

UNIVERSITÉ DE NAMUR
Faculté d'informatique
Année académique 2017–2018

**Archipelago : Un framework de peristence
pour bases de données orientées graph.**

Gilles Bodart

figures/unamur.pdf

Promoteur : _____ (Signature pour approbation du dépôt - REE art. 40)
CLEVE Anthony

Co-promoteur : LAMBIOTTE Renaud

Mémoire présenté en vue de l'obtention du grade de
Master en Sciences Informatiques.

Contents

I	État de l'art	4
1	L'évolution du NoSql	5
2	Backgroud technique	8
2.1	Neo4J	8
2.1.1	Description	8
2.1.2	Langage de requête	8
2.1.3	Communication	9
2.2	OrientDB	9
2.2.1	Description	9
2.2.2	Langage de requête	10
2.2.3	Communication	11
3	Solutions existantes	12
3.1	Les librairies	12
3.1.1	Hibernate	12
3.1.2	Spring Data	12
3.1.3	Librairie Neo4J	13
3.1.4	Librairie OrientDB	13
3.2	Les frameworks	14
3.2.1	Apache TinkerPop	14
II	Contribution	16
4	Application possibles des BDOG	17
4.1	L'exemple parfait	17
4.2	Critères de comparaisons	17
4.3	Comparaison des plus grandes BDOG	18
4.4	Piste de normalisation	18
5	Le framework	19
5.1	Utilisation	19
5.1.1	La persistance	19
5.1.2	La récupération	20
5.2	Schema conceptuel	21
5.3	Documentation	21
5.4	Processus	22
6	Evaluation	23

6.1	Points forts	23
6.2	Points faibles	23
6.3	Retour d'information	23
7	Conclusion	24
7.1	Piste de réflexions	24
7.2	Archipelago en résumé	24
III	Annexes	25

Introduction

De nos jours, les unités de stockages sont de plus en plus accessible au grand public, les différentes entreprises impliquées dans le développement des systèmes de gestion de base de données l'ont bien compris. Depuis plusieurs années, L'hégémonie des bases de données relationnelles se fait de moins en moins écrasante, le solutions alternatives du NoSql séduisent jours après jours de grandes entreprises.

La grande force des SGBDR provient des fondement théorique mis en place en 1970 par Edgar Frank Codd, IBM défini le langage "Structured Query Language" (SQL) pour utiliser ce système. Cette normalisation de la représentation de l'information génère un sentiment de sûreté pour les architectes de logiciels.

A contrario, le mouvement NoSql pour "Not Only SQL" ne se base pas sur un fondement théorique commun. Que ce soit la théorie des graphs, les systèmes de hash key/value avec partitions, ... chaque système choisit ce sur quoi il va se baser. Mais si il existait une sorte de contrat liant les SGBD NoSql entre eux afin d'assurer une cohérence d'utilisation, ce mémoire va se concentrer sur les implémentations de la théorie des graph, en essayant de mettre en place un système de comparaison, nous allons tenter de spécifier certains objectifs commun et développer un framework Java permettant d'utiliser de manière agnostique l'une ou l'autre base de donnée.

Part I

État de l'art

Chapter 1

L'évolution du NoSql

Un SGBD est par définition un ensemble de procédés permettant d'organiser et de stocker des informations (potentiellement de gros volumes). Si stocker et retrouver l'information est un des plus grand challenge d'un SGBD, une communauté de développeur, pensent que ces système devraient pouvoir offrir d'autres fonctionnalités.

A partir des années 1980, le modèle relationnelle supplante les autres formes de structures de donnée.

Les évolutions logicielles suivant naturellement les évolutions matérielles, la généralisation des interconnexion des réseaux, l'augmentation de la bande passante, la diminution du cout des machines, la miniaturisation des espaces de stockage, ... de nouvelles opportunités sont arrivé au XXI^e siècle.

Les entreprises comme Google, Amazon, Facebook, Twitter, ... sont tour à tour arrivés aux limites du modèle Relationnel. Que ce soit a cause de volumes astronomiques (plus de 100 pétaoctets) ou du nombre de requêtes par secondes, il fallut développer une nouvelle façon de gérer les données.

