# NEO4J

## Projet en architecture des logiciels - C1


Touma Chams

NEO4 J

ISAE CNAM.

Projet préparé sous la direction de Dr. Pascal Fares

+961 1 954342

# Contents

## 1- Introduction

La base de données dans l'architecture d'un logiciel ne se limite pas à stockée des informations, mais elle a également un impact sur les performances globales du logiciel. Alors, pour sélectionner une technologie de base de données qui convient à notre projet c'est crucial.

De nombreuses applications reposent sur une base de données relationnelle telle que MySQL ou PostgreSQL. Malgré les nombreux avantages des bases de données relationnelles, elles ne sont pas efficaces pour faire face à des quantités toujours croissantes de données connectées.

Dans ce document nous proposerons la base NEO4J, une base de données non relationnelle graphique optimisée pour gérer les relations. Cette base peut nous aider à créer des applications à haute performances. Elle est en même temps évolutive capable d'utiliser de gros volumes de données connectées.

## 2- Pourquoi la base de données graphique

Les bases de données graphiques sont basées sur la théorie des graphes mathématiques. Les graphes sont des structures qui contiennent des sommets (qui représentent des entités, telles que des personnes ou des objets) et des arêtes (qui représentent des connexions entre les sommets). Les arêtes peuvent avoir des valeurs numériques appelées *Weight*.

Cette structure permet aux développeurs de modéliser tout scénario défini par des relations.

Un modèle graphique est intuitif et facile à interpréter pour les gens. Après tout, le cerveau humain ne pense pas en termes de tableaux et de lignes

mais en termes d'objets abstraits et de connexions. En fait, tout ce que vous pouvez dessiner sur un tableau peut être affiché avec un graphique.

## 3- Installation NEO4J

La manière la plus simple de mettre en place un environnement pour développer une application avec Neo4j et Cypher est d'utiliser Neo4j Desktop.

Neo4j est une application gratuite donc il suffit d'avoir un réseau internet pour le téléchargement. Il est important de mentionné que l'application est compatible avec les différents types de systèmes d'exploitation.

Ci-dessous le lien pour le téléchargement,

https://neo4j.com/download/

## 4- Concept

Neo4j fournit sa propre implémentation des concepts de la théorie des graphes. Ci-dessous en détails les éléments qui composent la base NEO4J :

- **Nœuds – Nodes** : (équivalents aux sommets en théorie des graphes).

  Ce sont les principaux éléments de données qui sont interconnectés par des relations. Un nœud peut avoir une ou plusieurs étiquettes (qui décrivent son rôle) et des propriétés (c'est-à-dire des attributs).

- **Étiquettes – Labels :**

  Elles sont utilisées pour regrouper les nœuds et chaque nœud peut attribuer plusieurs étiquettes. Les étiquettes sont indexées pour accélérer la recherche de nœuds dans un graphe.

- **Relations** : (équivalentes aux arêtes en théorie des graphes).

  Une relation relie deux nœuds qui, à leur tour, peuvent avoir plusieurs relations. Les relations peuvent avoir une ou plusieurs propriétés.

- **Propriétés – Propreties :**

  Ce sont des attributs des nœuds et des relations. Neo4j permet de stocker des données sous forme de paires clé-valeur, ce qui signifie que les propriétés peuvent avoir n'importe quelle valeur (chaîne, nombre ou booléen).

## 5- Cypher vs SQL

- **Pourquoi pas SQL ?**
  - SQL est inefficace pour exprimer les requêtes de modèle graphique

En particulier pour les requêtes ci-dessous :

  - Récursifs.
  - Qui peuvent accepter des chemins de plusieurs longueurs différentes.
  - Les modèles de graphes sont plus intuitifs et déclaratifs que les jointures.
  - SQL ne peut pas gérer les valeurs de chemin.

