NoSQL, graphes et Neo4J

I.Mougenot

HMIN340 - FDS

Septembre 2018

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

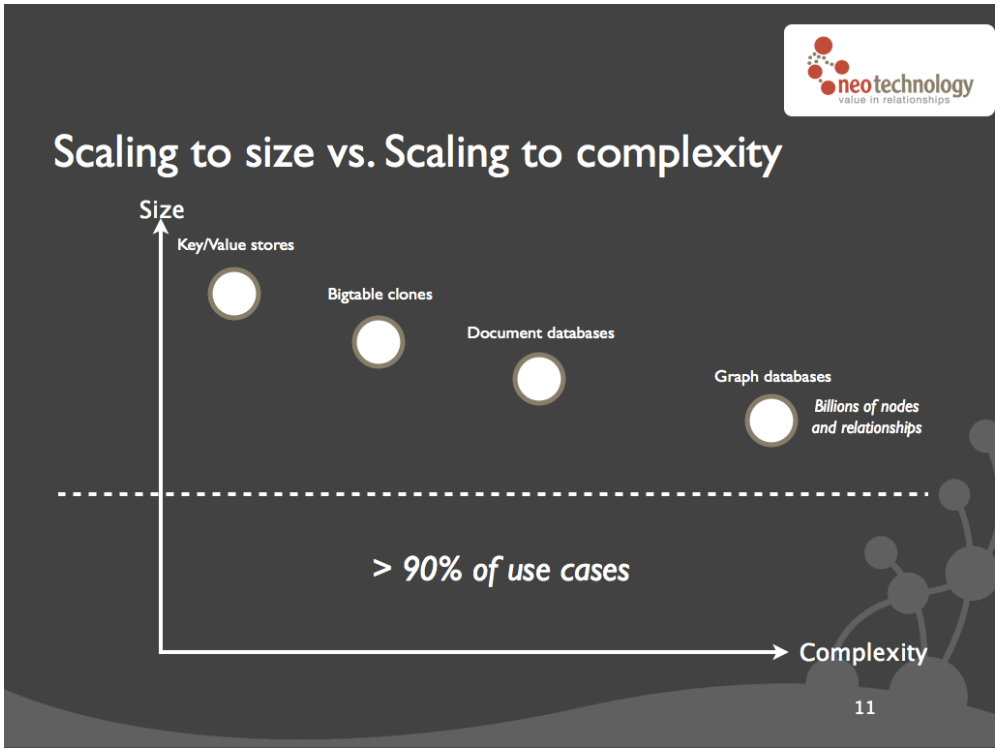


Figure: Une vision très générale (extrait de Neo Technology Webinar)

Adéquation avec le système "mental" humain

Associer et catégoriser : des mécanismes cognitifs[a] naturels

[a]processus psychiques liés à l'esprit

- catégories et associations représentées à l'aide de graphes (que l'on sait traiter efficacement)

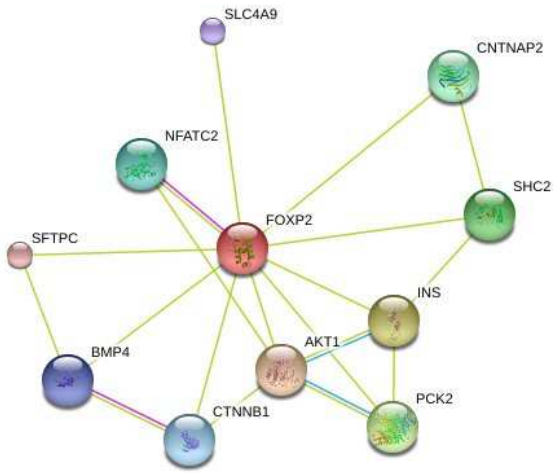


Figure: Gène FoxP2 : implication dans l'acquisition du langage (extrait de http://acces.ens-lyon.fr/evolution)

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)

