

**Document Préparé**

**Majed ABOU HAMDAN**

**ISAE CNAM – Liban**

**SMB214 – Mr.Pascal Fares**

**2016**

**Introduction:**

La taille moyenne des données augmente au cours des années.

Cette taille était très faible en 2007.

Mais au fil des années, on voit bien une augmentation de cette taille.

Actuellement elle est de l’ordre de 2500kb.

Par exemple, la taille d’une photo est passée de quelque kb à 2 Mégabit actuellement

**Tendance de Connectivité:**

Au début, La connexion entre les données était quasi inexistante.

Car il n’y avait pas de liens entre les documents textes.

Ensuite, l’évolution du Web qui est passe du web 01 au web 02 jusqu’au web 03 de nos jours avec XML, RDFa et les réseaux sociaux, montre que la connectivité des données est devenu plus importante avec le temps.

**Performance de RDBMS:**

Le problème de RDBMS est la perte de performance, lorsqu’on doit traiter un très grand volume de donner.

Tant que la taille des donnés augmentent, la performance diminue.

D’où l’utilité de concevoir et d’utiliser un outil permettant de gérer des données volumineuses sans pour autant nuire à la performance.

L’une des solutions, était d’utiliser NoSQL.

**NoSQL:**

NoSQL signifie Not Only SQL, littéralement “pas seulement SQL”

Ce terme date de 1998 et il est devenu populaire en 2009.

Il est capable de regrouper des BD très différentes.

Le NoSQL ne vient pas remplacer les BD relationnelles, mais proposer une alternative ou compléter les fonctionnalités des SGBDR (Oracle, PostgreSQL etc.)

POUR donner des solutions plus intéressantes dans certains contextes

Le NOSQL c’est du web 2.0, il est articulé autour de différentes technologies du web 2 , comme RSS, CSS ,ajax, soap, …..

Lorsque l’on parle de sites â fort trafic et de base de données on parle de NOSQL.

NoSQL existe à côté des bases de données relationnelles,

Il regroupe 4 grandes familles de représentations des bases de données:

1. Clé - Valeur
2. Colonne
3. Document
4. Base de données Graphe

Pour chaque représentation, on dispose de différents acteurs (implémentations)

Par exemple :

* Repr**ésentation en** Cl**é -**Valeur :
* Redis
* BerkeleyDB (et dans un sens Cassandra/RIAK)
* ecc…
* Repr**ésentation orientée** Document :
* CouchDB
* MongoDB
* ecc…
* Repr**ésentation orientée** colonne :
* Cassandra
* RIAK
* BigTable
* ecc…

Ce sont les principaux acteurs du genre qui sont basées sur le concept BigTable de Google.

Mais ces Bases ne sont pas transactionnelles car elles ne sont pas faites pour y stocker des données connectées. Pour cela, on a laRepr**ésentation orientée** graph qui est pour palier a des problèmes impossibles à résoudre avec des BDR.

On dispose d’une panoplie d’acteurs, dont le plus connu (Bigdata, Cloud Graph, Graph Base, HyperGraphDB, InfiniteGraph, Orient DB, ecc …), le plus utilisé est Neo4J.

**Quand utiliser une BD Graph.**

Nous avons les questions suivantes:

* Vos données sont-elles ***dynamiques?***
* Vos données sont- elles ***connectées?***
* Avez-vous besoin d’***un schéma flexible?***

Si nous répondons par « oui » au moins à 2 des questions,

Donc nous avons besoin d’une base oriente Graphe.

**Les plus courants Cas d'utilisation sont:**

* La recommandation.
* La détection de fraude.
* La Master Data Management.
* L’analyse d’impacts.
* La gestion des profils et des droits d’accès.

**A noter**:

Chaque type de data store NOSQL correspond à des contraintes métier différentes

**Pourquoi Neo4J ?**

Le Neo4j est une Base de données No SQL, a représentation orientée graphe, il est écrite en Java.