Le NoSql découle de ce genre de problèmes, ces modèles arrivent avec des approche optimisée pour des secteurs spécifiques.

Comme ces modèles représentent ce qui n'est pas Relationnel, par soucis de classification, nous allons distinguer 4 usages principaux :

- Performances : L'objectif du SGBD sera d'augmenter au maximum les performances de la manipulation des données.
- Structures simples : Pour s'affranchir de la rigidité du modèle relationnel, la structure sera généralement simplifiée, en utilisant une représentation plus souple comme le JSON par exemple.
- Structures spécifiques : Certain moteur NoSql sont liés a des besoins spécifiques, la structure de représentation de donnée sera dès lors focalisée sur un cas d'utilisation.
- Volumétries : Un des principal aspect important des SGBD NoSql est leur capacité de gérer la montée en charge de données. La distribution des traitements au travers de plusieurs clusters est un facteur très important dans la plupart des applications BigData.

Et nous allons aussi distinguer 4 grandes familles de représentation de Schéma de données :

- Document : L'utilisation de format spécifiques tels que le très répandu JSON permet de

stocker les données sur base de fichier.

- Clé / Valeur : Le système le plus simple, il manipule des paires de clé/valeurs, ou accède à un élément en fonction d'une table de hachage.
- Colonne : Inspiré de Google BigTable, la structure ressemble à la table relationnelle. On peut la comparer à une table de hachage qui va référencer une ou plusieurs colonnes.
- Graph : La famille Graph se distingue du fait que les entités ne sont pas considérées comme des entités indépendantes, mais que la relation entre ces objets est tout aussi importante que le contenu.

Les implémentations de bases de données de types graph sont de plus en plus nombreuses, les relations entre les éléments permettent de parcourir le graph de manière très performante les rendent de plus en plus intéressantes pour les entreprises possédant des millions de données. L'utilisation de ce genre de SGBD est dès lors tout à fait recommandé pour des entreprises intéressées entre les relations de ces données tels que des profils sociaux, des liens de cause à effet, des liens géographiques et bien d'autres.

POURQUOI UTILISER UNE BDOG PLUTÔT QU'UNE BD RELATIONNELLE COMME ORACLE OU MYSQL ?

L'utilisation des SGBD relationnelles pour tout type de problème est révolue. Grâce à l'apparition de SGBD NoSQL spécifique permet dès à présent une approche plus personnalisée. En effet, comme tout système, la relationnelle a ses limites, l'approche actuelle des entreprises est plus orientée "Big Data", elles veulent tout stocker afin d'avoir le plus d'informations possible, or du fait que la relationnelle cadenas les données dans une table préalablement définie, on se doit de tronquer l'information ou alors, mettre à jour le schéma de définition des tables. Dans une BDOG tel que Neo4J, nous pouvons ajouter toutes les informations que nous voulons sans condition préalable, cette approche oriente plus un contrôle de l'intégrité des données aux applications.

Si l'objectif de l'applicatif est de représenter un système récursif comme une arborescence de fichier, de la généalogie, ... le modèle relationnel se basera sur une table faisant référence à elle-même, l'utilisateur devra donc réaliser une jointure par profondeur, si l'arborescence descend jusqu'à 15 niveaux, cela peut devenir problématique. L'approche BDOG permet de parcourir une arborescence sans pour autant charger l'ensemble des nœuds ayant le même label, plus besoin de projection ensembliste pour pouvoir continuer le chemin.

En relationnel, il existe de nombreux moyens d'historiser les données, cependant, lorsque le schéma comprend des relations cycliques, cette opération devient plus complexe. L'utilisation d'un BDOG rend ça plus simple, il suffit de créer un nouveau nœud avec les anciennes données, lier ce nœud avec une relation historique en y spécifiant la date de mise à jour et ensuite changer les valeurs du nœud référencé par le cycle. Ce procédé peut être mis au point dans n'importe quelle représentation et ne nécessite aucune refonte globale du modèle de données.

