NoSQL et bases de données orientées graphe

I.Mougenot

FDS Sciences UM

Septembre 2015

Plan du cours

1. Principes généraux
   - Positionnement contextuel (taille vs complexité/expressivité)
   - Accointances avec les systèmes à objets et navigationnels
   - Adossement à la théorie des graphes
   - modèle de données : graphe attribué, marqué, orienté et multivalué
2. Un système en particulier : Neo4J

Volume de données versus richesse du modèle

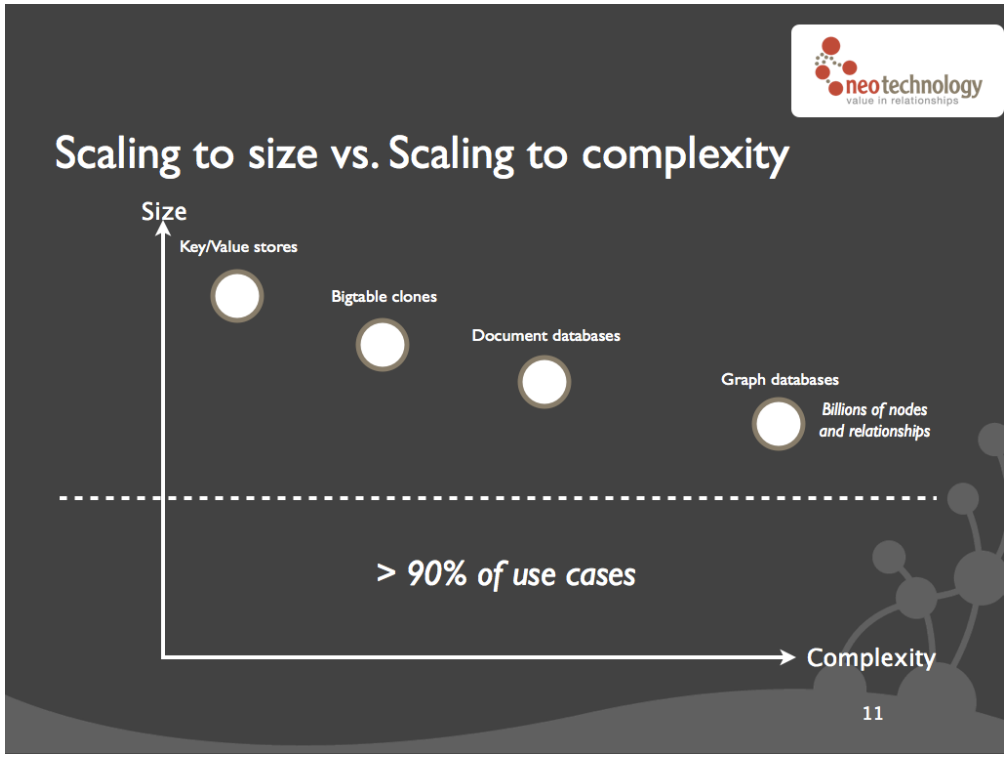


Figure: Une vision très générale (extrait de Neo Technology Webinar)

Adéquation avec le système "mental" humain

Associer et catégoriser : des mécanismes cognitifs[a] naturels

[a]processus psychiques liés à l'esprit

- catégories et associations se construisent de manière intuitive et naturelle pour l'esprit humain et se représentent à l'aide de graphes, structures que l'on sait traiter efficacement

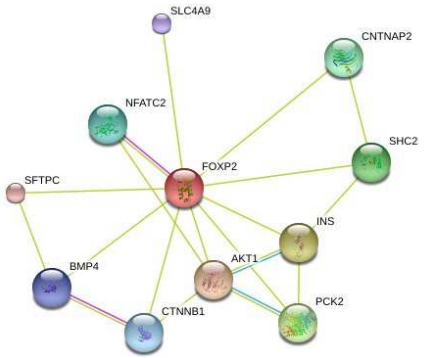


Figure: Gène FoxP2 : implication dans l'acquisition du langage (extrait de http://acces.ens-lyon.fr/evolution)

Le modèle de persistance le plus adapté : une longue histoire

BD : partage et pérennisation de l'information pour le compte de différentes applications - différents paradigmes[a] de représentations

[a] manières de voir les choses

1960 - système hiérarchique

1960 - système réseau (C. Bachman)

1970 - système relationnel (E.F. Codd)