C’est un Projet(s) open-source (licence AGPL) et il est Supportée commercialement par Néo Technologie

Le Neo4J est utilisé par plusieurs entreprises, actives dans différents domaines Comme Adobe, Cisco, Mozilla et autres, qui font confiance à Neo4j

**Neo4j est basé sur des graphes**

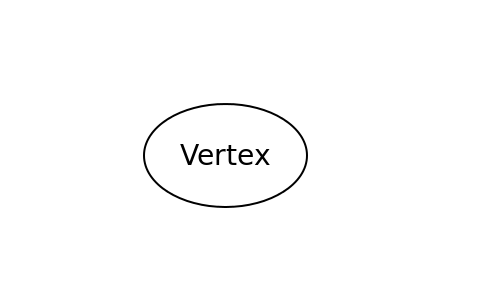
Les Bases de Données graphes manipulent des Données fortement interconnectées

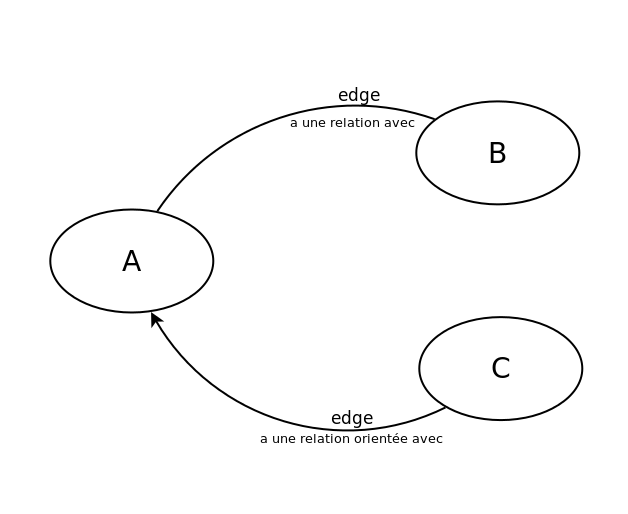
Les Graphes représentent bien un modèle Objet.

Dans les BDs graphes, les jointures ne sont pas coûteuses.

Le Modèle est rempli d'algorithmes efficaces et bien connus en Recherche opérationnelle.

**Quelques exemples de graphes**

1. C'est un graphe le plus simple. Il s’agit d’un seul nœud.
2. Un graphe un peu plus élaboré consiste de 3 nœuds A, B et C.



Dans ce Graphe il existe une relation orientée de C vers A, alors qu’une autre relation non orientée existe entre B et A

**A noter que :**

* Le graphe peut avoir un ou plusieurs nœuds.
* Les nœuds doivent être relies entre eux par une ou plusieurs relations.
* Chaque relation doit avoir une étiquette décrivant la relation.
* Chaque nœud peut disposer d’une ou de plusieurs attributs.

**Exemples d'un graphe:**

C'est un graph qui a plusieurs nœuds et chaque nœud a un attribut et les nœuds sont par un ou plusieurs relations qui ont une étiquette.

TRAVELS\_WITH

Name: the Doctor

Age: 37

Species: Time Lord

BORROWED

Year: 1963

LOVES

TRAVELS\_WITH

First name: Rose

Late name: Tyler

TRAVELS\_IN

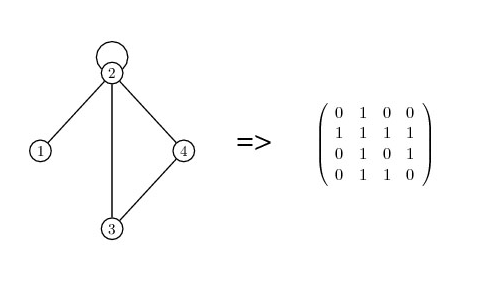
Vehicle: Tardis

Model: Type 40

Pour transformer cette forme en informatique on utilise la matrice Adjacence

**Quelques matrices d’adjacence**:

Ex1:



4

3

2

1

4

3

2

1

Ici on possède 4 nœuds, 1 2 3 4

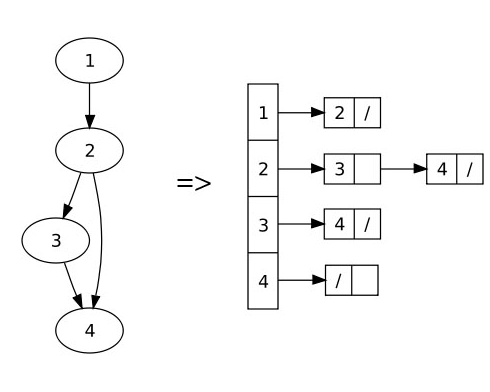
Le 1 n'a pas de relation avec elle-même, donc dans la matrice d’adjacence correspond à un zéro.

