

Séance 4

Le modèle orienté-graphe : Neo4j et OrientDB

Sébastien Combéfis, Virginie Van den Schriek mercredi 9 novembre 2016

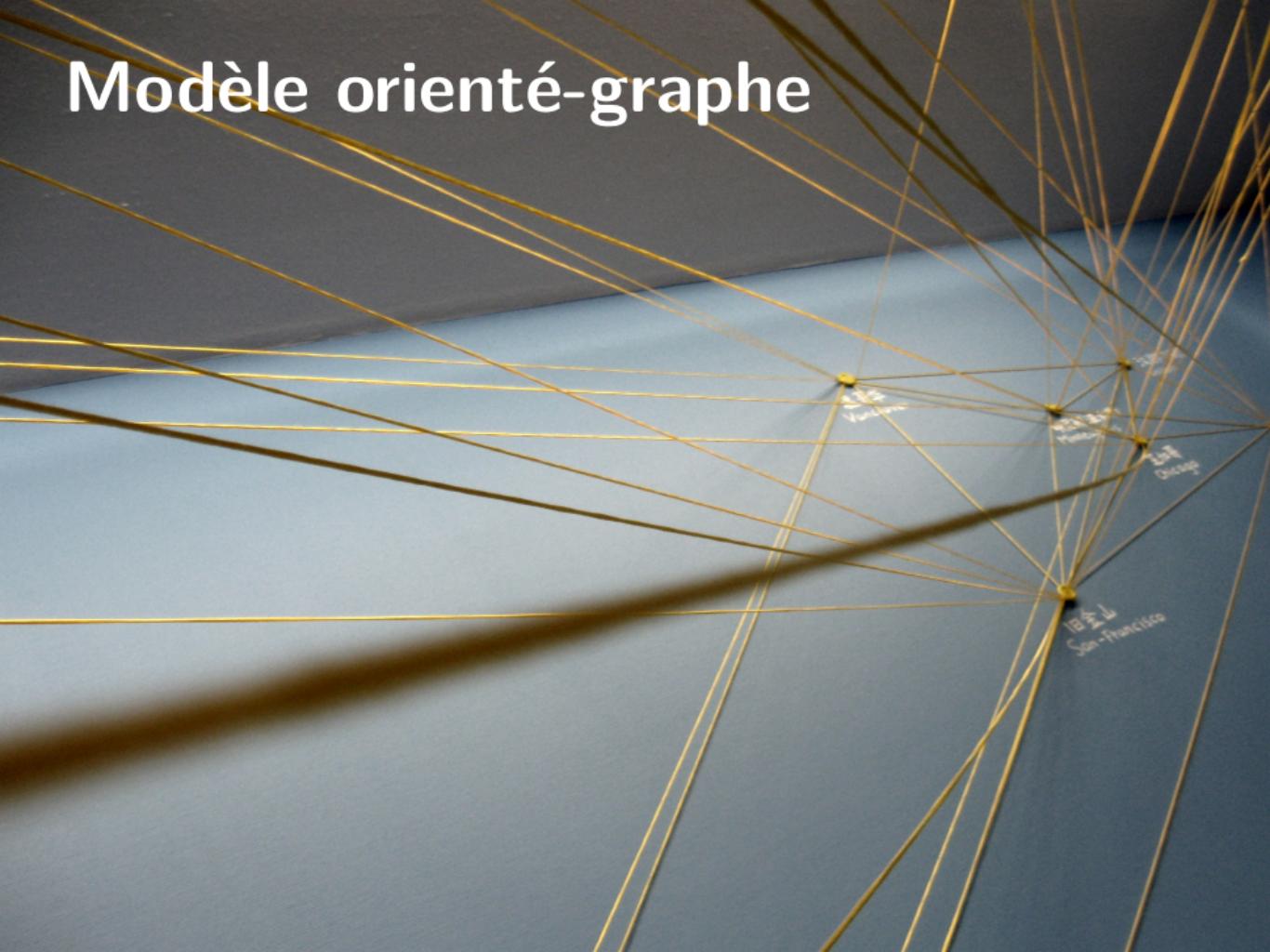


Ce(tte) œuvre est mise à disposition selon les termes de la Licence Creative Commons
Attribution – Pas d'Utilisation Commerciale – Pas de Modification 4.0 International.