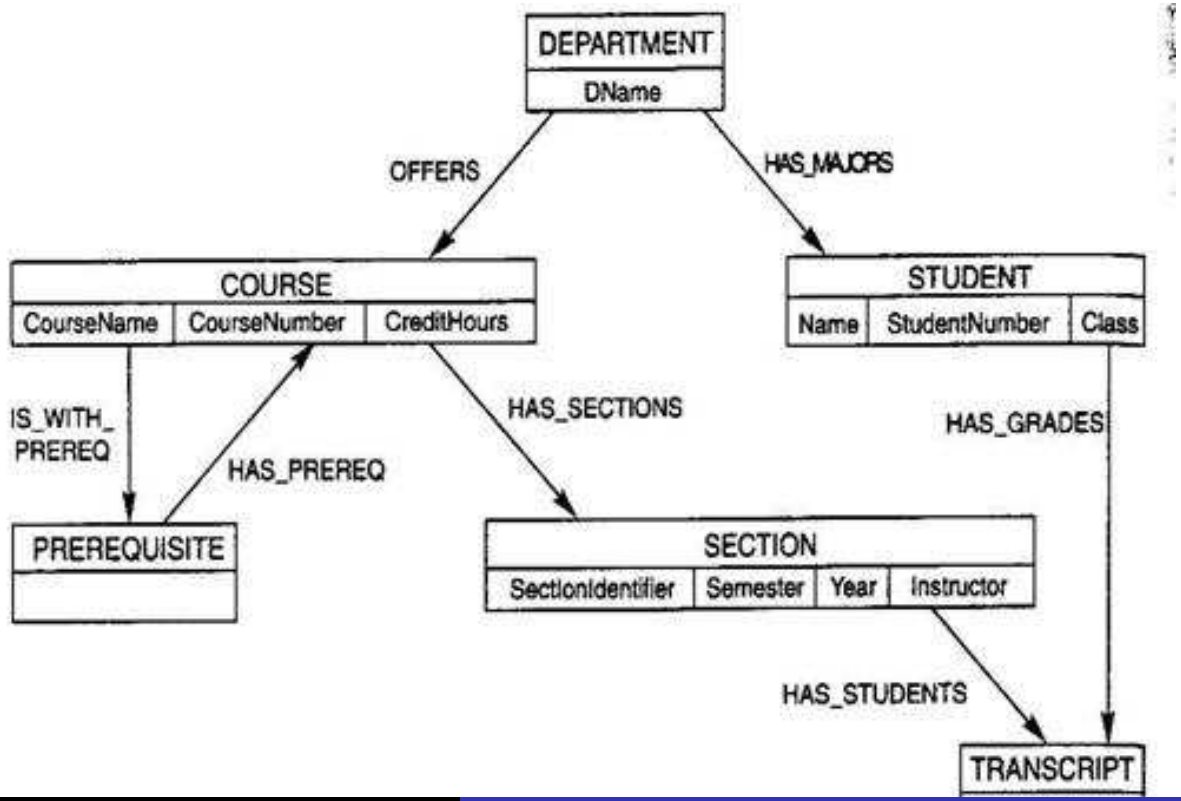
1980 - système objet Objectivity, Objectstore, db4o, Zope Object Database, Caché

1990 - système objet/relationnel 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. 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, ensemble des sommets et E, ensemble des arêtes ({v1;v2}),
- graphe orienté : les arêtes sont des arcs
- sous-graphe G'=<V';E'> de G=<V;E> est un graphe tel que V' $\in$ V et E' $\in$ E
- chemin C entre 2 nœuds v1 et v2 : séquence de nœuds et d'arêtes permettant de rejoindre v2 à partir de v1
- un graphe est dit 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 comme structure support

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 (cascades signalétiques, voies métaboliques, ...)
3. réseaux structurant les territoires (géomatique)
4. web de données (LOD), systèmes de recommandation en ligne ...

Grandes forces

Complexité des données : connectivité + volume + structuration partielle

Atouts des systèmes graphes

1. requêtes topologiques : produits d'expression de gènes interagissant en cascade, amis d'amis d'ennemis, meilleure façon de relier Paris à partir de Montpellier . . . . . .

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)