Mais ce 1 possède une relation avec le nœud 2, donc dans la matrice d’adjacence on met 1.

Et ainsi de suite ….

Ex2:

Une autre forme pour représenter les graphes, ce sont des listes de chaine:



Ici on a représenté le différent nœud …

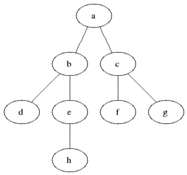
Et les flèches représentent les relations inter-nœuds

**Recherche sur le graphe**

On a 2 types de parcoures:

1. Parcoure en Largeur (**Breadth-first search**).
2. Parcoure en Profondeur (**Depth-first search**).
3. **Le parcours en largeur:**

Dans ce parcours, les nœuds seront vérifiés niveau par niveau.



Propriétés intéressantes:

Complexité: (n) où n étant le nombre de nœuds.

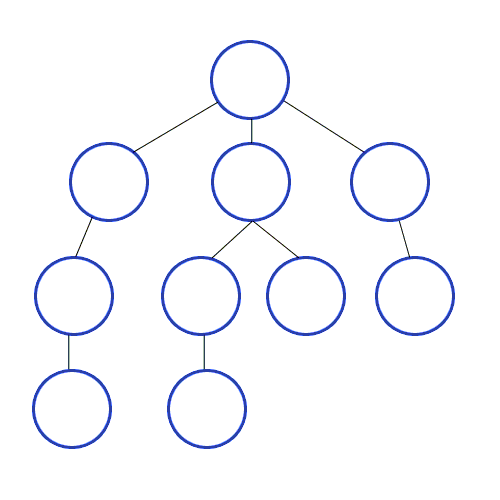
Niveau 1

Niveau 2

Niveau 3

1. **Le parcours en profondeur:**

Dans ce parcours, les nœuds seront vérifiés branche par branche



Propriétés intéressantes:

Complexité: (n) où n étant le nombre de nœuds.

Branche 1

Branche 2

Branche 3

Branche 4

Dans les 2 types de parcours notre but est de répondre à une requête formulée par un utilisateur.

Pour cela, la première étape est de parcourir le graphe pour trouver le ou les nœuds qui correspondent aux propriétés demandes.

On doit parcourir tous les nœuds pour répondre à une requête, d’où la complexité (n), "n" étant le nombre de nœuds.

**Programmation du graphe:**

Pour programmer le graph, on utilise le langage java:

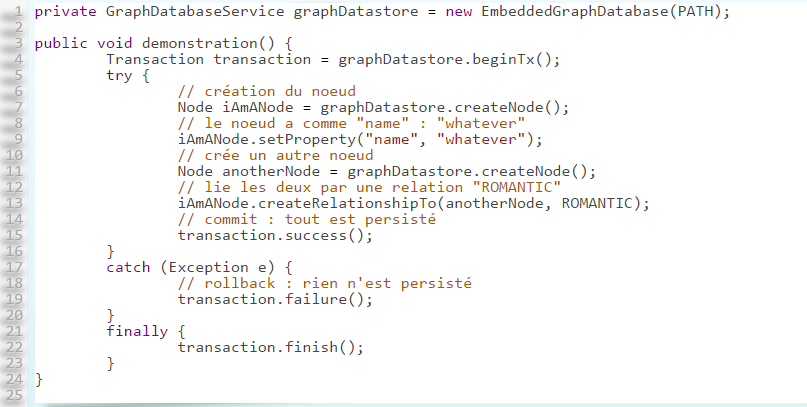
1. Création d'un nœud (vertex, node) qui est la forme la plus simple d'un graphe.
   * *Node iAmANode = graphDatastore.createNode();*

(**Req:** *iAmANodeest le nom du* nœud*)*

1. Création d'une relation (relationship, edge) qui permet d'organiser les nœuds.
   * *iAmANode.createRelationshipTo(anotherNode, ExampleRelationships.ROMANTIC);*
2. Création d'un propriété (property, key/value pairs) qui enrichit les noeuds et les relations.
   * *iAmANode.setProperty("having", "a property!");*