ON PARLE DE BASE DE DONNÉE ORIENTÉE GRAPH MAIS QUELLE EST LA DIFFÉRENCE ENTRE UNE RELATION ENTRE DEUX NOEUD ET UNE RELATION ENTRE DEUX TABLE ?

Les relations entre deux tables se font à l'aide de clé étrangère définie lors de la création du schéma de données elles apporte de l'information supplémentaire en permettant de réaliser des jointure entre plusieurs tables.

La relation dans les BDOG est une information à part entière, elle peut posséder autant de donnée que le noeud lui même, la force de ce genre de système est de pouvoir lier deux noeud distinct par n'importe quelle relation et ce à tout moment.

Chapter 2

Backgroud technique

2.1 NEO4J

2.1.1 DESCRIPTION

Créé par Neo Technologie, une société suédo-américaine, elle est actuellement (selon db-engines.com) la base de donnée orienté graph la plus utilisée dans le monde. Développé en java sous licence GPL V3, AGPL ou licence commerciale, Neo4J représente les données sous formes de "Noeuds" et de "Relations", chacun de ces éléments peuvent contenir une ou plusieurs propriétés. Les propriétés sont des couples clés/valeurs de type simple, comme des chaines de caractères ou des valeurs numériques, des coordonnées spatiales, ...

Une des particularité de Neo4J est l'absence de structure définie, un noeud peut être labellisé afin de permettre de travailler sur un ensemble d'éléments, mais il n'y aura aucune contrainte sur les propriétés du noeud. Cette particularité rend ce SGBD bien adapté pour les modèles évoluant fréquemment.

2.1.2 LANGAGE DE REQUÊTE

Le langage propre à Neo4J se nomme "Cypher", il a pour but de réaliser plus simplement que SQL les opérations de parcours ou d'analyse de proximité.

```
1 CREATE (mamours:Person {name:"Mamours"})
2 CREATE (mamymo:Person {name:"Mamymo"})
3 CREATE (gilles:Person {name:"Gilles"})
4 CREATE (marie:Person {name:"Marie"})
5 CREATE (bebe:Person {name:"Enfant"})
6 CREATE (mamours)-[:PARENT_OF]->(gilles)
7 CREATE (mamymo)-[:PARENT_OF]->(marie)
8 CREATE (marie)-[:PARENT_OF]->(bebe)
9 CREATE (gilles)-[:PARENT_OF]->(bebe)
```

Ces requêtes vont créer 5 noeud et 4 relations PARENT_OF, nous pouvons aisément comprendre que "Mamymo" est parente de "Marie"

```
1 MATCH (p:Person)-[:PARENT_OF]->(c:Person)
2 RETURN DISTINCT (p)
```

Cette query va retourner tout les nœuds distinct qui ont une relation :PARENT_OF avec un autre noeud.

```
1 MATCH (gp:Person)-[:PARENT_OF*2]->(c:Person)
2 RETURN DISTINCT (gp)
```

Celle-ci quant à elle va retourner toutes les personnes qui sont parent de parent et donc grand parents.

ces deux exemples peuvent montrer la force de l'utilisation d'un SGBD de type graph pour représenter un ensemble hierarchique de données par rapport au SGBD relationnelles qui nécessiterai une double jointure sur la Table "Person".

2.1.3 COMMUNICATION

Neo4J peut être utilisé sous plusieurs formes.

La première option est une solution embarquée, ce choix peut être très intéressant en alternatif au très célèbre SQLite relationnel.

La deuxième, pour toute application distribuée, est une solution autonome pouvant tourner comme un service sur tout type de plateforme. Le protocol "Bolt", développé par "Néo Technologie" est grandement conseillé pour communiquer avec ces serveurs distants.

Son utilisation est simple grâce à l'utilisation de la librairie native à Néo4J pour java (neo4j-java-drive) ou avec l'utilisation de l'API Rest déployée en même temps que le SGBD.

2.2 ORIENTDB

2.2.1 DESCRIPTION

OrientDB est un SGBD initialement développé en C++(Orient ODBMS) ensuite repris en 2010 en Java par Luca Garulli dans une version multi-modèle sous licence Apache 2.0, GPL et AGPL. actuellement 3ème mondial (selon db-engines.com) il offre de nombreuses fonctionnalités intéressante.