multi-graphe attribué et orienté : illustration Neo4J

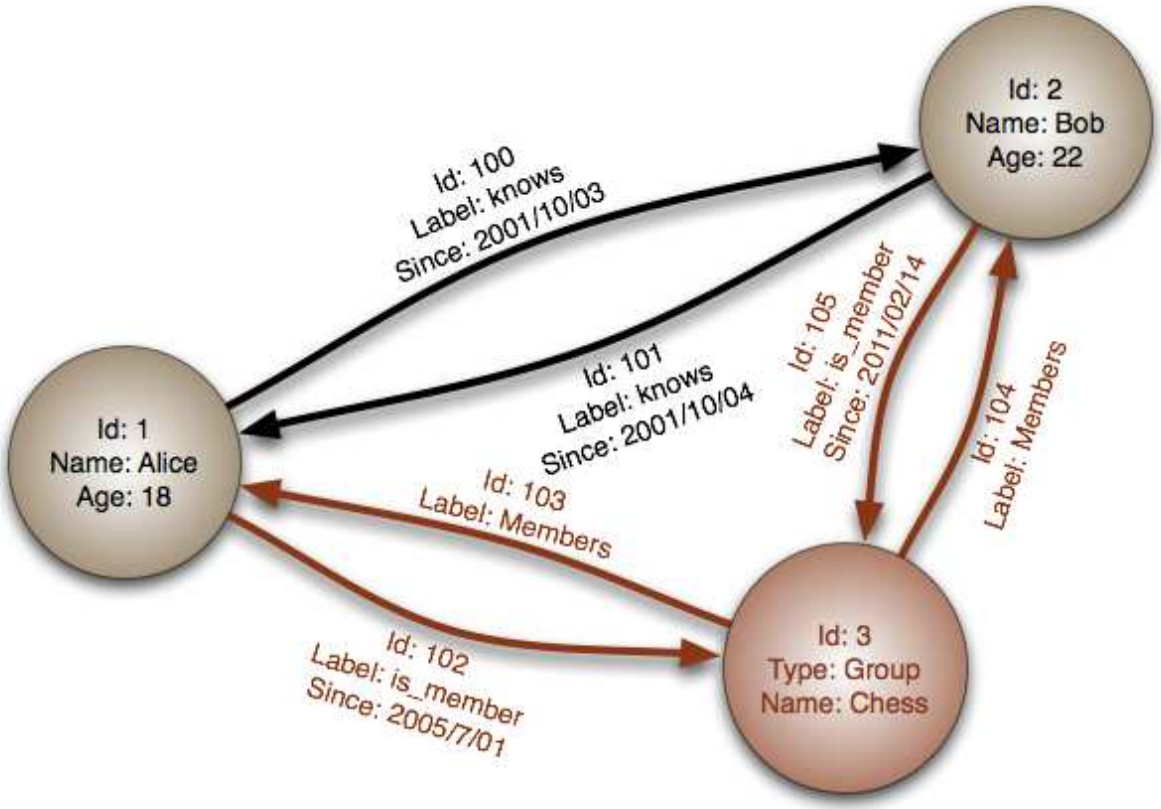


Figure: Illustré (source : Documentation Neo4J)

BD graphes (pas toujours graphe attribué)

Non exhaustif

- Neo4J
- FlockDB (Twitter)
- Pregel (décisionnel)
- InfiniteGraph
- DEX
- OrientDB
- HypergraphDB
- 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 "clustérisée" pour version payante (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 d'entité = label et type de relation = type

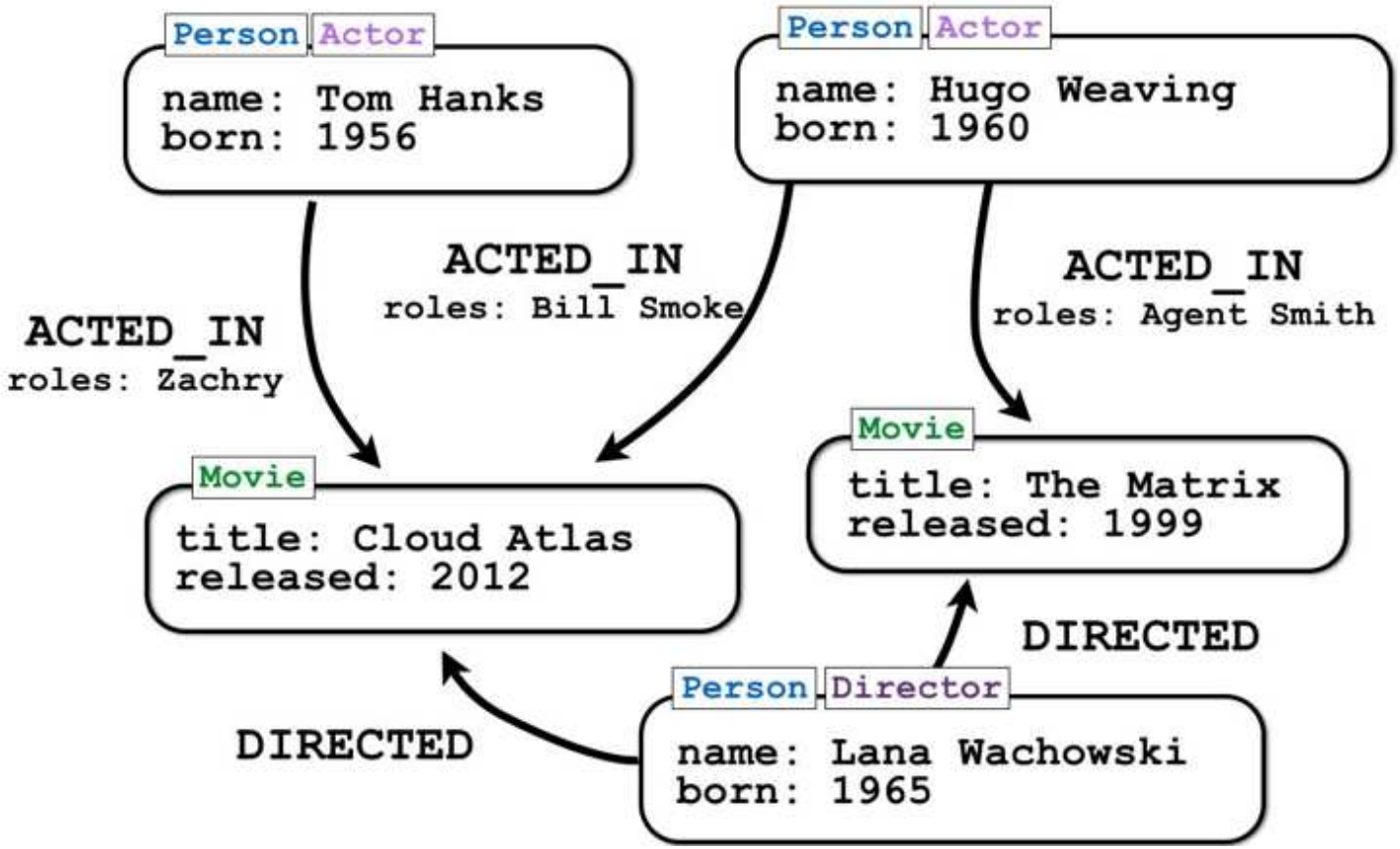


Figure: Illustré (source : Documentation Neo4J)