Objectifs

- Le modèle **orienté-graphe**
 - Définition d'un graphe et modèle de données
 - Web sémantique et langages d'interrogation
- **Exemples** de bases de données
 - Neo4j
 - OrientDB

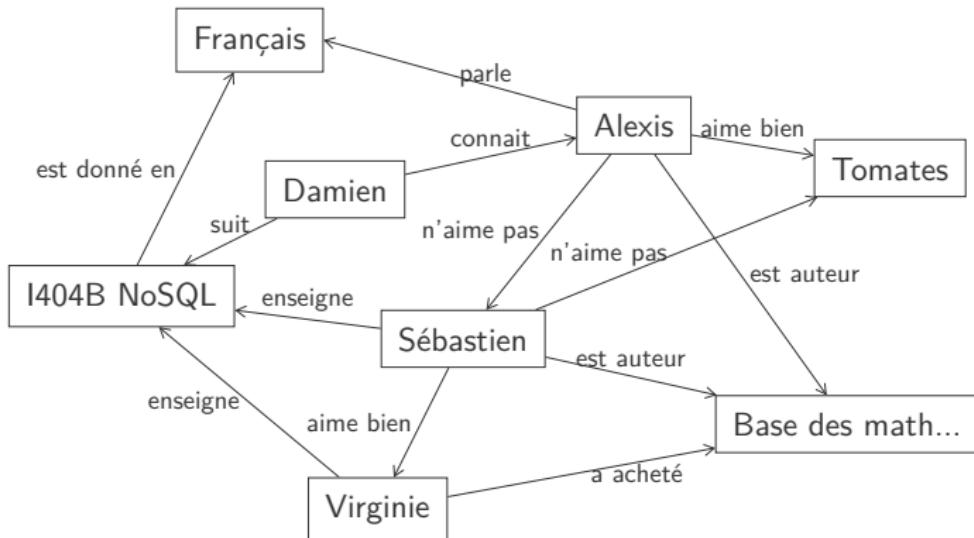
Modèle orienté-graphe



Graphe (1)

- Base de données **orientée-graphe** stocke des entités et relations
Les relations permettent de relier les entités entre elles
- Un **nœud** représente une entité et possède des propriétés
Une instance d'un objet dans une application
- Une **arête** est directionnelle et possède des propriétés
Établissement de liens entre les instances dans une application

Graphe (2)



Modèle de données (1)

- Un **graphe** est un ensemble de nœuds et d'arêtes

Tous deux possédant plusieurs propriétés

- Les arêtes sont **directionnelles** et ont un type

Certaines peuvent être bidirectionnelles dans la signification

- Deux **principaux avantages** sur le modèle relationnel

- Permet de stocker plusieurs types de relations en même temps
- Traversé du graphe sans devoir recalculer les relations

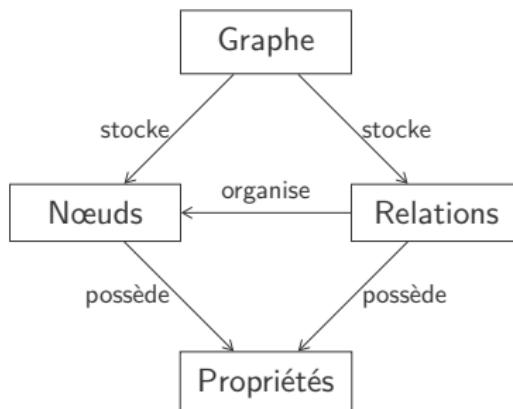
Modèle de données (2)

- Plusieurs relations entre les éléments stockés

Que l'on peut représenter avec un graphe

- Il n'y a jamais de liens morts

Impossible de supprimer un nœud qui intervient dans des relations



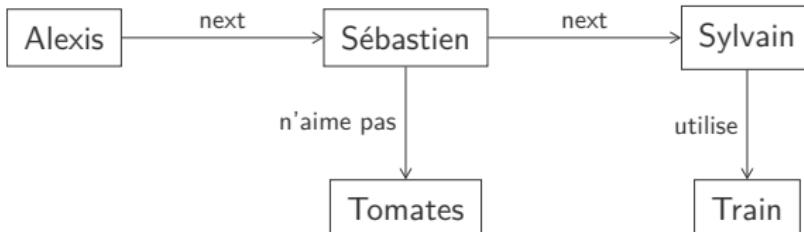
Relation

- Traversée des relations très rapide en orienté-graphe

Relations stockées comme telles, doivent pas être construites

- Plusieurs types de relation possibles

- Relations entre éléments de domaines différents
- Relation secondaire indiquant une catégorie, un chemin...
- Quad-tree pour l'indexation ou liste chainée pour l'accès trié



Gremlin

- Langage pour décrire des **traversée de graphes**

Fait partie du projet Apache TinkerPop™

- Utilisé avec **deux types** de bases de données
 - Bases orientée-graphe (OLTP) : OrientDB...
 - Manipulation de graphes (OLAP) : Apache Giraph, Hadoop...

```
1 g.V().has("name", "gremlin").  
2   out("knows").  
3   out("knows").  
4     values("name")
```

Cas d'utilisation

- Tout domaine qui est très **riche en liens** entre des entités
Réseaux sociaux : amitié, employé/employeur, compétences...
- Routage/dispatching et services dépendants d'une **localisation**
Lieux en nœuds et relations contiennent distance
- Moteur de **recommandations** automatiques
« Tes amis ont aussi acheté cela »...

