Projet Bibliographique

NoSQL & Big DATA

Comprendre et mettre en oeuvre

Réalisé par : ABOUELOMOUM Younes

Encadré par : M.BOUTABIA Mohammed

2ème Année Génie Informatique

Année Universitaire : 2020-2021

Table de matières

Introduction Générale

Devant le besoin grandissant de performance et de disponibilité des services logicielles ayant de très grands traffic, les systèmes de gestion de données représentent sans aucune doute un facteur décisif dans le succès ou l'échec de ces services. C'est pour répondre à cette problématique que sont nées les base de données NoSQL, sous l'impulsion de grands acteurs du web comme Facebook ou Google, qui les ont développées à l'origine pour leurs besoins propres.

Le projet bibliographique effectué tout au long de ce semestre m'a permis de découvrir les différentes technologies NoSQL existantes ainsi que leurs avantages et inconvénients par rapport aux SGBDR. Il m'a aussi permis d'acquérir de nombreuses compétences aussi bien techniques que d'autres compétences comportementales et personnelles, telles que l'auto-discipline, l'organisation, et l'analyse de problèmes.

Le présent rapport a pour but de synthétiser le travail effectué pendant ce semestre, avec un souci d'expliquer avec toute clarté possible les différents aspects liés aux SGBDs. et surtout de combiner l'aspect théorique et pratique pour pouvoir donner une image globale sur le sujet du projet.

I. A propos du livre

“Les bases de données NoSQL: Comprendre et mettre en oeuvre” est un livre rédigé par M. Rudi Bruchez et publié en deux éditions, la première édition était publiée en 2013. Puis après deux ans et vu l'évolution très rapide des technologies NoSQL et le Big Data, M. Bruchez a publié une deuxième édition entièrement refondue. C’est un livre assez exhaustif et qui traite de toutes les parties fondamentales et historique du mouvement NoSQL mais aussi les parties techniques concernant la mise en oeuvre des principaux moteurs NoSQL existants dans le marché ainsi que leurs cas d'utilisation.

L'auteur du livre est un consultant informatique indépendant, expert en bases de données depuis une quinzaine d'années. Il assure des services pour la modélisation, l'administration et l'optimisation des serveurs et du code SQL, ainsi que des services autour de SQL Server et des solutions NoSQL

II. L’Evolution des SGBDs

Afin de comprendre les points de convergence et de divergence des modèles que nous allons présenter tout au long de ce rapport, il faut tout d'abord savoir qu'est ce que c'est que les systèmes de gestion de base de données et comment ont ils évolué pour parvenir aux modèles en présence. C'est l'objet de notre premier chapitre.

2.1 Qu'est ce qu'un système de gestion de base de données?

Depuis toujours et l'être humain est en quête de comprendre une des plus grandes mystères du monde, le cerveau. Ainsi, les ordinateurs représentent une tentative de reproduire le cerveau humain. Cependant, il nous est impossible de produire le moindre raisonnement sans constituer un stock d'informations équivalent au stock humain appelé mémoire. Cela était alors un des premiers défis de l'informatique, conserver les données et pouvoir les restituer. Un système de gestion de base de données est tout simplement un système qui fournit essentiellement ces fonctions, souvent à l'aide d'un langage dédié, aussi bien que d'autres sous-fonctions telles que la sécurité, la disponibilité et la cohérence.

2.2 Les systèmes de gestion de base de données relationnels

Les systèmes dits relationnels représentent un modèle de gestion de données introduit pour la première fois par Edgar Codd en 1970, et basé principalement sur la théorie des ensembles mathématiques. Ce qui faisait du système du Codd une révolution dans son domaine n'était pas le fait qui est le premier modèle de gestion de base de données. il avait bien entendu des modèles plus anciens notamment les systèmes hiérarchiques et les application Cobol. Mais ce qui faisait la différence entre ces systèmes et le système relationnel sont deux facteurs essentiels qui sont :

1. **Les relations:** d'où venait la nomination système relationnel. ils représentent un nouveau concept qui a remplacé la manipulation des pointeurs par un système de relation basé uniquement sur les valeurs de données et qui a représenté une approche beaucoup moins complexe pour parcourir les données mais aussi plus optimale.
2. **Le Langage Sequel :** il s'agit d'un langage haut niveau déclaratif et algébrique, qui ne s'occupe pas des algorithmes de manipulation et de stockage physique des données mais qui permet la gestion logique des données et d'exprimer simplement une requête d'une manière plus compacte que la manière procédurale utilisée en Cobol. Comme Sequel était un marque déposé d'une société britannique, le nom du langage fut changé en SQL.

2.3 Les douze règles de Codd :

Ces règles sont basées sur les travaux originaux de Edgar Codd, et ont été publiées dans deux articles du magazine Computerworld en octobre 1985 : Is your DBMS really relational ? et Does your DBMS run by the rules? Ces règles constituent les fondements des bases de données relationnelles et permettent de connaître le niveau relationnel du produit d'un éditeur. Ne pas s'y contraindre implique plus d'inconvénients que d'avantages.

RÈGLE 0 – Introduction : Tout système de bases de données prétendant être relationnel doit être capable de gérer complètement les bases de données à l'aide de ses caractéristiques relationnelles.

RÈGLE 1 - Règle de l'information : Toutes les informations dans une base de données relationnelle sont représentées de façon explicite au niveau logique et d'une seule manière : par des valeurs dans des tables.

RÈGLE 2 - Garantie d'accès : Toute donnée atomique d'une base de données relationnelle est accessible par le biais d'une combinaison du nom de la table ou de la vue, d'une valeur de la clé primaire et d'un nom de colonne.

RÈGLE 3 - Gestion du marqueur NULL : Les marqueurs NULL sont systématiquement pris en charge pour représenter des informations manquantes ou inapplicables, indépendamment du type, du domaine de valeur ou d'une valeur par défaut.

RÈGLE 4 - Catalogue relationnel, dynamique et accessible directement : La description de la base de données et de son contenu est représentée au niveau logique de la même manière que les données ordinaires, c'est à dire sous forme de tables.

RÈGLE 5 - Langage de manipulation de données complet : Au moins un des langages du SGBDR doit avoir une syntaxe complète et doit permettre la définition des données, la formation des vues, la manipulation des donnée, la gestion des règles d'intégrité, les autorisations et les frontières des transactions.

RÈGLE 6 - Règle de mise à jour des vues : Toutes les vues qui sont théoriquement modifiables peuvent être mises à jour par le système.

RÈGLE 7 - Insertion, suppression et modification ensemblistes : Le SGBDR retourne un ensemble d'éléments en réponse aux requêtes qui lui sont soumises. Il doit pouvoir mettre à jour un ensemble d'éléments en exécutant une seule requête.

RÈGLE 8 - Indépendance physique des données : Les applications et les programmes terminaux sont logiquement non affectés lorsque les méthodes d'accès physiques ou les structures de stockage sont modifiées.

RÈGLE 9 - Indépendance logique des données : Les applications et les programmes terminaux sont logiquement non affectés, quand des changements de tous ordres, préservant les informations et qui ne leur portent théoriquement aucune atteinte, sont apportés aux tables de base (restructuration).

RÈGLE 10 - Indépendance vis-à-vis de l'intégrité : Les contraintes d'intégrité spécifiques à une base de données relationnelle sont définies à l'aide du langage relationnel et leur définition doit être stockée dans le catalogue et non dans des programmes d'application.

RÈGLE 11 - Indépendance de distribution : Le langage relationnel doit permettre aux programmes d'application et aux requêtes de demeurer identiques sur le plan logique lorsque des données, quelles qu'elles soient, sont physiquement réparties ou centralisées.

RÈGLE 12 - Non subversion : Il ne doit pas être possible de transgresser les règles d'intégrité et les contraintes définies par le langage relationnel du SGBDR en utilisant un langage de plus bas niveau.

2.4 Les transactions et les critères ACID :

On appelle une transaction en base de données, une suite d'opérations permettant le passage d'un état A vers un état B. Pour mieux comprendre on va prendre un exemple, supposons qu'un banquier veut transmettre une somme d'argent d'un compte A vers un compte B. Ceci requiert un ensemble de sous-tâches à suivre, il faut tout d'abord retirer de l'argent auprès du compte A, puis rajouter la somme retirée au solde du compte B...etc. L'ensemble de ces opérations se regroupent dans une seule transaction appelée virement qui est soit réussie ou échouée dans son ensemble. Le concept reste le même en base de données la seule différence est que les opérations sont effectuées sur les données de la base. Une transaction en base de données doit respecter quatre critères resumés par l'acronym ACID. Ces critères sont présentés dans le tableau suivant :

Critère	Définition
Atomique	Une transaction représente une unité de travail qui est validée intégralement ou totalement annulée. C'est tout ou rien.
Cohérente	La transaction doit maintenir le système en cohérence par rapport à ses règles fonctionnelles. Durant l'exécution de la transaction, le système peut être temporairement incohérent, mais lorsque la transaction se termine, il doit être cohérent, soit dans un nouvel état si la transaction est validée, soit dans l'état cohérent antérieur si la transaction est annulée.
Isolée	Comme la transaction met temporairement les données qu'elle manipule dans un état incohérent, elle isole ces données des autres transactions de façon à ce qu'elle ne puisse pas lire des données en cours de modification.
Durable	Lorsque la transaction est validée, le nouvel état est durablement inscrit dans le système.