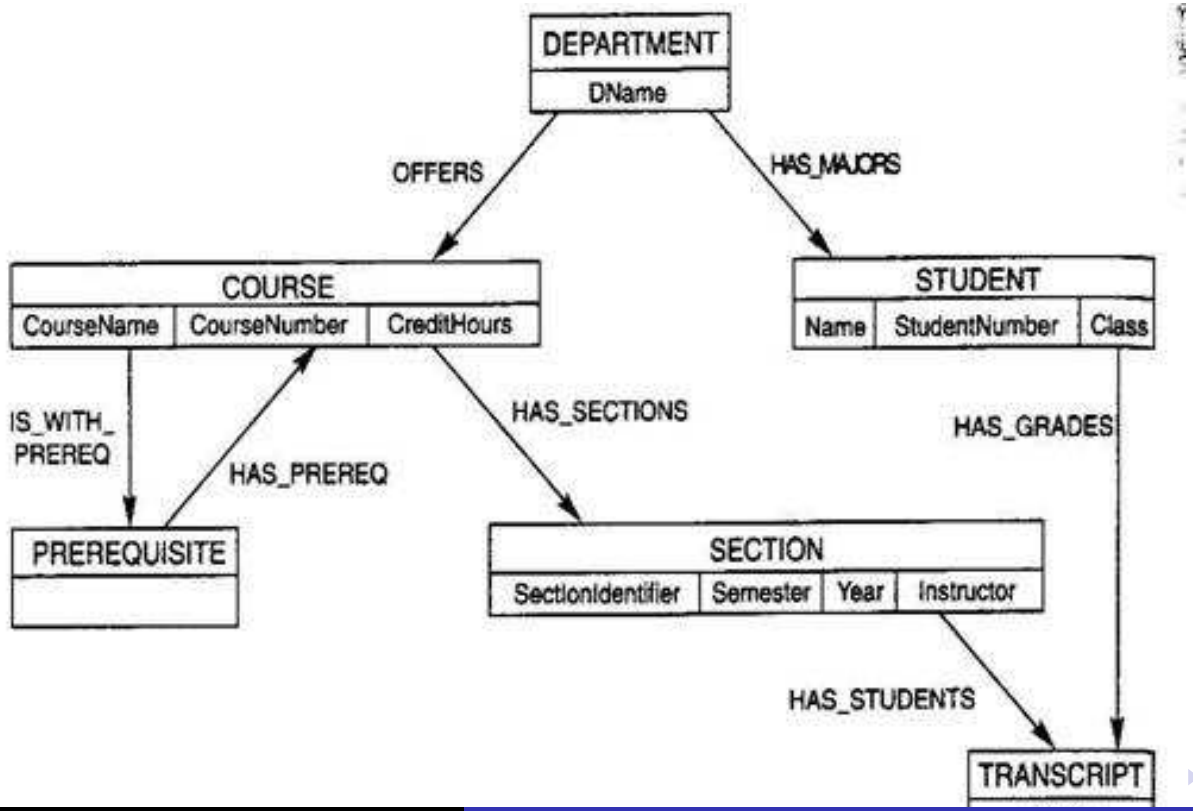
1980 - système objet Objectivity, Objectstore, db4o, Zope Object Database, Caché

1990 - système objet/relationnel

Plus récent - NoSQL regroupant différentes approches dont les systèmes à base de graphe

BD réseau représentée à l'aide d'un graphe des types

Les sommets représentent les types d'articles ; et les arcs les types d'ensembles



BD objet : état + comportement

SQL3 et ODL/OQL (ODMG) : décrire et interroger les BDOO

Listing 1: Un exemple ODL (source tech. ingenieur)

```
class DIPLOMES
tuple (Intitule : string,
  Cycle : integer,
  Detenu : set (ETUDIANTS) inverse Detient )
end;
class ETUDIANTS
tuple (Numero : integer,
  Nom : string,
  Prenoms : list(string),
  Detient : set(DIPLOMES),
  Est-inscrit : set( tuple (Mod : MODULES, inverse
      Inscrits Note : real )) )
end;
```

Quelques rappels : théorie des graphes

Eléments de vocabulaire

graphe G=<V;E> : où V représente l'ensemble des sommets et E l'ensemble des arêtes ({v1;v2}),

graphe orienté : les arêtes sont des arcs

un sous-graphe G'=<V';E'> de G=<V;E> est un graphe tel que V' $\in$ V et E' $\in$ E

un chemin C entre 2 nœuds v1 et v2 est une séquence de nœuds et d'arêtes permettant de rejoindre v2 à partir de v1

un graphe est connecté si il existe un chemin reliant toute paire de nœuds

un cycle est un chemin fermé (C(vi;vi))

un arbre est un graphe connecté et acyclique

matrice d'adjacence

Quelques rappels : théorie des graphes

De nombreux algorithmes

parcours en largeur ou en profondeur

recherche du plus court chemin (e.g. Dijkstra)

mesures de centralité (e.g. Eigenvector) : mise en avant d'indicateurs structurels

partitionnement

coloration

recherche de composantes connexes

...

Cas d'utilisation

Tout domaine qui se visualise naturellement sous forme de graphe : système NOSQL connecté (à la différence des systèmes à base d'agrégats)

1. réseaux sociaux
2. réseaux biologiques (génomique)
3. réseaux structurant les territoires (géomatique)
4. web de données (LOD), systèmes de recommandation ...