Jeux de données plus large : OpenStreetMap ou MusicBrainz

Gestion pouvant aller jusqu'à plusieurs milliards de nœuds (2^{32} identifiants possibles)

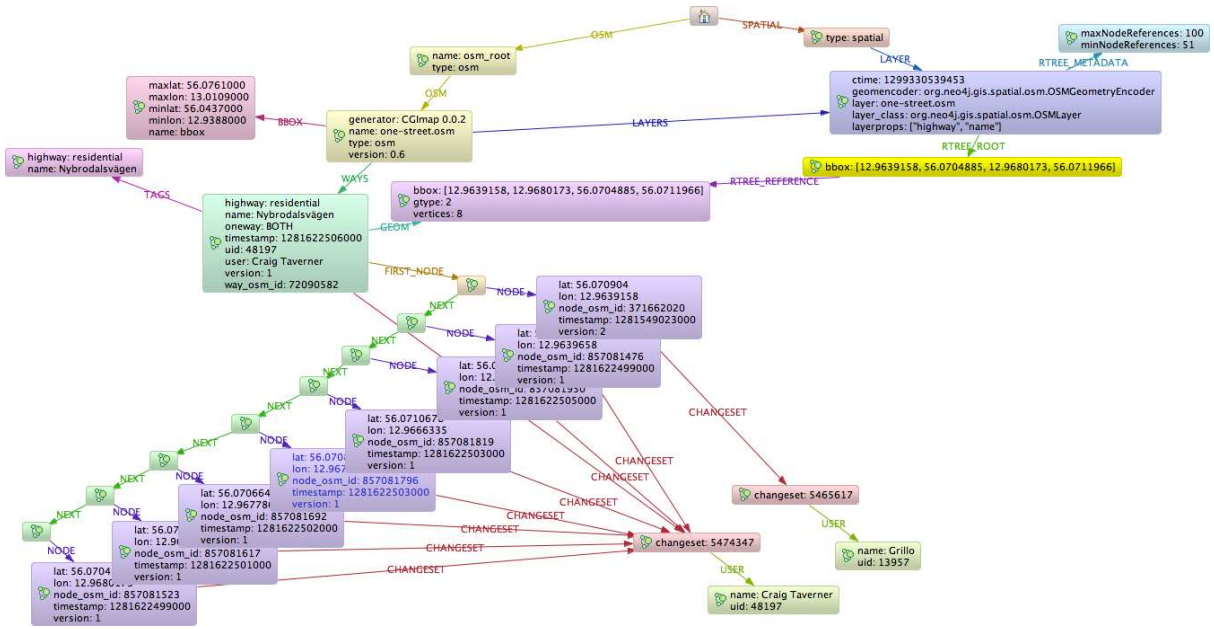


