NEO4J

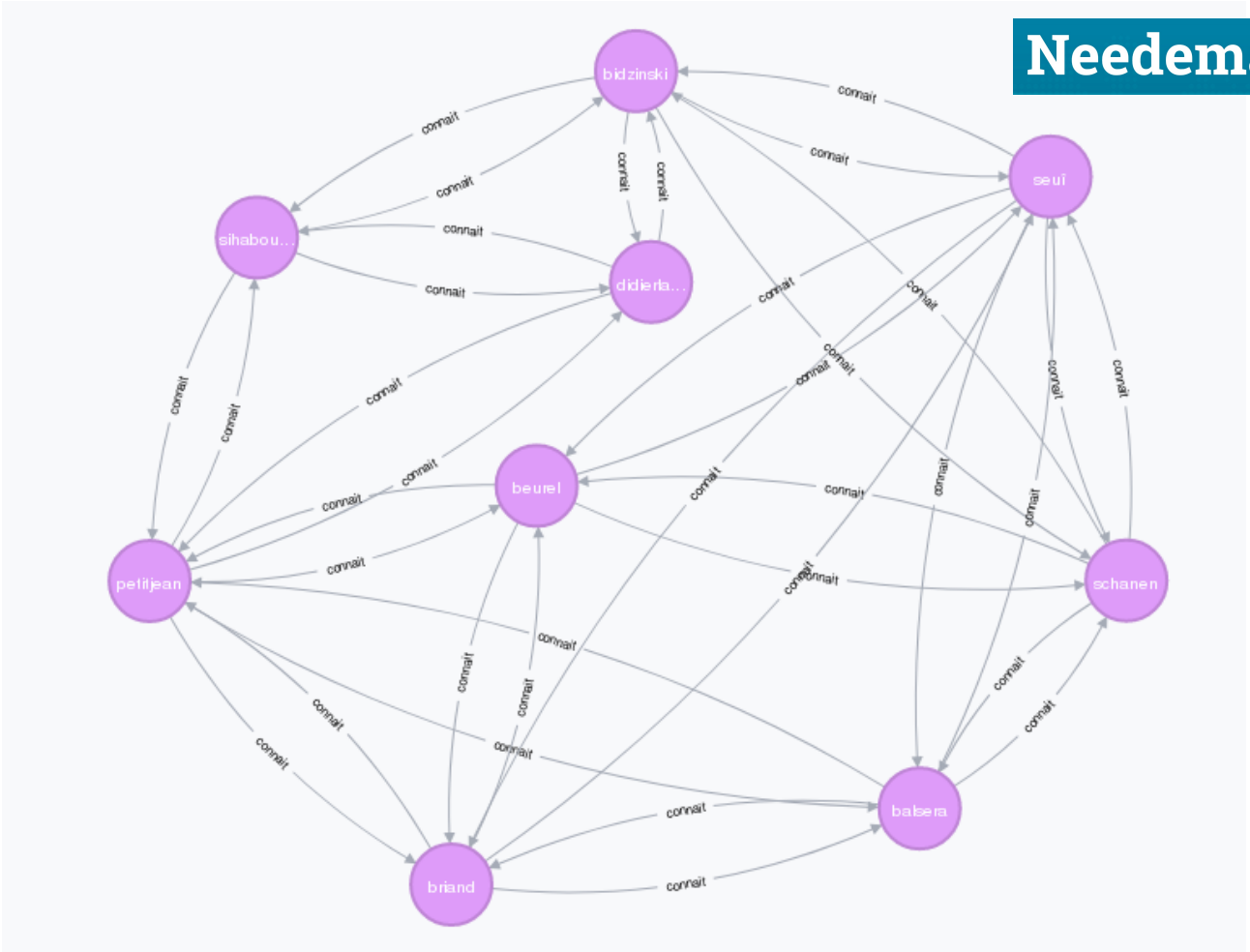
Bases de données orientées graphe

Fondées sur la théorie des graphes, elles sont conçues pour modéliser des structures de données relationnelles aussi diverses qu'un réseau social, un réseau ferroviaire, l'organisation d'une entreprise ou les interactions sociales.
Les bases de données relationnelles sont tout à fait adaptées pour gérer ce type de structures mais elles atteignent leurs limites sur le champ de la performance quand la profondeur du graphe et le volume de données à gérer sont importants.
Sur ce point, les bases de données orientées graphe surclassent les SGBDRs.