Modèle général

Les éléments clés

1. nœuds pour décrire des entités
2. propriétés pour en enrichir la description
3. arcs pour mettre en relation des entités avec d'autres entités ou encore connecter des nœuds avec leurs propriétés
4. patterns : dégager du sens à partir des connexions entre les éléments du graphe

Modèle de graphe plus ou moins riche en fonction du système considéré

Graphe attribué multivalué (Property Graph) : le plus exploité

- un ensemble de nœuds
  - chaque nœud a un identifiant unique
  - chaque nœud a un ensemble d'arcs entrants et un ensemble d'arcs sortants
  - chaque nœud possède une collection de propriétés
- un ensemble d'arcs
  - chaque arc a un identifiant unique
  - chaque arc a une extrémité sortante (queue) et une extrémité entrante (tête)
  - chaque arc possède un label qui indique le type de relation entre les deux nœuds
  - chaque arc possède une collection de propriétés (paires clé/valeur)
- ensemble de propriétés : paires clé/valeur définie comme un tableau associatif (valeur : type primitif et tableau de types

Graphe attribué multivalué

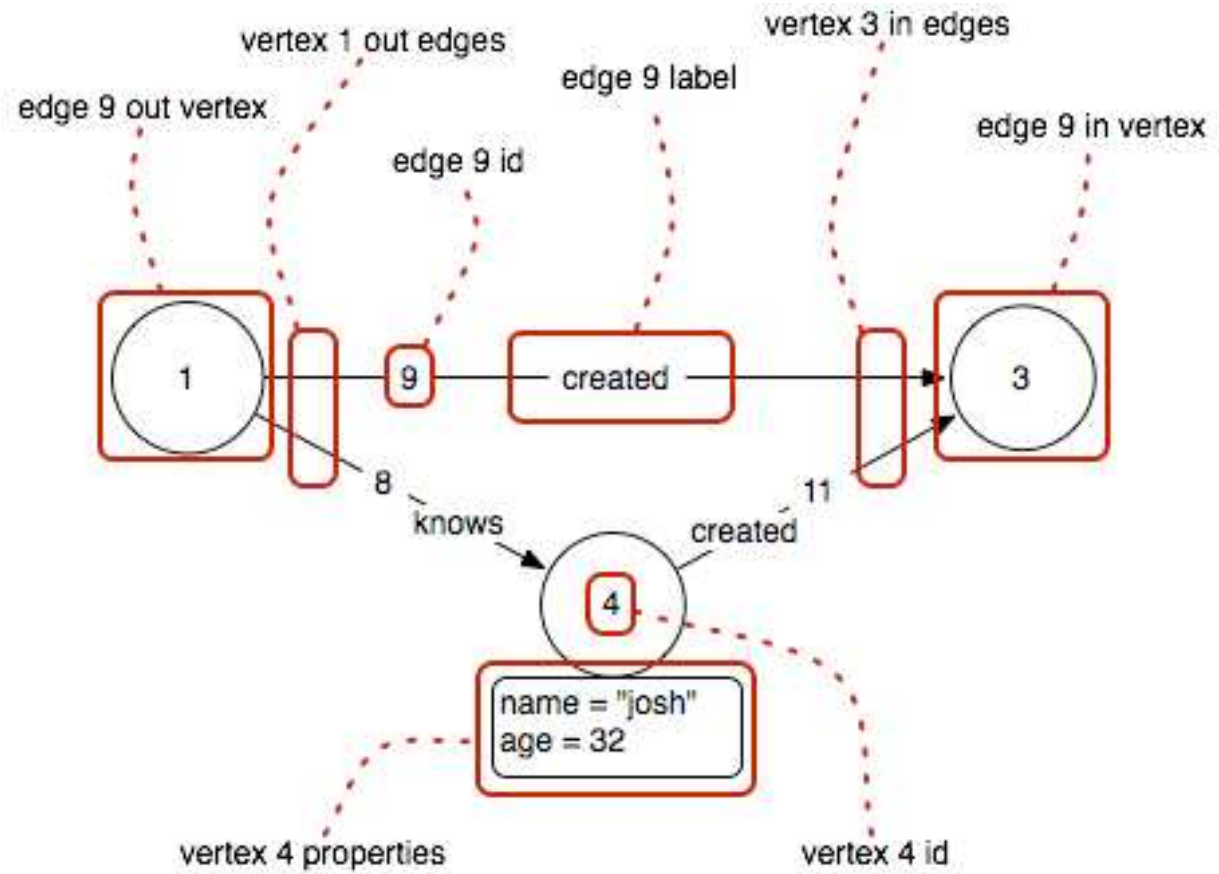


Figure: En visuel (source : tinkerpop/blueprints)