Figure 1 : Tableau résumant les critères ACID

2.5 Le mouvement NoSQL :

Une règle générale dans le monde informatique c'est que les évolutions logicielles dépendent toujours des évolutions matérielles. Ainsi, les premiers modèles de gestion de données ont été développés sous plusieurs contraintes notamment la capacité de stockage très limitée par rapport aux capacités actuelles. Le succès du modèle relationnel est dû non seulement aux qualités du modèle lui-même mais aussi aux optimisations de stockage que permet la réduction de la redondance des données. Cependant, avec l'augmentation de la bande passante sur Internet, la diminution du coût de machines moyennement puissantes et l'augmentation des capacités de stockages. Les contraintes de l'époque ne sont plus de soucis pour les entreprises d'aujourd'hui et de nouvelles possibilités ont vu le jours telles que la distribution et la virtualisation mais aussi de nouveaux défis sont apparus. Avec l'émergence de Big Data, les moteurs de base de données relationnels, hautement transactionnels se trouvaient inefficaces pour traiter des volumes de données grandissants d'une manière exponentielle. Il fallait alors chercher de nouvelles approches pour surmonter ces nouveaux défis. Les premières initiatives ont été prises par les acteurs les plus concernés comme google, facebook et amazon. Commençant du Google Files System (GFS) un système de fichier distribué permettant le stockage de grands volumes de données, et passant par Hadoop qu'il a été basé principalement sur le paradigme MapReduce inspiré de la programmation fonctionnels, jusqu'à arriver au DynamoDB d'Amazon qui a été le déclencheur officiel du mouvement NoSQL et le système qui a inspiré la création de plusieurs moteurs NoSQL comme Cassandra, Riak ou Voldemort de Linkedin.

2.6 Le Théorème CAP & PACELC :

Il s'agit d'un théorème énoncé par Eric Brewer, et qui dérive d'une réflexion du professeur américain, qui voulait changer l'approche par laquelle suivie par l'informatique distribuée en s'intéressant non seulement au distribution du calcul, la chose qui était relativement simple à l'aide de Hadoop et le paradigme MapReduce, mais aussi de réfléchir au distribution de données, ce qui plus difficile. le théorème présente alors les trois propriétés qu'un système distribué de données doit assurer, et prouve qu'un système ne peut fournir que deux propriétés de ces trois simultanément. L'acronym CAP est une abbréviation des trois propriétés :

Consistency (cohérence) : tous les nœuds sont à jour sur les données au même moment

Availability (disponibilité) : la perte d'un nœud n'empêche pas le système de fonctionner et de servir l'intégralité des données

Partition tolerance (résistance au morcellement) : chaque nœud doit pouvoir fonctionner de manière autonome.

le théorème CAP a coulé beaucoup d'encre dans la communauté des SGBDs. Certains l'ont considéré comme un faux argument pour argumenter l'incapacité des moteurs NoSQL d'assurer les critères ACID, surtout qu'ils ne peuvent pas assurer la disponibilité et la cohérence au même temps; Pendant que d'autres le voyaient comme un théorème plutôt simpliste, et qui incite à abandonner la cohérence sans raison clair. La chose qui a motivé Daniel Abadi d'énoncer sa variation du théorème CAP, appelée PACELC, qui est sous forme d'un arbre de décision, présenté ci-dessous :

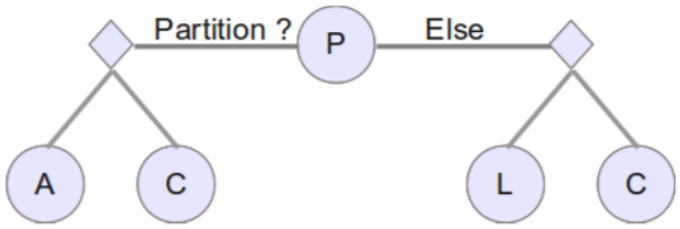


Figure 2 : Le théorème PACELC

Le raisonnement est le suivant : si l'on est en présence d'une partition (P), le système va-t-il choisir entre la disponibilité (A) ou la cohérence (C) ? Sinon (E pour Else), le système va-t-il choisir entre la latence (L) ou la cohérence (C) ? Un système comme Dynamo est de type PA/EL : en présence d'une partition, il privilégie la disponibilité au détriment de la cohérence, sinon il choisit la latence . En d'autres termes, il n'a jamais le souci de maintenir la cohérence . De l'autre côté du spectre, un moteur relationnel respectant l'ACIDité de la transaction sera PC/EC : la cohérence sera toujours privilégiée.

2.7 La classification des moteurs NoSQL :

Lors de la lecture du livre on se rend compte qu'il est très difficile de regrouper tous les moteurs NoSQL dans le même panier, vu les nombreuses différences entre ces moteurs. Cependant, l'auteur a essayé de proposer une classification selon l'usage de ces moteurs où on trouve des moteurs destinés à améliorer la performance notamment les base de données en mémoire comme Redis, d'autres moteurs pour s'affranchir à la rigidité du modèle relationnel et simplifier la structure en utilisant des schéma souple comme le JSON, et d'autre moteurs qui sont dédié à opérer dans des environnements Big Data et de monter très haut en charge comme cassandra et Hbase de Hadoop. La classification peut aussi être faite en distinguant le schéma de données que les moteurs NoSQL manipulent où on trouve des moteurs de type paire clé-valeur, des moteurs orientés document, des moteurs orientés colonnes, des moteurs manipulant des indexes inversés et des moteurs orientés graphes.

III. Mise en oeuvre de quelques moteurs NoSQL

Cette partie s'intéresse à l'installation et l'implémentation de trois moteurs NoSQL : MongoDB, Redis et Neo4j. Pourquoi ces trois moteurs? Pour deux raisons principales, la première est le fait que les trois moteurs manipulent des schéma de données différents ce qui donnera une idée plus globale sur la manière dont chaque type de données est manipulé en pratique. La deuxième raison est que les cas d'utilisation de ces trois moteurs diffèrent également et donc on profitera de cela pour présenter le plus grand nombre de cas d'utilisation des moteurs NoSQL.

3.1 MongoDB :

MongoDB est un système de gestion de données NoSQL orienté documents, càd qu'il ne manipule pas les données sous forme de tables mais plutôt en une format spécifique appelée BSON (Binary Serialized dOcument Notation) ou Binary JSON. BSON est une format similaire au JSON (JavaScript Object Notation, un format de données textuel dérivé de la syntaxe d'expression des objets en JavaScript) mais plus compacte et optimisée pour le moteur MongoDB. Le moteur offre d'excellentes performances vu le fait qu'il est codé en C++. Il est aussi un des moteurs les plus populaires grâce à sa simplicité et sa polyvalence. l'installation sera effectué sur un environnement linux, vu que l'installation sur les autres systèmes d'exploitation est relativement plus simple.

3.1.1 Installation :

Pour installer MongoDB, il faut tout d'abord télécharger la clé publique pour synchroniser la réception des paquets MongoDB :

```
younes-abou@labinfo12-Precision-T1700:~$ wget -qO - https://www.mongodb.org/stat
ic/pgp/server-4.2.asc | sudo apt-key add -
OK
```

Figure 3: étape 1 pour installer mongoDB sur ubuntu

Créer un fichier .list contenant le chemin à l'entrepôt où se trouve la version ubuntu de MongoDB :

Figure 4: étape 2 pour installer mongoDB sur Ubuntu

Mettre à jour le gestionnaire de paquets :

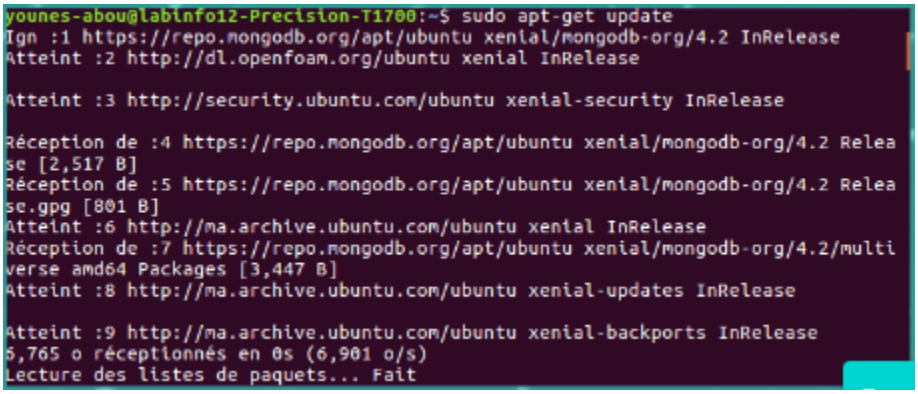


Figure 5 : étape 3 pour installer mongoDB sur Ubuntu

Installer MongoDB :

```
younes-abou@labinfo12-Precision-T1700:~$ sudo apt-get install -y mongodb-org
Lecture des listes de paquets... Fait
Construction de l'arbre des dépendances
Lecture des informations d'état... Fait
Les paquets suivants ont été installés automatiquement et ne sont plus nécessair
es :
  kde-l10n-ar kde-l10n-engb kde-l10n-fr libllvm4.0 linux-headers-4.15.0-36 linux
-headers-4.15.0-36-generic linux-image-4.15.0-36-generic
  linux-modules-4.15.0-36-generic linux-modules-extra-4.15.0-36-generic snapd-lo
gin-service
Veuillez utiliser « sudo apt autoremove » pour les supprimer.
Les paquets supplémentaires suivants seront installés :
  mongodb-org-mongos mongodb-org-server mongodb-org-shell mongodb-org-tools
Les NOUVEAUX paquets suivants seront installés :
  mongodb-org mongodb-org-mongos mongodb-org-server mongodb-org-shell mongodb-or
g-tools
0 mis à jour, 5 nouvellement installés, 0 à enlever et 0 non mis à jour.
Il est nécessaire de prendre 81.8 Mo dans les archives.
Après cette opération, 272 Mo d'espace disque supplémentaires seront utilisés.
Réception de :1 https://repo.mongodb.org/apt/ubuntu xenial/mongodb-org/4.2/multi
verse amd64 mongodb-org-shell amd64 4.2.2 [12.0 MB]
Réception de :2 https://repo.mongodb.org/apt/ubuntu xenial/mongodb-org/4.2/multi
verse amd64 mongodb-org-server amd64 4.2.2 [18.3 MB]
Réception de :3 https://repo.mongodb.org/apt/ubuntu xenial/mongodb-org/4.2/multi
```

Figure 6 : étape 4 pour installer mongoDB sur Ubuntu

le service est par défaut n'est pas activé après l'installation, pour l'activer :

```
younes-abou@labinfo12-Precision-T1700:~$ sudo service mongod start
younes-abou@labinfo12-Precision-T1700:~$ sudo service mongod status
● mongod.service - MongoDB Database Server
   Loaded: loaded (/lib/systemd/system/mongod.service; disabled; vendor preset:
   Active: active (running) since ن 2019-12-16 10:25:02 +01; 7s ago
     Docs: https://docs.mongodb.org/manual
 Main PID: 22988 (mongod)
   CGroup: /system.slice/mongod.service
           └─22988 /usr/bin/mongod --config /etc/mongod.conf

سري د16 10:25:02 labinfo12-Precision-T1700 systemd[1]: Started MongoDB Database S
```

Figure 7 : Activer le démon mongod pour se connecter au serveur

3.1.2 Invite de commande :

Comme la majorité des moteurs NoSQL, MongoDB offre une invite de commande interactive pour faciliter l'interaction directe avec les fichiers JSON, elle utilise une syntaxe similaire à la syntaxe JavaScript et offre un ensemble de fonctions permettant la manipulation des documents JSON. Les documents JSON sont regroupés dans ce qu'on appelle des collections, c'est l'équivalent d'une table dans un SGBDR. Pour accéder à l'invite de commande on exécute la commande mongo sur le terminale.