Dans le monde des bases orientées graphe, tout est affaire de noeuds, de relations et de propriétés.
Les propriétés portent aussi bien sur les noeuds que les relations.

Ce modèle est très adapté pour gérer des données sociales ou spatiales, mais également financières où les relations modémathieuraient les dépôts ou retraits.

Considérons l'exemple ci-dessous :



Il représente une réalité sociale suivante :

Pounnou (mathieu) est membre du groupe DataMining et connaît Desmae (benjamin) qui lui est membre du groupe NoSQL

Voyons maintenant comment implémenter et exploiter ce graphe avec Neo4J.

Neo4J est sans conteste l'implémentation (développée en Java) la plus connue. La seule édition gratuite (Community Edition) n'offre ni outils de monitoring, ni haute disponibilité.

Mise en œuvre

Télécharger la dernière version de la Community Edition et décompresser le fichier dans un dossier dédié.
Se positionner dans le répertoire REP_INST_NEO4J /bin puis lancer le serveur en exécutant la commande :

./neo4j start (sous linux)
ou
double clic (sous windows)

Par défaut, une console d'administration web est accessible sur

http://localhost:7474

Neo4J fournit des bindings Java et Python mais l'utilisation de son shell intégré s'avère très pratique pour l'explorer.

Créer les noeuds

Les commandes suivantes permettent de créer les noeuds du réseau social ainsi que leurs propriétés simultanément :

```
create (n:Personne {nom:'alili',prenom:'samir'})
, (n1:Personne {nom:'adomayakpo',prenom:'serge'})
,(n2:Personne {nom:'dochez',prenom:'florent'})
,(n3:Personne {nom:'Desmae',prenom:'benjamin'})
, (n4:Personne {nom:'mogenet',prenom:'florent'})
,(n5:Personne {nom:'theron',prenom:'anthony'})
,(n6:Personne {nom:'yborra',prenom:'amandine'})
,(n7:Personne {nom:'domingues',prenom:'chris'})
,(n8:Personne {nom:'Pounnou',prenom:'mathieu'})
```


L'identifiant de chaque noeud (id) est attribué de manière séquentielle, soit respectivement 1 , 2 , 3 et 4 .

Créer les relations entre les nœuds

Pour créer la relation sortante « connait » entre domingues et mogenet , on crée la relation via la commande :

match (n:Personne {nom:'domingues'}),(m:Personne {nom:'mogenet'}) create (n)-[r:connait]->(m) return n,r,m

Pour créer la relation entrante « connait » entre domingues et mogenet , on crée la relation via la commande :

match (n:Personne {nom:'domingues'}),(m:Personne {nom:'mogenet'}) create (m)-[r:connait]->(n) return n,r,m

On peut la résumer avec la commande

match (n:Personne {nom:'domingues'}),(m:Personne {nom:'mogenet'}) create (n)-[r:connait]->(m), (m)-[s:connait]->(n) return n,r,m,s

Et on continue de proche en proche pour créer les autres relations.

Ici vous avez toutes les relations (même celle plus haut) :

le fichier

La commande suivante permet de connaître les propriétés des noeud par exemple :

match (n:Personne) return n.nom, n.prenom

affichera ceci pour les noeud :

n.nom	n.prenom
alili	samir
adomayakpo	serge
dochez	florent
Desmae	benjamin
mogenet	florent