Graphe attribué multivalué

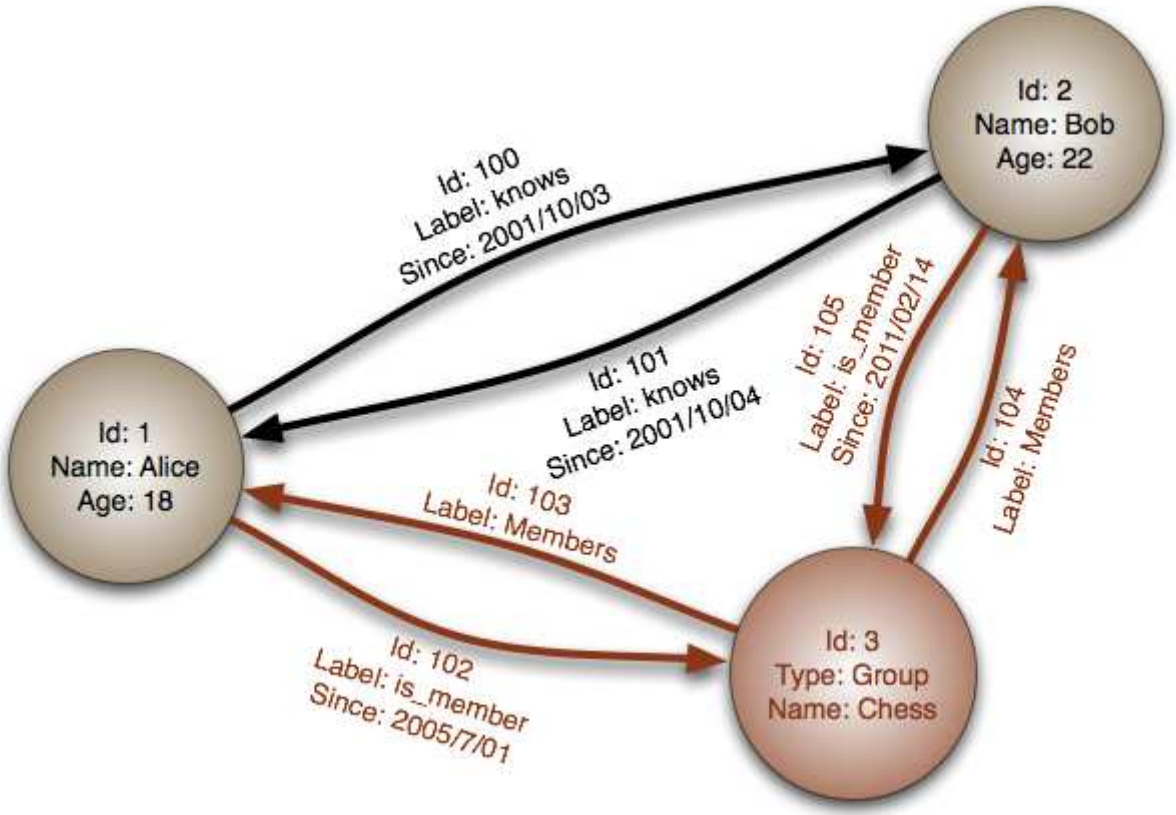


Figure: Illustré (source : Documentation Neo4J)

BD graphes

Non exhaustif

- Neo4J
- FlockDB (Twitter)
- Pregel
- InfiniteGraph
- DEX
- OrientDB
- et les solutions adossées à RDF (triplestores) à l'exemple de Stardog ou Sesame

Spécifications (non exhaustives) Neo4J (écrit en Java)

Différents supports pour l'accès et la manipulation des données

- différents stratégies de parcours de graphes (Traversal Java API)
- langages de requête Gremlin et Cypher
- index posés sur les valeurs pour un accès performant aux nœuds et arcs
- mécanismes transactionnels (ACID)
- architecture mono-serveur (la distribution est un exercice difficile dans les BD graphes)
- pensé pour le web : Java EE (framework Spring et Spring Data), web de données (SAIL et SPARQL), API et interface REST