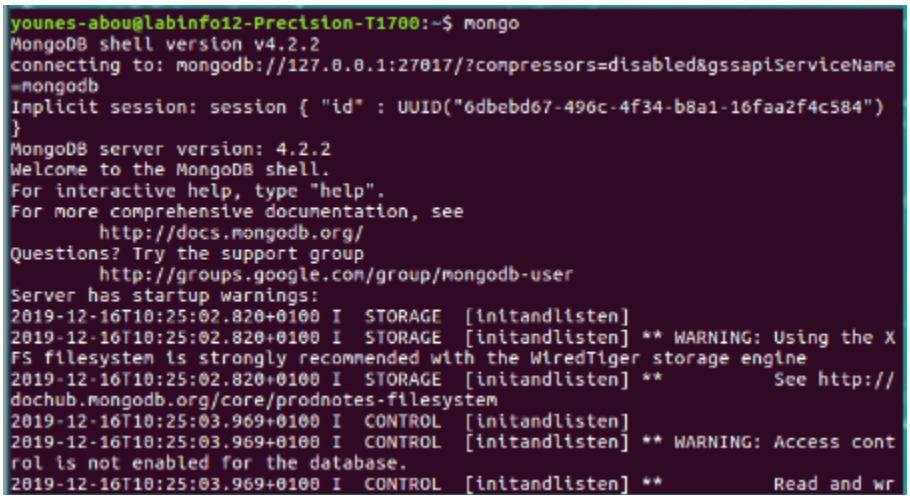


Figure 8 : l'exécution de la commande mongo

Pour basculer d'une base de donnée à une autre on utilise la commande use, et pour afficher les collections d'une base de données on utilise la commande show collections:

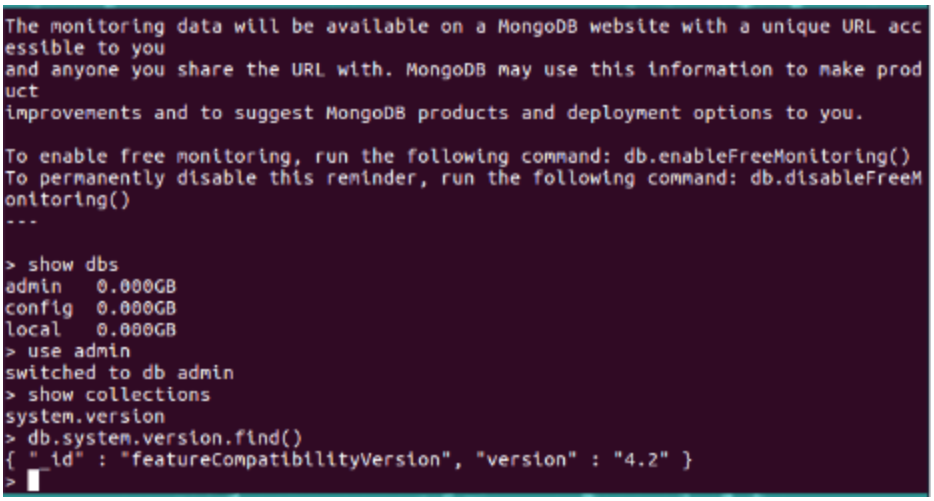


Figure 9 : les commandes use et show de l'invite de commande mongo

L'invite mongo offre un ensemble de fonctions pour manipuler les données, parmi les fonctions les plus utilisées on trouve : find, remove, update, insert, forEach.

La fonction find permet de récupérer un ou plusieurs documents d'une collection, s'ils correspondent à un critère de recherche donné à la fonction en argument. Elle s'agit de la fonction la plus utilisée.

La fonction insert permet d'ajouter un nouveau document JSON à une collection donnée. Contrairement au base de données relationnels, les données insérées ne respectent aucun schéma

prédéfini. En d'autre termes, on peut ajouter des documents avec des champs totalement différents sans aucune contrainte. Aussi, les documents d'une même collection peuvent ne pas avoir le même nombre de champs. Ci-dessous un exemple d'exécution des deux fonctions find et insert.

```
> show collections
articles
user
> db.user.find()
{ "_id" : ObjectId("5df7acff4c76518e5d9769a6"), "nom" : "ABOU", "prenom" : "Younes", "da
te_naissance" : "29/09/1998 23:15" }
> db.user.insert({"nom": "KAMAL" , "prenom" : "Hamza"})
WriteResult({ "nInserted" : 1 })
> db.user.find().pretty()
{
        "_id" : ObjectId("5df7acff4c76518e5d9769a6"),
        "nom" : "ABOU",
        "prenom" : "Younes",
        "date_naissance" : "29/09/1998 23:15"
}
{
        "_id" : ObjectId("5e0a9569b86302fe3d07fd8b"),
        "nom" : "KAMAL",
        "prenom" : "Hamza"
}
>
```

Figure 10 : L'exécution des fonctions insert et find

La fonction remove permet de supprimer un élément selon un critère donné. Sur l'exemple ci-dessous on va supprimer tous les éléments de la collection dont le nom est KAMAL.

```
> db.user.remove({"nom" : "KAMAL"})
WriteResult({ "nRemoved" : 1 })
> db.user.find().pretty()
{
        "_id" : ObjectId("5df7acff4c76518e5d9769a6"),
        "nom" : "ABOU",
        "prenom" : "Younes",
        "date_naissance" : "29/09/1998 23:15"
}
>
```

La fonction update permet de mettre à jour un ou plusieurs documents d'un collection ayant le même qui correspondant à un critère donné. La fonction prend deux arguments, le premier est le critère de sélection des documents et le deuxième est la modification qui faut effectuer aux documents sélectionnés. Dans l'exemple suivant on va modifier la date de naissance de tous les documents ayant le nom KAMAL. L'opérateur set est un parmi plusieurs opérateurs offert par mongoDB pour effectuer des modifications sur les documents JSON. Il

permet d'affecter la valeur spécifiée à côté de l'opérateur sur chacun des documents sélectionnées par la fonction update.

```
> db.user.update({"nom" : "KAMAL"}, {$set: {"date_naissance" : "07/07/1998"}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.user.find().pretty()
{
        "_id" : ObjectId("5df7acff4c76518e5d9769a6"),
        "nom" : "ABOU",
        "prenom" : "Younes",
        "date_naissance" : "29/09/1998 23:15"
}
{
        "_id" : ObjectId("5e0a9569b86302fe3d07fd8b"),
        "nom" : "KAMAL",
        "prenom" : "Hamza",
        "date_naissance" : "07/07/1998"
}
>
```

Figure 12 : L'exécution de la fonction update

L'invite de commande Mongo implémente une syntaxe JavaScript, et par conséquent offre plusieurs fonctions du langage, pour faciliter la manipulation de données. La fonction Foreach est de ces fonctions très pratiques. Elle permet d'appliquer une traitement sur un ensemble de documents souvent après une sélection avec la fonction find. Voyons un exemple concret qui permet d'afficher le nom et le prénom de tous les documents de la collection user.

```
> db.user.find().forEach(function(usr){print( "le nom de l'utilisateur : "+ usr.nom)})
le nom de l'utilisateur : ABOU
le nom de l'utilisateur : KAMAL
>
```

Figure 13 : L'exécution de la fonction forEach

3.1.3 L'aggrégation :

L'agrégation est manière d'appliquer des traitements sur les données afin de sortir avec un résultat signifiant. Il s'agit d'une des fonctionnalités importantes des SGBDs. MongoDB offre trois formes d'agrégation :

1. **Agregation par pipelines :**

Similaire au principe de tubes en Linux, l'agrégation par pipelines permet les documents d'une collection de passer par différents stages afin de sortir avec un résultat agrégé. Le schéma suivant montre en détail ce mécanisme d'agrégation :