Figure: Un exemple plus complet : cartographie en Norvège avec OSM

Modèle de données Neo4J

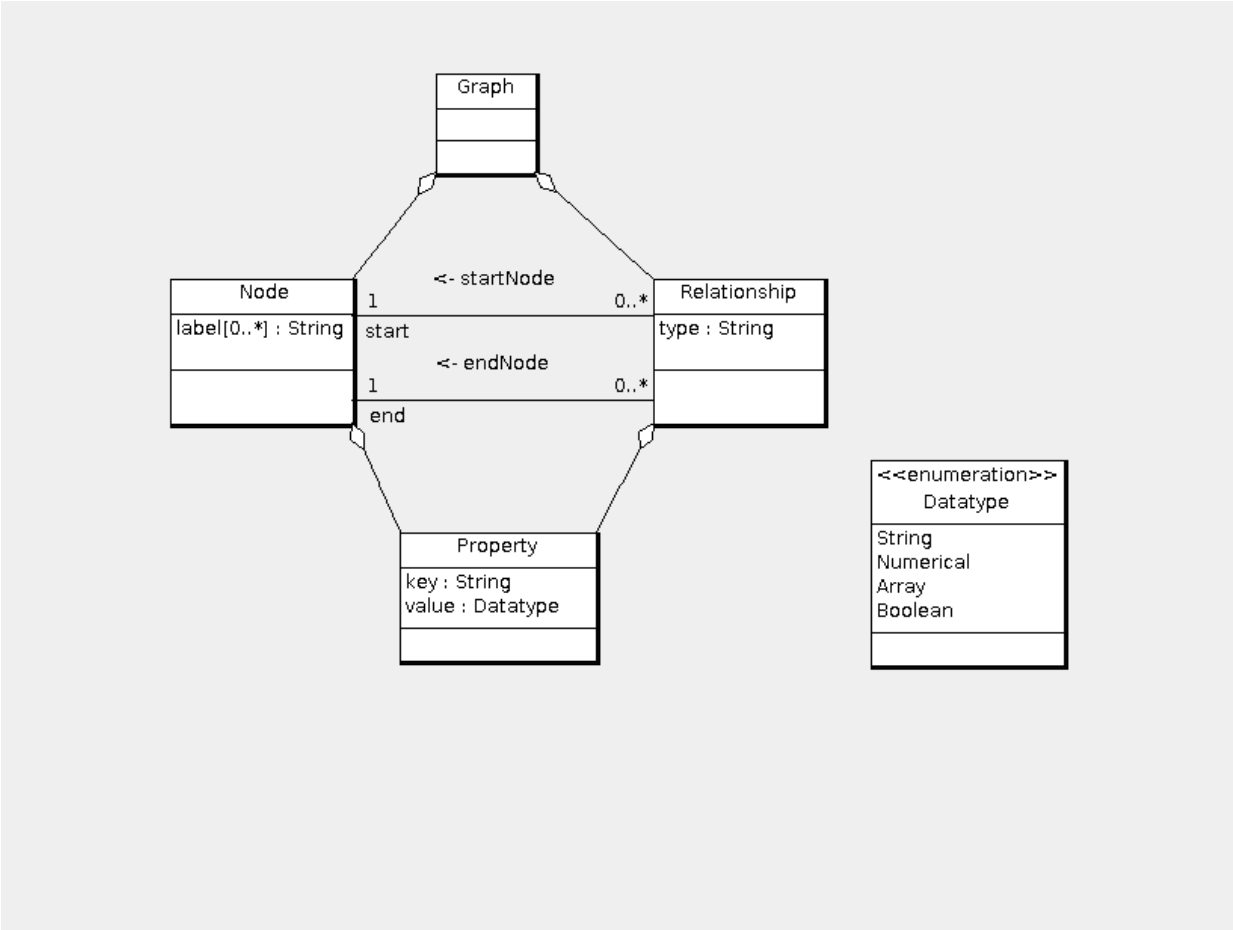
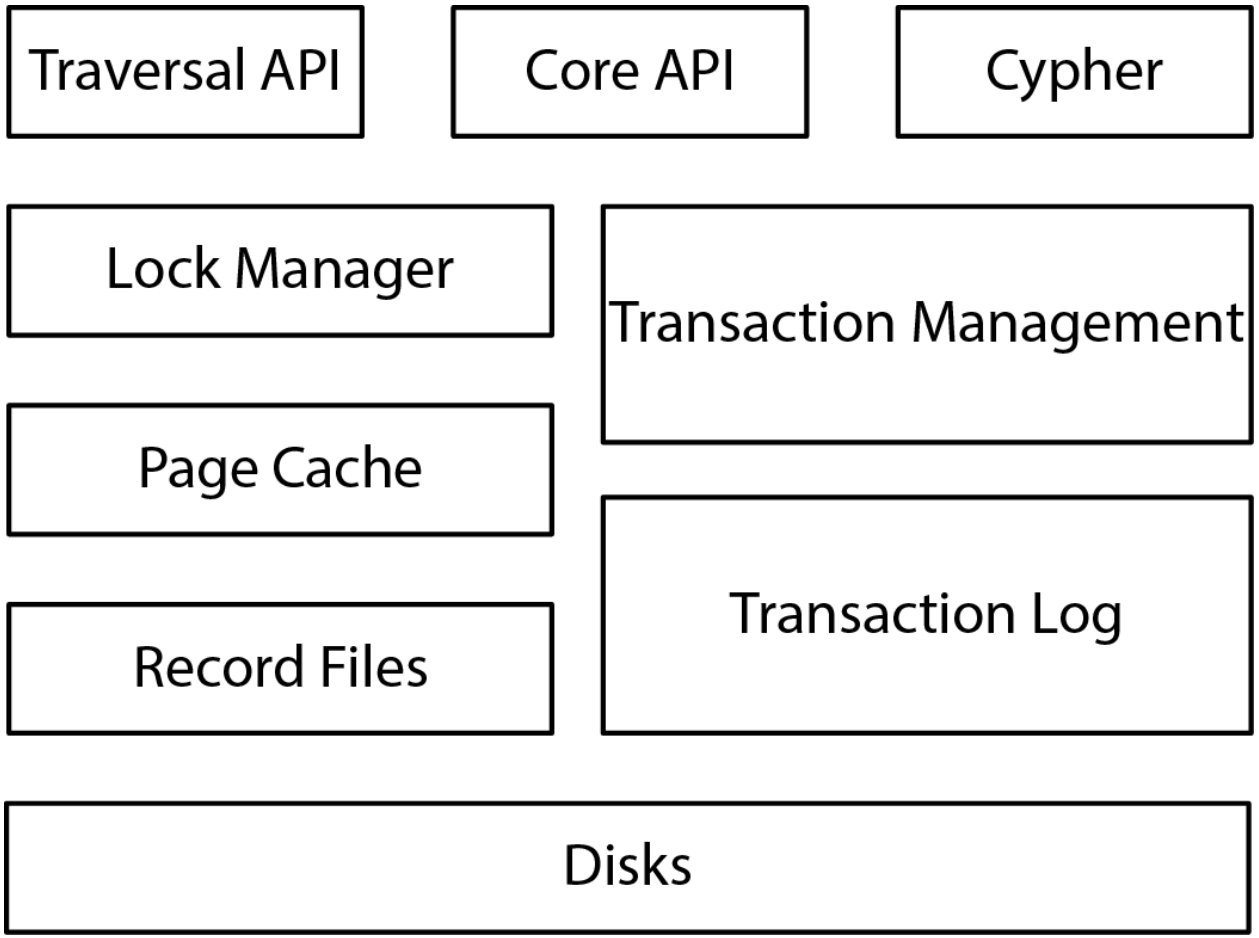


