

# M1Info



---

UNIVERSITÉ PARIS 8 - VINCENNES À SAINT-DENIS

**M1 MIASHS : Big Data et fouille de données**

**Projet sur les technologies NoSQL : OrientDB**

**PANCHALINGAMOORTHY GAJENTHRAN**

Organisme d'accueil : Université Paris 8  
Sujet : Projet sur les technologies NoSQL

# Table des matières

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	NoSQL et SQL . . . . .	1
1.2	Types de base de données NoSQL . . . . .	2
1.3	Modèle orientées graphes . . . . .	3
<b>2</b>	<b>Présentation d'OrientDB</b>	<b>5</b>
<b>3</b>	<b>Installation d'OrientDB</b>	<b>7</b>
<b>4</b>	<b>Fonctionnalités d'OrientDB</b>	<b>10</b>
4.1	Les fichiers importés/exportés . . . . .	10
4.2	Modèle orienté graphe . . . . .	11
4.3	Opérations CRUD . . . . .	11
4.4	Schema Manager . . . . .	12
4.5	Security Manager . . . . .	13
4.6	Transactions ACID . . . . .	14
4.7	Autres fonctionnalités . . . . .	14
<b>5</b>	<b>Comparaison avec MongoDB</b>	<b>16</b>
5.1	Modèle utilisé . . . . .	16
5.2	Langage de requête . . . . .	16
5.3	Relation entre les données . . . . .	17
5.4	Transactions ACID . . . . .	17
5.5	Indexation . . . . .	18
5.6	Méthodes de stockage . . . . .	18
5.7	Drivers . . . . .	18
<b>6</b>	<b>Conclusion</b>	<b>20</b>
<b>7</b>	<b>Sources</b>	<b>21</b>

# Chapitre 1

## Introduction

NoSQL, souvent expliqué comme étant l'acronyme de "Not Only SQL" ("pas seulement SQL" en anglais), correspond à une famille de système de gestion de base de données (SGBD) qui vont servir à gérer un très grand ensemble de données distribuées. Né depuis les années 1960, NoSQL a notamment pris de l'ampleur dans les années 2000 avec l'émergence de MongoDB, Cassandra... et le développement de grandes entreprises telles que Google ou encore Amazon. En effet, ces entreprises avec des bases de données de plus en plus volumineuses ont commencé à souffrir des limites des SGBD relationnels connues pour leur rigueur et leur rigidité mais également leurs problèmes d'extensibilité.

### 1.1 NoSQL et SQL

Bien que le nom "NoSQL" peut sous entendre le remplacement de SQL, les bases de données NoSQL n'ont pas pour but d'effacer ou de remplacer les bases de données SQL. En effet, il s'agit d'une alternative à ces bases, dans le sens où les deux vont stocker les données avec des approches différentes. De plus, nous ne pouvons pas dire si NoSQL est meilleur que SQL ou inversement, cela dépend des cas de figure, dans certains cas NoSQL sera plus approprié que SQL.

Les principales différences à noter concernent la manière dont les données sont stockées, avec SQL qui propose l'utilisation de tables tandis que NoSQL opte plutôt pour des collections d'objets (des documents). NoSQL est plus flexible car SQL demande la création d'un schéma (table et le type des champs) pour ajouter des données là où NoSQL ne nécessite pas spécifier de document. Cette flexibilité offerte par NoSQL peut s'avérer comme un atout en facilitant la manipulation mais cela peut causer des problèmes de cohérences au niveau de la

base de données. Par ailleurs, la jointure correspond à la principale différence entre les deux modèles. Les requêtes SQL s'appuient sur des JOIN afin de créer des relations entre plusieurs tables, l'essence même des SGBD relationnels. Cependant, NoSQL ne possède pas d'instruction de la sorte pour relier les tables, d'où l'importance de la dénormalisation. A préciser que certains de ces SGBD se rapprochent des requêtes SQL comme CouchBase et peuvent donc réaliser des jointures.

