République Algérienne Démocratique et Populaire
Université Abou Bakr Belkaid– Tlemcen
Faculté des Sciences
Département d'Informatique

Mémoire de fin d'études

pour l'obtention du diplôme de Master en Informatique

Option: Réseaux et Systèmes Distribues (RSD)

Thème

Migration d'une base de données relationnelle Vers une base de données distribuée NoSQL

Réalisé par :

-BENMIA Imane

Présenté le …. Juin 2015 devant le jury composé de MM.

- …….. *(Président)*
- …….. *(Encadreur)*
- …….. *(Examinateur)*
- …….. *(Examinateur)*

Année universitaire : 2014-2015

Remerciements

Avec un grand plaisir je remercie Allah qui m'a aidé et m'a donné la Patience, le courage et la force d'achever ce travail.

Je tiens à remercier sincèrement Mr Houcine MATALLAH, qui en tant qu'encadreur de ce PFE, s'est toujours montré à l'écoute et très disponible tout au long de la réalisation de ce travail, ainsi pour l'orientation, la confiance, l'aide et le temps qu'il a bien voulu me consacrer.

J'exprime également ma gratitude aux membres de jury, qui m'ont honoré en acceptant de juger ce modeste travail.

Mes sincères remerciements à tous nos professeurs du département d'informatique de la faculté des sciences à Tlemcen

Dédicaces

Toutes les lettres ne sauraient trouver les mots qu'il faut . . .

Tous les mots ne sauraient exprimer la gratitude, l'amour, le respect, la reconnaissance . . .

Aussi, c'est tout simplement que . . .

Je dédie ce mémoire à . . .

Ma très chère mère

Vous présentez pour moi le symbole de la bonté par excellence, la source de tendresse et l'exemple du dévouement qui n'a pas cessé de m'encourager et de prier pour moi.

Vos prières et vos bénédictions m'ont été d'un grand secours pour bien mener mes études. Aucune dédicace ne saurait être assez éloquente pour exprimer ce que vous méritez pour tous les sacrifices que vous n'avez cessé de me donner depuis ma naissance, durant mon enfance et même à l'âge adulte.

Vous avez fait plus qu'une mère qui puisse faire pour que ses enfants suivent le bon chemin dans leurs vies et leurs études.

Je vous dédie ce travail en témoignage de mon profond amour.

Ma très chère grande mère

Ce travail est pour moi le fruit de vos prières, je vous souhaite

Une longue vie pleine de joie.

Mes très chère sœurs et frères

Je vous dédie ce travail avec tous mes vœux de bonheur, de santé et de

Réussite.

Sommaire

Introduction générale

Le Big Data est un ensemble de techniques permettant le passage à l'échelle en volumes, en nombres et en types de données. La révolution scientifique qui envahisse le monde de l'information et l'Internet a imposé aux différents chercheurs depuis quelques années, de nouveaux défis et les a poussés à concevoir de nouveaux outils de stockage et de manipulation spécifiques.

Le développement de ces outils suscite un intérêt grandissant auprès des acteurs scientifiques et économiques pour leur offrir la possibilité de gérer toutes ces masses de données avec des temps de réponses raisonnables.

Le Big Data corrèle entre 4 notions regroupées généralement sous l'acronyme "concept 4V", à savoir :

- La notion de **V**olume qui décrit le phénomène selon lequel les quantités de données stockées dans les entreprises sont entrain de passer de l'ordre des téraoctets vers l'ordre de pétaoctets;
- La notion de **V**ariété des types de données qui décrit un monde de l'entreprise ouvert dans lequel les sources d'information sont multiples, hétérogènes et souvent non structurées;
- La notion de **V**élocité qui impose aux entreprises de traiter rapidement les données pour faire le lien avec les outils de reporting (Business Intelligence) indispensable à la prise de décision dans l'entreprise;
- La notion de **V**ariabilité de format et de sens des données qui peuvent varier au fil du temps, provoqué par le changement des formats des sources des données, impose aux entreprises de disposer d'un système d'information en perpétuelle évolution.

Les problématiques du Big Data s'inscrivent dans un contexte complexe, à la croisée de 2 préoccupations majeures :

- La mise en œuvre de nouvelles solutions de stockage de masse ;
- La capture d'information à grande vitesse et si possible en temps réel ;

Pour répondre à ces trois préoccupations, il faut développer des solutions technologiques comprenant :

- Des outils innovants de restitution ;
- Des outils innovants de visualisation de données adaptés aux volumétries ;
- De nouvelle solution de stockage.

Dans ce contexte, les bases de données NoSQL offrent de nouvelles solutions de stockage dans les environnements à grande échelle, en substitution des différents systèmes de gestion de bases de données traditionnelles dont une grande partie est relationnelle.

Introduction générale

Beaucoup d'utilisateurs des SGBD classiques dits « SQL » veulent basculer vers ces nouvelles solutions « NoSQL » pour anticiper l'explosion de leurs données dans le proche futur, néanmoins ils ne veulent certainement pas commencer avec des bases vides mais chercher comment récupérer les données hébergées dans leurs bases de données relationnelles.
En réponse à cette problématique et dans le cadre de notre PFE, nous proposons une approche de migration d'une base de données relationnelle classique vers une base de données NoSQL. La démarche proposée permettra de :

- Manipuler et héberger un système NoSQL choisi.
- Maîtriser le passage des bases de données classiques vers des bases de données NoSQL. Ce passage permet de proposer un algorithme (des règles) de passage.
- Exploiter cette masse de données qualifiée de *Big Data* par l'utilisation du *Map/Reduce*.

Notre mémoire est organisé comme suit : nous présentons dans le premier chapitre le principe de fonctionnement les bases de données relationnelles ainsi que leurs limites.
Le deuxième chapitre, va être consacré à la technologie NoSQL, les différents types de bases de données NoSQL ainsi que leurs avantages et inconvénients.

Dans le troisième chapitre, la solution dans laquelle on va migrer nos données, va être développée, à savoir MongoDB.

Le quatrième chapitre va faire objet de l'implémentation et la mise en œuvre de la migration du SGBD relationnel vers le NoSQL.

Enfin, nous clôturons cette étude par une conclusion générale sur le travail effectué ainsi que les perspectives éventuelles de cet axe de recherche très prometteur.

CHAPITRE 1 :

Base de données relationnelles

1. Introduction

Les technologies de bases de données relationnelles et transactionnelles, qu'on résumera ici par "Technologies SQL", règnent en maîtres pour le stockage et la manipulation de données depuis plus de 20 ans. Elles consisteront à stocker les informations décomposées et organisées dans des matrices appelées relations ou tables conformément au modèle de données relationnel. Le contenu de la base de données peut ainsi être synthétisé par des opérations d'algèbres relationnelles telles que l'intersection, la jointure ou le produit cartésien.

2. Principe de fonctionnement de SGBD relationnel

Afin de pouvoir contrôler les données ainsi que les utilisateurs, on a besoin de système de gestion de base de données. Le SGBD est un ensemble de logiciels informatiques qui sert à manipuler les bases de données, à effectuer des opérations ordinaires telles que consulter, modifier, construire, transformer, copier, sauvegarder ou restaurer des bases de données.

Le SGBD peut se décomposer en trois sous-systèmes :

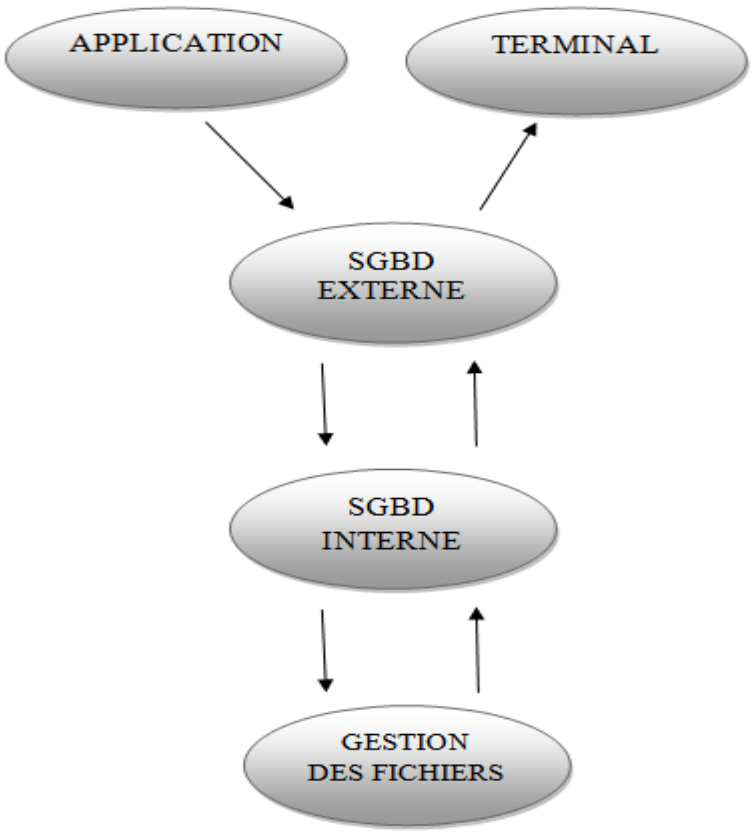


Figure 1-1 Les Sous système SGBD

1. Le système de gestion de fichiers : permet de stocker les informations sur un support physique.
2. Le SGBD interne : gère l'ordonnancement des informations.
3. Le SGBD externe : présente l'interface de l'utilisateur.

3. Principe d'acidité

La plupart des SGBD relationnels sont transactionnels ce qui impose le respect des contraintes : **A**tomicité, **C**onsistance, **I**solation, **D**urabilité ; l'acronyme ACID permet de garantir la fiabilité d'une transaction informatique : [1]

3.1 Atomicité

Une transaction doit être atomique qui veut dire qu'elle doit être complètement effectué, ou pas du tout.

Par exemple, sur 9000 lignes devant êtres modifies au sein d'une même transaction et si la modification d'une seule ligne échoue donc la transaction entière doit être annulée.