OrientDB est base de donnée associant Document et Graph. Elle combine la rapidité et la flexibilité du type document ainsi que les fonctionnalités de relations des bases de données graph.

Ce SGBD est composé de trois grands éléments

- Document & Vertex : Source de contenu, ces éléments peuvent être considéré comme des container de données, on peut le comparer avec la ligne d'une base de données relationnelle.

- Links & Edge : Une arrête orientés reliant deux éléments non nécessairement distinct.
- Property : Typée ou embarquée dans un document JSON, ceci va représenter le contenu de l'information. Ces propriétés sont bien entendu primordiales pour ordonner, rechercher, ...

Chaque Document ou Vertex appartient à une "Class", celle-ci peut être strictement définie ou plus laxiste. Comme dans la programmation orientée objet, OrientDB offre le principe de polymorphisme avec un système d'héritage entre les classes.

OrientDB vient dans sa version community avec un système de clustering permettant à l'utilisateur de correctement gérer les montées en charges. chaque document est identifié avec une partie désignant le cluster dans lequel l'information est stockée et une autre partie désignant sa position dans ce dernier (exemple @rid: 10:12). Chaque classe peut être associée à un ou plusieurs clusters, permettant d'optimiser les accès dans des ensembles plus petits.

2.2.2 LANGAGE DE REQUÊTE

OrientDB utilise une sorte de SQL avancée pour interpréter les requêtes. On peut de plus utiliser le langage Gremlin.

Voici quelques exemples d'utilisation du SQL avancé dans OrientDB.

```

1 CREATE CLASS Person EXTENDS V
2 CREATE CLASS Company EXTENDS V
3 CREATE CLASS WorkAt EXTENDS E
4 CREATE PROPERTY Person.firstname string
5 CREATE PROPERTY Person.lastname string
6 CREATE PROPERTY Company.name string
7 INSERT INTO Person(firstname, lastname) VALUES
8     ("Gilles","Bodart"),("Marie","Van Cutsem")
9     ou
10 INSERT INTO Company set name = "ACME"

```

Cet ensemble de requête ressemblant au langage SQL permet de créer deux vertex, Person et Company, un Edge WorkAt et leur associer certaines propriétés. Les deux types d'insert différents permettent comme en SQL d'ajouter un nœud.

```

1 SELECT FROM V

```

Metadata			Properties		
@rid	@version	@class	firstname	lastname	name
10:0	1	Person	Marie	Van Cutsem	ACME
10:1	1	Person	Gilles	Bodart	
11:0	1	Company			

Nous pouvons observer qu'OrientDB se charge de qualifier les documents de plusieurs métadonnées en leur octroyant par exemple un identifiant unique (n° cluster:position), un numéro de version pour permettre une gestion de transaction "full optimistic"¹ et la class représente la structure du document.

¹On laisse l'utilisateur continuer sa transaction et le refus se fera lors du commit si il y a eu une modification concurrente du même document

```
1 CREATE Edge WorkAt from 10:1 to 11:0
```

Cette petite requête va lier le document ayant l'id 10:1 au document dont l'id est 11:0 par une relation WorkAt. Nous pouvons traduire en français que Marie Van Cutsem travaille chez ACME.

2.2.3 COMMUNICATION

OrientDB embarque une API Rest complète, toute actions pouvant être faite sur les interfaces web et consoles peuvent être reproduites au travers de cette API.

Chapter 3

Solutions existantes

3.1 LES LIBRAIRIES

3.1.1 HIBERNATE

Hibernate, soutenu et développé par JBoss, à mis au point un système de relation entre le code Java et le SGBD Néo4J, en réutilisant les mêmes annotations de relations que celles employées pour les bases de données relationnelles, à savoir :

- @OneToOne : Relation 1-1
- @ManyToOne : Relation n-1
- @OneToMany : Relation 1-n
- @ManyToMany : Relation n-n

Bien qu'à l'heure actuelle, aucune intégration avec OrientDB, un ticket est ouvert (OGM-855) depuis 2 ans auprès d'Hibernate.