Cas de non utilisation

- Opération de **mise à jour des nœuds** pas immédiate
Modification de propriétés des nœuds peut coûter très cher
- **Opérations globales** sur le graphe peuvent être très couteuses
En fonction de la base de données choisie

Web sémantique (1)

- Extension du Web pour l'utilisation de format de données

Format de base RDF (Resource Description Framework)

- Permettre le partage des données entre applications

Web des données pour lier et structurer l'information

- Références à des schémas de vocabulaire et concepts

Ajout d'un sens aux informations déjà présentes sur le web



Web sémantique (2)

- **Sémantique** du texte « Paul Schuster est né à Dresde. »

Ajout de propriétés aux balises div et span

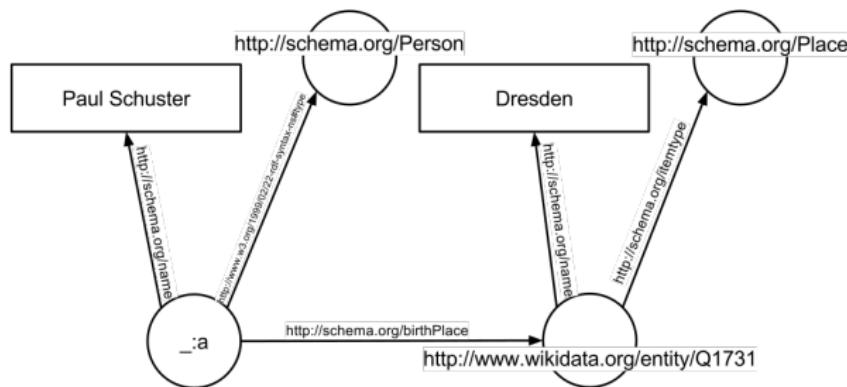
- Utilisation de la **syntaxe RDFa** pour décrire un mini-graphe

Vocabulaire provenant de Schema.org et Wikidata

```
<div vocab="http://schema.org/" typeof="Person">
  <span property="name">Paul Schuster</span> est né à
  <span property="birthPlace" typeof="Place" href="http://www.
  wikidata.org/entity/Q1731">
    <span property="name">Dresde</span>.
  </span>
</div>
```

Web sémantique (3)

```
<div vocab="http://schema.org/" typeof="Person">
  <span property="name">Paul Schuster</span> est né à
  <span property="birthPlace" typeof="Place" href="http://www.
  wikidata.org/entity/Q1731">
    <span property="name">Dresde</span>.
  </span>
</div>
```



SPARQL

- Langage de requêtes **SPARQL Protocol and RDF Query Language**
Adapté à la structure spécifique des graphes RDF
- **Deux types** de requêtes dans SPARQL
 - SELECT extrait un sous-graphe d'un graphe RDF
 - CONSTRUCT engendre un nouveau graphe complétant un autre

Apache Jena

- Framework open source pour le Web sémantique

Développé par la fondation Apache et avec Java

- API pour extraire des données et construire des graphes RDF
 - Interrogation des modèles à l'aide de SPARQL*
- Fournit du support pour OWL (Web Ontology Language)



Neo4j



BOSTON 2013

INNOVATE. SHARE. CONNECT.

{graphs}-[:ARE]->{everywhere}

Neo4j

the world's leading graph database
Neo4j



Neo4j

- Système de **gestion de graphes** par Neo Technology, Inc.

Basé à San Francisco, CA, USA et Malmö en Suède

- **Transactions ACID** avec stockage natif des graphes

Le plus populaire dans les bases orientées-graphe

- Interrogation de la base avec un **langage à travers HTTP**

Cypher Query Language (CQL)

Modèle de données (1)

- Le **nœud** est l'unité fondamentale qui constitue un graphe
Peut être associé à des labels et peut posséder des propriétés
- Une **relation** connecte deux nœuds avec une direction
Peut également posséder des propriétés
- **Propriétés** pour les nœuds et relations comme paires clé-valeur
Valeurs de type nombre, chaîne de caractères, booléen et listes
- Créer des ensembles de nœuds à l'aide de **labels**
Un nœud peut posséder plusieurs labels

Modèle de données (2)

- Une **traversée** d'un graphe répond à une requête
 - Navigation de nœud en nœud
 - Interrogation avec langage déclaratif CQL
- Un **chemin** est une séquence de nœuds avec des relations

Utilisé comme résultat d'une requête

Cypher Query Language (CQL)

- Interrogation d'une base Neo4j avec le langage **Cypher**

Langage très simple et très puissant

- **Deux clauses** principales pour construire des requêtes

- CREATE permet de créer une nouvelle entité
- MATCH permet de chercher des entités

- **Identification graphique** des entités

- Nœud représenté entre parenthèses et deux-points pour label
- Relation représentée avec flèches et crochets pour détails
- Propriétés représentées par un dictionnaire à la JSON