## 1.2 Types de base de données NoSQL

Ainsi, pour gérer cette importance des données, NoSQL est né. Selon IBM Marketing Cloud, 90% des données dans le monde d'aujourd'hui ont été créées au cours des deux dernières années. Devant cette montée en volume des données, NoSQL proposent plusieurs types de base de données :

- **Clé-Valeurs** : Les données sont stockées sous forme de paires clé-valeur dans lesquelles la clé est un identifiant unique et la valeur peut représenter un quelconque objet. Cela permettra d'avoir un temps d'exécution de la requête très faible et rendre la base extensible. Cependant les mises à jours peuvent paraître difficiles, dès lors où la base commence à être volumineuse.
- **Documents** : Les données sont stockées sous forme de documents c'est-à-dire sous forme de données semi-structurées. Cela permettra d'avoir une flexibilité dans la gestion des données, et également une base plus maintenable. Mais cette base engendre une duplication et une incohérence au niveau des données, s'écartant donc des SGBD relationnels
- **Orientée colonnes** : Les données sont stockées par colonne et non pas par ligne, ce qui va aider la base à avoir des colonnes variables. Cela permettra d'accéder plus rapidement aux données mais limite les possibilités de requêtage.
- **Graphes** : Les données sont représentées sous la forme d'un graphe où les noeuds représentent les données et leurs propriétés, et les arcs orientés qui créent les relations entre les noeuds. Cela permettra d'avoir une architecture plus modelable et surtout s'adapte bien aux données relationnelles.

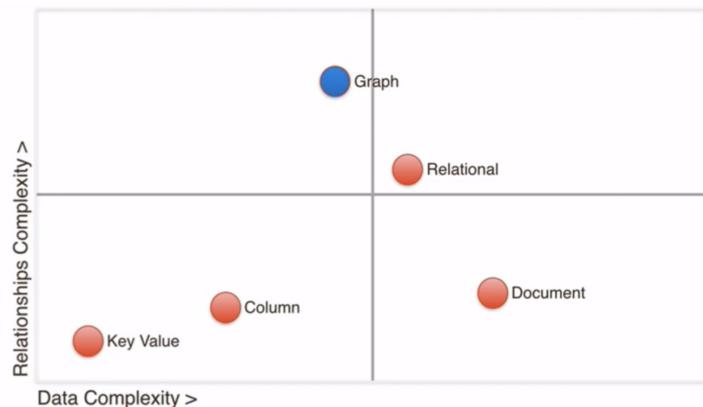


FIGURE 1.1 – Graphe représentant la complexité des modèles

### 1.3 Modèle orientées graphes

Après avoir décrit une grande partie des catégories de base de données dans **NoSQL**, nous nous demanderons lesquelles des types sont les plus adéquats pour avoir un modèle se rapprochant le plus des SGBD relationnels. En effet, dans les bases de données, le plus important ne concerne pas les données mais plutôt les relations entre elles. L'avantage des graphes est le temps des opérations qui est constant, ne dépend pas de la taille de la base ; contrairement aux jointures **SQL** qui nécessitent plusieurs étapes si les dimensions de la base sont trop élevées et qui est donc de la forme  $O(\text{Log}N)$ . En effet, dans un graphe, la relation est créée une seule fois là où nous devons recalculer les relations à chaque requête **JOIN** dans **SQL**.

Cependant, avec **NoSQL**, il n'y a pas de standard entre les produits, ce qui nécessite d'avoir des compétences dans plusieurs domaines. Le schéma ci-dessous illustre les difficultés que peut rencontrer les bases **NoSQL**, ce qui rend la performance et la fiabilité, deux concepts très difficile à prédire.

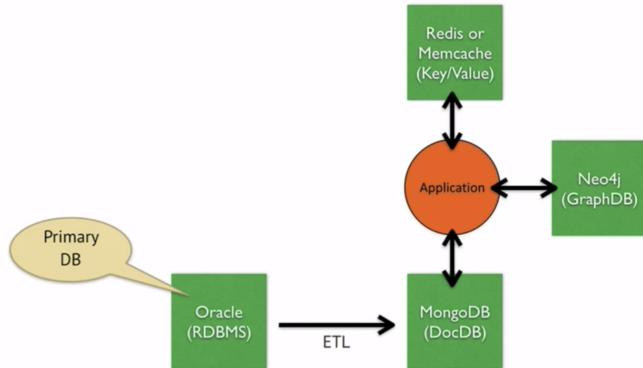


FIGURE 1.2 – Schéma représentant le manque de standard entre les produits NoSQL

La solution à ce problème est de fournir une base de données multi-modèles : c'est effectivement ce que propose OrientDB que nous verrons dans les parties suivantes. Ainsi, nous décrirons le modèle OrientDB, expliquerons l'installation de ce dernier, étudierons ses principales fonctionnalités et enfin nous le comparerons à MongoDB.

# Chapitre 2

## Présentation d'OrientDB

OrientDB est un système de gestion de base de données, open-source, écrit en Java. Il s'agit d'une base de données multi-modèles, supportant ainsi des données orientées graphes, documents, des paires clé/valeur et des modèles objets avec les relations étant gérées comme des bases de données orientées graphes avec des arcs orientés entre les noeuds.