3.1.2 SPRING DATA

Spring data à nouveau possède un module de communication avec le SGBD Néo4J, il intègre au célèbre Framework Spring les fonctionnalités de Néo4J-OGM¹, permettant de qualifier une classe avec certaines annotations facilitant la sérialisation et le désérialisation vers la représentation graphique.

Exemple :

```
1 @NodeEntity(label="Film")
2 public class Movie {
3
4     @GraphId Long id;
5
6     @Property(name="title")
7     private String name;
8 }
```

¹Object Graph Mapper

```

9
10 @NodeEntity
11 public class Actor extends DomainObject {
12
13     @GraphId
14     private Long id;
15
16     @Property(name="name")
17     private String fullName;
18
19     @Relationship(type="ACTED_IN", direction=Relationship.OUTGOING)
20     private List<Role> filmography;
21
22 }
23
24 @RelationshipEntity(type="ACTED_IN")
25 public class Role {
26     @GraphId
27     private Long relationshipId;
28     @Property
29     private String title;
30     @StartNode
31     private Actor actor;
32     @EndNode
33     private Movie movie;
34 }

```

3.1.3 LIBRAIRIE NEO4J

Une librairie java développée par NeoTechnologie existe, elle permet de travailler aisément avec une base de donnée Neo4J embarquée ou à distance. Elle est en constante évolution et se trouve sur une plateforme open source.

Exemple de communication avec la librairie Neo4J le protocol "Bolt":

```

1 Driver driver = GraphDatabase.driver("bolt://localhost:7687",
    AuthTokens.basic("matrix", "neo"));
2 Session session = driver.session();
3 session.run("CREATE (a:Person {firstName:{name},lastName:{lastName}})",
    parameters("firstName","Gilles","lastName","Bodart"));
4 session.close();
5 driver.close();

```

Bien que cette librairie soit efficace, elle ne peut être utilisée avec un autre SGBD.

3.1.4 LIBRAIRIE ORIENTDB

La librairie OrientDB pour java est composée de trois éléments:

- Graph API
- Document API
- Object API

La graph API d'OrientDB couvre 80% des uses cases classique d'un utilisateur, elle supporte tout les modèles de représentations comme un unique multi-modele de donnée. Elle travaille sur des "Vertex" et des "Edges", respectivement nœud et relations et est compatible avec le standard TinkerPop

La document API elle permet de couvrir les 20% restants, elle est plus simple d'utilisation mais n'offre une utilisation plus atomique, les relations ne peuvent par exemple qu'être unidirectionnelles tandis que la bidirectionnalité est offerte dans l'API précédent. Un document est un élément unique, pour gérer les relations entre les différents nœuds de notre modèle, il faudra gérer manuellement chaque connexion.

La dernière librairie se focalise sur les objets java, elle n'a plus été améliorée depuis la version 1.5 d'OrientDB²

3.2 LES FRAMEWORKS

3.2.1 APACHE TINKERPOP

Apache TinkerPop est Framework open source d'utilisation de graph, il regroupe un grand ensemble de fonctionnalités et d'algorithmes. Son écosystème est de plus en plus agrémenté de librairies externes développées par de tierces personnes.

Il se considère lui-même comme étant complexe d'utilisation pour les nouveaux développeurs car il nécessite l'utilisation d'un environnement qui leur est propre, la "Gremlin-console".

Les développeurs de ce framework ont mis au point un langage de travail sur les graphs appelé "Gremlin". Le langage dont nous avons parlé en section 2.1.2 s'est grandement inspiré de ce dernier.