## 6- SGBDR et GRAPHS

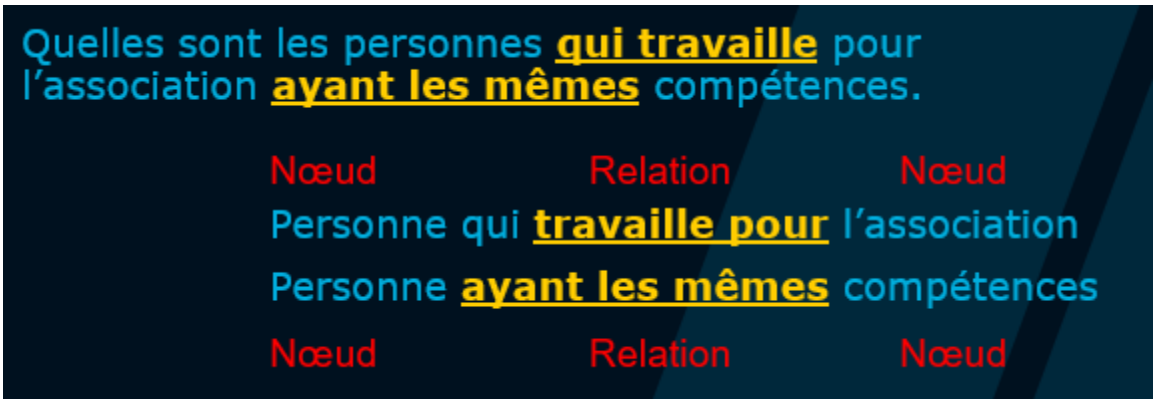
| SGBDR | Base de données graphique |
|---|---|
| les tables | Des graphiques |
| Des rangées | Les nœuds |
| Colonnes et données | Propriétés et ses valeurs |

| SGBDR | Base de données graphique |
|---|---|
| Contraintes | Des relations |
| Joint | Traversal |

- Impossible de modéliser ou de stocker les données et les relations sans complexité.
- Les performances se dégradent avec le nombre et les niveaux de relations, de même avec la taille de la base de données.
- La complexité des requêtes augmente avec le besoin des jointures.
- L'ajout de nouveaux types de données et de relations nécessite la reconsidération du schéma ce qui implique une augmentation du temps de développement.

## 7- Translation en langage Cypher

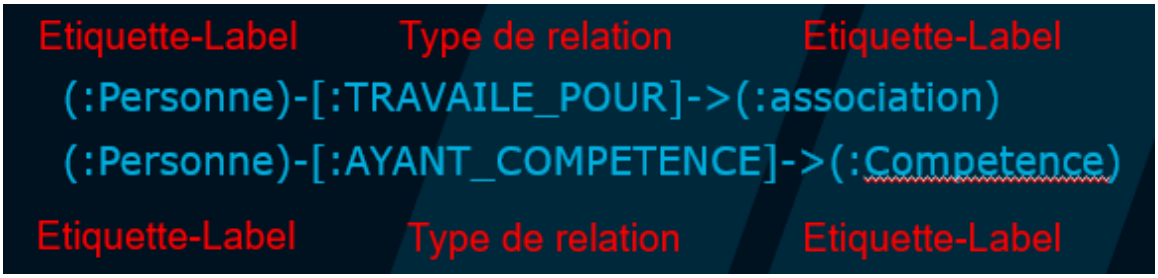
Cypher est un langage très proche à notre méthode de compréhension voici un exemple pour comprendre cette logique :