Installation de Neo4j

- Neo4j est un programme développé **en Java et Scala**
- **Plusieurs programmes** proposés après installation
 - neo4j permet de lancer plusieurs commandes de gestion
 - neo4j start démarre le serveur
 - neo4j stop arrête le serveur
 - neo4j status récupère le statut du serveur
 - neo4j-shell propose un client en ligne de commande

Lancement du serveur

■ Lancement du serveur et vérification de la connexion

Le démarrage de Neo4j lance également un serveur web

```
& neo4j start
```

```
& neo4j-shell
Welcome to the Neo4j Shell! Enter 'help' for a list of commands
NOTE: Remote Neo4j graph database service 'shell' at port 1337

neo4j-sh (?)$
```

Browser Neo4j

The screenshot shows the Neo4j Browser application window. On the left is a vertical toolbar with icons for file operations (New, Open, Save, etc.), database management (Cloud, Settings, Help), and search. The main area has a header bar with a dollar sign icon and three circular buttons (star, close, refresh). A blue banner at the top states: "Database access not available. Please use :server connect to establish connection. There's a graph waiting for you." Below this is a command input field containing ":server connect". The central part of the screen is titled "Connect to Neo4j" and contains a message: "Database access requires an authenticated connection." It features two input fields: "Username" (containing "neo4j") and "Password" (empty). Below these is a note: "Default username/password: neo4j/neo4j" and a "Connect" button.

\$

Database access not available. Please use `:server connect` to establish connection. There's a graph waiting for you.

:server connect

Connect to Neo4j

Database access requires an authenticated connection.

Username

neo4j

Password

Default username/password: **neo4j/neo4j**

Connect

Création d'un nœud

- Création d'un **nouveau nœud** avec la commande CREATE
Spécification optionnelle de propriétés

```
neo4j-sh (?)$ CREATE (:Person {firstname: "Alexis"});  
+-----+  
| No data returned. |  
+-----+  
Nodes created: 1  
Properties set: 1  
Labels added: 1  
557 ms  
neo4j-sh (?)$
```

Chercher un nœud

- Recherche d'un nœud dans le graphe avec MATCH

Spécification d'un filtre pour les nœuds désirés

- Exemple de recherche des deux nœuds déjà créés

Stockage des résultats dans les variables p et f

```
neo4j-sh (?)$ MATCH (p:Person)
> MATCH (f:Food)
> RETURN p, f;
+-----+
| p                         | f                         |
+-----+
| Node[0]{firstname:"Alexis"} | Node[1]{name:"Tomato"}   |
+-----+
1 row
50 ms

neo4j-sh (?)$
```

Ajouter une relation

- Ajout d'une relation dans le graphe avec CREATE

Recherche éventuelle des nœuds avant

```
neo4j-sh (?)$ MATCH (alexis:Person {firstname: "Alexis"})  
> MATCH (tomato:Food {name: "Tomato"})  
> CREATE (alexis) -[r:LIKES]-> (tomato)  
> RETURN r;  
+-----+  
| r |  
+-----+  
| :LIKES [0] {} |  
+-----+  
1 row  
Relationships created: 1  
321 ms  
  
neo4j-sh (?)$
```

Chercher une relation

- Recherche d'une relation dans le graphe avec MATCH

Variables du pattern de recherche seront remplies

```
neo4j-sh (?)$ MATCH (p:Person) -[LIKES]-> (Food {name: "Tomato"})
> RETURN p;
+-----+
| p
+-----+
| Node [0]{firstname:"Alexis"} |
+-----+
1 row
90 ms

neo4j-sh (?)$
```

Visualisation avec le browser

Database Information

Node labels
Food Person

Relationship types
LIKES

Property keys
firstname name

Database
Version: 3.0.6
Name: graph.db
Size: 140.64 kB
Information [sysinfo](#)

```
$ MATCH (n) RETURN n LIMIT 25
```

*(2) Food(1) Person(1)
*(1) LIKES(1)

Graph Rows Text Code

Displaying 2 nodes, 1 relationship (completed with 1 additional relationship).

AUTO-COMPLETE

Modification d'un nœud

- Ajout/modification de propriétés avec la commande SET

Récupération avec une requête MATCH puis modification

- Suppression d'une propriété en spécifiant NULL comme valeur

```
neo4j-sh (?)$ MATCH (p:Person {firstname: "Alexis"})  
> SET p.sex = "M"  
> SET p.age = 22  
> RETURN p;  
+-----+  
| p |  
+-----+  
| Node [0]{firstname:"Alexis",sex:"M",age:22} |  
+-----+  
1 row  
Properties set: 2  
202 ms  
  
neo4j-sh (?)$
```

Filtrage des résultats

- Définition de **filtre de recherche** avec WHERE

En spécifiant plusieurs conditions sur les entités récupérées

```
neo4j-sh (?)$ CREATE (:Person {firstname: "Damien", age: 24, sex: "M"});  
[...]  
  
neo4j-sh (?)$ MATCH (p:Person)  
> WHERE p.age>22  
> RETURN p.firstname;  
+-----+  
| p.firstname |  
+-----+  
| "Damien"    |  
+-----+  
1 row  
333 ms  
  
neo4j-sh (?)$
```