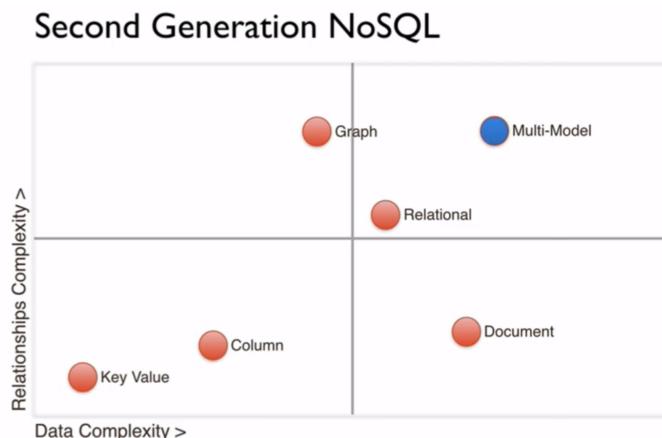


FIGURE 2.1 – Schéma représentant un SGBD multi-modèle

Ce dernier est réputé pour ses performances, en effet, il est capable de stocker 22 000 enregistrements par seconde. Il s'appuie sur plusieurs mécanismes d'indexation se basant sur **B-tree** et **Extendible hashing**. Les enregistrements possèdent une clé (nommé ID ou RID) indiquant la position de l'enregistrement comme valeur unique, ce qui va aider les liens avec les autres enregistrements, et traitant ainsi les enregistrements de manière rapide et efficace, avec une complexité O(1) (parcours, ajouts, suppressions). Cela évite donc les jointures coûteuses en terme de temps d'exécution, car les

connexions entre les enregistrements se déroulent grâce à des pointeurs persistants. Il est possible de supporter des modèles sans schéma, avec schémas, ou encore les schémas mixtes

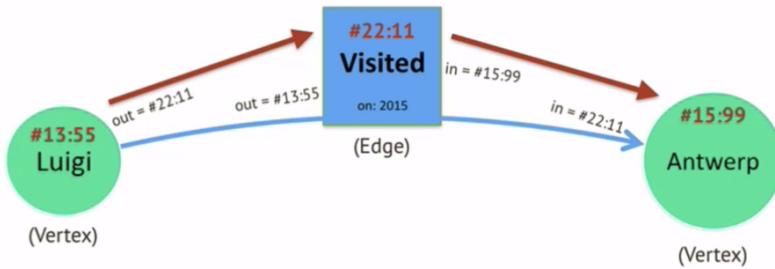


FIGURE 2.2 – Schéma représentant un graphe et les relations entre noeud

En terme de sécurité, celui-ci dispose également d'un système de profilage avec un solide niveau de sécurité basé sur les utilisateurs et prenant en compte le langage SQL parmi les langages de requêtes, ce qui rend le SGBD très intéressant lui permettant de manier les données relationnelles.

## Chapitre 3

# Installation d'OrientDB

L'installation d'OrientDB se fait de façon extrêmement rapide : elle prend moins de 2 minutes pour être réalisée. Le logiciel est écrit en Java et est capable de tourner dans n'importe quelle plateforme sans configuration ni installation. Il suffit de suivre les étapes suivantes :

- Se diriger vers le site officiel d'OrientDB : <https://orientdb.com/>
- Cliquer sur le bouton **Download** pour pouvoir télécharger le logiciel
- Une fois le fichier téléchargé, il faudra commencer par l'extraire
- Aller vers le dossier **bin**
- Exécuter le fichier **server.sh** ou **server.bat** si vous êtes sur Windows.
- Vous serez amener à entrer un mot de passe pour le root (mais cela n'est pas obligatoire)
- Après avoir entré le mot de passe, le serveur est en train de tourner localement. Il faudra aller sur un navigateur puis se diriger vers le port 2480 qui est le port destiné à ce logiciel : <http://localhost:2480/studio/index.html>