C'est nécessaire, car chaque ligne modifiée peut dépendre du contexte de modification d'une autre, et toute rupture de ce contexte pourrait engendrer une incohérence des données de la base.

3.2 Consistance

Chaque transaction doit préserver la cohérence de données ; Cela signifie que les modifications apportées à la base doivent être valides, en accord avec l'ensemble de la base et de ses contraintes d'intégrité. Si un changement enfreint l'intégrité des données, alors soit le système doit modifier les données dépendantes, comme dans le cas classique d'une suppression en cascade, soit la transaction doit être interdite.

La notion de cohérence est importante, et c'est l'un des éléments les plus sensibles entre le monde du relationnel et le monde NoSQL. Les SGBDR imposent une règle stricte : d'un point de vue transactionnel, les lectures de données se feront toujours sur des données cohérentes. Les modifications en cours sont donc cachées, un peu comme sur la scène d'un théâtre : chaque acte d'une pièce de théâtre se déroule intégralement sous l'œil des spectateurs. Si le décor va être changé, le rideau se baisse, les spectateurs doivent attendre, les changements s'effectuent derrière le rideau et lorsque le rideau se lève, la scène est de nouveau dans un état totalement cohérent. Les spectateurs n'ont jamais eu accès à l'état intermédiaire.
Mais cet état intermédiaire peut durer longtemps, pour deux raisons : plusieurs instructions de modifications de données peuvent être regroupées dans une unité transactionnelle, qui peut être complexe. Ensuite, un SGBDR fonctionnant de façon ensembliste, une seule instruction de mise à jour, qui est naturellement et automatiquement transactionnelle, peut très bien déclencher la mise à jour de milliers de ligne de table. Cette modification en masse conservera des verrous d'écriture sur les

ressources et écrira dans le journal de transaction ligne par ligne. Tout ceci prend, bien évidemment, du temps.
Ce respect strict de la cohérence est concevable si nous restons sur le même serveur, mais dés que nous essayons de distribuer les données, il commence à poser de sérieux problèmes. La modification des données qui résident sur plusieurs serveurs n'est aujourd'hui possible qu'à partir d'un mécanisme nomme transaction distribué.

3.3 Isolation

Chaque transaction voit l'état du système comme si elle était la seule à manipuler la base de données c'est-à-dire si les transactions sont lancées en même temps, on n'aura jamais des interférences entre elles. Par exemple si pendant la transaction de T1, une transaction T2 est exécutée en même temps, T1 ne doit pas la voir , tout comme T2 ne doit pas voir T1.

Une transaction isole les données accédées pour éviter une corruption de donnée qui peut être causée par une modification simultanée par plusieurs utilisateurs concurrents. Ce verrouillage provoque des attentes dans certains moteurs relationnels, comme Microsoft SQL Server. D'autres moteurs comme Oracle utilisent des techniques qui diminuent le verrouillage en maintenant la dernière version transactionnellment cohérente antérieure à la modification afin de permettre des lectures même en cas de modifications.

En revanche, deux tentatives d'écriture simultanée sont interdites. Cette isolation est maintenue par la pose de verrous sur les ressources accédées. La gestion des verrous est de la responsabilité de moteurs de stockage. Les problématiques de verrouillage sont relativement complexes.

3.4 Durabilité

Une transaction confirmée le demeure définitivement, peu importe les différents imprévus (panne d'électricité, panne d'ordinateur etc). Pour ce faire, un journal contenant toutes les transactions est créé, afin que celles-ci puissent être correctement terminées lorsque le système est à nouveau disponible.

3.5 Problème lié à l'application des propriétés ACID en milieu distribué

Les bases de données relationnelles s'appuient sur les propriétés ACID, ces propriétés bien que nécessaires à la logique du relationnel nuisent fortement aux performances, en parlant surtout aux propriétés de cohérences, car elles sont très difficiles à mettre en place dans les environnements distribués avec plusieurs serveurs, alors pour assurer cette cohérence il faut que les serveurs doivent être des miroirs les uns des autres, de ce fait deux problèmes apparaissent :

1. Le coût de stockage est énorme car chaque donnée est présente sur chaque serveur.
2. Le coût d'insertion, modification et suppression est très grand car en ne peut valider une transaction que si on est sûre et certain qu'elle a été effectuée sur tous les serveurs.

4. Limites des SGBD relationnels

Le modèle relationnel est fondé sur un modèle mathématique solide s'appuyant sur la logique des prédicats du premier ordre. Il s'appuie sur des concepts simples qui font sa force en même temps que sa faiblesse.

Le stockage distribué est une vraie contrainte qui pèse à ce jour sur les systèmes relationnels, sur laquelle s'ajoute la complexité des structures des données manipulées par les systèmes. Nous expliquerons dans ce qui suit les principales limites de SGBDR.[2]

4.1 Problème de requête non optimale suite à l'utilisation des jointures

Imaginons qu'on a une table contenant toutes les personnes ayant un compte sur Facebook, soit plus de 800 millions, les données dans une base de données relationnelle classique sont stockées par lignes, ainsi si on effectue une requête pour extraire tous les amis d'un utilisateur donné, il faudra effectuer une jointure entre la table des usagers (800 millions) et celle des amitiés (chaque usager ayant au moins un ami donc au minimum on a 800 millions), puis on va parcourir le produit cartésien de ces deux tables .De ce fait, on perd en performances car il faut du temps pour stocker et parcourir une telle quantité de données.

4.2 Type de données

Les systèmes relationnels sont limités à des types simples (entiers, réels, chaines de caractères) ou des données composés comme dans le modèle Objet-Relationnel, les seules types étendus se limitant à l'expression de dates ou de données financières, ainsi que des conteneurs binaires de grande dimension (BLOB , Pour Binary Large OBjects) qui permettent de stocker des images ainsi que des fichiers audio ou vidéos .Ces BLOBs ne sont toutefois pas suffisants pour représenter des données complexes non structurés, les mécanismes de contrôles BD sont inexistants et le langages de requêtes SQL ne possède pas les operateurs correspondant aux objets stockés dans ces BLOBs.

4.3 Langage de manipulation

Il est limité aux opérateurs de base de langage SQL aves ces différentes versions, qu'il n'est pas possible d'étendre à de nouveaux operateurs.

4.4 Requêtes récursives

Il est possible, avec des requêtes SQL imbriquées, de travailler sur des relations de type parent-enfant, mais il est possible de travailler sur des relations de type ancêtre. Pour

traiter ce type de problème, les requêtes doivent être insérées dans un langage de programmation.

4.5 Scalabilité limitée

Atteinte par les poids lourds du Net comme Yahoo, Google, Facebook, Twitter ou LinkedIn, cette limite est la plus gênante pour une entreprise. Dans le cas de traitement de données en masse, le constat est simple : les SGBD relationnels ne sont pas adaptés aux environnements distribués requis par les volumes gigantesques de données et par les trafics aussi gigantesques générés par ces operateurs.

5. Théorème CAP

CAP est l'acronyme de « **C**onsistency, **A**vailability and **P**artition Tolerance » ou « Coherence, Disponibilité et Résistance au morcellement ». Ce théorème est aussi connu sous le nom de théorème de Brewer, il a été énoncé pour la première fois en tant que conjecture par le chercheur en informatique Eric Brewer. En 2002, deux chercheurs au MIT **" Massachusetts Institute of Technology "** Seth Gilbert Nancy Lynch, ont formellement démontré la vérifiabilité de la conjecture de Brewer afin d'en faire un théorème établi. Ce théorème énonce qu'il est impossible d'obtenir ces trois propriétés en même temps dans un système distribué et qu'il faut donc en choisir deux parmi les trois :[3]

5.1 Cohérence

Tous les nœuds du système voient exactement les mêmes données au même moment et les données sont à tout moment dans un état cohérent. Dans un système réparti, la cohérence est appliquée à l'ensemble des nœuds comme s'ils formaient un nœud unique.

5.2 Disponibilité

Les données sont à tout moment disponibles, si un nœud ne répond plus, ça ne doit pas empêcher les autres de continuer à fonctionner correctement. Pour pouvoir tenir la troisième propriété, la disponibilité induit que même en cas de panne sévère chaque requête doit aboutir à une réponse de réussite ou d'échec.

5.3 Résistance au morcellement

Aucune panne moins importante qu'une coupure totale du réseau ne doit empêcher le système de répondre correctement, c'est-à-dire en cas de perte de messages entre les nœuds, le système doit continuer à fonctionner normalement. Cela implique par exemple, que le système doit pouvoir continuer à fonctionner même en cas de panne d'un ou plusieurs nœuds.