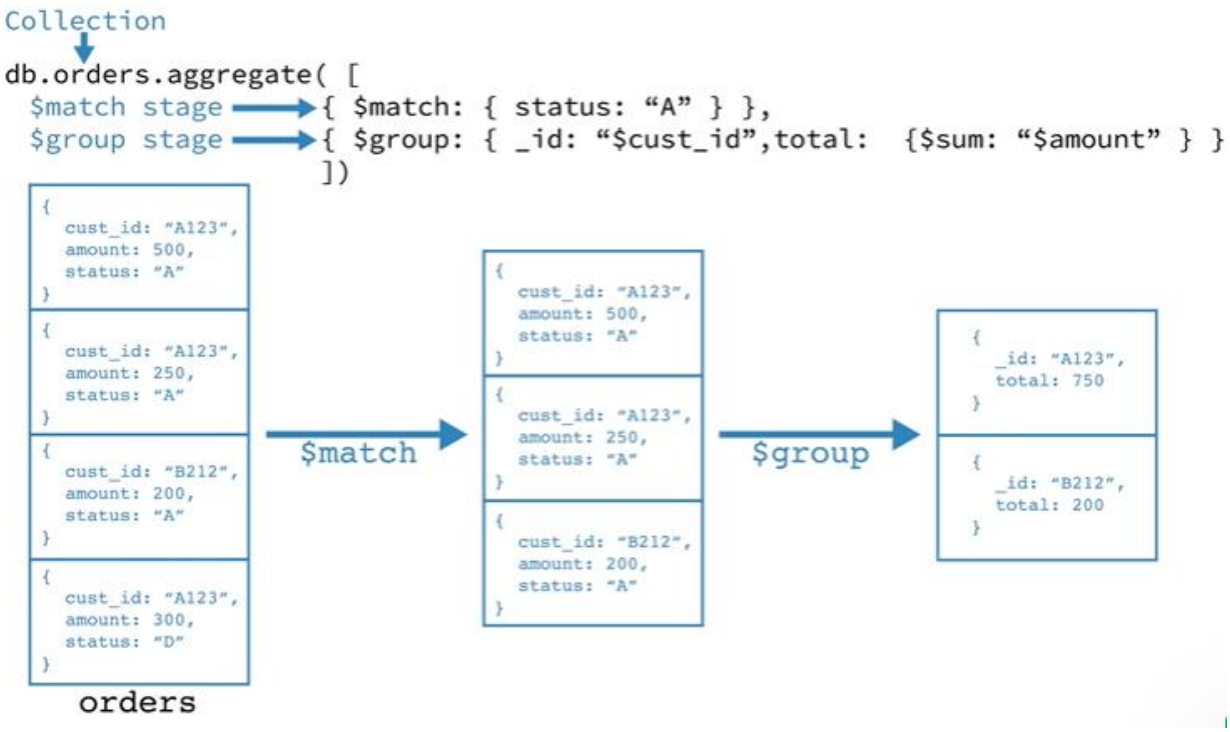


Figure 14 : le mécanisme d'agrégation par pipelines

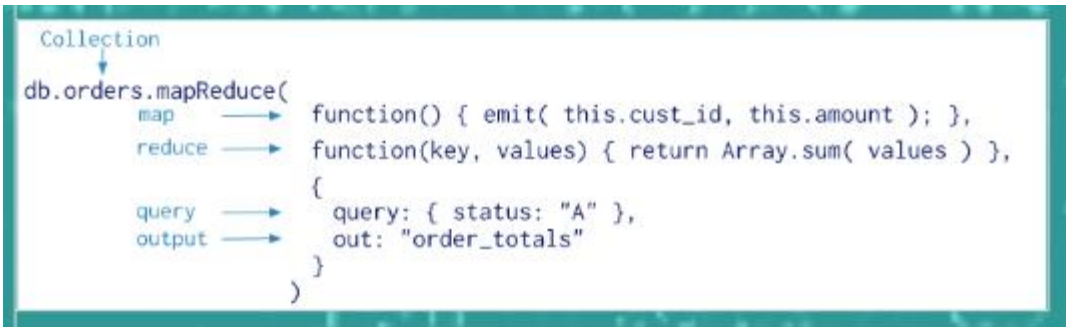
Dans cet exemple, la collection orders passe par deux niveaux :

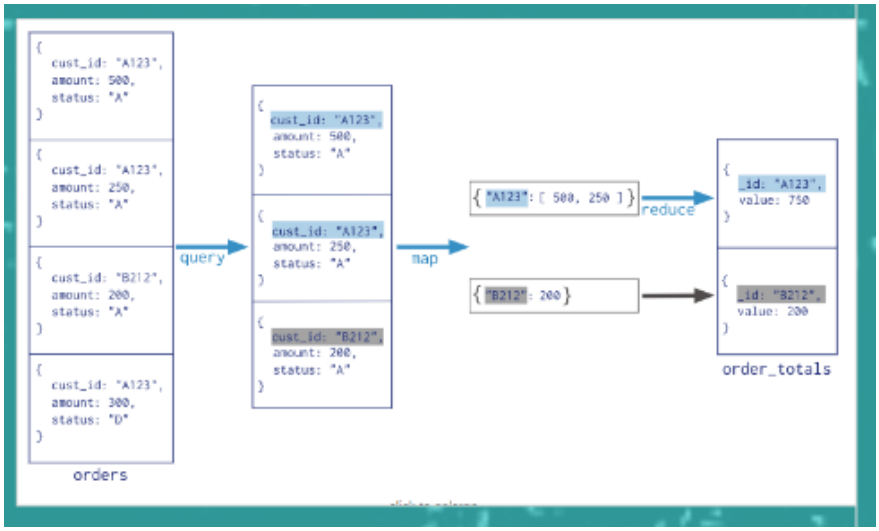
Le premier niveau : L'étape $match filtre les documents par le champ “statut” et passe à l'étape suivante les documents dont le statut est égal à "A".

Le deuxième niveau : L'étape $group regroupe les documents par le champ cust_id pour calculer la somme du montant pour chaque identifiant cust_id unique.

2. Agregation par MapReduce :

MongoDB fournit également des opérations de MapReduce pour effectuer l'agrégation. En général, les opérations MapReduce comportent deux phases : une étape de map qui traite chaque document et émet un ou plusieurs objets pour chaque document d'entrée, et réduit la phase qui combine la sortie de l'opération de mappage. Le cas échéant, MapReduce peut avoir une étape finalize pour apporter des modifications finales au résultat. Comme d'autres opérations d'agrégation, MapReduce peut spécifier une condition de requête pour sélectionner les documents d'entrée ainsi que pour trier et limiter les résultats. Le schéma suivant montre ce mécanisme d'agrégation :





3. Opérations d'agrégation prédéfinies :

MongoDB offre également des fonctions d'agrégation prédéfinies comme : db.collection.estimatedDocumentCount, db.collection.count et db.collection.distinct. Bien que ces opérations fournissent un accès simple aux processus d'agrégation communs, elles manquent de la souplesse et des capacités du pipeline d'agrégation et du MapReduce. L'exemple suivant montre l'exécution de la fonction db.collection.distinct.

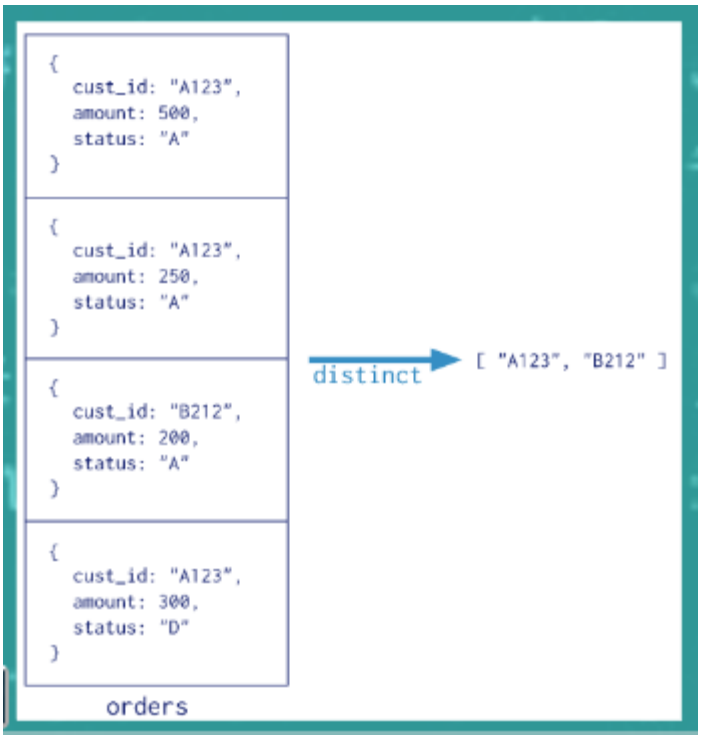


Figure 16 : L'exécution de la fonction distinct sur la collection orders

3.1.4 La programmation client :

Dans un environnement de production, la manipulation de données se fait rarement d'une manière manuelle. Il s'agit dans le plupart des cas d'une manière automatisée à travers des scripts de code. Ainsi, de nombreux langages de programmation offrent des pilotes dédiés à l'interrogation des serveurs MongoDB. l'exemple suivant utilisera le pilote Pymongo offert par Python pour effectuer une simple insertion dans une base mongodb.

On commencera d'abord par créer le document à insérer qu'on va appeler article :

```
#!/usr/bin/python
# coding=utf-8

from datetime import datetime

article = {
        "auteur": {
                "prenom": "Annie",
                "nom": "Brizard"
                },
        "titre": "Pourquoi les elephants ont-ils des grandes oreilles?",
        "mots-cles" : [ "faune", "afrique", "questions intrigantes"],
        "categories" : [ "science", "éducation", "nature"],
        "date_creation" : "12/05/2012 23:30",
        "statut" : "publié",
        "nombre_de_lectures" : "54",
        "commentaires" :
        [
                {
                        "date" : "14/05/2012 10:12",
                        "auteur" : "fred92@gmail.fr",
                        "contenu" : "il n'y a pas que les elephants qui ont des grandes oreilles xD"
                }
        ]
}
```

Figure 17 : Création d'un jeu de données pour l'insérer dans la base

Après le code suivant nous permettra de se connecter à la base de données "test" et d'insérer le document article dans la collection articles de la base. Si la collection n'existe pas, elle sera créée par MongoDB :

```
import sys
import pymongo
from pymongo import MongoClient
from pymongo.errors import ConnectionFailure

def main():
        try:
                cn = MongoClient()
                print "connection reussie"
        except ConnectionFailure, e:
                sys.stderr.write("Erreur de connection à mongodb: %s"% e)
                sys.exit(1)
        db = cn['test']
        db.articles.insert(article)
if __name__ == "__main__":
 main()
```

Figure 18 : Script permettant l'insertion d'un article dans la base de donnée test

On exécute le script à travers la commande python et on vérifie si les données ont bien été insérées :

```
younes-abou@younesabou-virtual-machine:~/python_env/nosql/bin$ sudo service mongod start
younes-abou@younesabou-virtual-machine:~/python_env/nosql/bin$ sudo service mongod start
[sudo] password for younes-abou:
younes-abou@younesabou-virtual-machine:~/python_env/nosql/bin$ python ~/insertion.py
connection reussie
```

Figure 19 : l'execution du script

```
> show dbs
admin   0.000GB
config  0.000GB
local   0.000GB
test    0.000GB
> use test
switched to db test
> show collections
articles
user
> db.articles.find().pretty()
{
        "_id" : ObjectId("5dfb5405aef44b3ae1836362"),
        "nombre_de_lectures" : "54",
        "auteur" : {
                "nom" : "Brizard",
                "prenom" : "Annie"
        },
        "statut" : "publié",
        "titre" : "Pourquoi les elephants ont-ils des grandes oreilles?",
        "commentaires" : [
                {
                        "date" : "14/05/2012 10:12",
                        "contenu" : "il n'y a pas que les elephants qui ont des grandes
```

Figure 20 : Vérification de la base de données

3.1.5 Replication et Sharding :

Les systèmes de base de données manipulant de grands volumes de données ou d'applications à haut débit peuvent dépasser la capacité d'un seul serveur. Par exemple, des taux de requêtes élevés peuvent épuiser la capacité CPU du serveur. La taille de données de travail supérieure à la RAM du système limite la capacité d'E/S des unités de disque. les moteurs NoSQL ont été conçu pour résoudre ce problème en facilitant le principe de distribution de données sur plusieurs machines et assurant une montée en charge horizontale. Étant un moteur NoSQL, MongoDB offre la possibilité de distribution à travers deux principes la réplication et le sharding.