Schema-less : type = label

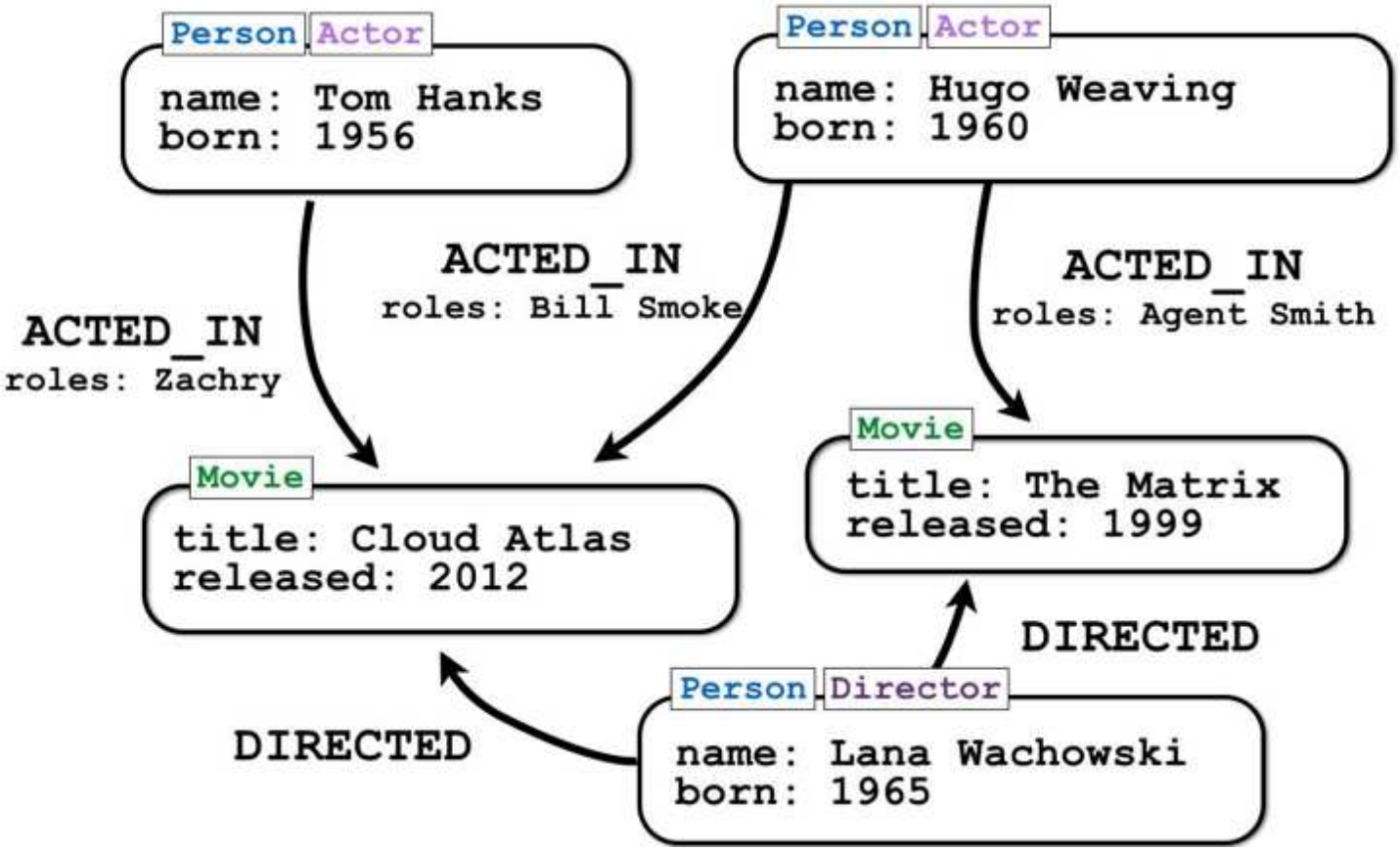


Figure: Illustré (source : Documentation Neo4J)