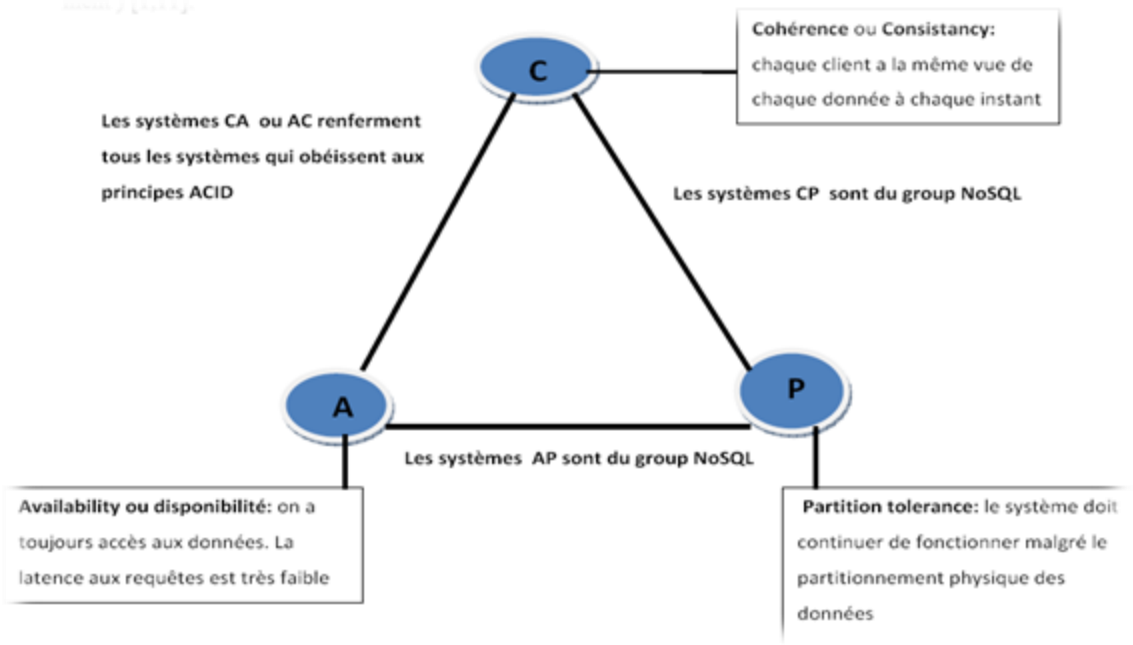


Figure 1-2Les types des systèmes selon Théorème de CAP

Le théorème de CAP dit que seules deux de ces trois contraintes peuvent être respectées à un instant *T* car elles se trouvent en opposition.
Les bases de données relationnelles implémentent les propriétés de Cohérence et de Disponibilité (système AC). Les bases de données NoSQL sont généralement des systèmes CP (Cohérent et Résistant au partitionnement) ou AP (Disponible et Résistant au Partitionnement). De ce fait, le NoSQL concentrent plus sur la résistance au morcellement, afin de garantir une disponibilité en tout temps et par conséquent abandonnent la cohérence.

Systèmes CA : Par exemple des bases de données en cluster ou sur le même site (pour éviter les problèmes de connexion). LDAP, système de fichiers.

- La résistance au morcellement est compromise, elle ne peut être garantie, la cohérence ne peut être maintenue en cas de perte entre les nœuds. En cas de coupure, les verrous nécessaires à la cohérence produisent au final un blocage du système.

Système CP : par exemple distributed data base

- la disponibilité ne peut être garantie, l'accès à une partie des données peut être temporairement limité. L'autre partie restera cependant toujours cohérente et disponible.

Système AP : par exemple DNS, cache WEB, le protocole CODA et certains systèmes NoSQL.

- La cohérence ne peut pas être garantie, on parle de systèmes optimistes.
- On peut appliquer des mécanismes d'expiration et de relâchement.
- On peut mettre en place un mécanisme ou une politique de gestion des conflits.

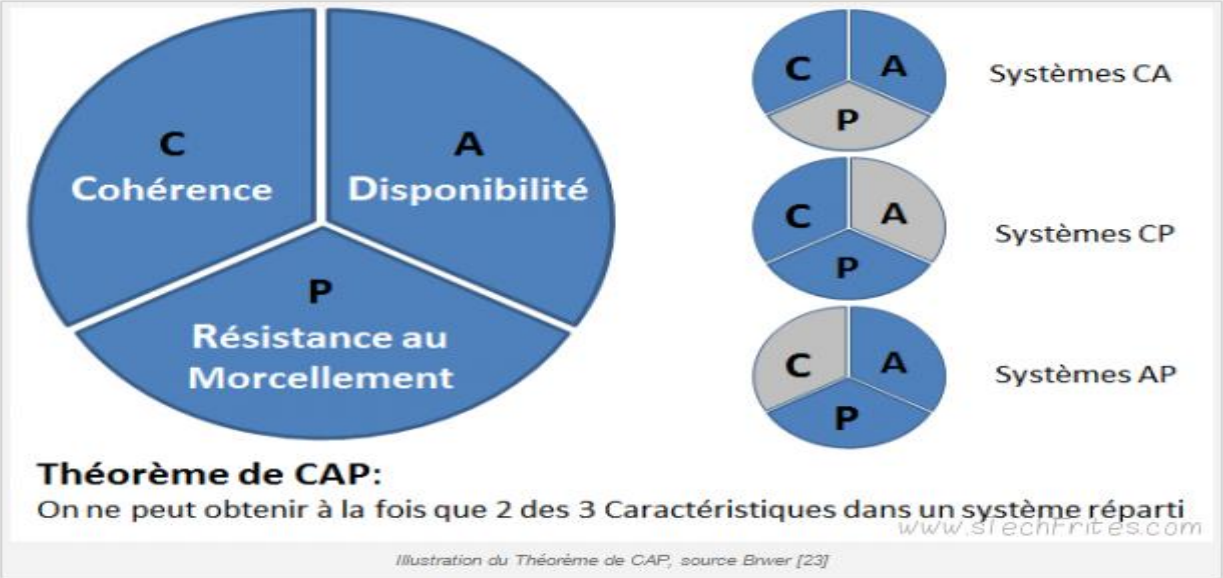


Figure 1-3 Les sous systèmes du théorème CAP

6. Théorème BASE

Le modèle BASE est une alternative au modèle ACID. Le modèle ACID est dit pessimiste puisqu'il oblige à vérifier la cohérence à la fin de chaque opération (2PC). Le modèle BASE est au contraire optimiste et accepte qu'à certains moments la cohérence de la base ne soit pas garantie. BASE est orienté systèmes de stockage répartis et défini les trois principes à atteindre :[4]

- Basically Available
- Soft State
- Eventually Consistent

La cohérence éventuelle est une forme de cohérence faible dans laquelle on ne peut pas garantir, pendant la période dite de incohérence, que chaque processus accède à la même version d'une donnée et cela même pour le processus à l'origine de l'écriture.

Cela est induit par la nécessité de répliquer la modification à travers les nœuds du système. Cependant à l'aide de mécanisme on apporte les modifications pour résoudre les actions qui peuvent amener à des violations de la cohérence. Ces mécanismes permettent ainsi de garder une certaine cohérence.

La différence ente la cohérence faible et la cohérence éventuelle est que le système garantie la cohérence de la donnée si celle-ci n'a pas été modifié très récemment. Pendant la période de l'incohérence nécessaire à la distribution de la modification, la garantie de la cohérence de données ne peut être appliquée, on est dans statut d'une cohérence faible.

ACID	BASE
Cohérence forte	Cohérence faible *Garantie pour les données anciennes*
Isolation	
Pessimiste	Optimiste
Disponibilité non garantie	Disponibilité forte
	Rapide
Transactions évoluées et imbriquées	Simple
Evolutivité difficile (modification du schéma)	Evolutivité simple du schéma

Comparaison ACID - BASE, source Brewer [23]

Figure 1-4 comparaison ACID-BASE

7. Conclusion

Les évolutions logicielles suivent assez naturellement les évolutions matérielles. Les premiers SGBD étaient construits autour de mainframes et dépendaient des capacités de stockage de l'époque. Le succès du modèle relationnel est dû non seulement aux qualités du modèle lui-même mais aussi aux optimisations de stockage qui permet la réduction de redondances des données. Avec la généralisation des interconnexions de réseaux, l'augmentation de la bande passante sur internet et la diminution du coût de machines moyennement puissantes, de nouvelles possibilités ont vu le jour, dans le domaine de l'informatique distribuée et de la virtualisation.

Le passage au XXI^{e} siècle a vu les volumes de données manipulées par certains entreprises ou organismes, notamment ceux en rapport avec Internet, augmenté considérablement. Données scientifiques, réseaux sociaux, opérateurs téléphoniques, base de données médicale, indicateurs économiques et sociaux, etc., l'informatisation croissante des traitements de tout genre implique une multiplication exponentielle de ce volume de données qui se compte maintenant en pétaoctets (10^{6} GigaOctets). C'est ce que les Anglo-Saxons ont appelé le Big Data. La gestion et le traitement de ces volumes sont considérés comme un nouveau défi de l'informatique, et les moteurs de bases de données relationnelles traditionnelles, hautement transactionnels, semblent totalement dépassés.

CHAPITRE 2 :

Bases de données NoSQL

1. Introduction

L'informatique dans les nuages « Cloud Computing » est un ensemble de solutions logicielles et matérielles offrant divers services et proposant des solutions de Big Data à des acteurs économiques dont les ressources financières et matérielles ne leurs permettent pas de mettre en place eux-mêmes les moyens nécessaires pour le traitement de données volumineuses en temps réel comme promettent de le faire les solutions de Big Data.

Le Cloud consiste à proposer des ressources de calcul via Internet, ces ressources de calcul étant fournies par un tiers et donnant lieu à une facturation en fonction du volume de données ou des consommations des ressources de calcul.

L'informatique dans le nuage consiste à externaliser le traitement des données vers des tiers ou des fournisseurs qui gèrent ces données pour le compte de clients.

2. La technologie NoSQL [3]

NoSQL ou « Not Only SQL » est un mouvement très récent (2009), qui concerne les bases de données. L'idée du mouvement est simple : proposer des alternatives aux bases de données relationnelles pour coller aux nouvelles tendances et architectures du moment, notamment le Cloud Computing. Les axes principaux du NoSQL sont une haute disponibilité et un partitionnement horizontal des données, au détriment de la consistance. Alors que les bases de données relationnelles sont basées sur les propriétés ACID (Atomicité, Consistance, Isolation et Durabilité). NoSQL signifie "Not Only SQL", littéralement "pas seulement SQL". Ce terme désigne l'ensemble des bases de données qui s'opposent à la notion relationnelle des SGBDR. La définition, "pas seulement SQL", apporte un début de réponse à la question "Est ce que le NoSQL va tuer les bases relationnelles?". En effet, NoSQL ne vient pas remplacer les BD relationnelles mais proposer une alternative ou compléter les fonctionnalités des SGBDR pour donner des solutions plus intéressantes dans certains contextes. Le NoSQL regroupe de nombreuses bases de données, récentes pour la plupart, qui se différencient du modèle SQL par une logique de représentation de données non relationnelle. Leurs principaux avantages sont leurs performances et leur capacité à traiter de très grands volumes de données. En revanche, dans les projets, il ne faut pas opposer ces deux approches mais bien souvent les faire cohabiter ! Cette technologie (le NoSQL) ne vise finalement pas à remplacer les SGBD traditionnels mais plutôt à les compléter en déportant une partie de la charge des traitements et de stockage de données vers des serveurs-tiers (dans les architectures web classiques).