Exemple d'utilisation

```

1 gremlin> g.V() /* récupère tout les noeuds du graph */
2 ==>v[1]
3 ==>v[2]
4 ==>v[3]
5 ==>v[4]
6 ==>v[5]
7 ==>v[6]
8 gremlin> g.V(1) /* récupère le noeud d'Id 1 du graph*/
9 ==>v[1]
10 gremlin> g.V(1).values('name') /* récupère le noeud d'Id 1 du graph et
    renvoie sa propriété 'name' */
11 ==>marko

```

²À l'heure de l'écriture de ce mémoire, nous sommes à la version 2.2 et la version 3 à bientôt fini sa phase de "bêta test"

```

12 gremlin> g.V(1).outE('knows') /* récupère les relations 'knows' partant du
    noeud d'Id 1 */
13 ==>e[7][1-knows->2]
14 ==>e[8][1-knows->4]
15 gremlin> g.V(1).outE('knows').inV().values('name') /* retourne le nom des
    personnes que le noeud 1 connaît */
16 ==>vadas
17 ==>josh
18 gremlin> g.V(1).out('knows').values('name') /* même résultat */
19 ==>vadas
20 ==>josh
21 gremlin> g.V(1).out('knows').has('age', gt(30)).values('name') /* retourne le
    nom des personnes plus vieille que 30 ans que le noeud 1 connaît */
22 ==>josh

```

Comme expliqué plus haut, le langage doit être exécuté dans un invite de commande ou dans un serveur Gremlin, nous sommes dès lors contraint d'installer et de configurer ce dernier pour pointer vers l'un ou l'autre SGBD.

Part II

Contribution

Chapter 4

Application possibles des BDOG

4.1 L'EXEMPLE PARFAIT

Pour la suite de ce mémoire, il est primordiale d'avoir un exemple concret de modèle habituellement persisté avec un modèle relationnelle mais qui serait sans le moindre problème transposable dans une BDOG.

L'ÉCOLE

Imaginons la représentation d'une école composée d'élèves, de professeurs, de classes, de salles techniques, de travailleurs, ... dont le diagramme de classes se trouve en Annexe (7.1).

Les liens entre les différents objets peuvent être fortement nombreux, nous pouvons imaginer que les étudiants sont amis, membres de famille, ennemis, amoureux, ... de même que pour les professeurs, ils peuvent avoir les mêmes relations que les étudiants mais de plus peuvent être assigné a des cours, a des classes, ... toutes ces liaisons peuvent être fortement compliquées à représenter sur une base de donnée relationnelles. L'utilisation d'une BDOG semble être tout à fait adéquate. Durant la suite de ce mémoire, nous utiliserons donc ce modèle.

4.2 CRITÈRES DE COMPARAISONS

Dans un premier temps, il est primordiale de mettre en place une liste de critères permettant de faciliter le choix de la BDOG.

4.3 COMPARAISON DES PLUS GRANDES BDOG

Critère \ BDOG	Neo4J	OrientDB
Schema de donnée strict	X	✓
Format de donnée	JSON	JSON
Principe d'héritage	X	✓
Clustering	Configurable	Natif
Communication	BOLT (protocole propriétaire)	REST (via HTTP)
Évolutivité	L'absence de schéma strict entraîne une grande liberté dans la création et la modification des nœuds	Les classes doivent préalablement être configurées sur le serveur
Rapidité ¹	7m 10s	4m 31s
Modèle de représentation	Uniquement graph	multimodèle

Une idée d'arbre de décision devrait, à la fin de cette section, permettre à un utilisateur muni de ses besoins, de choisir la BDOG la plus adaptée à son projet.

4.4 PISTE DE NORMALISATION

Comme nous l'avons présenté dans le point 1 nous nous apercevons que le problème principal du NoSql est le manque de standard d'utilisation de ces SGBD, pour ce faire, nous allons tenter de lister un des objectifs de normalisations.

- Utilisation d'un langage unique
- Système de transaction
- Opération de base de persistance CRUD
- Possibilité de créer des méthodes stockées
- Contrôle sur les propriétés.

La suite de ce mémoire portant sur le développement d'un framework java, tentera d'offrir ces objectifs de manière abstraite aux développeurs.

¹10 000 requêtes sur le même graph de donnée avec la même requête au travers d'un appel Java

Chapter 5

Le framework

5.1 UTILISATION

Ce framework à été basé sur l'utilisation d'annotations, à l'heure actuelle, il permet d'effectuer deux opérations:

5.1.1 LA PERSISTANCE

La persistance est l'élément primordiale de ce framework, afin de ne pas avoir de lien avec une BDOG propre, nous nous sommes basé sur le modèle orienté objet. La persistance de celui-ci ainsi que des éléments qui lui sont lié est paramétrisation dans un fichier de configuration nécessaire à l'initialisation de cet outil.