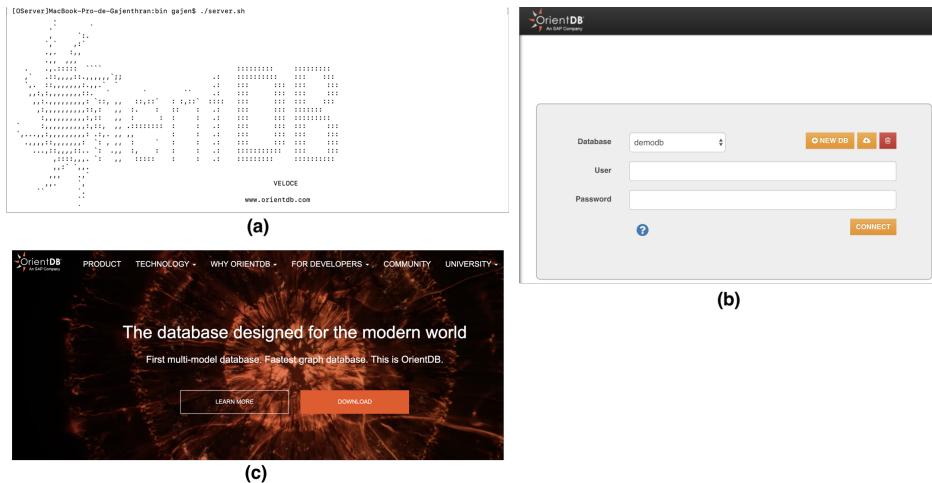


FIGURE 3.1 – (a) Exécution du serveur, (b) gestion de la base de données, (c) téléchargement d'OrientDB

En ce qui concerne les drivers, OrientDB est capable de supporter Native Binary Remote permettant la communication TCP/IP en utilisant le protocole binaire, HTTP REST/JSON permettant la communication TCP/IP en utilisant le protocole HTTP ou encore Java-wrapped pour les langages utilisant JVM comme Scala, Groovy et JRuby.

Voici un tableau dans lequel nous trouverons la liste des drivers permettant l'utilisation d'OrientDB dans d'autres langages.

Language	Name	Type	Description
	Java (native) API	Native	Native implementation.
	JDBC driver	Native	For legacy and reporting/Business Intelligence applications and <a href="#">JCA integration</a> for J2EE containers
	OrientDB Spring Data	Native	Official <a href="#">Spring Data Plugin</a> for both Graph and Document APIs
	OrientJS	Native	Binary protocol, new branch that has been updated with the latest functionality. Tested on 1.7.0, 2.0.x and 2.1-rc*.
	node-orientdb-http	HTTP	RESTful HTTP protocol. Tested on 1.6.1
	Gremlin-Node		To execute Gremlin queries against a remote OrientDB server
	PyOrient	Binary	Community driver for Python, compatible with OrientDB 1.7 and further.
	Bulbflow project	HTTP	Uses <a href="#">Rexter</a> Graph HTTP Server to access to OrientDB database <a href="#">Configure Rexster for OrientDB</a>
	Compass	HTTP	
	Active-Orient	HTTP	Use OrientDB to persistently store dynamic Ruby-Objects and use database queries to manage even very large datasets. The gem is rails 5 compatible.
	OrientDB-JRuby	Native	Through Java driver
	OrientDB Client	Binary	
	OrientDB4R	HTTP	

FIGURE 3.2 – Tableau représentant les drivers supportés par OrientDB

# Chapitre 4

# Principales fonctionnalités d'OrientDB

De nombreuses fonctionnalités sont offertes par OrientDB, c'est pourquoi il est souvent qualifié comme étant un SGBD polyvalent.

Pour cela, nous allons lister les principales fonctionnalités proposées par OrientDB (pour illustrer les propos nous nous aiderons de la base de données dbdemo fournie, par défaut, par OrientDB en nous appuyant sur la table `profiles`.

#### 4.1 Les fichiers importés/exportés

Ce SGBD accepte les fichiers sous format CSV ou encore JSON et renvoie également des fichiers JSON lorsqu'on souhaite exporter la base de données

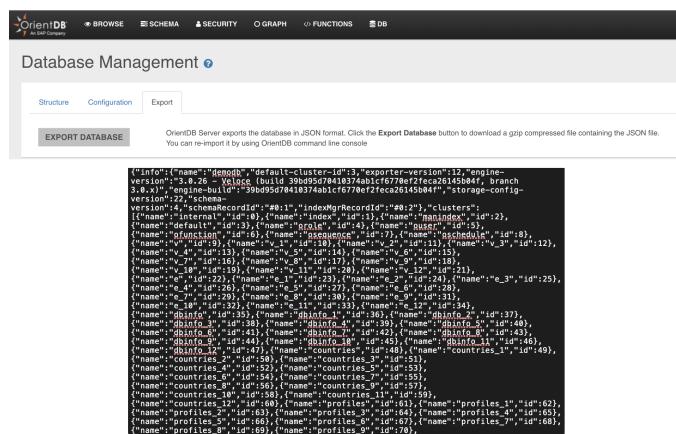


FIGURE 4.1 – Exportation de la base et format des fichiers exportés

## 4.2 Modèle orienté graphe

Ce SGBD met en évidence un modèle de données orienté graphes. Comme expliqué précédemment, le graphe compte sur 2 principes : les noeuds et les arcs qui permettent de relier les noeuds. Ces deux éléments sont représentés par des identifiants uniques permettant de créer des relations sans pour autant avoir de conflits. De plus, chaque élément aura un champ `in` lui permettant de prendre en entrée un ou plusieurs identifiants en entrée et `out` pour donner en sortie un ou plusieurs identifiants vers lequel il pointe.