3. Les Enjeux des bases NoSQL

A l'heure de la révolution du Web 2.0 et vu l'usage actuel d'Internet, le nombre de données disponibles sur la toile augmente d'année en année de manière exponentielle.

Aujourd'hui les sociétés doivent faire face à une très forte augmentation de la quantité de données qu'elles doivent stocker et traiter. Ceci est particulièrement vrai pour les

applications web très à la mode comme Twitter, Facebook, Google. Celles-ci doivent faire face à une activité immense générée par leurs millions d'utilisateurs. Donc le premier besoin fondamental auquel répond NoSQL est la performance.

Les systèmes de gestion bases de données relationnelles (SGBDR) sont la solution prédominante pour le stockage et le traitement de données, mais l'expérience de ces sociétés a montré que le modèle des SGBDR atteint ses limites face à de telles importantes quantités de données.

Voici quelques exemples d'acteurs concernés par ces gros volumes de données [5] :

4.1 Le Big Data en France : un phénomène récent

En 2009, Mathias Herberts, pionnier du Big Data en France (et l'un des plus grands experts du domaine) a déployé l'une des toutes premières plates-formes de Big Data : Arkéa, pour le crédit Mutuel.

C'est un service en ligne qui permet aux clients de la banque d'accéder à leurs historiques sur 10 ans. Représentant 5 milliards d'enregistrements, cette gigantesque base de données supporte des services de recherches complexes capable de croiser de multiple critères (dates, montants, libelles d'opérations).

En 2008, ce dernier a fait un passage chez Google, en prenant en charge le Big Table, qui n'est autre que la couche de gestion des données structurées supportant Google Maps, Google Mail.

4.2 Facebook : 500 téraoctets de données par jour

Facebook à développer des techniques pour :

- Analyser le comportement des utilisateurs sur le site ;
- Consolider les données recueillies sur les différents Datacenters ;
- Trouver de nouvelles solutions logicielles pour traiter ces données.

Leur difficulté numéro 1 réside dans le volume de données, selon les estimations actuelles, ça représente 500 To/jour. Pour Facebook, ces données comprennent des sources multiples :

- Les données liées à l'activité de l'utilisateur (pages visitées, les photos marquées).
- Les données déposées par l'utilisateur (les photos par exemple).

Ces données sont analysées et permettent à Facebook de dresser un profil très détaillé de l'utilisateur, comprenant ses goûts musicaux, ses relations professionnelles et personnelles. Au cours des 4 derniers années le volume de données stockées a été multiplié par 400.

"*Pour faire face à ces problèmes, Facebook a mis en point sa propre solution logicielle : Prism, qui permet de mener des fonctions d'analyse clés, à travers tous les*

centres de données que l'entreprise possède à travers le monde, et découper les analyses en morceaux", a expliqué Ravi Murthy selon le site du Monde informatique.

Leur difficulté numéro 2 réside dans la gestion de "l'infrastructure transactionnelle", qui gère au jour le jour le traitement des données de base telles que, les 'j'aime', les commentaires et les mises à jour de statuts, pour que le réseau social reste fluide.

Quelques chiffres ont été récemment rendus publics lors d'une conférence de presse et publiés par TechCrunch. Des chiffres "Big Data" exorbitants, puisque chaque jour Facebook gère :

- 2.5 milliards d'objets ;
- 500 To de nouvelles données ;
- 2.7 milliards de "like" ;
- 300 millions de photos intégrées ;
- 70 000 requêtes.

4.3 Google : le pionnier du Big Data en perpétuelle évolution

Google a été le précurseur du Big Data. Il suffit pour s'en convaincre de garder l'esprit qu'Hadoop a été développé à partir des spécifications techniques de Google. Inutile donc de préciser que Google à toujours été la pointe dans ce domaine.

Dernièrement Google a développée une suite logicielle autour de Big Data en créant des services en ligne pour ses outils. Le plus récent est sans aucun doute BigQuery qui permet d'analyser en ligne des jeux de données très volumineux, son avantage est d'offrir aux entreprises un service ne nécessitant pas la création d'une infrastructure ad hoc.

4.4 Twitter

Twitter a mis en place une politique globale de développement open source pour le Big Data par le développement interne et par des rachats de sociétés tierces. Ainsi en 2011, la société a racheté BackType qui commercialisait un outil nommé Strom. Cet outil est spécialisé dans le traitement de temps réel de données réparties dans un cluster de serveurs. L'idée était de proposer une alternative à Hadoop qui réalise ses traitements distribués en mode batch.

Notons que Groupon, Yahoo! Et Alibaba ont adopté cette solution.

"*L'une des premières applications de Strom est capable de gérer 1 000 000 de messages par seconde sur un cluster de 10 nœuds "* commente Nathan Marz, ingénieur cher Twitter.

4. Avantages et inconvénients

4.5 Avantages

Les bases de données NoSQL reposent essentiellement sur plusieurs aspects qui font leurs forces et justifient leur usage, donc parmi les raisons principales qui ont mené à la création de ces systèmes on site [6] :

4.5.1 Scalabilité maîtrisée à travers le partitionnement horizontal

Pour gérer la "montée en charge", le NoSQL se réfère à la répartition de la charge sur les systèmes de Cloud Computing. On parle ici de la Scalabilité Horizontale.
La Scalabilité horizontale est le mot à la mode pour designer la caractéristique d'un système capable de supporter une grande charge. Pour augmenter la puissance d'un système, la stratégie consiste à multiplier le nombre de machines de petites puissances. Elle se positionne en opposition à la scalabilité verticale qui prône l'augmentation des capacités d'une machine pour répondre à des besoins de charge croissante.

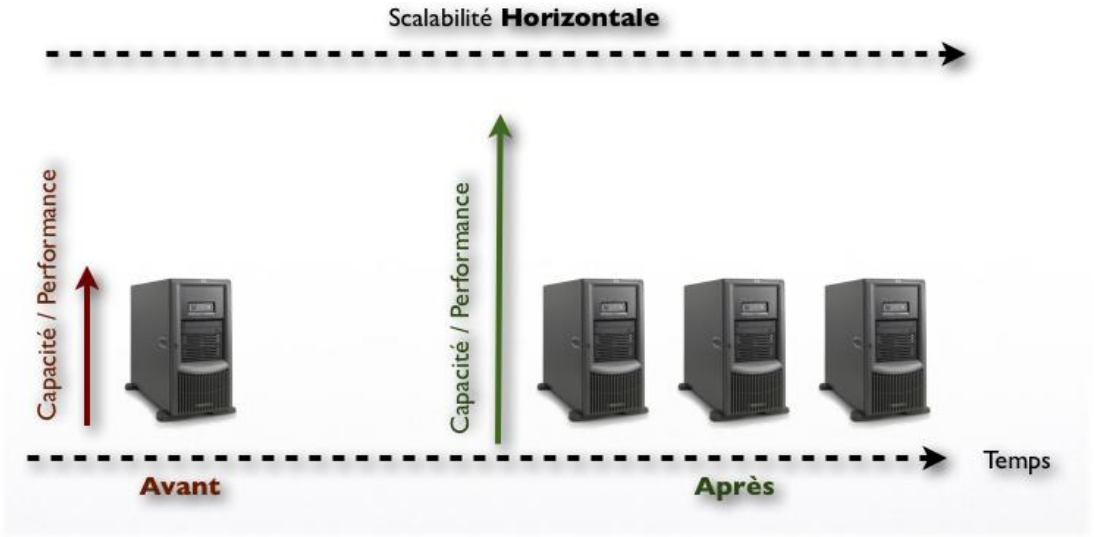


Figure 2-1 Scalabilité horizontale

4.5.2 Gros volume de données/« Big data »

Au cours de la dernière décennie, le volume de données à stocker a augmenté de manière massive. Les bases de données relationnelles ont augmenté leurs capacités afin de suivre la tendance. Mais avec cette constante augmentation de volume de données, il est devenu quasi impossible, pour un unique serveur de base de données relationnelle, de répondre aux exigences des entreprises en terme de performance. Aujourd'hui, ces gros volumes de données ne sont plus un problème pour les SGBD de type NoSQL, même le plus grand des SGBD relationnel ne peut rivaliser avec une base NoSQL.

4.5.3 Administrateur de bases de données (DBA) moins indispensable

Malgré les nombreuses améliorations vantées pendant des années par les fournisseurs de SGBD relationnels concernant la gestion de bases de données, un SGBD relationnel haut de gamme demande une certaine expertise pour sa maintenance. C'est la raison

pour laquelle on fait généralement appel à un DBA, ce qui représente donc un certain coût. Les DBA sont intimement impliqués dans la conception, l'installation ainsi que dans le réglage des SGBD relationnels haut de gamme.

Il est erroné de dire que la présence d'un DBA n'est plus requise pour gérer une base de données NoSQL. Mais ses tâches seront facilitées notamment grâce à la distribution des données et surtout grâce à la simplicité du schéma de données. Il y aura toujours une personne responsable des performances ainsi que de la disponibilité quand des données critiques sont en jeu.

4.5.4 Solution économique

Les bases de données NoSQL ont tendance à utiliser des serveurs bas de gamme de moindre coût afin d'équiper les « clusters », tandis que les SGBD relationnels, eux, tendent à utiliser des serveurs ultra puissants dont le coût est extrêmement élevé. De ce fait, les systèmes NoSQL permettent de revoir à la baisse les coûts d'une entreprise en termes de gigabytes ou de transactions par seconde. Cela permet de stocker et manipuler plus d'informations à un coût nettement inférieur.