Contrainte (1)

- Forcer l'**intégrité des données** à l'aide de contraintes

Unicité des nœuds et des propriétés

- Définition d'une **nouvelle contrainte** avec CONSTRAINT

```
neo4j-sh (?)$ CREATE CONSTRAINT ON (p:Person)
> ASSERT p.firstname IS UNIQUE;
+-----+
| No data returned. |
+-----+
Unique constraints added: 1
641 ms

neo4j-sh (?)$
```

Contrainte (2)

- Contraintes **vérifiées** lors d'ajout ou modification des nœuds

L'ajout d'une contrainte peut échouer s'il y a déjà des conflits

```
neo4j-sh (?)$ CREATE (:Person {firstname: "Damien"});
113 ms

WARNING: Node 3 already exists with label Person and property "
firstname"=[Damien]

neo4j-sh (?)$ CREATE CONSTRAINT ON (p:Person)
> ASSERT p.sex IS UNIQUE;
260 ms

WARNING: Unable to create CONSTRAINT ON ( person:Person ) ASSERT
person.sex IS UNIQUE:
Multiple nodes with label 'Person' have property 'sex' = 'M':
  node(0)
  node(5)

neo4j-sh (?)$
```

Requête complexe (1)

\$ MATCH (n) RETURN n LIMIT 25

Graph Rows Text Code

*(8) Book(1) Course(1) Food(1) Language(1) Person(4)

*(13) AUTHORED(2) BOUGHT(1) DOESNT_LIKE(2) KNOWS(1) LIKES(2) SPEAKS(1) TAKES(1) TAUGHT_IN(1)

```
graph TD; Sébastien -- AUTHORED --> Bases["Bases des maths pour le"]; Sébastien -- AUTHORED --> Alexis; Sébastien -- LIKES --> Virginie; Sébastien -- LIKES --> Tomato; Sébastien -- DOESNT_LIKE --> Tomato; Sébastien -- TEACHES --> NoSQL; Sébastien -- TEACHES --> Damien; Virginie -- AUTHORED --> Bases; Virginie -- LIKES --> Tomato; Virginie -- TEACHES --> NoSQL; Tomato -- LIKES --> Damien; Tomato -- KNOWS --> Alexis; Damien -- SPEAKS --> Français; Damien -- TAKES --> NoSQL; Damien -- TAUGHT_IN --> Français
```

Displaying 8 nodes, 13 relationships (completed with 13 additional relationships).

AUTO-COMPLETE ON

Requête complexe (2)

- Récupérer **tous les auteurs** d'un bouquin donné

On fixe la relation et le nœud d'arrivée

```
neo4j-sh (?)$ MATCH (p:Person) -[r:AUTHORED]-> (b:Book {title: "Bases des maths pour le supérieur"}) RETURN p.firstname;
+-----+
| p.firstname |
+-----+
| "Sébastien" |
| "Alexis"     |
+-----+
2 rows
14 ms

neo4j-sh (?)$
```

Requête complexe (3)

- Quelqu'un qui **n'aime pas** une personne qui en **aime** une autre

Un enchainement de deux relations est à préciser dans la requête

```
neo4j-sh (?)$ MATCH (a:Person) -[r:DOESNT_LIKE]-> (b:Person) -[s:LIKES]-> (c:Person) RETURN a.firstname, b.firstname, c.firstname;  
+-----+  
| a.firstname | b.firstname | c.firstname |  
+-----+  
| "Alexis"    | "Sébastien"  | "Virginie"   |  
+-----+  
1 row  
45 ms  
  
neo4j-sh (?)$
```



Requête complexe (4)

- Quelqu'un qui **n'aime pas** et **aime** deux choses différentes

Deux relations sont à préciser dans la requête

```
neo4j-sh (?)$ MATCH (p:Person) -[r:LIKES]-> (a) MATCH (p) -[s:DOESNT_LIKE]-> (b) RETURN p.firstname, a, b;
+-----+
| p.firstname | a                         | b
+-----+
| "Alexis"    | Node [1]{name :"Tomato"}      | Node [2]{firstname
:"Sébastien"} |
| "Sébastien" | Node [4]{firstname :"Virginie"} | Node [1]{name :"Tomato"} |
+-----+
2 rows
14 ms

neo4j-sh (?)$
```

Requête complexe (5)

```
$ MATCH (p:Person) -[r:LIKES]-> (a) MATCH (p) -[s:DOESNT_LIKE]-> (b) RETURN p, r, s;
```

*(4) Food(1) Person(3)

*(4) DOESNT_LIKE(2) LIKES(2)

Graph

Rows

A Text

</> Code

```
graph TD; Sébastien -- LIKES --> Virginie; Sébastien -- DOESNT_LIKE --> Tomato; Tomato -- LIKES --> Alexis; Alexis -- DOESNT_LIKE --> Tomato
```

Displaying 4 nodes, 4 relationships.