**Exemple 1 (Programmation de 2 nœuds):**

Le premier ayant une propriété. Le 2eme sans proprietes et Les 2 sont liés par une relation romantique



**Exemple 2 (Programmation de traversée de graphe):**

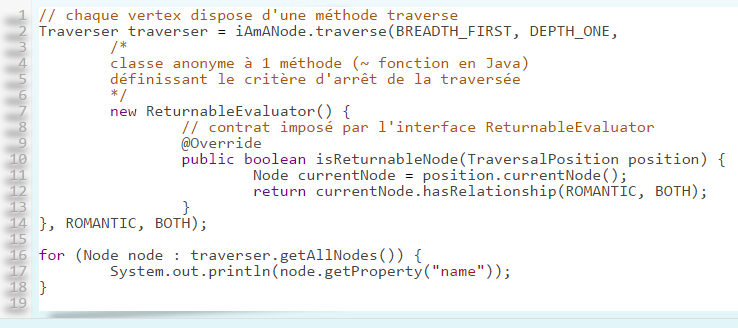
Dans cet exemple la requête est de chercher les nœuds reliés au nœud initial par la relation ROMANTIC, et d'imprimer à la fin la propriété “Nom” des nœuds correspondant à la requête.

En java cette requête se traduit par:

Pour choisir le parcours à suivre : on utilise la commande …#traverse # …

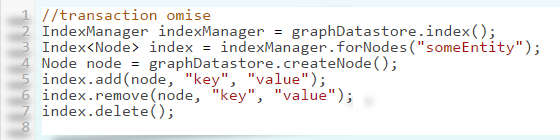
* Si le parcours choisi est en largeur, ce sera: traverse « BREADTH\_FIRST »
* Si le parcours en profondeur ce sera: Traverse « DEPTH\_FIRST “

Ensuite “pour réponde aux requêtes” qui sont les conditions à vérifier, c.à.d. la relation « ROMANTIC » dans notre exemple, on utilise la fonction ReturnableEvaluator.



**Exemple 3 (index):**

Pour accélérer la recherche des nœuds, il est souhaitable d’indexer les nœuds en utilisant les fonctions prédéfinies en Neo4j

****

**Requête**:

Pour effectuer une requête, on peut utiliser des outils plus élaborés comme

1. Gremlin :

C'est un langage de traversée de graphe.

Il supporte Neo4J, OrientDB, DEX...

1. Cypher :

C'est un langage déclaratif permettent de requêter et mettre à jour le graph.

Il inspire du SQL, on y retrouve beaucoup de concepts familiers, comme les clauses WHERE, ORDER BY, SKYP, LIMIT ecc…

Son objectif est de permettre a l'utilisateur de définir des motifs, qui seront par la suite recherches dans tout le graphe.

**A Noter:**

Les requêtes plus complexes peuvent être effectuées avec le module Cypher qui propose plus d’options

**Mise en pratique**

1. **Typée les relations:**

En Neo4j, chaque relation doit être typée.

Ex:

Création d’une énumération qui représentera tous les types de notre graphe.



1. **Création d’une classe:**

Pour créer et démarrer la base de données Neo4j en mode embarqué, il suffit de:

- Déclarer un objet de type GraphDatabaseService.

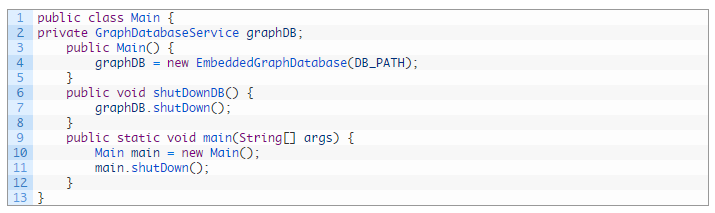
- instancier avec la classe EmbeddedGraphDatabase en passant

comme paramètre au constructeur le chemin de la base.

- La méthode shutdown de notre base de données permet de

l’éteindre correctement.