4.5.5 Modèle de données flexible

Changer le modèle de données est une vraie prise de tête dans une base de données relationnelle en production. Même une petite modification doit être maniée avec précaution et peut nécessiter l'arrêt du serveur pendant la modification ou limiter les niveaux de services.
Les systèmes NoSQL sont plus souples en termes de modèles de données, comme dans les catégories clé/valeur et documentaire. Même les modèles un peu plus stricts comme dans la catégorie orientée colonne permettent d'ajouter une colonne sans grandes difficultés.

4.6 Inconvénients

Aucun système n'est parfait, même la solution NoSQL a ses inconvénients. Les principaux inconvénients sont les suivants [6]:

4.6.1 Fonctionnalités réduites

Les bases de données NoSQL sont principalement conçues pour stocker de façon optimale des ensembles données mais en contreparties, le langage permettant d'effectuer des requêtes vers le système NoSQL est beaucoup moins riche. C'est pour cette raison que les bases de données relationnelles survivent toujours et le NoSQL ne pourra jamais remplacer entièrement le SQL.

4.6.2 Normalisation et Open Source

Etrangement, le critère d'open source pour les bases de données NoSQL est à la fois sa plus grande force et sa plus grande faiblesse. La raison est qu'il n'existe pas encore de normalisation fiable pour NoSQL.

4.6.3 Performances et évolutivité au détriment de la cohérence

En raison de la façon dont les données sont gérées et stockées dans ces bases, la cohérence des données pourrait bien être une préoccupation. Comme déjà vu précédemment, les bases de données NoSQL font l'impasse sur les propriétés dites ACID afin de mieux répondre aux besoins de performances et d'évolutivité. La cohérence des données est donc un facteur moins important. Donc, puisque les systèmes NoSQL ne respectent pas l'ensemble des propriétés ACID, comme le font les systèmes relationnels classiques, cela se traduit en pratique par un effort supplémentaire de développeur dans certains cas pour s'assurer de la cohérence des données.

4.6.4 Manque général de maturité

Bien que les bases de données NoSQL soient présentes depuis bon moment, leurs technologies sont encore immatures par rapport à celles des bases relationnelles. Cela se traduit également par un manque d'administrateurs et de développeurs ayant les compétences dans ce système. Les bases de données ont beau être « administrator-friendly » (simples à administrer), cela n'a pas de sens si les administrateurs en question ne savent pas comment les utiliser. A l'heure actuelle, les bases de données relationnelles sont beaucoup mieux implémentées dans les entreprises. Elles disposent d'un nombre plus grand de fonctionnalités et de professionnels qui comprennent comment les gérer.

Les bases de données NoSQL ne sont donc, pas la solution miracle pour répondre à toutes les problématiques de stockage sur le web ou ailleurs. Il est surtout très important, de bien comprendre, ce que le choix d'une base de données de ce type va avoir comme conséquences en terme d'architecture logicielle et complexité de développement.

5. Les types de bases de données NoSQL

Dans la mouvance NoSQL, il existe une diversité d'approches classées en quatre catégories. Ces différents systèmes NoSQL utilisent des technologies fortes distinctes. Les différents modèles de structure sont décrits comme suit :

5.1 Les bases de données clé-valeur [1]

Il s'agit de la catégorie de base de données la plus élémentaire. Son principe est très simple, chaque objet est identifié par **une clé unique** qui constitue la seule manière de le requêter.

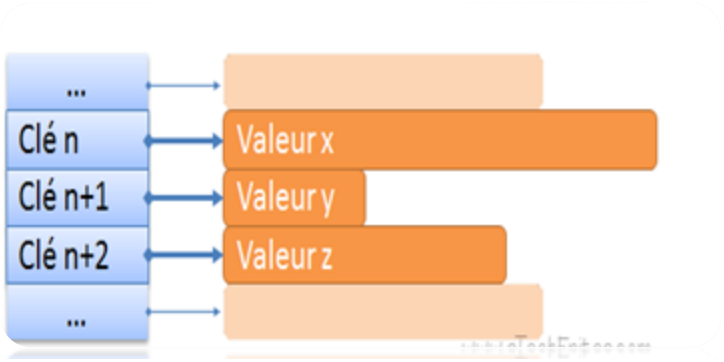


Figure 2-2 Illustration d'une base de données orientée clé-valeur

La structure de l'objet est libre et le plus souvent laissé à la charge du développeur de l'application (XML, JSON, ...), la base ne gérant généralement que des chaînes d'octets. Un avantage considérable de ce type de base de données est qu'il est très facilement extensible, on parle donc de Scalabilité horizontale. En effet dans le cas où le volume de données augmente, la charge est facilement répartissable entre les différents serveurs en redéfinissant tout simplement les intervalles des clés entre chaque serveur.

Donc pour palier à la problématique de montée en charge et en volume, la solution simple est de partitionner la table, divisant ainsi le volume et la charge entre toutes les instances. Cette approche simple n'est pas idéale car elle peut entrainer une répartition inégale des données entre les instances. Ainsi sur la figure suivante où l'on considère le cas du partitionnement d'un ensemble de clés de type String, l'une des partitions pourrait avoir une taille bien plus importante que les autres [7] :

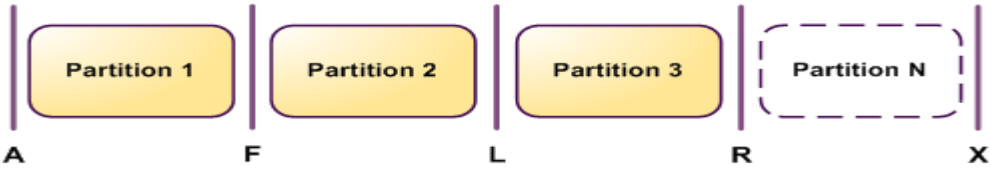


Figure 2-3 principes partitionnement de clé

Pour résoudre le problème de la répartition inégale des données entre les instances on va utiliser le mécanisme de consistent haching, Ce mécanisme se base sur un *hash* de la clé pour sélectionner l'instance qui se chargera du stockage. Si la fonction de hachage utilisée est uniforme, les données seront équitablement réparties.

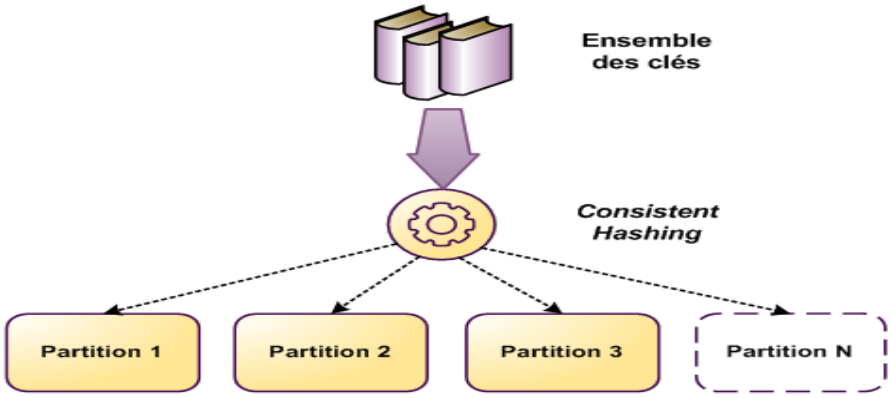
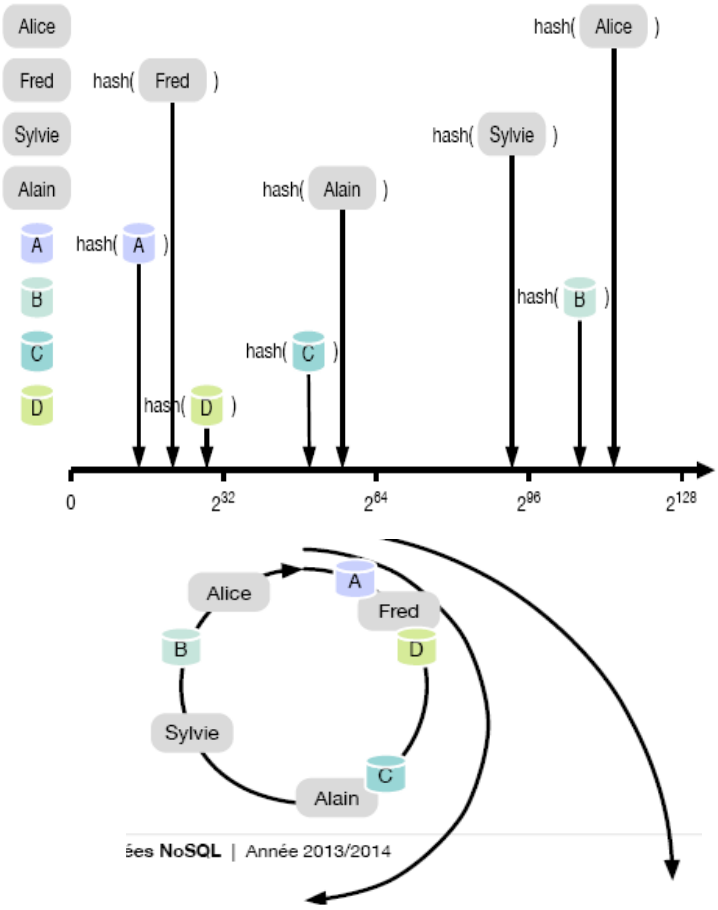


Figure 2-4 consistent Hashing

Le hachage cohérent évite les déplacements des objets en cas d'addition ou d'enlèvement de machines, les clés sont mappées par la même fonction de hachage, mais la valeur de hachage est mappée dans un cercle et la même chose pour les machines sont mappées de la même manière dans le cercle en utilisant leur nom comme clé.