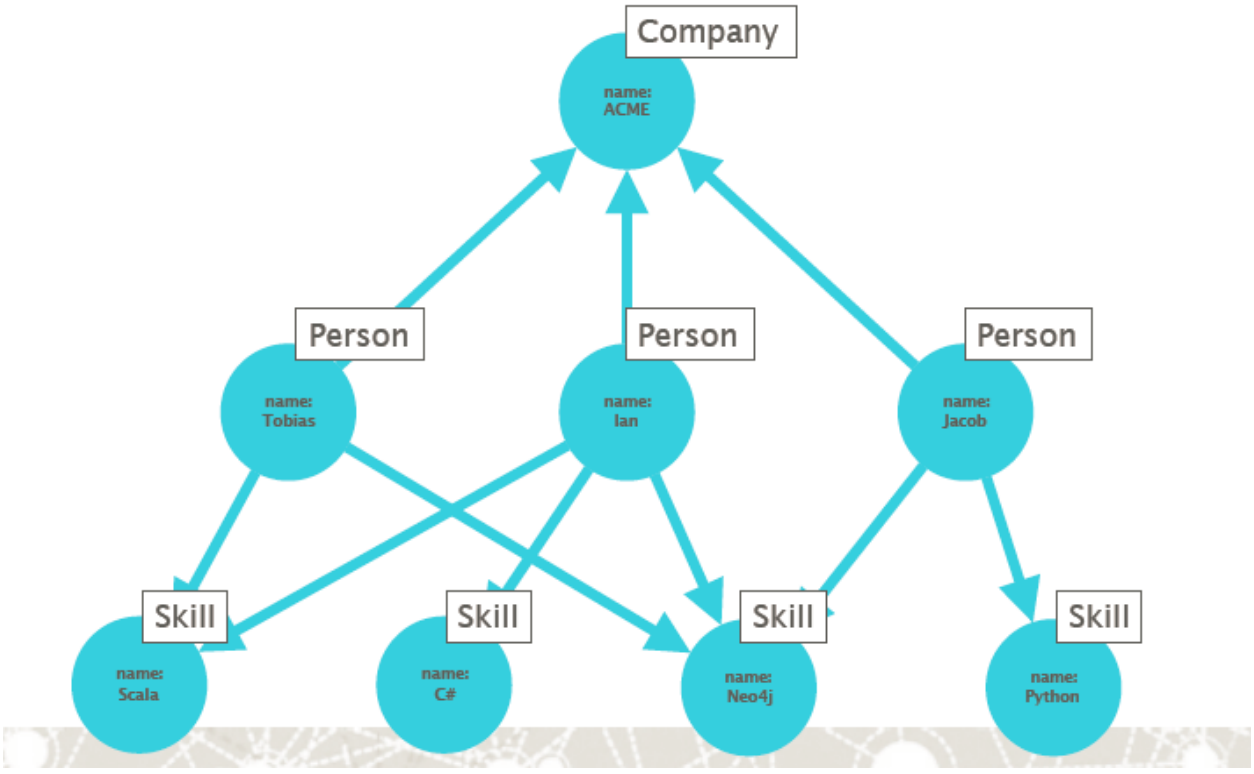

Personne et association sont des nœuds reliee avec la relation « travaille pour » comme ci-dessus.

Personne et compétence sont reliées entre eux en utilisant la relation « ayant les mêmes ».

En langage cypher c'est ecrit comme suivant :




L'architecture du concept ci-dessus peut être traduite comme suivant :




Personne, compagnie et compétences (skill) sont reliées entre eux où les flèches peuvent être les relations.

## 8- Langage Cypher

Neo4j fournit sa propre implémentation des concepts de la théorie des graphes.

Un langage de requête pour les graphiques. Il est :

- Déclaratif.
- Expressif.
- Conçu pour la correspondance de motifs.

- **Les nœuds (NODES) :** ()
- **Les relations (RELATIONSHIP) :** -->
  - Pour les détails additionnels : -[:DIRECTED]->
- **Les modèles (PATTERNS) :** Les modèles sont dessinés en connectant des nœuds et des relations avec des tirets, optionnellement spécifiant une direction avec les signes > et <.