theron	anthony
yborra	amandine
domingues	chris
Pounnou	mathieu

Exploiter le graphe

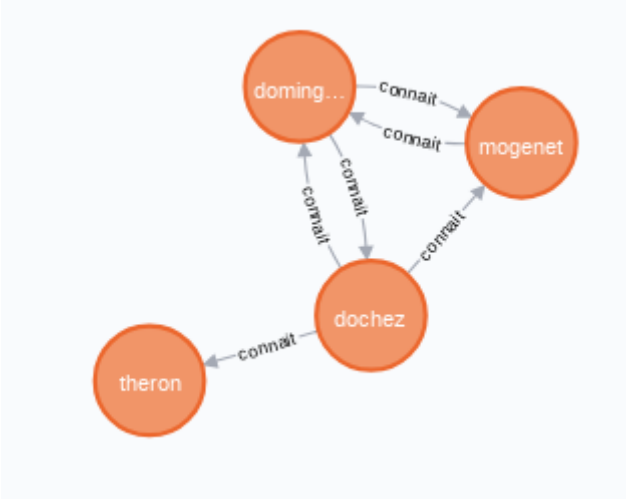
Cypher est un langage de requêtage déclaratif pour Neo4J.

Voyons comment exploiter le graphe à travers une série de cas concrets. .

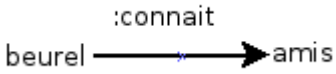
adomayakpo,qui connaît il?

Match (n:Personne {nom:'adomayakpo'})-[r:connait]->(m) return m

Ce qui donne :



L'identifiant personne fait référence au label du nœud.
Notez bien la syntaxe de l'expression match : noeud-[:connait]->amis . On devine les noeuds séparés par une relation orientée. C'est la représentation stylisée de la relation :



Quelles sont les personnes qui connaissent adomayakpo ?

Match (n:Personne {nom:'adomayakpo'})<-[r:connait]-(m) return m

Dans notre cas le résultat est le même que plus haut. Le sens de la relation vous fournira des informations différentes.
Alors quelles sont les relations de mister 'adomayakpo', toutes :

ceux qu'il connaît et ceux qui le connaissent :

Match (n:Personne {nom:'adomayakpo'})-[r:connait]-(m) return m

Enlevons les doublons :

Match (n:Personne {nom:'adomayakpo'})-[r:connait]-(m) return distinct m

Créons des nœuds qui auront le label 'Groupe' et affectons les :

voir le fichier plus haut

Qui est membre du groupe NoSQL ?

match (n:Personne)-[:membre]->(m:Groupe {titre:'NoSql'}) return n

Notez bien la syntaxe de l'expression match : -[:membre]-> . On devine les noeuds séparés par une relation orientée.

Quelles sont les connaissances des connaissances de alili (relationnel de niveau 2) ?

match (n:Personne {nom:'alili'})-[:connait]->(m)-[:connait]->(f) return f

Si nous avons un soucis : les noms apparaissent autant de fois que les nœuds ont de relations.
Pour traiter les multiples réponses identiques :

match (n:Personne {nom:'alili'})-[:connait]->(m)-[:connait]->(f) return distinct f

Si vous ne connaissez que le début du nom

match (n:Personne) where n.nom starts with "didier" return n

Quand on conjugue la connaissance approximative du nom (on ne connaît que le début) et des nœuds de niveau 2

match (n:Personne) where n.nom starts with 'bals' optional match (n)-[:connait]->(m)-[:connait]->(f) return distinct f

Vous pouvez utiliser « ends » pour la fin d'un mot

Cette recherche par niveau peut permettre de voir le chemin par ou passer l'information.
Pounnou a du chocolat (et en plus c'est vrai) il le dit à domingues.
Deux minutes plus tard theron lui en demande.

Comment savoir qui à vendu la mèche, à qui a t il parlé ?

```
MATCH (n:Personne {nom:'Pounnou'})-[:connait]->(m:Personne {nom:'domingues'})-[]->(g)-[]->(v:Personne {nom:'theron'}) RETURN (g)
```

Quoi de mieux que les concepteurs de Neo4j pour vous expliquer au mieux leur produit. Allez sur cette page :
https://neo4j.com/graphacademy/online-training/introduction-graph-databases/

Inscrivez vous sur le petit formulaire à droite.
Sujet des formations :

- What is a Graph Database
- What is Neo4j
- Neo4j and Cypher
- Querying a Graph Database with Cypher

Comme dirait mathieu : Mais pourquoi passer par le site de l'éditeur ?

Tout simplement parce que le support est parfait et que les exercices sont interactifs avec la possibilité de faire des requêtes en live (sans installation).