On établit la convention suivante : Un objet est assigné á la machine qui le suit dans le cercle (sens des aiguilles d'une montre). [8]



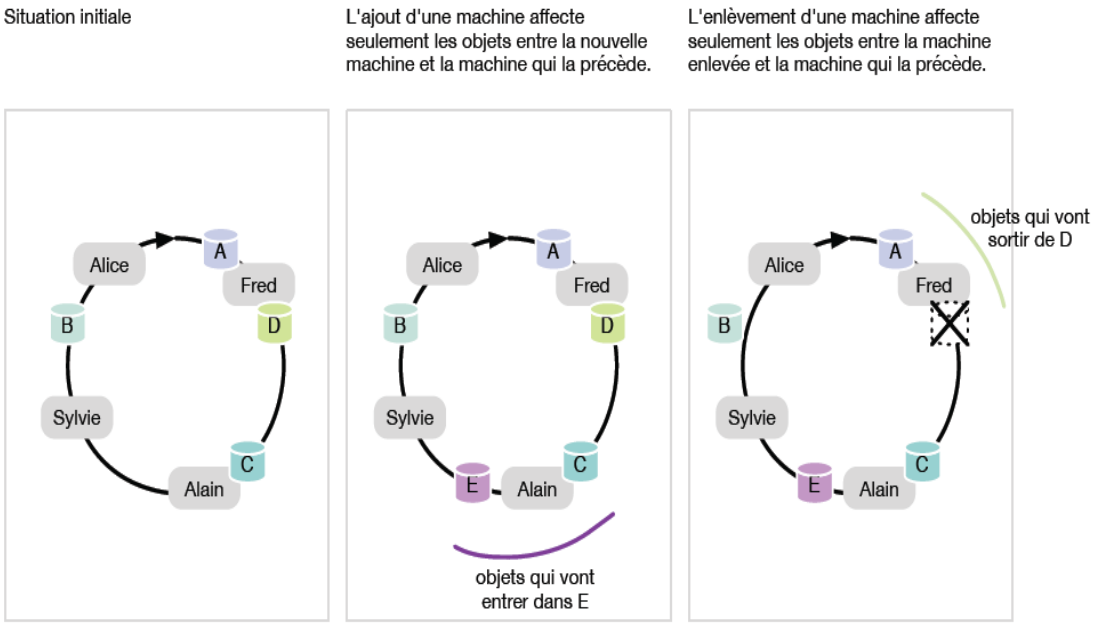


Figure 2-5 principe de consistent hashing

Dans ce modèle on ne dispose généralement que des quatre opérations **C**reate **R**ead **U**pdate
Delete (CRUD):

- **Create** : créer un nouvel objet avec sa clé → create (Key, value)
- **Read** : lit un objet à partir de sa clé → Read(Key)
- **Update** : met à jour la valeur d'un objet à partir de sa clé → update (Key, value)
- **Delete**: supprime un objet à partir de sa clé → Delete (Key)

5.1.1 Points forts

Parmi les avantages des bases de données clé-valeur, le fait qu'elles sont très faciles et rapides à réaliser. De plus, elles ont le mérite d'être facilement extensibles. En effet, dans le cas où le nombre de données augmente, on peut facilement répartir la charge entre différents serveurs de stockage, en définissant par exemple des intervalles de clés sur chaque serveur.

De plus, les applications utilisant une persistance avec des bases de type clé-valeur ont la particularité d'avoir un code simplifié et facile à lire par rapport à des applications standards utilisant SQL.

Enfin, coté performance, c'est leur point fort car ces modèles permettent de réduire le nombre total de requête effectué sur la base puisque les seuls types de requêtes possibles sont la lecture et l'écriture de données.

5.1.2 Points faibles

La simplicité est considérée comme un point faible des bases de données clé-valeur, on peut faire que des requêtes basées sur les clés. En effet, ce type de bases permet un

stockage de données sans schéma, c'est-à-dire sans champs dans les tables et sans modèle relationnel comme dans les SGBD classiques.

5.1.3 Acteurs

DynamoDB est une base de données NoSQL de type clé-valeur très connue, et propose un grand nombre de fonctionnalités afin de développer sa base de données de manière très rapide et efficace.

Cette solution est recommandée afin de concevoir une base de données qui a vocation à traiter d'importantes quantités d'informations et à être « scalable ».

DynamoDB dispose d'un modèle de données bien particulier. En effet, les informations sont stockées sous la forme de paires clé-valeur que l'on appelle attribut. Ensuite, on définit ce qu'on appelle des « Items », qui sont représentés par une série d'attributs. Pour chaque Item, il faut qu'un attribut le composant soit défini comme étant clé-primaire. Tous les Items sont alors stockés dans des tables.
Le schéma ci-dessous (tiré de la documentation officielle de DynamoDB sur le site aws.amazon.com), montre le modèle de données de DynamoDB : on voit notamment que les différents Items d'une même table n'ont pas forcément les mêmes attributs, ni le même ordre pour ceux qu'ils peuvent avoir en commun.

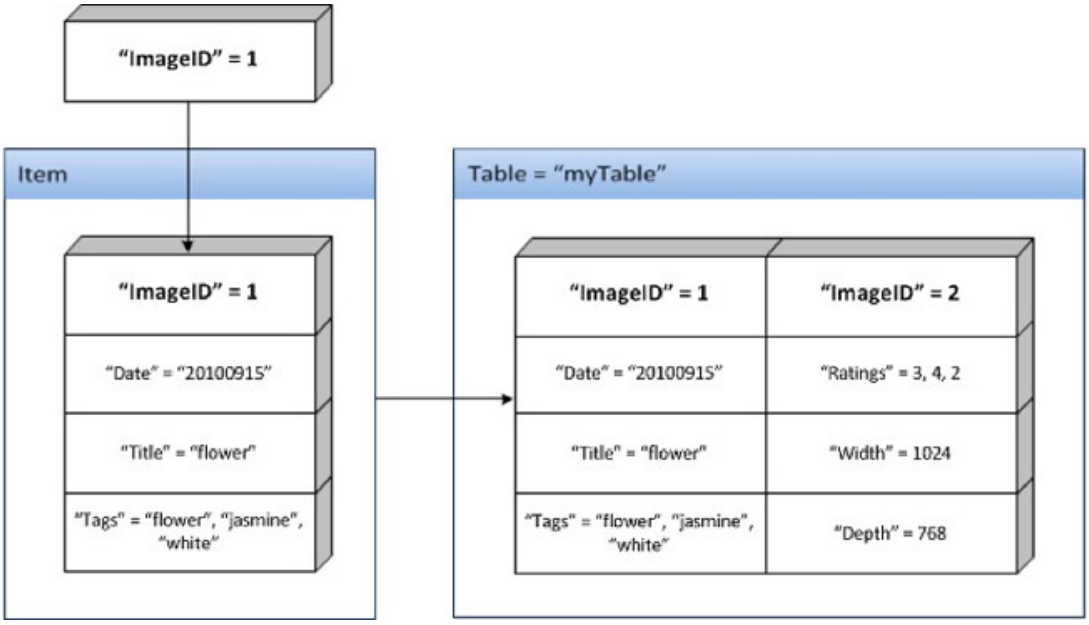


Figure 2-6 Modèle de données DynamoDB

5.2 Les bases de données orientées documents

La représentation en document est particulièrement adaptée au monde du Web. Il s'agit d'une extension du concept de clé-valeur qui représente la valeur sous la forme d'un document de type JSON ou XML. L'avantage est de pouvoir récupérer via une seule clé un ensemble d'informations structurées de manière hiérarchique. Étant consciente du contenu qu'elle stocke, la base de données peut alors effectuer des indexations de différents champs et offrir des requêtes plus élaborées. [9]

Les bases de données documentaires sont constituées de collections de documents. Un document est composé de champs et des valeurs associées, ces dernières pouvant être requêtées.
Par ailleurs, les valeurs peuvent être, soit d'un type simple (entier, chaine de caractères, date,...), soit elles mêmes composées de plusieurs couples clé/valeur.
Bien que les documents soient structurés, ces bases sont dites "schemaless". A ce titre il n'est pas nécessaire de définir au préalable les champs utilisés dans un document. Les documents peuvent être très hétérogènes au sein de la base.
Le stockage structuré des documents leur confère des fonctionnalités dont ne disposent pas les bases clés/valeurs simples dont la plus évidente est la capacité à effectuer des requêtes sur le contenu des objets. [10]

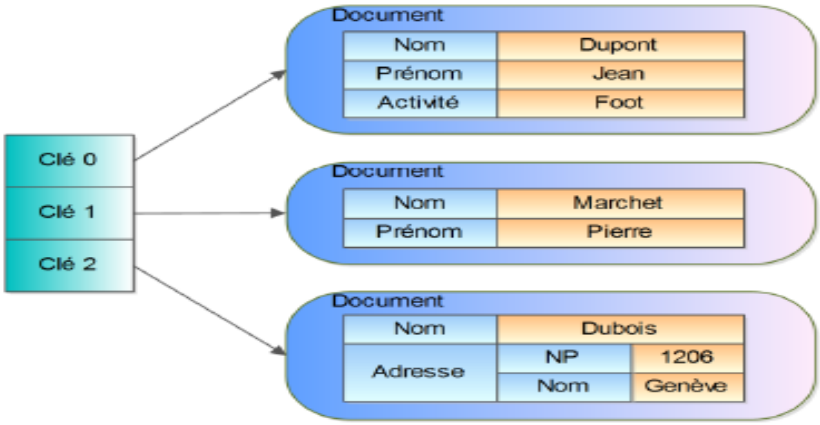


Figure 2-7 Illustration base de données oriente document

Comme l'image peut nous le montrer, chacun des documents ci-dessus a une structure différente, ils peuvent donc être très hétérogènes. Un document peut contenir des chaînes de caractères comme illustré sur cette image, mais également des valeurs numériques ainsi que d'autres documents.