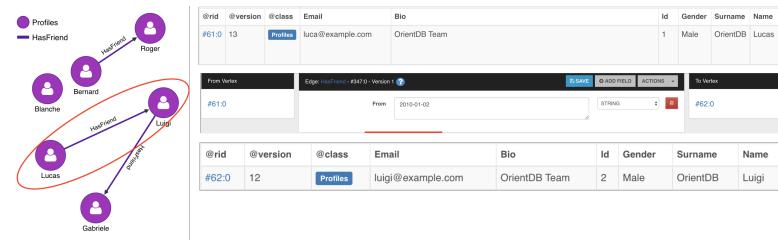


FIGURE 4.2 – Représentation des relations entre les classes

## 4.3 Opérations CRUD

Les opérations CRUD (create, read, update, delete) mais également les autres opérations sont gérées via des requêtes SQL. Les réponses de ces requêtes peuvent être représentées sous plusieurs formes : tables, documents (raws) ou encore graphe.

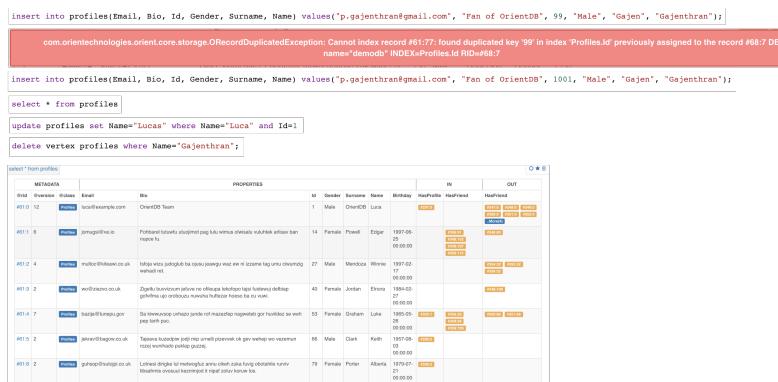


FIGURE 4.3 – Les opérations CRUD et la représentation du tableau

Avant tout, il est important de distinguer les tables d'un arc et d'un noeud car ils n'ont pas la même composition. Le tableau d'un noeud se démarque

par 3 catégories de colonnes : **Metadata** qui correspond aux méta-données à savoir le nom de la classe, **Properties** qui correspond aux propriétés des enregistrements, **in** qui correspond aux arcs en entrée et **out** qui correspond aux arcs en sortie. Les arcs permettent bien évidemment de créer les relations entre les classes. Tandis que celui des arcs ne possèdent que 2 catégories : **Metadata** et **Properties**. Parmi les **Metadata**, nous retrouvons notamment **@rid** qui concerne l'identifiant unique qui permet de reconnaître une enregistrement, **@class** qui concerne la classe à laquelle l'enregistrement appartient et **@version** qui permet de savoir le nombre d'arcs liés à un noeud ; et lorsqu'il s'agit d'un arc, **@version** sera par défaut à 1. **Properties** concernent les propriétés entrées par l'utilisateur pour chaque enregistrement, les **in** et **out** possèderont des classes "arcs" comme sous-colonne avec comme contenu le **@rid** de l'arc. Et si nous cliquons sur celui-ci, nous obtiendrons le noeud pointé par l'arc et pointant vers l'arc.

Par ailleurs, il est également possible de représenter les données sous forme de documents (raws), avec des paires clé-valeurs. Comme les tableaux, nous devons séparer les documents consacrés aux noeuds et aux arcs. A noter que les champs traitant à propos des **Metadata** seront accompagnés du préfixe **@**, les noeuds, reliés par des arcs, comportent le préfixe **in\_** ou **out\_** là où les arcs possèdent deux champs **in** et **out**. Ces champs prennent comme valeur des **@rid**. Quant aux champs destinés aux **Properties**, ils ne possèdent pas de nomenclature particulière. De plus, il existe un champs **fieldTypes** donnant des indications sur le type des champs.

```
delete vertex profiles where Name="Gajenthran";
1 {
2   "result": [
3     {
4       "count": 1
5     }
6   ],
7   "elapsedMs": 70,
8   "notification": "Query executed in 0.098 sec. Returned 1 record(s)"
9 }
```