Figure: Diagramme de classes UML illustratif

Principales briques du système



Cypher : expressions pour poser des filtres sur le graphe

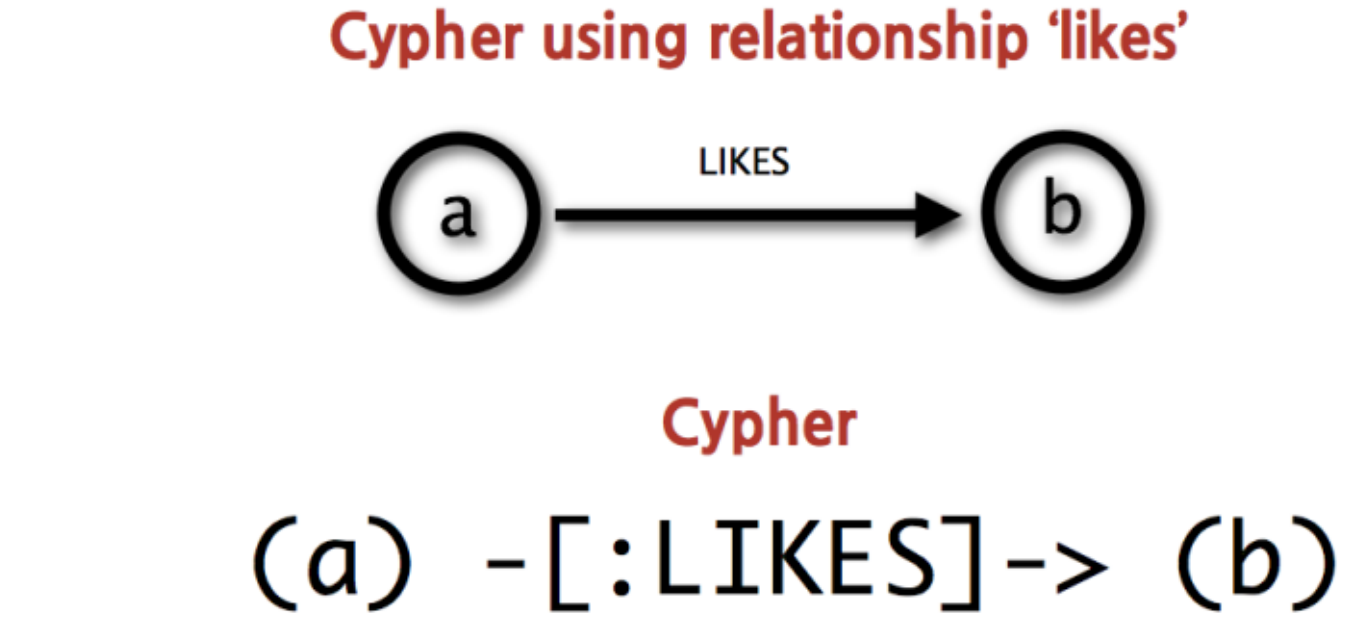


Figure: Illustré (source : Documentation Neo4J)

Cypher : principales clauses

- CREATE - création de nœuds et d'arêtes
- DELETE - removes de nœuds, d'arêtes et de propriétés
- SET - mise à jour de valeurs de propriétés
- MATCH - rechercher des points d'entrée dans le graphe
- MERGE - combinaison de MATCH et CREATE
- WHERE - poser des sélections
- RETURN - nœuds et arêtes à retourner
- UNION - combiner les résultats de plusieurs requêtes
- WITH - sorte de pipe en commande Unix . . .