- **La replication :**

La réplication est le processus de synchronisation des données sur plusieurs serveurs. La réplication assure la redondance et augmente la disponibilité des données avec plusieurs copies de données sur différents serveurs de base de données. La réplication protège une base de données contre la perte d'un seul serveur. La réplication vous permet également de se remettre des défaillances matérielles et des interruptions de service. Nous allons essayer dans cette partie de créer un replica set qui est tout simplement un ensemble d'instances mongod dont un est primaire et les autres sont secondaire. On ne peut écrire que sur l'instance primaire, qui va ensuite se charger de faire la réplication sur les instances secondaires. Voyons alors comment créer un replica set de trois membres.

On ouvre trois instances mongod en spécifiant un répertoire où on va enregistrer les données, le numéro du port avec lequel il va être connecté, et le nom de de l'ensemble de replica auquel il va appartenir.

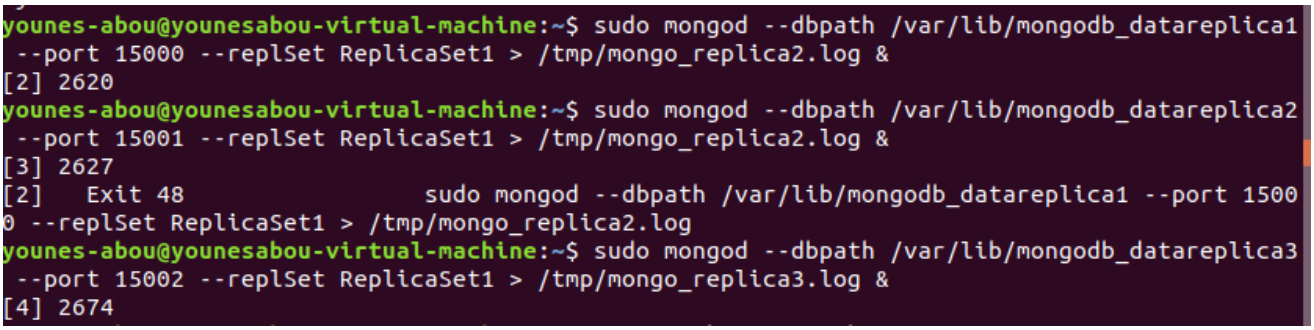


Figure 21 : Démarrer les membres de ReplicaSet 1

Maintenant, on se connecte à une des trois instances par la commande mongo :

Figure 22 : Se connecter au replica 1

Le replica set n'est pas encore configuré et par conséquent n'est pas opérationnel. Il faut spécifier le nom du replica set et ses instances membres à l'aide de la fonction rs.initiate.

```
> config = {_id: "ReplicaSet1", members: [
... {_id: 0, host: "localhost:15000"},
... {_id: 1, host: "localhost:15001"},
... {_id: 2, host: "localhost:15002"}]};
{
        "_id" : "ReplicaSet1",
        "members" : [
                {
                        "_id" : 0,
                        "host" : "localhost:15000"
                },
                {
                        "_id" : 1,
                        "host" : "localhost:15001"
                },
                {
                        "_id" : 2,
                        "host" : "localhost:15002"
                }
        ]
}
> rs.initiate(config);
{
        "ok" : 1,
```

Figure 22 : Configuration du replica set

Maintenant on remarque que le curseur de l'invite affiche PRIMARY ce que signifie qu'on est connecté au membre primaire de ReplicaSet 1, pour vérifier on exécute la fonction rs.config

Figure 23 : Vérifier la configuration du replica set avec rs.config

Maintenant si on se connecte avec un noeud secondaire et on exécute la fonction rs.isMaster() on va voir que le champs " ismaster " est false ce qui signifie que ce noeud n'est qu'un esclave.

Figure 24 : la fonction isMaster() sur un noeud secondaire

Si on arrête le noeud primaire le basculement se fait automatiquement et un parmi les noeuds secondaires est élu pour être le maître.

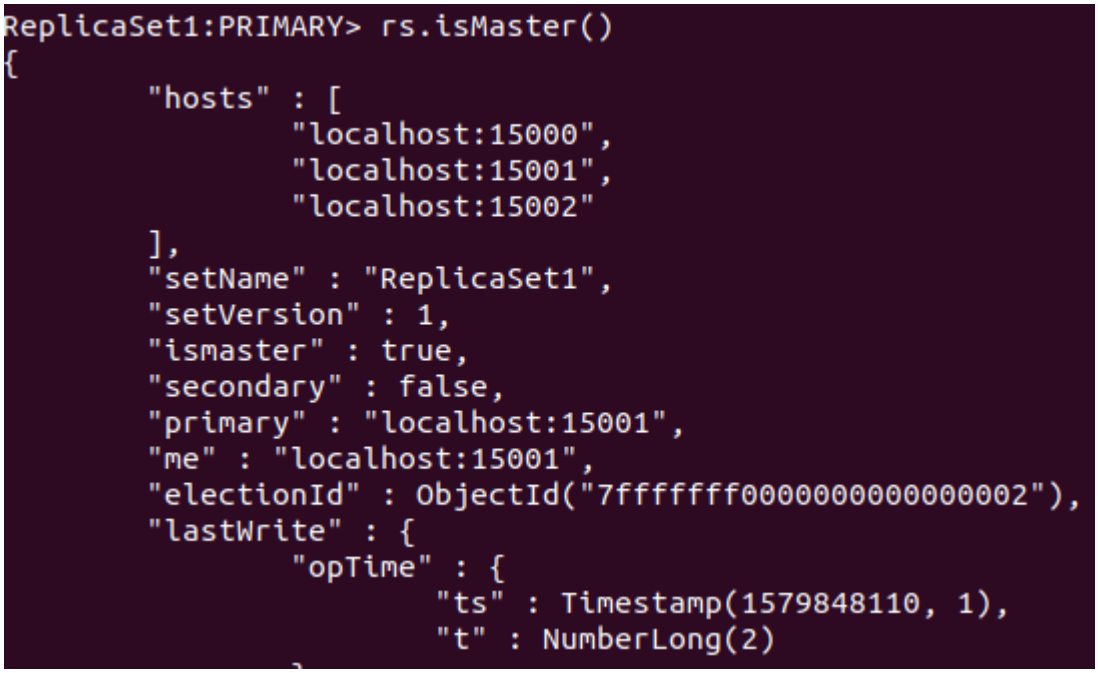


Figure 25 : L'élection noeud secondaire après arrêt du noeud primaire

- **Sharding :**

Bien que MongoDB peut s'exécuter en tant que moteur individuel en une seule machine, la vrai puissance du moteur est quand il est converti en cluster et distribué en plusieurs machines. MongoDB offre cette possibilité à travers le mécanisme du sharding qui consiste à répartir les

données sur plusieurs serveur ce qui supporte la montée en charge rapide d'une manière très efficace. Pour des raisons de test on va effectuer une répartition sur une seule machine. Mais dans un environnement de production chaque shard est installé sur une machine. La puissance de MongoDB au niveau de distribution de données est le fait que c'est MongoDB qui gère automatiquement la répartition, le routage de requêtes et la récupération de données à l'aide de du routeur mongos. Pour créer une répartition on procède comme suit :

1. Créer les shards qui sont tout simplement des groupes de réplicas où on va répartir les données. le nombre maximum de shards peut aller théoriquement jusqu'à mille. Dans l'exemple suivant on va créer des groupes deux shards composés d'un seule membre.

```
younes-abou@younesabou-virtual-machine:~$ sudo mkdir /var/lib/shardRep1 /var/lib/shardRep2
younes-abou@younesabou-virtual-machine:~$ chmod a=rwx /var/lib/shardRep1
chmod: changing permissions of '/var/lib/shardRep1': Operation not permitted
younes-abou@younesabou-virtual-machine:~$ sudo chmod a=rwx /var/lib/shardRep1
younes-abou@younesabou-virtual-machine:~$ sudo chmod a=rwx /var/lib/shardRep2
younes-abou@younesabou-virtual-machine:~$ mongod --shardsvr --replSet ShardReplica1 --dbpath
 /var/lib/shardRep1 --bind_ip localhost --port 16000 > /tmp/shard1.log &
[1] 4425
```

Figure 26 : La creation de deux instances mongod representant deux shards

2. Créer un serveur de configuration qu'il s'agit également d'un groupe de replica composé généralement de trois membres et il contient les métadonnées des différents shards du cluster.

```
younes-abou@younesabou-virtual-machine:~$ mongod --configsvr --replSet config_replicaSet --dbpath
 /var/lib/config_replica/config_replicaSet1 --bind_ip localhost --port 11111 > /tmp/config.log &
[3] 4663
```

Figure 27 : La création du serveur de configuration

3. Créer un serveur mongos en lui attribuant les serveurs de configurations créés précédemment

```
younes-abou@younesabou-virtual-machine:~$ sudo mongos --configdb config_replicaSet/localhost:11111
--bind_ip localhost --port 27018
[sudo] password for younes-abou:
2020-01-24T13:33:09.608+0100 W  SHARDING [main] Running a sharded cluster with fewer than 3 config
servers should only be done for testing purposes and is not recommended for production.
```