AUTO-COMPLETE

Module Python neo4j

■ Module Python neo4j pour interroger la base

Connexion au serveur et ouverture d'une session

```
1 from neo4j.v1 import GraphDatabase, basic_auth  
2  
3 driver = GraphDatabase.driver("bolt://localhost", auth=basic_auth("neo4j", "neo4j"))  
4 session = driver.session()  
5  
6 print(session)  
7 session.close()
```

```
<neo4j.v1.Session object at 0x105475550>
```

Exécution d'une requête

- Utilisation de la **méthode run** sur la session

Exécuter une requête CQL et récupérer un objet Record

```
1 result = session.run('MATCH (p:Person) RETURN p.firstname AS
2   firstname')
3 for record in result:
4     print(record)
5     print(record['firstname'])
```

```
<Record firstname='Alexis'>
Alexis
<Record firstname='Damien'>
Damien
```

Requête paramétrée

- Une **requête paramétrée** peut être exécutée plusieurs fois

Valeurs fournies avec le second paramètre de la méthode run

```
1 request = 'MATCH (p:Person {firstname: {name}}) RETURN p'
2 people = ['Alexis', 'Sylvain', 'Damien']
3 for p in people:
4     result = session.run(request, {'name': p})
5     print(p, end=' : ')
6     try:
7         p = result.single()
8         print(p['p'].properties['age'], 'ans.')
9     except ResultError:
10        print('pas trouvé.')
```

```
Alexis : 22 ans.
Sylvain : pas trouvé.
Damien : 24 ans.
```

OrientDB

40

Ma-2100

ORIENT

Conjunt Històric



Serra de Tramuntana
Patrimoni de la Humanitat



OrientDB

- Base de données **multi-domaines**

Supporte les graphes, documents, clé-valeur et objets

- Les **relations** entre entités sont gérées comme avec les graphes

Supporte les requêtes de type Gremlin

- **Transactions ACID** supportées par le moteur d'OrientDB

Et tolérance aux pannes à l'aide de caches

Modèle de données (1)

- Notion de **classes** pour représenter des enregistrements

Notion similaire aux tables du modèle relationnel

- Une classe contient **plusieurs éléments**

- Une propriété définit une colonne de la classe
- Des contraintes peuvent être ajoutées aux propriétés
- Une classe stocke des enregistrements

- Partition des classes en **clusters**

Permet notamment de répartir les données physiquement

Modèle de données (2)

- Localisation d'un **enregistrement** au sein d'un cluster

#cluster_id :cluster_position

- Établissement de **relations** entre les classes

Similaire au modèle relationnel, relation de type LINK

- Possibilité de stocker des **graphes** avec des classes particulières

Deux classes V et E toujours présentes à étendre

Relation

- Stockage du **record id** de la cible dans la source

Comme un pointeur entre deux objets en mémoire

- Plusieurs **types de relation**

- LINK pointe vers un objet
- LINKSET pointe vers plusieurs objets (ensemble)
- LINKLIST pointe vers plusieurs objets (liste)
- LINKMAP pointe vers plusieurs objets (dictionnaire)

Installation de OrientDB

- OrientDB est un programme développé **en Java**
- **Plusieurs programmes** proposés après installation
 - orientdb permet de lancer plusieurs commandes de gestion
 - neo4j start démarre le serveur
 - neo4j stop arrête le serveur
 - neo4j status récupère le statut du serveur
 - orientdb-console propose un client en ligne de commande

Lancement du serveur

■ Lancement du serveur et vérification de la connexion

Indication immédiate de si un serveur a été trouvé

```
& orientdb start
```

```
& orientdb-console
```

```
OrientDB console v.2.2.5 (build 2.2.  
x@r393af9c5a3e4a4408440a9376283a26d2d3d3c7b; 2016-07-20  
06:03:46+0000) www.orientdb.com  
Type 'help' to display all the supported commands.  
Installing extensions for GREMLIN language v.2.6.0
```

```
orientdb>
```