Modèle Neo4J

modèle : graphe marqué, orienté , attribué et multi-valué

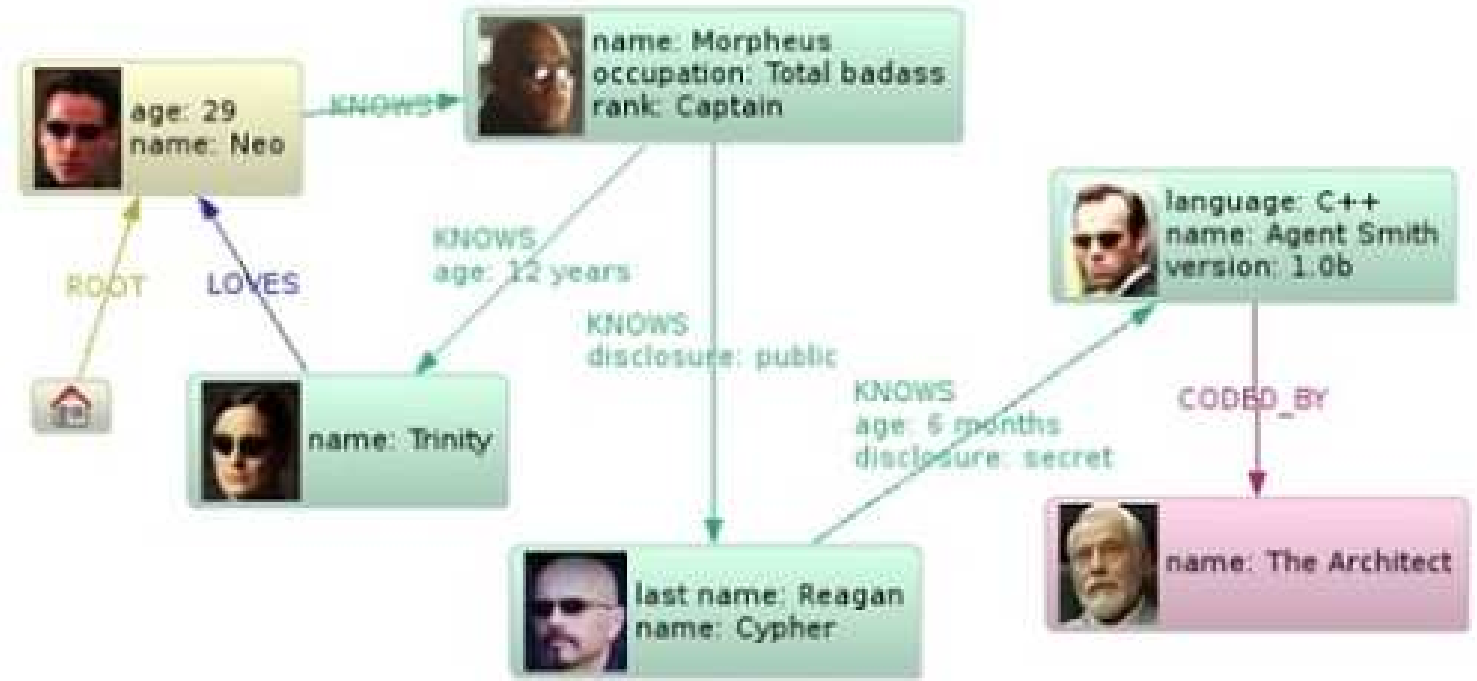


Figure: L'exemple proposé par Neo4J

Jeux de données plus large : OpenStreetMap ou MusicBrainz

Gestion pouvant aller jusqu'à plusieurs milliards de nœuds

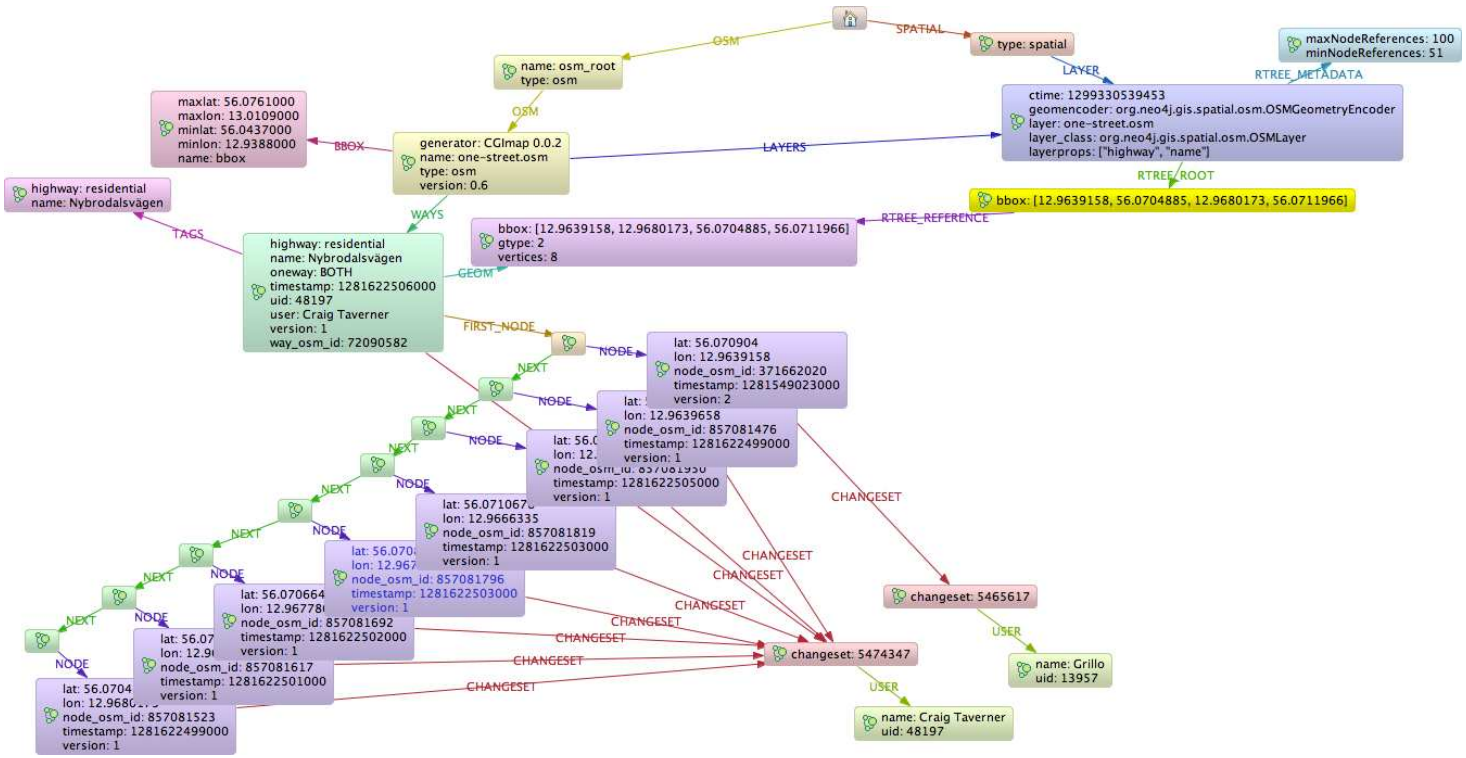


Figure: Un exemple plus complet : cartographie en Norvège avec OSM

Cypher : expressions pour poser des filtres sur le graphe

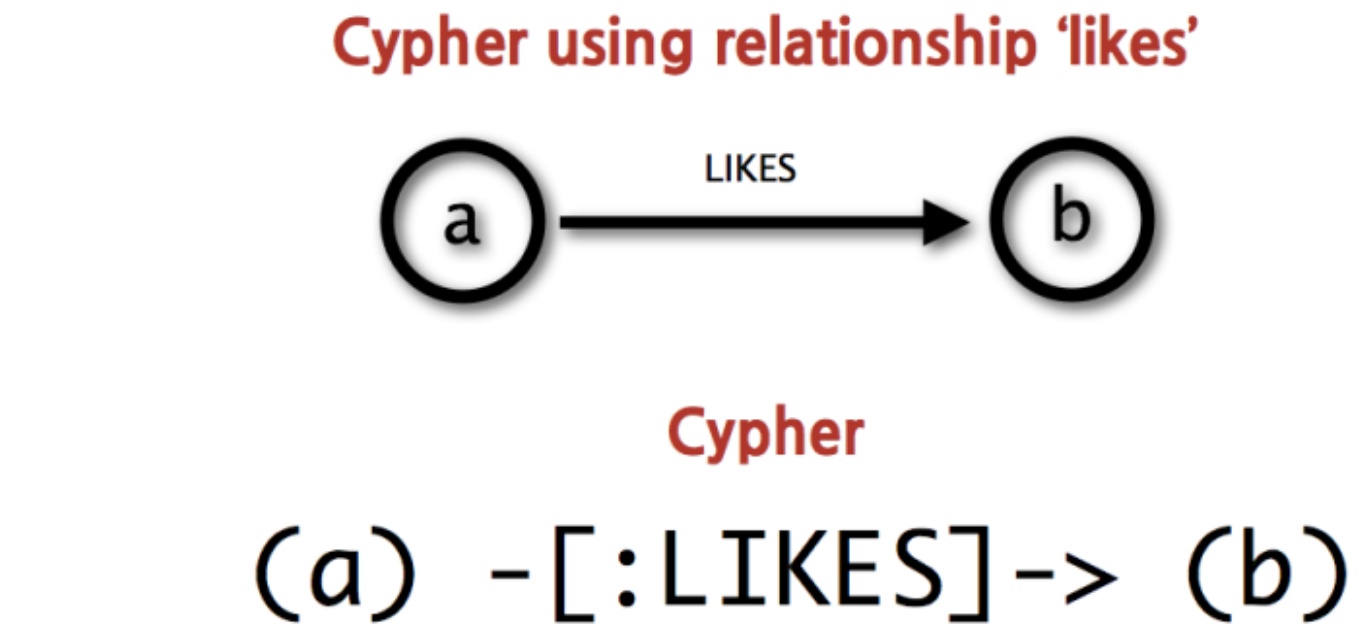


Figure: Illustré (source : Documentation Neo4J)

Exemple Cypher

Clauses dans une grammaire déclarative à rapprocher de SQL : START, MATCH, WHERE, RETURN

Listing 2: Read-Only Query Structure (doc. Neo4J)

```
START me = node:people(name = 'Pierre')
MATCH me -[:FRIEND]-> friend
WHERE friend.age > 28
RETURN friend.name
ORDER BY friend.age ASC
SKIP 5 LIMIT 10
```

Autre exemple Cypher

Listing 3: Compter sur le type des relations

```
MATCH ()-[r]->()
RETURN TYPE(r) AS rel_type, count(*) AS rel_cardinality

MATCH (:Person)-[:likes]->(r:Restaurant)
WITH r, count(*) as tr
WHERE tr > 2
RETURN r.name as RESTAURANT, tr as TOTAL
```

Une force : les appels récursifs

Listing 4: parcourir le graphe

```
(A)->()->()->()->(B)
(A)-[*]->(B)

MATCH (John {name:'John'}) -[:friend]-> () -[:friend]->
    (fof)}
RETURN John, fof
```