Figure 28 : Création du serveur mongos

4. Se connecter au serveur mongos pour configurer les shards

```
younes-abou@younesabou-virtual-machine:~$ mongo --host localhost:27018
MongoDB shell version v4.2.2
connecting to: mongodb://localhost:27018/?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("fddc0046-4428-4164-b9b3-cb31f52b7861") }
MongoDB server version: 4.2.2
Server has startup warnings:
2020-01-24T13:33:09.665+0100 I  CONTROL  [main]
2020-01-24T13:33:09.665+0100 I  CONTROL  [main] ** WARNING: Access control is not enabled for the
 database.
2020-01-24T13:33:09.665+0100 I  CONTROL  [main] **          Read and write access to data and con
figuration is unrestricted.
2020-01-24T13:33:09.665+0100 I  CONTROL  [main] ** WARNING: You are running this process as the r
oot user, which is not recommended.
2020-01-24T13:33:09.665+0100 I  CONTROL  [main]
mongos>
```

Figure 29 : Se connecter à mongos

5. Pour configurer les shards à mongos on utilise la fonction sh.addShard()

```
mongos> sh.addShard("ShardReplica1/localhost:16000")
{
        "shardAdded" : "ShardReplica1",
        "ok" : 1,
        "operationTime" : Timestamp(1579871277, 2),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1579871277, 2),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        }
}
mongos> sh.addShard("ShardReplica2/localhost:16001")
{
        "shardAdded" : "ShardReplica2",
        "ok" : 1,
        "operationTime" : Timestamp(1579871296, 5),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1579871296, 5),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
```

Figure 30 : Ajouter les shards à mongos

Maintenant, pour commencer la répartition d'une collection il faut d'abord activer l'option de répartition sur la base de données concernée avec la fonction sh.enableSharding(). Puis effectuer le sharding sur une collection avec la fonction sh.shardCollection().

```
mongos> show dbs
admin   0.000GB
config  0.001GB
mongos> sh.enableSharding("test")
{
        "ok" : 1,
        "operationTime" : Timestamp(1579872242, 5),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1579872242, 5),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        }
}
mongos> sh.shardCollection("test.articles", {date_creation: 1, categories: 1})
{
        "collectionsharded" : "test.articles",
        "collectionUUID" : UUID("ea715c69-5a45-4f74-94ee-0ff67e7c980c"),
        "ok" : 1,
        "operationTime" : Timestamp(1579872593, 13),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1579872593, 13),
                "signature" : {
```

Figure 31 : les fonctions permettant le sharding

On peut vérifier si le sharding est effectivement activé par afficher la collection “collections” de la base de données config.

```
mongos> use config
switched to db config
mongos> show collections
changelog
chunks
collections
databases
lockpings
locks
migrations
mongos
shards
system.sessions
tags
transactions
version
mongos> db.collections.find()
{ "_id" : "config.system.sessions", "lastmodEpoch" : ObjectId("5e2aed3c4d98bbd0a8cf452e"), "lastm
od" : ISODate("1970-02-19T17:02:47.296Z"), "dropped" : false, "key" : { "_id" : 1 }, "unique" : f
alse, "uuid" : UUID("f36ff5c7-771b-429e-86d5-3af7fe83d4d7") }
{ "_id" : "test.articles", "lastmodEpoch" : ObjectId("5e2af151f2c3edc6da4dea2f"), "lastmod" : ISO
Date("1970-02-19T17:02:47.296Z"), "dropped" : false, "key" : { "date_creation" : 1, "categories"
: 1 }, "unique" : false, "uuid" : UUID("ea715c69-5a45-4f74-94ee-0ff67e7c980c") }
mongos>
```

Figure 32 : Afficher les collections distribuées

3.2 Redis :

Redis est une abréviation du terme Remote Dictionary Server. Il s'agit d'un serveur de structure de données non relationnel développé en C, ce qui le permet d'être très rapide, très efficace et très léger. Redis est un moteur de base de données en mémoire (in-memory data store) ce qui

signifie que contrairement à la plupart des base de données qui stockent leur données sur le disque, les données Redis résident entièrement dans la RAM et par conséquent élimine l'accès multiple au disque et maximise les performances du moteur. la persistance de données est configurable en Redis, ce qui le permet d'effectuer du backup ou de fonctionner comme une base de données ordinaire, même si ce n'est pas le cas d'usage le plus répandu.

3.2.1 Schéma de données :

Contrairement au MongoDB qui manipule des documents en format JSON, Redis travaille en paires clé-valeur, dont les valeurs peuvent être de différents types, cinq types de données pour être précis.

les chaînes : Les chaînes sont le type de valeur Redis le plus basique. Les chaînes Redis sont Binary Safe, ce qui signifie qu'une chaîne Redis peut contenir n'importe quel type de données, par exemple une image JPEG ou des valeurs numériques. Une valeur de type String peut avoir une longueur maximale de 512 mégaoctets.

les listes : Les listes Redis sont simplement des listes de chaînes, triées par ordre d'insertion. Il est possible d'ajouter des éléments à une liste Redis en poussant de nouveaux éléments sur la tête (à gauche) ou sur la queue (à droite) de la liste. La commande LPUSH insère un nouvel élément sur la tête, tandis que RPUSH insère un nouvel élément sur la queue.

les ensembles : les ensembles représentent des collections non triées de chaînes . Il est possible d'ajouter, de supprimer et de tester l'existence de membres avec une complexité d'ordre constant quel que soit le nombre d'éléments contenus dans l'ensemble, ce qui rend les opérations sur les ensembles Redis très performantes. Les ensembles Redis ont la propriété de ne pas autoriser les membres répétés. L'ajout d'un même élément plusieurs fois entraîne l'obtention d'une copie unique de cet élément. En pratique, cela signifie que l'ajout d'un membre ne nécessite pas de vérification s'il existe avant de l'ajouter. Ce qui est très intéressant avec les ensembles en Redis, c'est qu'ils offrent un certain nombre de commandes pour effectuer des opérations mathématiques sur les ensembles telles que l'union, l'intersection et la différence.

les Hachages : ils représentent une table de hachage, ce qui permet de stocker une représentation d'objet, similairement à un document JSON. Un hachage avec une centaine de

champs est stocké d'une manière qui prend très peu d'espace, de sorte que vous pouvez stocker des millions d'objets dans une petite instance Redis.

les ensembles triés : Les ensembles triées Redis sont, tout comme les ensembles, des collections non répétées de chaînes. La différence est que chaque membre d'un ensemble trié est associé à un score, qui est utilisé pour trier l'ensemble, du plus petit au plus grand score. Avec les ensembles triées, on peut ajouter, supprimer ou mettre à jour des éléments de manière très rapide.

3.2.2 Installation :

l'installation de Redis est plus simple que celle de MongoDB il suffit juste de télécharger le fichier compressé depuis le site officiel de redis : https://redis.io/download. et le décompresser puis le compiler.

pour s'assurer que tout marche bien on exécute la commande `redis-cli ping`, le serveur est activé si ça retourne `PONG`.

3.2.3 Invite de commande :

Tout comme MongoDB, Redis offre une invite de commande client pour interroger le serveur, appelée redis-cli. elle vient avec un très grand nombre de commandes pour manipuler les différents types de données supportées par Redis. les captures suivantes présente quelques unes :

```
127.0.0.1:6379[1]> select 1
OK
127.0.0.1:6379[1]> hmset article:1 prenom "Annie" nom "Brizard" email "brizard.gmail.com" titre
 "Ce sont les girafes qui ont de grandes oreilles"
OK
127.0.0.1:6379[1]> SADD tag:animeaux 1
(integer) 1
127.0.0.1:6379[1]> SADD tag:questions 1
(integer) 1
127.0.0.1:6379[1]> SADD article:1:tags animaux questions
(integer) 2
127.0.0.1:6379[1]> SINTER tag:animaux tag:questions
(empty list or set)
127.0.0.1:6379[1]> SINTER tag:animeaux tag:questions
1) "1"
127.0.0.1:6379[1]> SADD tag:animaux 1
(integer) 1
127.0.0.1:6379[1]> SINTER tag:animaux tag:questions
1) "1"
```

Figure 33 : Les commandes SADD et SINTER

```
127.0.0.1:6379[1]> HVALS article:1
1) "Annie"
2) "Brizard"
3) "brizard.gmail.com"
4) "Ce sont les girafes qui ont de grandes oreilles"
127.0.0.1:6379[1]> HGET article:1 email
"brizard.gmail.com"
```

Figure 34 : Les commandes HVALS et HGET

```
127.0.0.1:6379[1]> HGET article:1 email
"brizard.gmail.com"
127.0.0.1:6379[1]> MULTI
OK
127.0.0.1:6379[1]> RENAMENX article:1 article:2
QUEUED
127.0.0.1:6379[1]> SREM tag:animaux 1
QUEUED
127.0.0.1:6379[1]> SREM tag:questions 1
QUEUED
127.0.0.1:6379[1]> SADD tag:animaux 2
QUEUED
127.0.0.1:6379[1]> SADD tag:questions 2
QUEUED
127.0.0.1:6379[1]> EXEC
1) (integer) 1
2) (integer) 1
3) (integer) 1
4) (integer) 1
5) (integer) 1
127.0.0.1:6379[1]> TYPE article:2
hash
127.0.0.1:6379[1]>
```

Figure 35 : Le principe du Queuing

3.2.4 Programmation client :

Contrairement aux autres moteurs NoSQL, qui offrent pratiquement les mêmes fonctionnalités que les SGBDR, comme on a vu avec MongoDB, Redis représente un cas un peu spécial des moteurs NoSQL et ses cas d'usage sont assez différentes. Redis n'est pas souvent utilisé comme un SGBD à part entière, il travaille plutôt avec d'autres SGBDR comme Oracle, MySQL, Postgres ou même avec d'autres moteurs NoSQL comme DynamoDB et MongoDB, afin d'offrir un espace de cache permettant l'amélioration des performance lors du requêtes de ces base de données. Nous allons essayer dans cette partie de visualiser l'effet de cache à travers un script

Node Js, qui récupère des données a partir de l'API de github, et le stocke temporairement dans une mémoire de cache sous Redis. Le script de code est le suivant :