Configuration d'un utilisateur

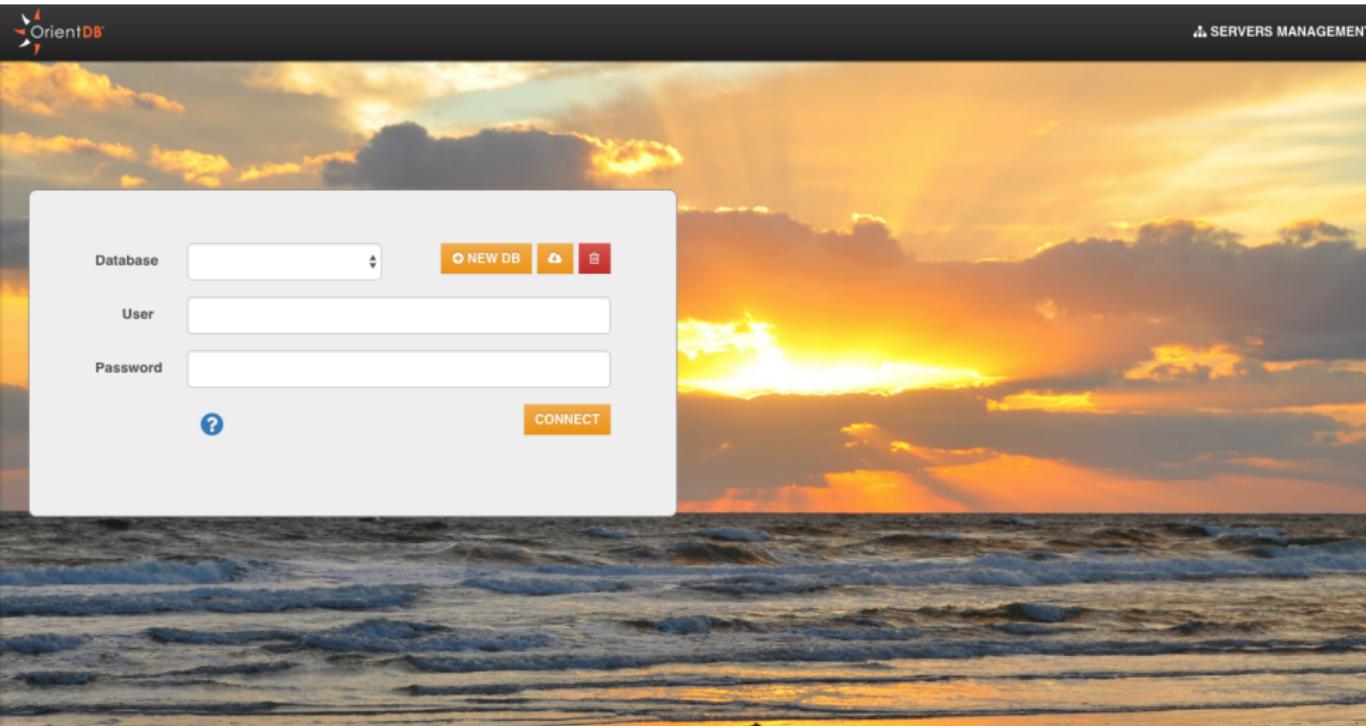
- Ajout d'un utilisateur dans orientdb-server-config.xml

Puis création d'une connexion avec la commande CONNECT

```
<user resources="*" password="admin" name="admin"/>
```

```
orientdb> CONNECT remote:localhost admin admin
Connecting to remote Server instance [remote:localhost] with user
'admin'...OK
orientdb {server=remote:localhost/}>
```

Browser OrientDB



Connexion à une base de données

- Connexion à une base de données avec la commande CONNECT

Déconnexion du serveur pour connexion à la base de données

```
orientdb {server=remote:localhost/}> LIST DATABASES
Found 1 databases:
* myschool (plocal)

orientdb {server=remote:localhost/}> CONNECT REMOTE:localhost/
myschool admin admin
Disconnecting from remote server [remote:localhost/]...
OK
Connecting to database [REMOTE:localhost/myschool] with user 'admin'...OK

orientdb {db=myschool}>
```

Création d'une classe

- Création d'une **nouvelle classe** avec CREATE CLASS

Ensuite ajout éventuel de propriété avec CREATE PROPERTY

- **Propriétés pas obligatoires**, sauf pour index ou contraintes

OrientDB travaille en mode schéma-mixte

```
orientdb {db=myschool}> CREATE CLASS student
Class created successfully. Total classes in database now: 12.
orientdb {db=myschool}> CREATE PROPERTY student.firstname STRING
Property created successfully with id=1.
```

Information sur une classe

■ Récupération d'**information sur une classe** avec INFO CLASS

Information sur la distribution et sur les propriétés et contraintes

```
orientdb {db=myschool}> INFO CLASS student

CLASS 'student'

Records.....: 0
Default cluster.....: student (id=21)
Supported clusters...: student(21), student_1(22), student_2(23),
                      student_3(24)
Cluster selection....: round-robin
Oversize.....: 0.0

PROPERTIES
+-----+-----+-----+-----+-----+-----+
| #   | NAME      | TYPE     | LINKED-TYPE/CLASS | MANDATORY | READONLY | NOT-N
+-----+-----+-----+-----+-----+-----+
| 0   | firstname | STRING   |                   | false     | false    | false
+-----+-----+-----+-----+-----+-----+-----+-----+
```

orientdb {db=myschool}>

Ajout d'un enregistrement

- Ajout d'un **nouvel enregistrement** avec INSERT INTO

Information sur la distribution et sur les propriétés et contraintes

- L'enregistrement est automatiquement placé **sur un cluster**

Possibilité de spécifier le cluster sur lequel le placer

```
orientdb {db=myschool}> INSERT INTO student SET firstname='Alexis', age=22
Inserted record 'student#21:0{firstname:Alexis,age:22} v1' in
0,090000 sec(s).

orientdb {db=myschool}>
```

