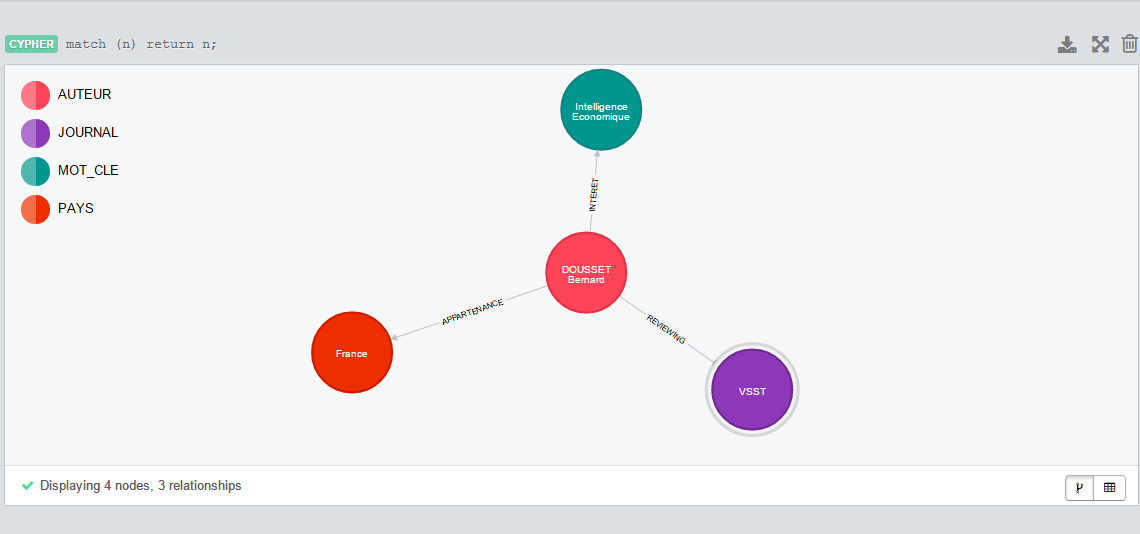
Sur la ligne de commande CQL, on tape :

**match (n)**

**return n;**

Puis on appuie sur « Ctrl-Entrée » pour exécuter la requête.

Là on retrouve les nœuds et leurs relations bien stockés.



**Conclusion**

La base de données Neo4j se démarque de ses concurrents grâce à sa simplicité d’utilisation, ses modules et sa communauté. Pour l’instant peu utilisée en entreprise, les bases graphe permettent de résoudre des problématiques plutôt spécifiques. Mais comparée à une base de données relationnelle, la représentation des données dans ces applications est bien plus simplifiée.