Les bases de données orientées documents, reposent sur le principe dit des documents, sont un peu plus élaborées que les "Clé / Valeur", bien qu'elles stockent des valeurs toujours liées à une clé d'accès. Mais contrairement à ces précédentes, une clé nous permet d'y entreposer des données complexes, qui peuvent très bien elles-mêmes contenir d'autres documents et ainsi de suite, ce qui permet de structurer ces données. Par exemple, on peut très bien penser stocker un article, qui contiendrait le nom de

l'auteur, la date ainsi que le corps de l'article, mais également les commentaires liés à cet article, qui seraient eux-mêmes composés du commentaire en lui-même ainsi que du nom de l'auteur. Bien que l'on puisse structurer les données stockées, elles n'ont pas besoin de suivre un modèle de données, les documents stockés pouvant avoir des types très hétérogènes entre eux. La plupart des bases de données documentaires enregistrent les données sous format JSON.

```
{
title : "Mon travail" ,
author : "Rudi Bruchez" ,
last_modified : new Date ("8/09/2012") ,
body : "NoSQL et CAP…" ,
tag : ["NoSQL", "Document Database"] ,
}
```

Les bases de données documentaires se rapprochent fortement des anciennes bases de hiérarchiques. La grande différence vient du fait que les bases documentaires actuelles permettent pour la plupart, via des fonctions MapReduce, de parcourir aisément les données.

Les bases de données documentaires ressemblent aux bases de données hiérarchiques de par la structure de leurs documents. La différence vient du fait que la hiérarchie ne se fait plus au niveau de la table mais des documents en eux-mêmes. Qui plus, avec les fonctions MapReduce, il est plus simple de parcourir et de rechercher les données.

La différence entre les bases de données orientées documents et les bases de données relationnelles, c'est qu'il n y a pas un schéma de base de données dans l'orienté document, c'est-à-dire que deux documents peuvent avoir des propriétés différentes comme l'illustre l'image ci-dessous [9]:

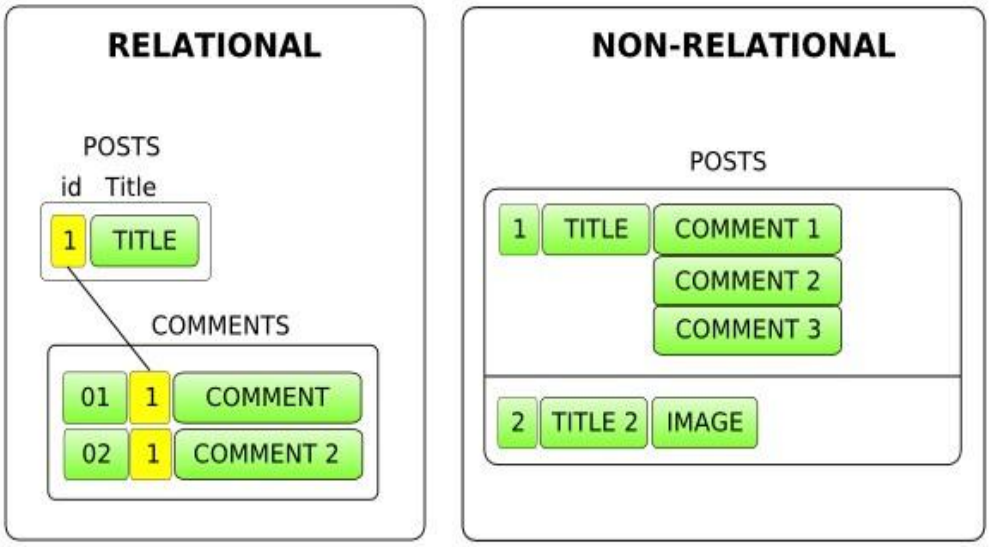


Figure 2-8 Différence entre relationnel et non relationnelle

Le premier document contient des commentaires, alors que le second contient une image. Ceci illustre la flexibilité des bases de données orientées document. Un document peut contenir des valeurs comme une chaine de caractères mais aussi d'autres documents. Dans le schéma ci-dessus « COMMENT 1 » pourrait être une chaîne de caractères ou un document contenant une chaîne de caractères et une date par exemple.

Il est très simple de récupérer les commentaires du document ayant pour titre TITLE pour les bases orientées documents, contrairement aux SGBDR où il faudra faire une jointure.

5.2.1 Points forts

- **Adaptées pour traiter des données hétérogènes**

Les données n'étant pas structurées dans des tables comme pour les SGBDR, l'organisation s'en retrouve beaucoup plus flexible. On peut ainsi avoir des propriétés différentes entre les documents. Ce qui est adapté dans le cas où l'on a un modèle de données très hétérogène.

- **D'excellentes performances possibles**

Les bases de données orientées documents sont particulièrement adaptées pour regrouper du contenu utilisé par une même page web. Cela évite les jointures en SQL des SGBD qui peuvent être coûteuses en termes de performances. En outre, elles sont adaptées aux gros volumes de données.

Elles peuvent aussi servir à mettre en cache des informations sous une forme intelligible (possibilité d'effectuer des requêtes dessus). Pour la plupart d'entre elles, il est possible de mettre la base directement en mémoire RAM, ce qui améliore grandement les performances.

- **Une bonne Scalabilité horizontale**

Les SGBDR ne sont pas très adaptés à la scalabilité horizontale puisqu'ils doivent respecter les caractéristiques ACID. Ainsi quand on rajoute des machines, il faudra entre autre s'assurer qu'une donnée n'est pas verrouillée par une autre transaction sur un autre serveur, si on souhaite l'utiliser. Bien sûr, il existe plusieurs solutions pour rendre plus efficace la scalabilité horizontale telles que RAC (Real Application Cluster) pour Oracle ou Service Broker pour MS SQL Server. Mais ce genre de solution n'est pas aisé à mettre en place. A l'inverse, la plupart des bases de données orientées documents sont pensées pour avoir une bonne scalabilité horizontale facilement. En effet, ce type de bases fonctionne avec quelque chose de semblable aux DHT (Distributed Hash Table). Alors, on peut dire que les bases de données orientées documents étant une évolution des bases de données clé-valeurs, elles se prêtent bien aux DHT, d'où une scalabilité horizontale efficace.

5.2.2 Points faibles

Parmi les points faibles des ces bases de données, la publication des données; les données étant regroupées dans un document, cela peut causer des problèmes d'intégrité, puisque certaines données seront inévitablement dupliquées. Par exemple, dans le cas d'une base de données contenant les produits et les ventes d'un magasin, on aurait des documents représentant les produits avec leurs noms et des documents représentant les commandes avec les noms des produits de la commande. Il y a donc duplication concernant les noms des produits et une possible incohérence si on modifie le nom d'un produit sans le changer dans les commandes ou il apparait.

5.2.3 Les acteurs

Il existe bon nombre de systèmes de gestion de bases de données orientées documents tel que Terrastore, RavenDB, RaptorDB, SimpleDB ou encore Redis. Cependant deux des plus connus vont être présentés en détails, MongoDB et CouchDB.

- **MongoDB**

MongoDB est un système de gestion de bases de données de plus en plus connu développé par la société 10gen, On va le voir en détail dans le chapitre suivant.

- **CouchDB**

CouchDB est un système de gestion de bases de données développé à l'origine par Damien Katz pour IBM. Il est actuellement maintenu par l'Apache Software Foundation0.. CouchDB est particulièrement adapté au web. En effet, la communication avec un serveur CouchDB se fait avec le protocole HTTP via une interface REST (REpresentationnal State Transfer) et le format des documents est JSON. Il est donc aisé de récupérer et manipuler des documents depuis le navigateur web avec le langage JavaScript et le concept AJAX (Asynchronous JavaScript And XML) qui permet de communiquer avec des serveurs sans recharger la page web. Ce qui est très utile pour des applications web ou des pages web dynamiques.

Pour stocker, lire, modifier ou supprimer un document, on envoie une requête HTTP sur une URL. Par exemple, pour obtenir le document ayant pour clé « doc » se situant dans la base « base », on utilisera la méthode GET sur l'URL « http://localhost/base/doc » qui retournera un fichier au format JSON.[11]

CouchDB étant orientée document, et pas seulement clé-valeur, il permet d'effectuer des requêtes sur les documents via ce que l'on appelle des vues. Ceci, non pas en SQL, mais en JavaScript (par défaut, il est possible d'utiliser d'autres langages tels que Python ou PHP). Les vues reposent sur le principe Map/Reduce.

CouchDB permet la répartition de charge. En effet, il est possible d'avoir des copies des données sur d'autres serveurs CouchDB avec un système de réplication qui gère la resynchronisation, cette synchronisation peut être réglée très finement. Il est possible d'indiquer quels documents doivent être synchronisés via un filtre qui pour chaque document indique s'il faut, ou pas, le synchroniser. Cela est utile pour créer des répliques spécialement pour certains documents. CouchDB est écrit en Erlang, langage conçu pour la programmation distribuée, la montée en charge et la tolérance aux pannes. En effet, ce langage a été créé par Ericsson, un opérateur téléphonique, dans le but d'avoir une architecture disponible 24 heures sur 24. [11]

5.3 Les bases de données orientées colonnes [12]

Les bases de données orientées colonnes avaient initialement été créées par Facebook pour le stockage des messages (non instantanés) entre utilisateurs. Les ingénieurs qui en sont à l'origine, leur étaient possibles d'accéder de manière très efficace aux messages échangés entre deux personnes ou aux messages contenant certains mots clés sur une base de données de taille conséquente.

Les bases de données orientées colonnes forment une évolution du stockage clé-valeur. Il s'agit ici de représenter les données sous la forme de blocs de colonnes stockés de manière triée sur le disque.

Les bases de données orientées colonnes c'est une extension de base de données clé-valeur, en effet le modèle de colonnes est en fait plus évolué, on parle de super-colonne ou de familles de colonnes qu'a identifiant de ligne permettent de stocker un ensemble structuré de données. Une famille de colonnes a les caractéristiques suivantes : les données sont triées, associées et peuvent contenir un tableau de colonnes de taille illimitée.