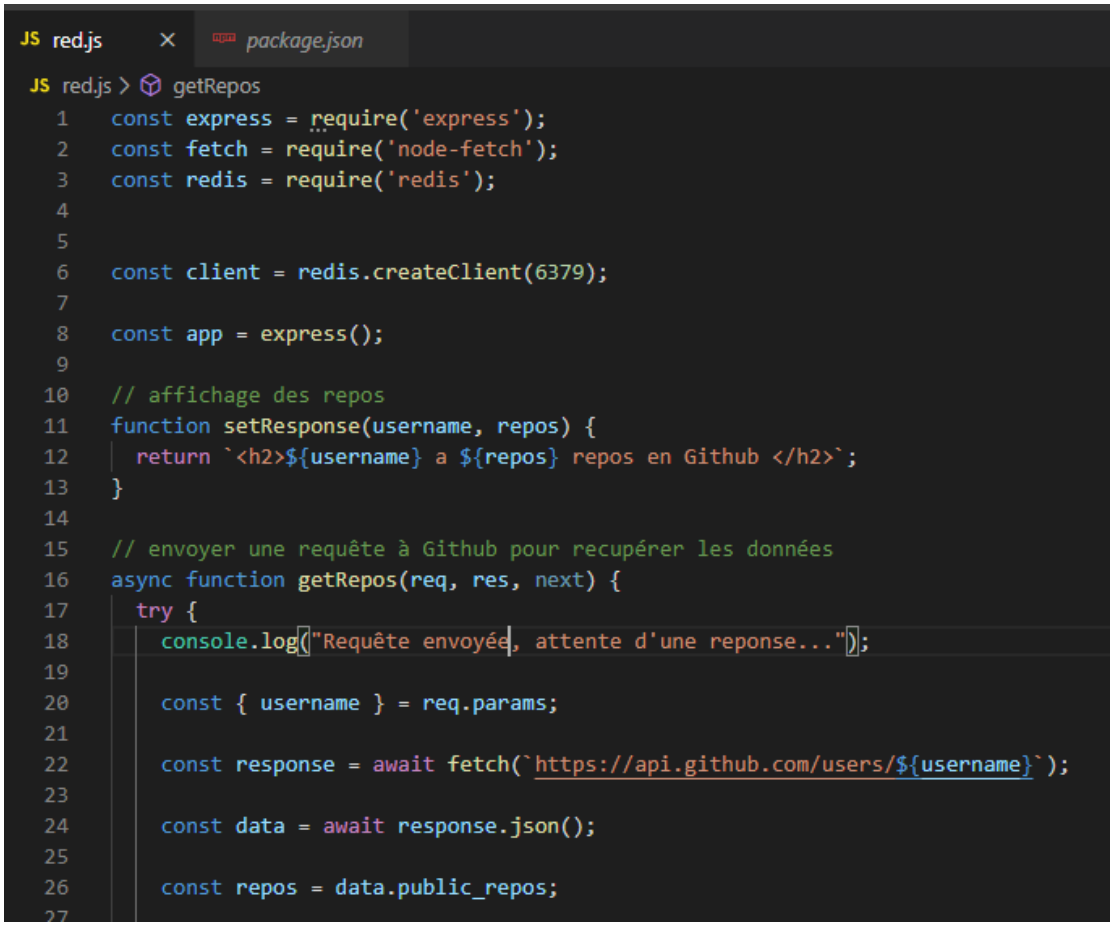


Figure 36 : Code permettant de créer une mémoire de cache Redis

Figure 37 : suite du code de la figure 24

Quand on compare le temps de la première exécution de la méthode GET, qui permet de récupérer les données de l'api, avec le temps effectué en deuxième reprise, on constate que la deuxième exécution est pratiquement 100 fois plus rapide. Les résultats du test sont présentés dans les deux captures suivantes :

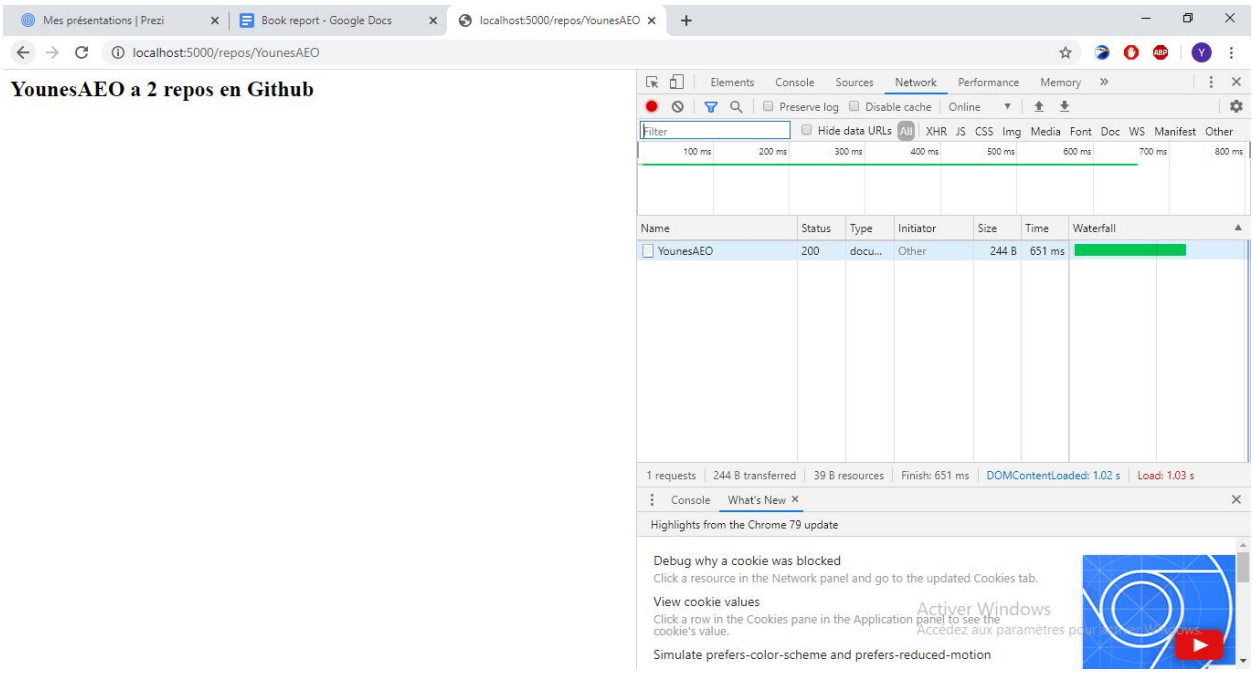


Figure 38 : La première exécution du script Node Js (651 ms)

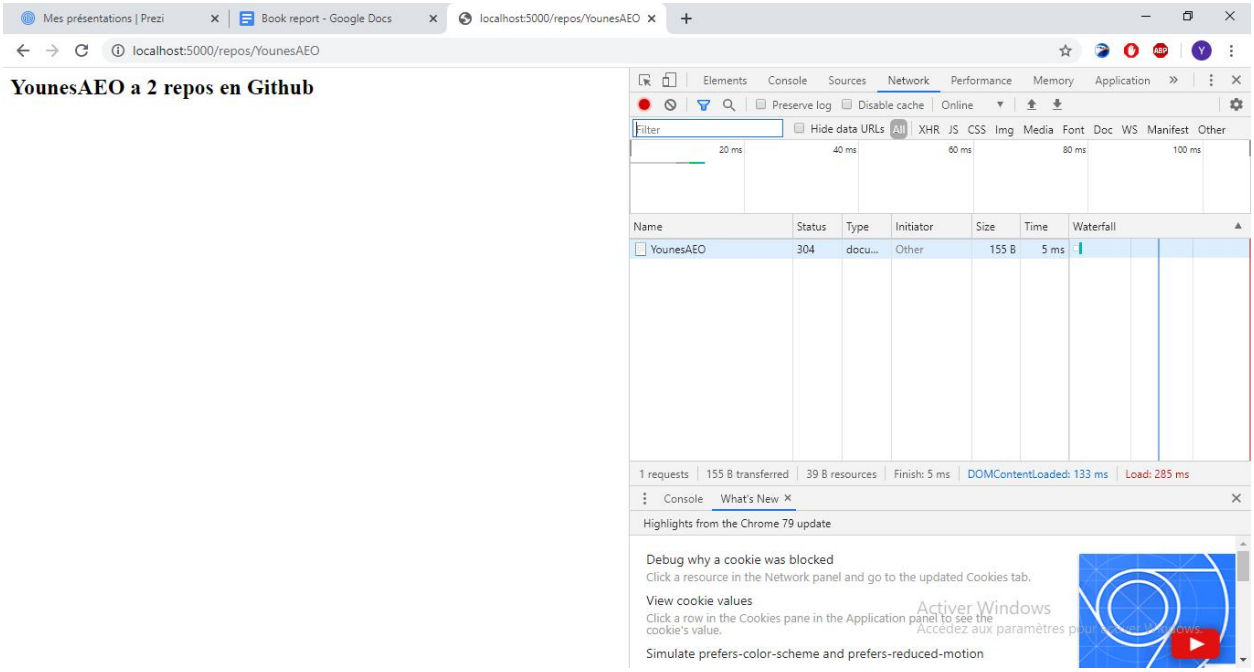


Figure 39 : La deuxième exécution après stockage sur Redis (5ms)

3.3 Neo4j :

Neo4j est notre dernier moteur NoSQL, et il s'agit d'un moteur très particulier dans le sens où il manipule une structure de données très différentes des structures qu'on a vu jusqu'à maintenant telles que les tables, les paires clé-valeur..etc. Neo4j stocke les données sous la forme d'un graphe. Ce qu'on entend par un graphe est tout simplement un noeud ou plusieurs noeuds reliés entre eux par des liens. Neo4j offre d'excellentes performances surtout au niveau de la recherche même avec un très grand nombre de noeuds, grâce à l'implémentation des algorithmes spécifiques aux graphes. Le moteur offre son propre langage, inspiré du langage SQL, pour exprimer les requêtes, appelé Cypher.