FIGURE 4.4 – Représentation des requêtes sous forme d'un document

## 4.4 Schema Manager

La rubrique **Schema Manager** est consacrée à la gestion des schémas. Il est donc possible de jeter un oeil et de modifier les propriétés d'une base de données notamment leur type, de supprimer ou de rajouter de nouvelles propriétés. Comme nous l'avons ci-dessous, les enregistrements appartiennent à

des classes. Les classes sont coupées en 3 parties : avec les classes  **noeuds**, les classes  **arcs** et les classes  **génériques**. Les classes sont des concepts provenant du paradigme du langage orienté objet. Les classes peuvent être sans schéma, avec schéma, ou schéma mixte. Surtout, celles-ci peuvent hériter d'autres classes et sont polymorphiques.

Dans l'illustration du tableau représentant les classes, nous pouvons observer une colonne **SuperClasses** qui correspond aux classes mères dont la classe a héritée. Des clusters sont présents pour stocker les enregistrements. En effet, OrientDB se charge de générer plusieurs clusters par classe. Comme nous pouvons le voir dans l'illustration suivante, tous les enregistrements appartenant à la même classe sont stockés dans le même cluster. Par exemple, pour la classe  **profiles**, le cluster par défaut commence à 61 et les enregistrements prendront les valeurs suivantes. L'illustration montre également le nombre d'enregistrement pour cette classe et le mode de sélection du cluster (**default**, **round-robin**, **balanced** et **local**). En apercevant la figure suivante et la requête `select * from profiles;`, nous remarquons que le `@rid` du premier enregistrement commence par 61.

Lors de l'affichage graphique des graphes, il est possible de modifier la couleur des figures afin d'obtenir des schémas plus clairvoyants.

**Vertex Classes**

Name	Color	SuperClasses	Alias	Abstract	Clusters	Default Cluster	Cluster Selection	Records
Profiles	Purple	V			[ 61, 62, 63, 64, 65, 66, 67, 68, ... ]	61	round-robin	1,000

**Edge Classes**

Name	Color	SuperClasses	Alias	Abstract	Clusters	Default Cluster	Cluster Selection	Records
HasFriend	Blue	E			[ 347, 348, 349, 350, 351, 352, ... ]	347	round-robin	1,617
HasProfile	Red	E			[ 295, 296, 297, 298, 299, 300, ... ]	295	round-robin	400

**Generic Classes**

Name	Color	SuperClasses	Alias	Abstract	Clusters	Default Cluster	Cluster Selection	Records
DBInfo	Light Blue				[ 35, 36, 37, 38, 39, 40, 41, 42, ... ]	35	round-robin	1

FIGURE 4.5 – Représentation de la rubrique Schema Manager

## 4.5 Security Manager

Dans la rubrique **Security Manager**, il est possible désormais d'ajouter des utilisateurs avec un système de mot de passe pour s'authentifier, avec la possibilité de lui octroyer un rôle. Selon le rôle attribué, l'utilisateur aura des limitations au niveau de l'accès à certains éléments. En plus de cela, depuis

la version 2.2 d'**OrientDB**, les dossiers sont chiffrés sur le disque, empêchant ainsi les utilisateurs qui ne sont pas autorisés à accéder à la base de données.

The screenshot shows the 'Security Manager' interface with two main sections: 'Users' and 'Roles'. In the 'Users' section, there are three entries: 'admin' (Status: ACTIVE), 'reader' (Status: ACTIVE), and 'writer' (Status: ACTIVE). Each user has 'Edit' and 'Delete' buttons. In the 'Roles' section, there are three entries: 'admin' (Role: 'admin'), 'reader' (Role: 'reader'), and 'writer' (Role: 'writer'). Each role has an 'Add role' button. Below these sections is a 'Permissions' table:

Name	Inherited Role	Mode	Actions
admin	Allow all but	<input checked="" type="checkbox"/>	<b>DELETE</b>
reader	Deny all but	<input checked="" type="checkbox"/>	<b>DELETE</b>
writer	Deny all but	<input checked="" type="checkbox"/>	<b>DELETE</b>

Below the permissions table is another table for specific database rules:

Name	Execute	Delete	Update	Read	Create
database:bypassRestricted	<input checked="" type="checkbox"/>				

FIGURE 4.6 – Représentation de la rubrique Security Manager

## 4.6 Transactions ACID

**OrientDB** supporte les transactions ACID (atomicité, cohérence, isolation, durabilité), qui vont nous permettre d'assurer que les transactions se passent de manière efficace et fiable. C'est un point rare à souligner dans les SGBD NoSQL de proposer des transactions partageant les propriétés ACID, soulignant ainsi qu'**OrientDB** est un logiciel à la fois performant mais également fiable.