Exemple Cypher

Ordres de création

```
CREATE (m:Commune:Ville {nom:'MONTPELLIER',
    latitude:43.610769, longitude:3.876716,
    codeinsee:'34172'})
RETURN m

CREATE (m:Commune {nom:'NIMES', codeinsee:'30189'})
    -[:WITHIN]-> (h:Departement {nom:'GARD',
    numero:'30'})
```

Listing 2: CREATE

Exemple Cypher

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

```
MATCH (d:Departement {nom:'GARD'}) <-[p:WITHIN]-
    (n:Commune)
RETURN d, n, p

MATCH (d:Departement) <-[p:WITHIN]- (n:Commune)
WHERE d.nom = 'GARD'
RETURN d.nom, n.nom
```

Listing 3: MATCH

Exemples génériques Cypher

Listing 4: infos schema

```
match n
return distinct labels(n)

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

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

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

Exemple de partitionnement

Listing 5: Compter les communes

```
MATCH (:Commune)-[:WITHIN]->(d:Departement)
WITH d, count(*) as nC
WHERE nC > 8
RETURN d.nom as dep, nC as communes
```

Une force : les appels récursifs

Listing 6: parcourir le graphe

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

MATCH (c1:Commune)-[:NEARBY]->()<-[:NEARBY]-(c2:Commune)
RETURN c1, c2

MATCH (m:Commune {nom:'MONTPELLIER'}) -[:NEARBY*]-
    (n:Commune)
RETURN m, n

MATCH (m:Commune {nom:'MONTPELLIER'}) -[:NEARBY*0..2]-
    (n:Commune)
RETURN m, n
```

Cypher : requête de navigation

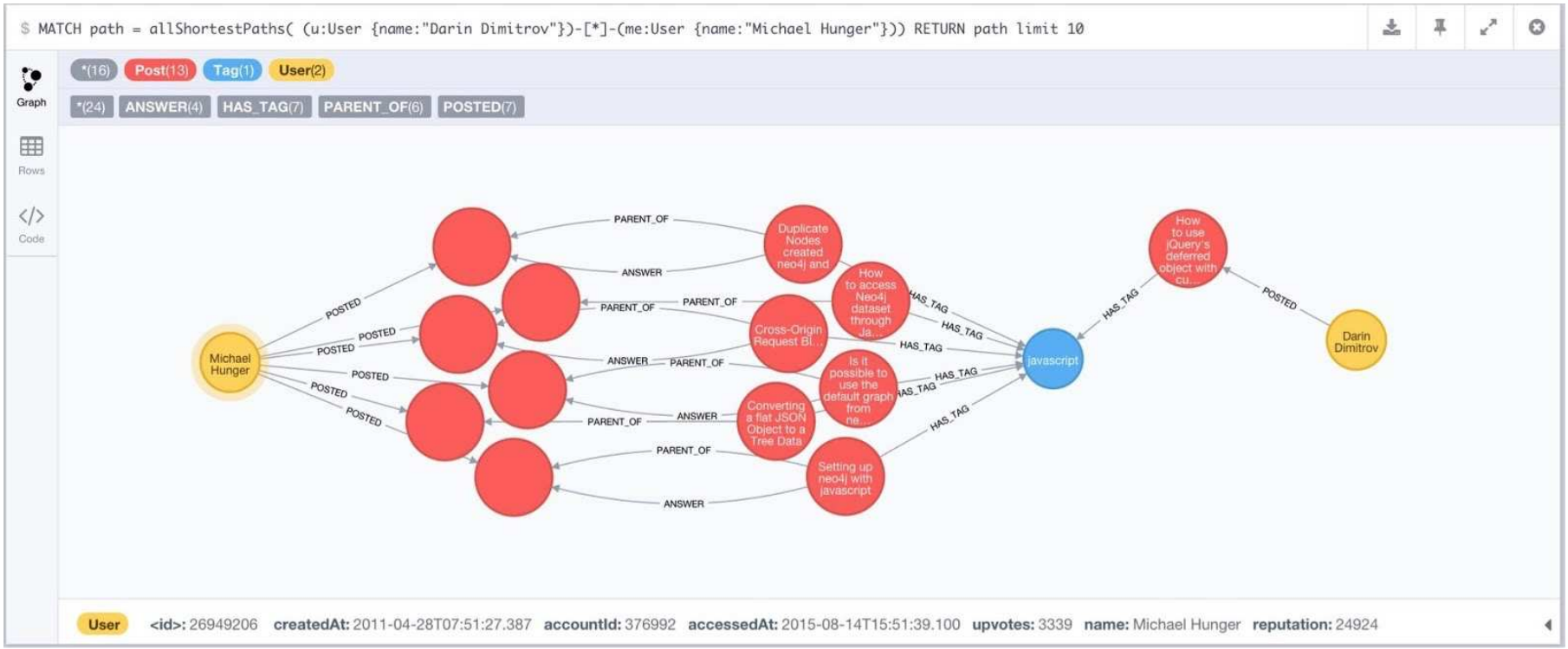


Figure: Recherche des chemins les plus courts

Exemple Cypher

Nouvelle consultation sur les chemins

Listing 7: taille du/des chemin(s) le(s) plus court(s) entre MONTPELLIER et NIMES

```
MATCH p=shortestPath(
    (m:Commune)-[:NEARBY*]-(n:Commune) )
WHERE m.nom='MONTPELLIER' and n.nom = 'NIMES'
RETURN length(p) as taillePLusCourtChemin
```

