

NoSQL et bases de données orientées graphe

I.Mougenot

FDS Sciences UM

Septembre 2017

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 et orienté

2 Un système en particulier : Neo4J

Volume de données versus richesse du modèle

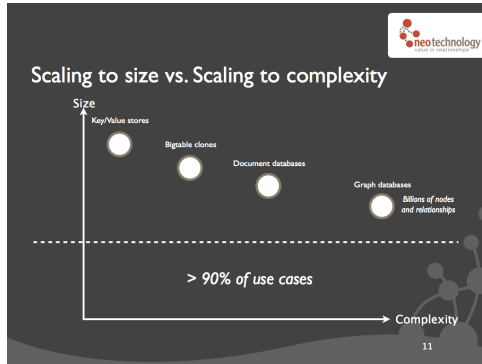


Figure: Une vision très générale (extrait de Neo Technology Webinar)

Adéquation avec le système "mental" humain

Associer et catégoriser : des mécanismes cognitifs^a naturels

^aprocessus 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

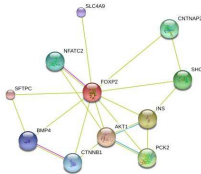


Figure: Gène FoxP2 : implication dans l'acquisition du langage (extrait de <http://acces.ens-lyon.fr/evolution>)

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

^amanières de voir les choses

1960 - système hiérarchique

1960 - système réseau (C. Bachman)

1970 - système relationnel (E.F. Codd)

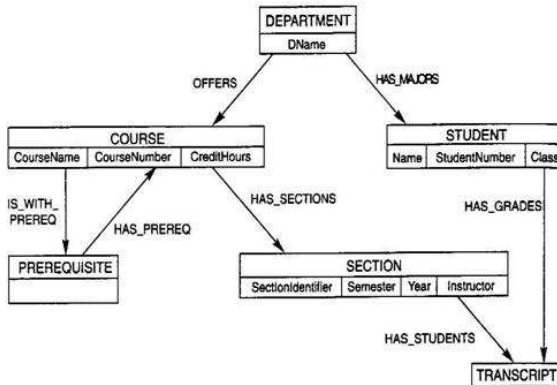
1980 - système objet Objectivity, Objectstore, db4o, Zope
Object Database, Caché

1990 - système objet/relationnel Oracle, PostgreSQL

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. ingénieur)

```
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

Éléments de vocabulaire

graphe $G = \langle V; E \rangle$: 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 $G' = \langle V'; E' \rangle$ de $G = \langle V; E \rangle$ est un graphe tel que $V' \subseteq V$ et $E' \subseteq E$

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 ($C(v_i; v_i)$)

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 (Property Graph) : le plus souvent exploité

- un ensemble de nœuds
 - chaque nœud a un identifiant unique, un ensemble d'arcs entrants et sortants, et possède une collection de propriétés
- un ensemble d'arcs
 - chaque arc a un identifiant unique, une extrémité sortante (queue) et une extrémité entrante (tête), un label indiquant le type de relation entre les deux nœuds, et 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 primitifs)

Graphe attribué

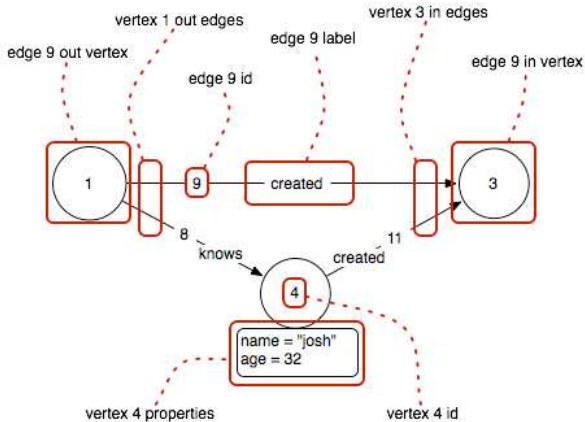


Figure: En visuel (source : tinkerpop/blueprints)

Graphe attribué : illustration

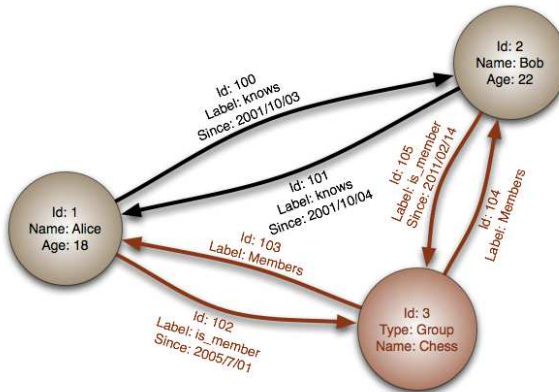


Figure: Illustré (source : Documentation Neo4J)

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

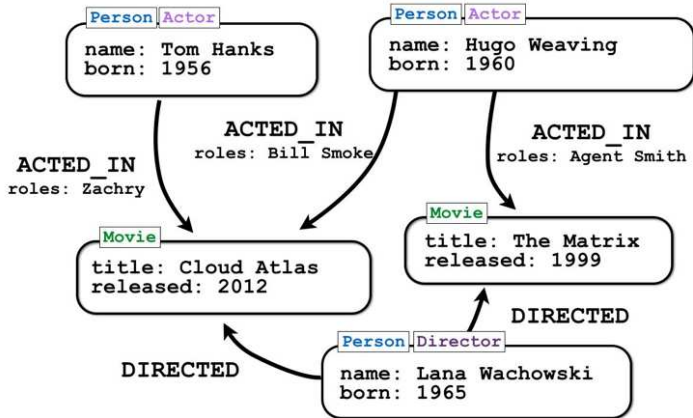


Figure: Illustré (source : Documentation Neo4J)

Modèle Neo4J

modèle : multigraphe orienté , attribué



Figure: L'exemple proposé par Neo4J

Gestion pouvant aller jusqu'à plusieurs milliards de nœuds

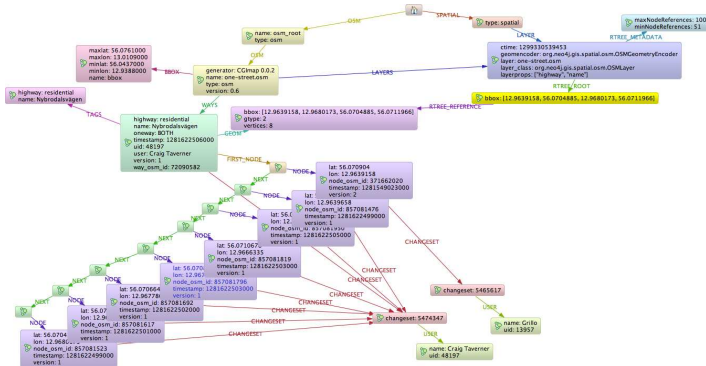


Figure: Un exemple plus complet : cartographie en Norvège avec OSM

Cypher : expressions pour poser des filtres sur le graphe

Cypher using relationship 'likes'



Cypher

(a) -[:LIKES]-> (b)

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

Listing 7: principes autour BD