Le fichier de configuration doit se nommer "config.archipelago.json" et être présent dans les ressources du projet.

Voici le canevas de ce fichier :

```
1 {
2   "database": {
3     "type": "", // Choix entre NEO4J et ORIENT_DB
4     "username": "",
5     "password": "",
6     "url": "",
7     "name": "", // Nom de la base de donnée
8     "embedded": false, // boolean Optionnel
9     "port": 1234 //
10  },
11  "deepness": 3, // Profondeur de persistance voir ci-après
12  "domainRootPackage": "org.archipelago.test.domain.school"
13  // package où se trouve les classes du domaine
14 }
```

Afin d'assurer la persistance des éléments liés, nous avons opté pour une approche récursive en "depth-first" avec marquage de sommet, en effet, pour créer un lien entre deux éléments, il faut

absolument que ceux si soit présent dans la base de donnée.

La propriété "deepness" est primordiale pour la performance de ce framework, elle permet de définir la profondeur de persistance d'un objet. Ce procédé empêche de tomber dans des boucles infinies, courante lorsque nous travaillons avec de modèles ayant de relations bidirectionnelles comme suit :

```
1 Archipelago arch = Archipelago.getInstance();
2
3 Student gilles = new Student();
4 Student antoine = new Student();
5
6 gilles.setFriend(antoine);
7 antoine.setFriend(gilles);
8
9 arch.persist(gilles);
```

Ainsi avec cette valeur sentinelle nous rompons la boucle de persistance apres avoir atteint la profondeur souhaité.

figures/Persist.png

5.1.2 LA RÉCUPÉRATION

La récupération de l'information est la raison pour laquelle nous la stockons. A quoi bon enregistrer une valeur dans une base de donnée si nous ne souhaitons jamais l'utiliser ultérieurement ?

Pour cela, Archipelago vient avec un système de création de requête. Grâce à ce procédé, que votre BDOG soit OrientDB ou Neo4J, vous serez en mesure de récupérer votre objet sans changer votre code source.

Voici un exemple:

```
1 Archipelago arch = Archipelago.getInstance();
2
3 ArchipelagoQuery aq = a.getQueryBuilder()
4     .of(Student.class)
5     .build();
6 List<Object> nodes = arch.execute(aq);
```

Cette requête va rechercher l'ensemble des étudiants de la base de donnée ainsi que les objets qui leurs sont liés.

Un système de condition est également mit en place, il permet d'affiner le résultat obtenu:

```
1 Archipelago arch = Archipelago.getInstance();
2
3 ArchipelagoQuery aq = a.getQueryBuilder()
4     .of(Student.class)
```

```

5         .where(of("firstName", "Gilles"), ConditionQualifier.Equal)
6         .build();
7 List<Object> nodes = arch.execute(aq);

```

Cette requête va quant à elle rechercher les étudiants ayant "Gilles" comme prénom.

Nous pouvons aussi ajouter des opérateurs logique "OU" et "ET" comme suit :

```

1 Archipelago arch = Archipelago.getInstance();
2
3 ArchipelagoQuery aq = a.getQueryBuilder()
4     .of(Student.class)
5     .where(of("lastName", "Bodart"), ConditionQualifier.EQUAL)
6     .and(of("firstName", "Gilles"), ConditionQualifier.EQUAL)
7     .or(of("firstName", "Thomas"), ConditionQualifier.EQUAL)
8     .build();
9 List<Object> nodes = arch.execute(aq);

```

Le procédé mit en place à l'heure actuel ne permet pas de faire des conditions fortement complexes, il va ajouter l'élément de condition à la suite de la requête actuelle, pour l'exemple précédent, nous aurons donc :

```

1 lastName = "Bodart" AND ( firstName = "Gilles" OR firstName = "Thomas")

```

ce qui se retournera donc la liste des étudiants ayant "Bodart" comme nom de famille et "Gilles" ou "Thomas" comme prénom. À L'heure actuelle, il n'est cependant pas possible d'effectuer des requêtes sur les relations entre les objets, cette piste est primordiale pour une deuxième version de ce framework.

figures/query.png

5.2 SCHEMA CONCEPTUEL

Les schémas conceptuels représentant une modèle exemple annoté des éléments du framework sera fourni et commenté dans cette section. Cela permettra au lecteur une meilleur compréhension.