3.3.1 Installation :

L'installation est pratiquement identique à l'installation de MongoDB. Ca commence par récupérer la clé publique avec la commande :

```
wget -O - https://debian.neo4j.com/neotechnology.gpg.key | sudo apt-key add -
```

la création de fichier .list qui contient le dépôt de nos sources de paquets :

```
echo 'deb https://debian.neo4j.com stable latest' | sudo tee -a
/etc/apt/sources.list.d/neo4j.list
```

Puis la mise à jour du gestionnaire de paquets et l'installation des paquets :

```
sudo apt-get update && sudo apt-get install neo4j=1:4.0.0
```

3.3.2 Interface graphique :

Neo4j offre une interface graphique pour manipuler les données et aussi visualiser les noeuds créés. Pour y accéder, il faut l'ouvrir dans le browser avec une adresse localhost et le port dont écoute le serveur web d'administration par défaut dans notre cas le port 7474.

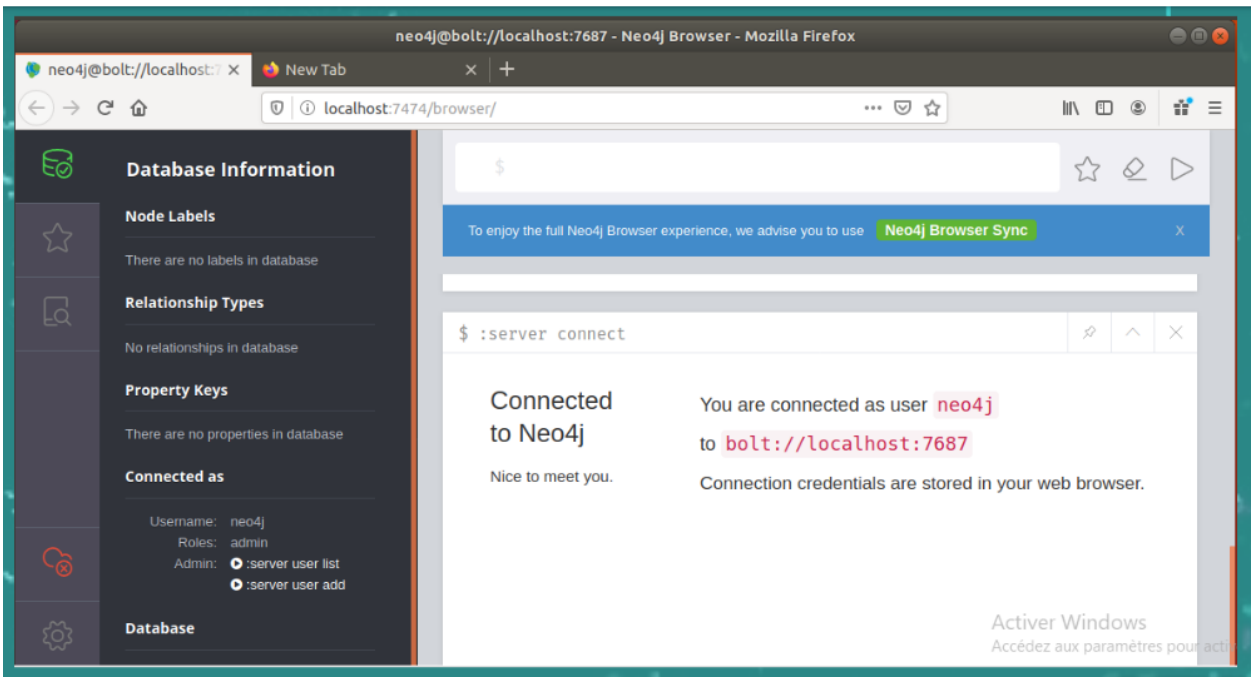


Figure 40 : L'interface web de Neo4j

La création des noeuds se fait à travers l'invite située en haut, les requêtes exprimées en langage Cypher sont envoyées au serveur. Le serveur vérifie la syntaxe de la requête et renvoie le résultat de la requête ou un message d'erreur dans le cas échéant. La commande create permet de créer une nouveau noeud.

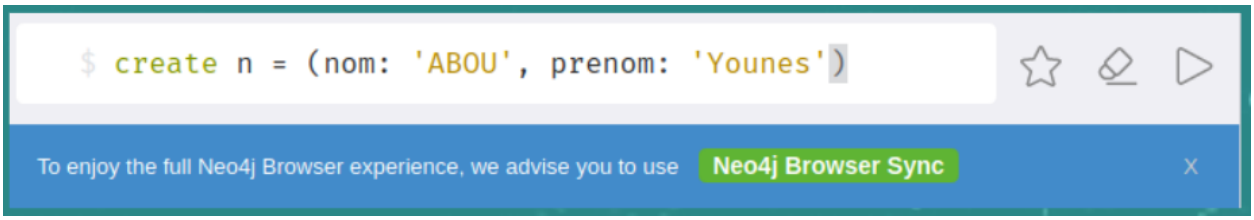


Figure 41 : Création d'un nouveau noeud

Pour créer une relation entre deux noeuds :

```
1 MATCH (ABOU:Personne),(KAMAL:Personne)
2 WHERE ABOU.nom = 'ABOU' AND KAMAL.nom = 'KAMAL'
3 CREATE (ABOU)-[r:RELTYPE:{name: 'AMI'}]→(KAMAL)
4 RETURN type(r)
```

Figure 42 : Création d'une nouvelle relation entre noeud KAMAL et ABOU

Ce qui intéressant avec l'interface graphique c'est la possibilité de visualiser les données sous forme de graphes.

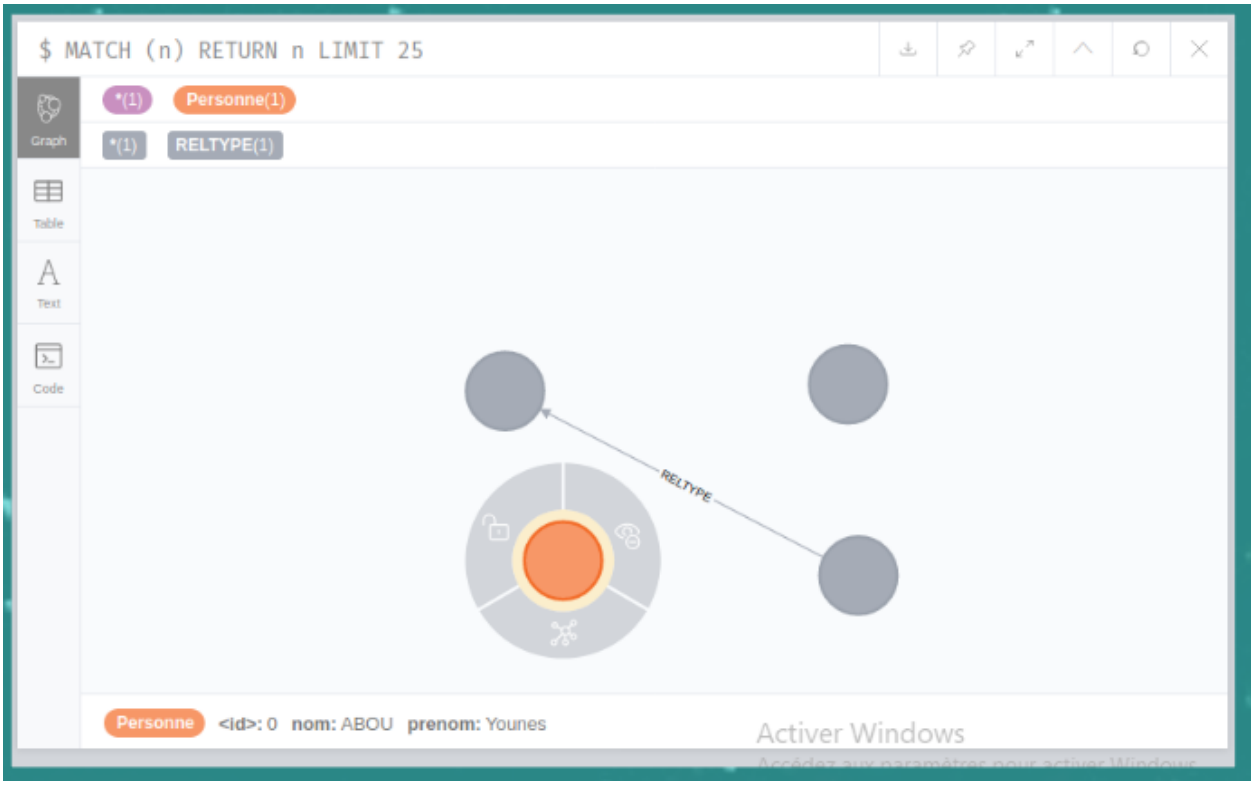


Figure 43 : Visualisation d'un graphe en Neo4j

Conclusion

Pour conclure, la lecture de ce livre m'a permis de réaliser que le NoSQL est un domaine très vaste et riche de technologies prometteuses dont chacune essaye de résoudre un problème spécifique ou encore une limitation d'un système existant. Cette impulsion vers de nouvelles solutions a permis le développement du NoSQL d'une manière très rapide et de s'ouvrir à de nouvelles possibilités qui ont jamais étaient atteignable par les solutions classiques.

Ce projet a été une excellente opportunité pour moi d'apprendre de nouveaux concepts et d'étendre mon domaine de connaissances sur des sujets d'actualité dans le monde informatique. Il m'a permis également de développer mes compétences de rédaction, de prise de parole et d'analyse et de sélection des informations. Sans négliger les nombreuses compétences techniques que j'ai acquis sur des technologies jamais étudiées auparavant.

Des remerciements spéciaux sont adressés à notre professeur, M.Mohammed BOUTABIA pour cette opportunité ainsi que son accompagnement et ses orientations précieuse.

Bibliographie & Webographie

- Rudi Bruchez, “NoSQL et le Big Data : Comprendre et mettre en oeuvre “, Edition Eyrolles, 2015.
- Documentation de MongoDB : https://docs.mongodb.com
- Documentation de Redis : https://redis.io/documentation
- Documentation de Neo4j : https://neo4j.com/docs/
- NodeJs : https://nodejs.org/en/
- Documentation PyMongo de Python : https://api.mongodb.com/python/current/api/index.html