  ()-[]-()

  ()-[]->()

  ()<-[]-()

- **Les composants d'une requête Cypher :**
  **MATCH (m:Movie)**
  **RETURN m**
  MATCH et RETURN sont des mots clés Cypher
  m est une Variable et Movie est une étiquette nœud

  **MATCH (p:Person)-[r:ACTED_IN]->(m:Movie)**
  **RETURN p, r, m**
  p,r,m sont des variables
  :ACTED_IN est un type de relation

  **MATCH path = (:Person)-[:ACTED_IN]->(:Movie)**

**RETURN path**
path est une Variable

**MATCH (m:Movie)**
**RETURN m.title, m.released**

Dans cet exemple nous avons un résultat tabulaire, l'accès aux propriétés est faite de la manière suivante : **{variable}.{property_key}**

**Ci-dessous un tableau pour définir la manière d'écriture des composants :**

**Case sensitivity**

| Case sensitive | Case insensitive |
|---|---|
| Node labels | Cypher keywords |
| Relationship types | |
| Property keys | |

## 9- Ecrire des requêtes :

- **Create :**

  La clause CREATE est utilisée pour créer des nœuds et des relations. Dans la clause CREATE, les modèles sont largement utilisés.

  **Exemple 1 : Création d'un nœud**

  CREATE (m:Movie {title:'Mystic River', released:2003})

```
RETURN m
```

**Exemple 2 : Création d'une relation entre deux nœuds**

```
MATCH (m:Movie {title: 'Mystic River'})
MATCH (p:Person {name: 'Kevin Bacon'})
CREATE (p)-[r:ACTED_IN {roles: ['Sean']}]->(m)
RETURN p, r, m
```

- **Créer un chemin complet**

```
CREATE p = (andy {name:'Andy'}) - [: WORKS_AT]->(neo)<-
[:WORKS_AT]-(michael { name: 'Michael' })
RETURN p
```

*Cette requête crée trois nœuds et deux relations en une seule fois, l'attribue à une variable chemin (PATH) et la retourne.*

- **Set**

On peut ajouter de nouvelles propriétés à un nœud ou une relation existant, et également ajouter ou mettre à jour les valeurs de propriétés existantes.

```
MATCH (m:Movie {title: 'Mystic River'})
SET m.tagline = 'We bury our sins here, Dave.'
RETURN m
```

## 10- Avantages

Conçue spécifiquement pour traiter d'énormes quantités de données connectées, la base de données Neo4j offre les avantages suivants :

- **Performance**

  Dans les bases de données relationnelles, les performances souffrent à mesure que le nombre et la profondeur des relations augmentent. Dans les bases de données graphiques comme Neo4j, les performances restent élevées même si la quantité de données augmente considérablement.

- **Souplesse**

  Neo4j est flexible, car la structure et le schéma d'un modèle de graphe peuvent être facilement ajustés aux changements d'une application. En outre, nous pouvons facilement mettre à niveau la structure de données sans endommager les fonctionnalités existantes.

- **Agilité**

  La structure d'une base de données Neo4j est facile à mettre à niveau, de sorte que le magasin de données peut évoluer avec votre application.

## 11- Backup

On doit utiliser l'option terminal puis la commande suivante :

*neo4j-admin backup --backup-dir=<path>*

Voici un exemple :

```
C:\Users\User\.Neo4jDesktop\neo4jDatabases\database-a25abae8-bed8-4423-995a-c18ef3a
37526\installation-4.0.1>C:\Users\User\.Neo4jDesktop\neo4jDatabases\database-a25aba
e8-bed8-4423-995a-c18ef3a37526\installation-4.0.1\bin\neo4j-admin backup --backup-d
ir=C:\Users\User\Desktop\Projet_C1
```

## 12- Conclusion

N'importe quelle est la technologie, la question est toujours la performance.

- Un exemple de graphique social avec ~ 1000 personnes
- En moyenne 50 amis par personne
- PathExists(a, b) limité à la profondeur 4
- Elimination des E/S du disque dure.

| | # Personnes | Temps(ms) |
|---|---|---|
| SGBD Relationnelle | 1,000 | 2000 |
| Neo4j | 1,000 | 2 |
| Neo4j | 1,000,000 | 2 |

## 13- Références


https://rubygarage.org/blog/neo4j-database-guide-with-use-cases#:~:text=To%20handle%20a%20growing%20volume,large%20volumes%20of%20connected%20data.

https://neo4j.com/docs/pdf/neo4j-getting-started-4.1.pdf

https://neo4j.com/docs/pdf/cypher-refcard-4.0.pdf