```
public class Exemple2_Main {
    static GraphDatabaseFactory graphDbFactory = new
        GraphDatabaseFactory();
    static GraphDatabaseService graphDb =
        graphDbFactory.newEmbeddedDatabase(
            new File("data/soc"));

    public static void main(String[] args) {
        Transaction tx = graphDb.beginTx();
        try {
            Node car = graphDb.createNode(Labels.CAR);
            car.setProperty("brand", "citroen");
            ... } catch (Exception e) { tx.failure(); } ...
    }
```

Les labels des Nœuds : classe énumérée

Listing 8: nodes et relations

```
import org.neo4j.graphdb.Label;

public enum Labels implements Label {
    EMPLOYEE,
    SOCIETY,
    CAR,
    DEPARTMENT
}
```

Les relations : autre classe énumérée

Listing 9: nodes et relations

```
import org.neo4j.graphdb.RelationshipType;

public enum Relations implements RelationshipType {
    WORKS_WITH, WORKS_FOR, FRIEND, OWNS
}
```

Nœuds et arcs de la base à créer au sein d'une transaction

Listing 10: nodes et relations

```
Node car = graphDb.createNode(Labels.CAR);  
car.setProperty("brand", "citroen");  
car.setProperty("model", "2cv");  
  
Node owner = graphDb.createNode(Labels.EMPLOYEE);  
owner.setProperty("firstName", "Isa");  
owner.setProperty("lastName", "M");  
owner.setProperty("job", "teacher");  
owner.createRelationshipTo(car, Relations.OWNS);
```

Propriété sur une relation

Listing 11: nodes et relations

```
Node sophie = graphDb.createNode(Labels.EMPLOYEE);
sophie.setProperty( "firstName", "Sophie" );
sophie.setProperty( "job", "scientist" );
Relationship rel = sophie.createRelationshipTo(
    owner,
        RelTypes.WORKS_WITH );
rel.setProperty( "since", "3 years" );
```

Première consultation avec Cypher

Listing 12: Cypher

```
Result result = graphDb.execute(
"MATCH (c:CAR) <-[OWNS]- (p:EMPLOYEE) " +
"WHERE c.brand = 'citroen' " +
"RETURN p.firstName, p.lastName");
while ( result.hasNext() )
{
    Map<String, Object> row = result.next();
    for ( String key : result.columns() )
    {
        System.out.printf( "%s = %s%n", key, row.get(
            key ) );
    }
}
```

Autre requête Cypher

Listing 13: nodes et relations

```
public static void findWithCypher()
{
    try ( Transaction tx = graphDb.beginTx() ;
        Result result = graphDb.execute( "match (n
            {firstName: 'Isa'}) 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());}
        }
    }
}
```


Exploiter le graphe : nœuds Employés

Listing 14: parcours graphe

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

Nœuds Employés

Listing 15: exemple parcours

```
public static void findAllOwners()  
{ try ( Transaction tx = graphDb.beginTx() )  
  { ResourceIterator<Node> owners = graphDb.findNodes(  
    Labels.EMPLOYEE );  
while( owners.hasNext() )  
{ Node owner = owners.next();  
  System.out.print( "\t" + owner.getProperty(  
    "firstName" ) + " owns " );  
  for( Relationship relationship :  
    owner.getRelationships(  
      Relations.OWNS ) )  
  { Node car = relationship.getOtherNode( owner );  
    System.out.print( "\t" + car.getProperty( "brand"  
      ) ); }  
  System.out.println();  
}
```

mécanismes de parcours (fragment du graphe)

Listing 16: exemple parcours

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