## 4.7 Autres fonctionnalités

Deux autres fonctionnalités sont à noter car ils peuvent paraître intéressantes. Il s'agit du téléporteur décrit dans <https://github.com/orientechnologies/teleporter>. Le téléporteur est outil permettant d'importer rapidement et facilement des bases de données relationnelles. En effet, il s'occupe de toutes les sortes de conversions entre les différents SGBD et importe ses données en modèle orienté graphe dans **OrientDB**. Ce dernier a été testé sur Oracle®, SQLServer®, MySQL®, PostgreSQL® et HyperSQL®. Ainsi, cela permet de migrer notre SGBDR vers **OrientDB**, ou encore de synchroniser notre base de données **OrientDB** avec un SGBDR.

La figure suivante montre que l'exécution du téléporteur se déroule en 4 étapes. Tout d'abord, la création du schéma de la base de données, puis la construction du modèle orienté graphe en s'a aidant de la base de données, ensuite l'écriture du schéma **OrientDB** et enfin l'importation des données vers **OrientDB**.

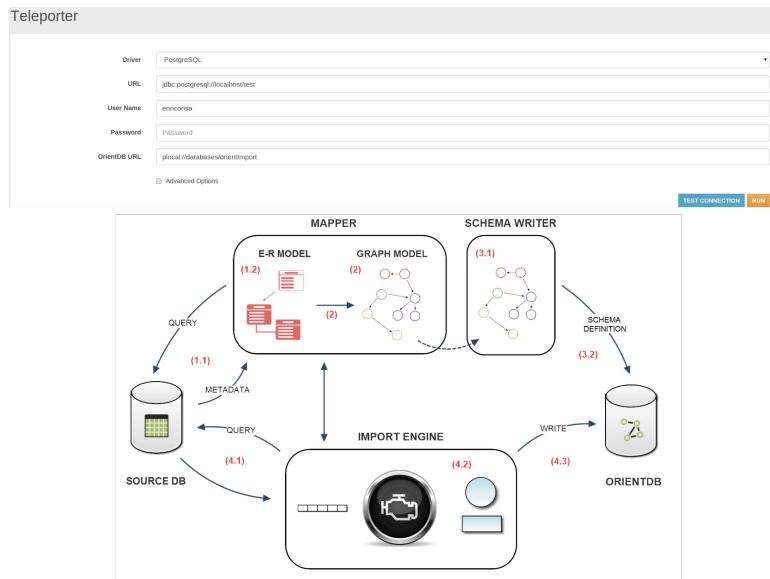


FIGURE 4.7 – Représentation de la fonctionnalité "Teleporteur"

L'autre fonctionnalité qui vaut le coup d'être présenter correspond au Cloud ready permettant au logiciel de s'installer dans le cloud à travers différents fournisseurs tels que Amazon Web Services, Microsoft Azure, Cloud Qwest...

# Chapitre 5

## Comparaison avec MongoDB

`OrientDB` et `MongoDB` sont tous les deux des SGBD `NoSQL` qui comportent de nombreuses similitudes mais présentent également des différences au niveau des fonctionnalités.

### 5.1 Modèle utilisé

Tout d'abord il se différencie par leur modèle ; en effet, `OrientDB` est un système multi-modèle essentiellement basé sur une base de données orientée graphe tandis que `MongoDB` se présente comme un système orienté documents

### 5.2 Langage de requête

Puis, le langage de requête est différent. `MongoDB` possède son propre langage basé sur `JSON` tandis que `OrientDB` est construit principalement sur `SQL` (nous précisons principalement car il s'agit plutôt de `SQL` avec d'autres instructions qui ont été ajoutées afin de manipuler les arbres et les graphes, vu qu'il n'existe pas de `JOIN`). Il est donc plus rapide et/ou plus facile de travailler sur `OrientDB` car `SQL` est un langage très populaire en ce qui concerne les langages de requête là où `MongoDB` peut prendre du temps du fait de la phase d'apprentissage.