Exemple Cypher

Autres opérations CRUD

Listing 8: Tout supprimer dans la base

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


MATCH (n)
DETACH DELETE n
```

Exemple Cypher

Performances d'accès : index utilisé par défaut

Listing 9: performance consultation

```
CREATE INDEX ON :Commune(nom)

CREATE CONSTRAINT ON (d:Departement) ASSERT d.nom IS
    UNIQUE
```

Exemple Cypher

Performances d'accès : index utilisé par défaut

Listing 10: performance consultation

```
CREATE INDEX on :Commune(nom)

EXPLAIN MATCH (c:Commune {nom:'MONTPELLIER'}) RETURN c
```

Cypher : visualiser le plan d'exécution

Figure: Planner en lieu et place d'Optimizer

Au travers d'un serveur d'application Jetty

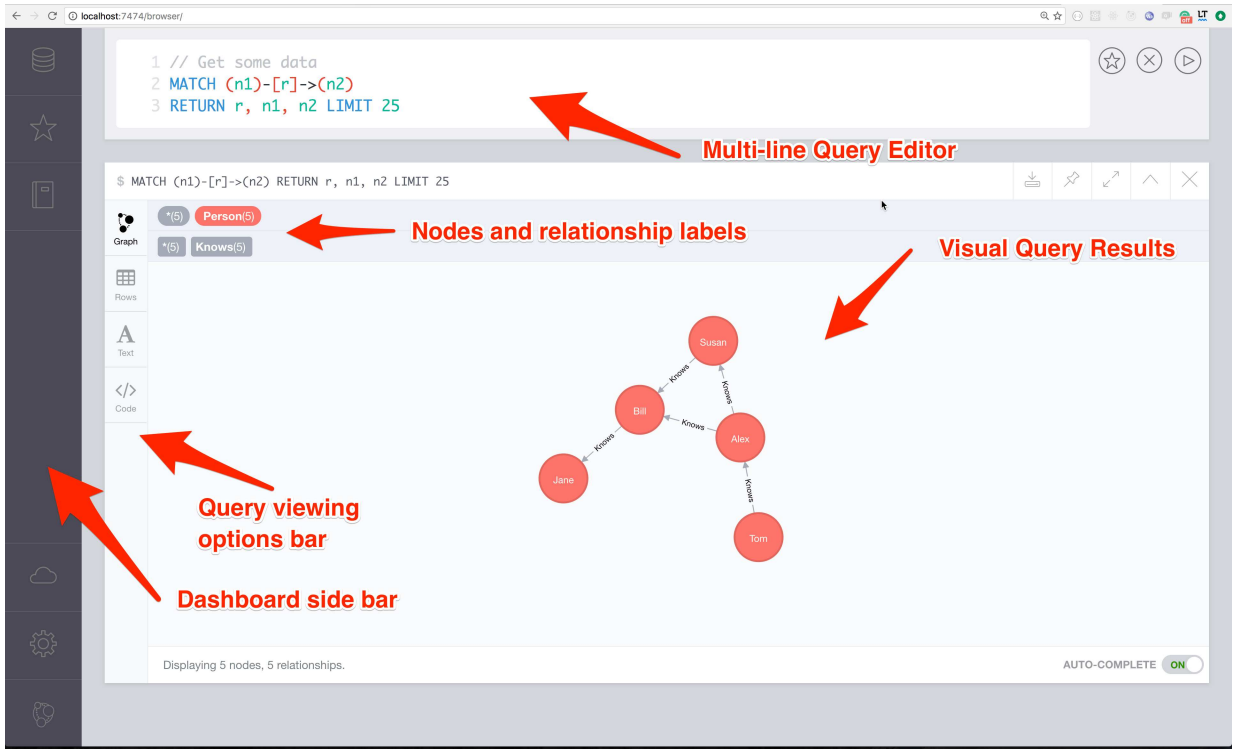


Figure: Interface d'accueil Web

Services Web REST

Listing 11: des exemples

```
browser web : http://localhost:7474/db/data/node/2
client curl :
    http://neo4j:neo4j17@localhost:7474/db/data/

curl -v -X POST -d '{"query":"CREATE (n:City
    {name:\"Montpellier\",lon:4.3,lat:43.2}) RETURN n;"}'
--header "Content-Type:application/json"
    http://neo4j:neo4j17@localhost:7474/db/data/cypher
}
```

,

Offre différents services REST : GET, POST, PUT, DELETE

Neo4J en mode embarqué

Listing 12: nodes et relations

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

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

Application autonome Java : Les labels des Nœuds : classe énumérée

Les relations : autre classe énumérée

Listing 13: nodes et relations

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

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

Créer une BD avec l'API Java

Listing 14: 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(); } ...
```

Le mode embarqué pose un verrou exclusif sur toute la base

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

Listing 15: 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("lastName", "M");
owner.setProperty("job", "teacher");
owner.createRelationshipTo(car, Relations.OWNS);
```

Première consultation avec Cypher

Listing 16: 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 ) );
   }
}
```