Retrouver le schéma

Listing 5: Infos sur le schéma

```
match n
return distinct labels(n)

match n-[r]-()
return distinct type(r)

match n-[r]-()
return distinct labels(n), type(r)
```

Exemple Cypher

Autres opérations CRUD

Listing 6: Tout supprimer dans la base (doc. Neo4J)

```
MATCH (n)
OPTIONAL MATCH (n)-[r]-()
DELETE n,r
```

Créer une BD avec l'API Java (1)

Listing 7: nodes et relations

```
public enum RelTypes implements RelationshipType
   { SOC_NODE, WORKS_WITH, WORKS_FOR }
         Node societe = graphDb.createNode();
         societe.setProperty( "name", "Soc SA" );
         socNodeId = societe.getId();
         Label empLabel =
             DynamicLabel.label("Employee");
         Label depLabel =
             DynamicLabel.label("Department");
         Node thomas = graphDb.createNode();
         thomas.addLabel(empLabel);
         thomas.setProperty( "name", "Thomas Dubois" );
         thomas.setProperty( "age", 29 );
         societe.createRelationshipTo( thomas,
             RelTypes.SOC_NODE );
```

Créer une BD avec l'API Java (2)

Listing 8: nodes et relations

```
Node sophie = graphDb.createNode();
sophie.addLabel(empLabel);
sophie.setProperty( "name", "Sophie Martin" );
sophie.setProperty( "job", "scientist" );
Relationship rel =
    thomas.createRelationshipTo( sophie,
      RelTypes.WORKS_WITH );
rel.setProperty( "since", "3 years" );
```

Créer une BD avec l'API Java (3)

Listing 9: nodes et relations

```
Node pierre = graphDb.createNode();
pierre.addLabel(empLabel);
pierre.setProperty( "name", "Pierre Claudel" );
pierre.setProperty( "salary", "high" );
thomas.createRelationshipTo( pierre,
    RelTypes.WORKS_WITH );
rel = pierre.createRelationshipTo( sophie,
    RelTypes.WORKS_WITH );
rel.setProperty( "since", "12 years" );
```

Créer une BD avec l'API Java (4)

Listing 10: nodes et relations

```
Node production = graphDb.createNode();
production.addLabel(depLabel);
production.setProperty( "nom", "production" );
production.setProperty( "localisation", "Usine
    A" );
pierre.createRelationshipTo( production,
    RelTypes.WORKS_FOR );
rel = thomas.createRelationshipTo( production,
    RelTypes.WORKS_FOR );
rel.setProperty( "role", "supervisor" );
```

Cyper avec l'API Java

Listing 11: exemple cypher

```
public static void findWithCypher()
   {
    try ( Transaction tx = graphDb.beginTx() ;
    Result result = graphDb.execute( "match (n {name:
        'Sophie Martin'}) return n, n.job" ) )
    {
     while ( result.hasNext() )
     {
     Map<String, Object> map = result.next();
  for (Map.Entry<String, Object> entry : map.entrySet())
      {
  System.out.println("key: " + entry.getKey() + ", value
      " + entry.getValue());
  } } }}
```

Nœuds Employés

Listing 12: exemple parcours

```
  public static void findAllNodesByTypes()
   {
    try ( Transaction tx = graphDb.beginTx() )
    {
   Label eLabel = DynamicLabel.label("Employee");
   ResourceIterator<Node> employees =
       graphDb.findNodes(eLabel);
   System.out.println( "Employees:" );
   while( employees.hasNext() )
   {
      Node employee = employees.next();
      System.out.println( "\t" + employee.getProperty(
          "name" ) );
   }
}}
```

mécanismes de parcours (fragment du graphe)

Listing 13: exemple parcours

```
// START SNIPPET : getColleagues
 private Traverser getColleagues( final Node person )
 {
    TraversalDescription td =
        graphDb.traversalDescription()
           .breadthFirst()
           .relationships( RelTypes.WORKS_WITH,
              Direction.OUTGOING)
           .evaluator(
              Evaluators.excludeStartPosition() );
    return td.traverse( person );
 }
 // END SNIPPET: getColleagues
```

mécanismes de parcours

Listing 14: exemple parcours

```
Node tomNode = getSocieteNode();
        int numberOfFriends = 0;
        String output = tomNode.getProperty( "name" )
            + "'s colleagues:\n";
        Traverser friendsTraverser = getColleagues(
            tomNode );
        for ( Path friendPath : friendsTraverser )
        {
        output += "At depth " + friendPath.length() +
            " => "
                + friendPath.endNode().getProperty(
                    "name" ) + "\n";
           numberOfFriends++;}
        output += "Number of friends found: " +
            numberOfFriends + "\n";
        return output;
```