FIGURE 5.1 – Comparaison des langages de requêtes MongoDB/OrientDB

Comme expliqué précédemment, il existe d'autres requêtes SQL ajoutée pour gérer les arbres et les graphes. Par exemple, la fonction `both()` permet d'obtenir les noeuds pointants l'arc ou pointés par l'arc. Voici un exemple d'utilisation des nouvelles fonctions proposées par OrientDB, qui demande à la base de rechercher les profils ayant des amis.

```
select expand("hasFriend") from profiles where @rid="#61:0";
```

FIGURE 5.2 – Exemple de requête SQL utilisant les fonctions d'OrientDB

### 5.3 Relation entre les données

Ensuite, le système de relation entre les données est un point essentiel qui rend OrientDB intéressant. Pour créer des relations entre les données, MongoDB comme OrientDB se servent d'ID (nommé respectivement `_id` et `@rid`). Cependant, la différence se fait lors de la recherche : MongoDB a tendance à agir comme un JOIN en intégrant les données directement dans les documents (voir figure ci-dessous), ce qui a pour conséquence d'avoir un coût d'exécution élevé mais également de créer des doublons. OrientDB se contente de connecter les documents de les relier via des arcs et à la fin il assemblera l'intégralité des données en prenant en compte toutes les connexions, ce qui pour intention d'éviter les doublons pour obtenir une base de données plus légère et plus facilement maintenable, et également de rendre la requête plus rapide et efficace car avec une base données plus petite, il est plus facile de gérer efficacement la RAM.

### 5.4 Transactions ACID

De plus, MongoDB ne supporte pas les transactions ACID contrairement à OrientDB qui s'inspire des SGBDR. OrientDB utilise un journal WAL (figure

ci-dessous) (Write ahead Logging) afin de gérer de façon "exhaustive" les manipulations de l'utilisateur, même en cas d'erreur

```
database.begin();
try {
    account.field("amount", amount + 100);
    account.save();
    operation.field("status", "Completed");
    operation.save();
    database.commit();
} catch( Exception e ) {
    database.rollback();
}
```

FIGURE 5.3 – Représentation du journal WAL utilisé par OrientDB

## 5.5 Indexation

Concernant l'indexation, MongoDB utilise l'algorithme **B-tree** pour retrouver les index. Quant à OrientDB, il se fie sur 3 algorithmes : **SB-Tree**, index de hachage et **Lucene**, ce qui permet de rendre les recherches beaucoup plus performantes.

## 5.6 Méthodes de stockage

À propos de la méthode, les méthodes de stockage sur MongoDB sont gérées en utilisant des techniques de Memory Mapping du fait de son rapidité et qu'elles soient gérées par l'OS. Cependant, cela peut poser des soucis lorsque la base de données a besoin de plus espace par rapport à la RAM disponible ; chaque OS gère les Memory mapped de manière différentes. C'est pour cela qu'OrientDB a décidé d'abandonner cette idée et de se focaliser plutôt dans la gestion directe de pages de disques, en essayant de compresser le plus possible les pages dans le but d'offrir le plus d'espace pour la RAM.

## 5.7 Drivers

Cependant, pour le moment, MongoDB possède plus de drivers qu'OrientDB.

Voici les principales différences entre ces deux SGBD, et nous pouvons observer qu'en terme de performance OrientDB est difficilement égalable par

MongoDB. La figure ci-dessous permettra d'avoir un récapitulatif des points communs et des différences entre ces deux SGBD.

Fonctionnalités	OrientDB	MongoDB
Base de données opérationnelle	✓	✓
Base de données orientée documents	✓	✓
Types de données modifiables	✓	✓
Documents imbriqués	✓	✓
Sharding	✓	✓
Fonctions côté serveur	✓	✓
HTTP Rest/JSON	✓	✓
Sécurité utilisateur et rôle	✓	✓
Base de données orientée graphes	✓	
Concept orienté objets	✓	
Schéma, sans schéma, schéma mixte	✓	
SQL	✓	
Langage TinkerPop Gremlin	✓	
Transactions ACID	✓	
Relationnel (documents connectés)	✓	

FIGURE 5.4 – Comparaison des 2 SGBD : MongoDB et OrientDB

# Chapitre 6

## Conclusion

Pour conclure, ce projet m'a permis de connaître de façon approfondie les différents SGBD NoSQL, notamment celui que je viens de vous décrire, à savoir OrientDB. Le fait qu'OrientDB soit un SGBD multi-modèle et principalement orienté graphes, m'a donné une autre vision de la conception NoSQL. Et enfin, cela m'a également aidé à avoir des avis critiques sur les différents SGBD, ce que je n'avais pas l'habitude de faire en comparant notamment les différents modèles.

# Chapitre 7

## Sources

- OrientDB, an SAP Company : <http://orientdb.com/>
- Orientdb, Github (@orientechnologies) : <https://github.com/orientechnologies/orientdb>
- SQL and databases, *Meenu Daves*
- Quelles différences entre sql et nosql, Sourcemax
- OrientDB, Wikipedia : <https://en.wikipedia.org/wiki/OrientDB>
- OrientDB - the 2nd generation of (MultiModel) by Luigi Dell'Aquila, Youtube
- Comparison orientdb vs. Mongodb, vschart