Récupérer un enregistrement

- ## ■ Récupérer un enregistrement avec DISPLAY RECORD

Possibilité d'utiliser *LOAD RECORD #21:0* (*plus global*)

```
orientdb {db=myschool}> BROWSE CLASS student
+-----+-----+-----+-----+
| #   | @RID  | @CLASS | firstname | age |
+-----+-----+-----+-----+
| 0   | #21:0 | student | Alexis    | 22   |
+-----+-----+-----+-----+

orientdb {db=myschool}> DISPLAY RECORD 0

DOCUMENT @class:student @rid:#21:0 @version:1
+-----+-----+-----+
| #   | NAME      | VALUE   |
+-----+-----+-----+
| 0   | firstname | Alexis  |
| 1   | age       | 22      |
+-----+-----+-----+-----+
```

Création d'un graphe (1)

- Création de classes **de type nœud et arêtes**

Une classe peut étendre une autre avec la clause EXTENDS

```
orientdb {db=social}> CREATE CLASS person EXTENDS V  
Class created successfully. Total classes in database now: 12.  
  
orientdb {db=social}> CREATE CLASS food EXTENDS V  
Class created successfully. Total classes in database now: 13.  
  
orientdb {db=social}> CREATE CLASS likes EXTENDS E  
Class created successfully. Total classes in database now: 14.  
  
orientdb {db=social}>
```

Création d'un graphe (2)

■ Ajout de nœud et arêtes avec CREATE VERTEX/EDGE

Même syntaxe que l'ajout d'enregistrements

```
orientdb {db=social}> CREATE VERTEX person SET firstname="Alexis"  
Created vertex 'person#21:0{firstname:Alexis} v1' in 0,002000 sec  
(s).
```

```
orientdb {db=social}> CREATE VERTEX food SET name="Tomato"  
Created vertex 'food#25:0{name:Tomato} v1' in 0,003000 sec(s).
```

```
orientdb {db=social}> CREATE EDGE likes FROM (SELECT FROM person  
WHERE firstname="Alexis") TO (SELECT FROM food WHERE name="Tomato")  
Created edge '[likes#29:0{out:#21:0,in:#25:0} v1]' in 0,148000 sec(s).
```

```
orientdb {db=social}>
```

Interrogation d'un graphe

- Récupération des **arêtes adjacentes** avec IN/OUT/BOTH

Transformation des identifiants en enregistrements avec EXPAND

```
orientdb {db=social}> SELECT OUT() FROM person WHERE firstname="Alexis"
```

```
+----+-----+
|#   |OUT    |
+----+-----+
|0   |[#25:0]|
+----+-----+
```

```
1 item(s) found. Query executed in 0.034 sec(s).
```

```
orientdb {db=social}> SELECT EXPAND(OUT()) FROM person WHERE
firstname="Alexis"
```

```
+----+-----+-----+-----+
|#   |@RID  |@CLASS|name   |in_likes|
+----+-----+-----+-----+
|0   |#25:0|food   |Tomato |[#29:0] |
+----+-----+-----+-----+
```

```
1 item(s) found. Query executed in 0.004 sec(s).
```

Module Python pyorient

■ Module Python **pyorient** pour interroger la base

Connexion au serveur et authentification d'un utilisateur

```
1 import pyorient  
2  
3 client = pyorient.OrientDB('localhost', 2424)  
4 session = client.connect('admin', 'admin')  
5  
6 print(client.db_list())
```

```
{'databases': {'myschool': 'plocal:/usr/local/var/db/orientdb/  
myschool', 'social': 'plocal:/usr/local/var/db/orientdb/social  
'}}}
```

Exécution d'une requête

- Utilisation de la **méthode command** sur la session

Après avoir ouvert une base de données avec open

```
1 import pyorient
2
3 client = pyorient.OrientDB('localhost', 2424)
4 session = client.db_open('myschool', 'admin', 'admin')
5
6 result = client.query('SELECT FROM student')
7 for student in result:
8     print(student)
9     print(student.firstname)
```

```
{'@student': {'firstname': 'Alexis', 'age': 22}, 'version': 1, 'rid': '#21:0'}
Alexis
```

Crédits

- Photos des logos depuis Wikipédia
- <https://www.flickr.com/photos/jennifer-stylls/8012538039>
- <https://fr.wikipedia.org/wiki/Fichier:Sw-horz-w3c-v.svg>
- https://fr.wikipedia.org/wiki/Fichier:RDF_example.svg
- <https://www.flickr.com/photos/neotechnology/9024811631>
- <https://www.flickr.com/photos/michaeljohnbutton/10049404576>