5.3 DOCUMENTATION

Une documentation claire et précise sur l'utilisation du framework Archipelago sera présente dans cette section. Un ensemble entre une documentation fonctionnelle et une documentation technique faite avec JavaDoc.

5.4 PROCESSUS

Description du processus implémenté sur base d'un exemple claire. Explications des différents choix d'implémentations et de chaque étape.

Chapter 6

Evaluation

6.1 POINTS FORTS

Autocritique du framework, sur base de test qualitatif et ou quantitatif. Evaluations : usability, performance, qualité, cohérence

6.2 POINTS FAIBLES

Autocritique du framework, sur base de test qualitatif et ou quantitatif. Evaluations : usability, performance, qualité, cohérence

6.3 RETOUR D'INFORMATION

Si le temps nous le permet, une analyse des retours utilisateur sera faite en fin de mémoire.

Chapter 7

Conclusion

7.1 PISTE DE RÉFLEXIONS

Une introspection sur le projet sera expliqué dans cette section, les idées innachevés y seront décrites en tant que piste de réflexions.

7.2 ARCHIPELAGO EN RÉSUMÉ

Le mémoire sera conclu avec un explication transversale et complète du framework, permettant au lecteur de garder une bonne impression sur le nouvel outil que sera ce framework.

Part III

Annexes

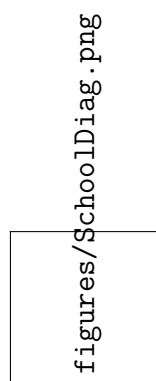


FIGURE 7.1 – Diagramme de Classe du modèle École

Lexique

SQL	Structured Query Language
NoSQL	Not only SQL
SGBD	Système de gestion de base de données
SGBDR	Système de gestion de base de données relationnelles
BDOG	Base de donnée orientée graph
JSON	JavaScript Object Notation
Vertex	Sommet
Edge	bord
Cluster	littéralement grappe, cela représente en informatique un groupe d'élément
API	Application Programming Interface
REST	Representational state transfer, style d'architecture pour les systèmes hyper-média distribués

Code Sources

Bibliographie

- <https://neo4j.com/> consulté à de nombreuses reprises (Neo Technology, Inc)
 - <https://orientdb.com/> consulté à de nombreuses reprises (OrientDB LTD)
 - <https://snap.stanford.edu/data/>
 - <https://networkx.github.io/>
 - <http://igraph.org/redirect.html>
 - <https://snap.stanford.edu/data/egonets-Facebook.html>
 - <http://konect.uni-koblenz.de/>
 - <https://icon.colorado.edu/#!/networks>
 - <https://neonx.readthedocs.io/en/latest/>
 - J. McAuley and J. Leskovec. Learning to Discover Social Circles in Ego Networks. NIPS, 2012.
 - J. Leskovec, K. Lang, A. Dasgupta, M. Mahoney. Community Structure in Large Networks: Natural Cluster Sizes and the Absence of Large Well-Defined Clusters. Internet Mathematics 6(1) 29–123, 2009.
 - <https://www.infoq.com/fr/articles/graph-nosql-neo4j>
 - <http://www.silicon.fr/base-donnees-nosql-impose-sgbd-93305.html>
 - <https://prezi.com/4flswlgipwbo/nosql-not-only-sql/>
- LIVRE <http://www.eyrolles.com/Chapitres/9782212141559/9782212141559.pdf>
- <https://db-engines.com>
 - <https://www.udemy.com/orientdb-getting-started>
 - https://fr.wikipedia.org/wiki/Representational_state_transfer