**Déclarer un objet de type**



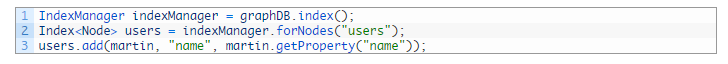
**Permet de l’éteindre correctement**

**Instancier avec la classe**

1. **Création des nœuds:**

  
Il suffit de déclarer pour chaque nœud un nouvel objet de type Node créé à partir de notre objet graphe :

Pour lui affecter une propriété, comme le nom par exemple, il faut appeler la méthode setProperty qui prend en paramètres une clé et une valeur.  
  
Pour indexer le nœud « Martin » afin de le récupérer facilementPour Pour indexer le nœud « Martin » afin de le récupérer facilement



1. **Création d’une relation:**

La relation prend en paramètre le nœud à lier et un type.

Pour créer une relation entre deux nœud une méthode createRelationship est disponible dans chaque objet de type Node.

Elle prend en paramètre le nœud à lier et un type.

Voici un exemple qui lie comme ami Martin avec Romain



1. **Création de plusieurs nœuds:**

Pour créer plusieurs nœuds et plusieurs relations.

Nous devons les encapsuler dans une transaction.

Car si une erreur se produit pendant une insertion il faut invalider les présentes.



**Encapsuler plusieurs noeud et relations**

**Création de plusieurs nœuds**

1. **Création de la traversée:**

Nous allons maintenant créer la traversée qui permettra de connaitre les amis de Martin qui ont au moins deux amis en commun avec lui.

**Récupérer le nœud grâce à l’index**

1. Mais avant il faut récupérer le nœud grâce à l’index.



1. Ensuite la traversée partira de Martin et pour chaque nœud traversé nous allons récupérer ceux qui ont au moins deux relations sortantes de type Friend.

Pour effectuer cette opération il faut créer un objet de type Traverser récupéré par le nœud de départ qui est Martin :

Les paramètres de la méthode traverse sont :

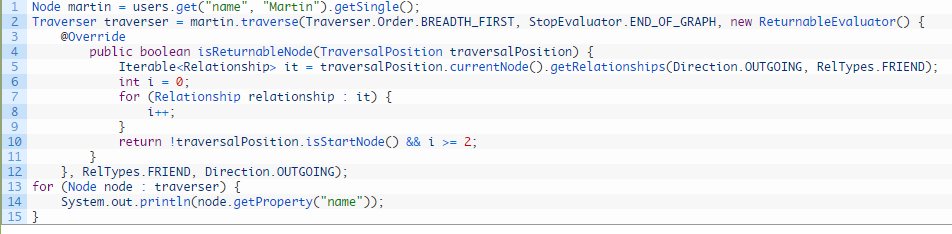
1. Traverser.Order : Ce paramètre peut prendre deux valeurs :
   * Order. BREADTH\_FIRST qui force en premier la traversée de chaque relation du nœud courant
   * Order.DEPTH\_FIRST qui force d’abords la traversée des nœuds enfant du nœud courant.
2. StopEvaluator : Ce paramètre définit la portée de la traversée dans le graphe.

(Ici dans l’exemple nous traversons tout le graphe.)

1. ReturnableEvaluator : Ce paramètre prend un objet de type ReturnableEvaluator qui contient une méthode qui définit si le nœud traversé doit être récupéré.

Dans l’exemple les nœuds validés doivent avoir au moins deux relations sortantes et on exclut le nœud de départ.

1. RelationshipType : Ce paramètre définit le type de relation à parcourir.
2. Direction : Ce paramètre définit la direction des relations à parcourir.



But est de connaitre les amis de Martin qui ont au moins deux amis en commun avec lui.

Le resulta: 

**Alors:**

Vous pouvez remarquer qu’il est très simple de faire des requêtes dans ce graph grâce à Neo4j.

Les requêtes plus complexes peuvent être effectuées avec le module Cypher qui propose plus d’options.

**On conclusion:**

Neo4j est:

* Un graph qui «stocke » des données dans des nœuds qui ont des propriétés.
* Les nœuds sont organisés par des relations qui ont-elles même des propriétés.
* Une traversée navigue dans le graphe à partir d’un nœud et identifie les chemins ou sous-chemins avec les nœuds ordonnés en fonction d’options.