Le stockage d'une base de données colonnes se fait par colonne et non par ligne. Ces bases peuvent évoluer avec le temps, que ce soit en nombre de lignes ou en nombre de colonnes. Autrement dit, et contrairement à une base de données relationnelle ou les colonnes sont statiques et présentes pour chaque ligne, celles des bases de données orientée colonnes sont dite dynamiques et présentes uniquement en cas de nécessité. De plus le stockage d'un « null » est 0. Prenons l'exemple de l'enregistrement d'un client nécessitant son nom, prénom et adresse. Dans une base de données relationnelle il faut donc au préalable créer le schéma (la table) qui permettra de stocker ces informations. Si un client n'a pas d'adresse, la rigidité du modèle relationnel nécessite un marqueur NULL, signifiant une absence de valeur qui coûtera toutefois de la place en mémoire. Dans une base de données orientée colonne, la colonne adresse n'existera pas. Les bases

de données colonnes ont été pensées pour pouvoir stocker plusieurs millions de colonnes.

ID	Prénom	Operateur	LOCALITE
1	Matallah		Alger
2	Benmia	Djezzy	
3	Naceur		
4			Oran
5			Constantine

Base de donnée relationnel

Prénom	Opérateur	Localite
Mtallah(1)	Djezzy(2)	Alger(1)
Benmia(2)		Oran(4)
Naceur(3)		Constantine (5)

Base de donnée orienté colonne

Figure 2-9 La différence entre relationnel et orienté colonne

Ces deux schémas démontrent que le stockage de données en colonne permet de gagner un espace considérable, spécialement lors des stockages des tuples incomplets. En effet, la valeur « null » n'est pas stockée dans ce type de base de données.
Dans des bases de données telles que Cassandra ou HBase il existe quelques concepts supplémentaires qui sont les familles de colonnes, qui sont un regroupement logique de lignes. Dans le monde relationnel ceci équivaudrait en quelque sorte à une table. Cassandra offre une extension au modèle de base en ajoutant une dimension supplémentaire appelée « Super colonnes » contenant elle-même d'autres colonnes.

5.3.1 Concepts

Pour bien comprendre le modèle de données utilisé par Cassandra il est important de définir un certain nombre de termes utilisés par la suite :

- **Keyspace**
  S'apparente à un namespace, c'est en général le nom donné à l'application.
- **Column**
  Représente une valeur, elles disposent de 3 champs: son nom, sa valeur et un timestamp représentant la date à laquelle a été inséré cette valeur.

Implémentation

```
public class Column {
 private String name;
 private String value;
 private Long timestamp;

 public Column(String name, String value) {
 this.name = name;
 this.value = value;
              }
                  }
```

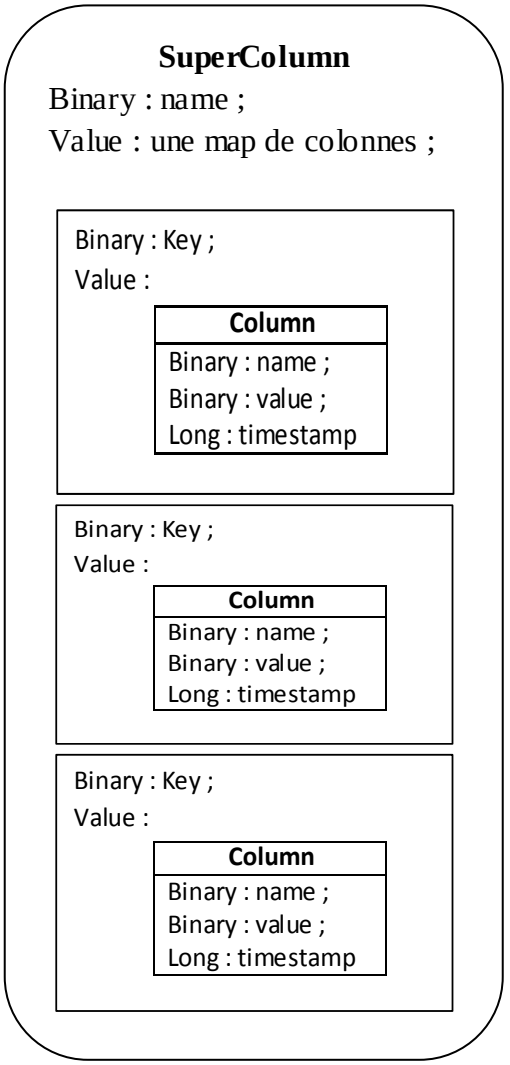
Column
Binary : name ;
Binary : value ;
Long : timestamp ;

- **Super Column**
  Les supers colonnes sont des listes de colonnes, si on veut faire le correspondre avec une base SQL, cela représente une ligne. On retrouve cette correspondance clé-

valeur, la clé permet d'identifier la super colonne tandis que la valeur est la liste des colonnes qui la compose.

Implémentation en JAVA

```
public class SuperColumn {

        private String name;
        private Map<String, Column> value;

        public void setName(String bs) {
                this.name = bs;
        }
        public void setValue(String name, Column value) {
                this.value.put(name, value);
        }
    }
```



Pour l'insertion de donnée en utilisant ce modèle, on doit tout d'abord initialiser notre super colonne et lui attribuer une clé, nous pouvons ensuite commencer à insérer les données :

```
SuperColumn sc = new SuperColumn();
sc.setName("person1");
sc.put("firstname", new Column("firstname", "Hocine"));
sc.put("familyname", new Column("familyname", "Matallahh"));

sc = new SuperColumn();
sc.setName("person2");
sc.put("firstname", new Column("firstname", "Imane"));
sc.put("familyname", new Column("familyname", "Benmia"));
```

Voila comment les données sont organisées en gardant à l'esprit la notion de clé-valeur. D'après notre exemple, nous pouvons constater que nous avons crée 2 super colonnes (person1 et person2) disposant chacune de 2 colonnes qui décrivent le firstname et le familyname de chaque personne.

SuperColumns	
Key	**Value**
Person1	**Column** **Name** : **Value** FirstName : Hocine LastName : Matallahh
Person2	**Column** **Name** : **Value** FirstName : Benmia LastName : Imane

Tableau 2-1 Super colonne

- **Column family**
  Ressemble le plus aux tables SQL, elle est considérée comme un conteneur de plusieurs colonnes ou super-colonnes.

ColumnFamily	
Key	**Value**
Adress-book	**SuperColumns**

Key	**Value**
Person1	**Column**

Name	**Value**
FirstName	Hocine
LastName	Matallahh

Key	**Value**
Person2	**Column**

Name	**Value**
FirstName	Benmia
LastName	Imane

Tableau 2-2 Family colonne

Implémentation

On retrouve le même mécanisme que les super colonnes, c'est uniquement la Map qui change pour y insérer les super colonnes.

```
public class ColumnFamily {

    private String name;
    private Map<String, SuperColumn> value;

    public void setName(String bs) {
            this.name = bs;
    }
    public void setValue(String name, SuperColumn value) {
            this.value.put(name, value);
    }
}
```

Pour l'insertion des données en utilisant ce modèle, il faut tout d'abord instancier une famille de colonne (ColumnFamily) et lui attribuer un nom qui sera sa clé (dans cet exemple "AddressBook"). On peut ensuite créer autant de supers colonnes que l'on souhaite et l'ajouter à notre famille de colonne.

```
ColumnFamily cf = new ColumnFamily();
cf.setName("AddressBook");

SuperColumn row= new SuperColumn();
row.setName("person1");
row.put("firstname", new Column("firstname", "Hocine"));
row.put("familyname", new Column("familyname", "Matallahh"));
cf.setValue("person1", row);

row = new SuperColumn();
row.setName("person2");
row.put("firstname", new Column("firstname", "Benmia"));
row.put("familyname", new Column("familyname", "Imane"));
cf.setValue("person2", row);
```

- **Row** : C'est l'identifiant unique de chaque ligne de colonne
- **Value** : C'est le contenu de la colonne, ou la colonne elle-même

Le concept de bases de données orientées colonnes est créé par les grands acteurs du web, pour répondre au besoin du traitement de grands volumes de données, et précisément pour gérer de larges volumes de données structurées. Souvent, ces bases intègrent un système de requêtes minimaliste proche de SQL comme le NoSQL utilise le CQL.
BigTable est un SGBD orienté colonne créé par Google, pour leur moteur de recherche et leurs indexations des données. Sa structure complexe se base sur un stockage en colonne, et permet toute sorte d'optimisation et notamment pour l'indexation. BigTable n'est pas librement distribué, et est réservé à l'usage interne de Google, étant, de plus, spécifiquement conçu pour traiter les données relatives à la société.
D'autres grands acteurs du web et des sociétés indépendantes se sont inspiré de la solution mise en place par Google, et ont créé leur propre base NoSQL orientée colonnes notamment Facebook. Cependant il n'existe pas encore de système officiel de classification de celles-ci, ni de règles conventionnelles officiellement définies pour qu'une base soit déclarée comme base de données colonnes de qualité.
Les 12 règles de Codd ont donc été créées pour définir ce qui est exigé d'un système de gestion de base de données (SGBD) afin qu'il puisse être considéré comme relationnel (SGBDR). Ces 12 règles pour les BDD relationnelles n'ont pas encore d'équivalent pour les BDD colonnes et autres destinées au traitement de gros volumes de données, c'est là le propos des 6 règles établies par Michael Stonebraker, créateur de Ingres et PostgréSQL et co-fondateur de Vertica.
La base de données orientée colonnes Vertica satisfait bien-sûr toutes ces 6 règles proposées par M. Stonebraker :

- Un bloc de stockage ne doit pas contenir des données de plusieurs colonnes.
- Le taux de compression doit être amélioré grâce à une implémentation de compression de données volumineuse, c'est-à-dire en effectuant, au besoin, des rotations de block.
- Il faut pouvoir se passer des ID d'enregistrement.
- La base de données orientée colonnes doit disposer d'un moteur d'exécution au niveau de la colonne et non au niveau de l'enregistrement (ce qui limite le nombre de boucles nécessaires pour accéder aux enregistrements).
- Le moteur doit pouvoir travailler directement sur les données compressées et non à travers d'un buffer de décompression.
- Le moteur doit être capable de traiter aussi bien des données stockées chronologiquement que sur une clé.

Ces règles sont une première proposition de normes et indicateurs de qualité pour une base orientée colonne. Des règles officielles seront probablement bientôt établies, tant les bases colonnes tendent à envahir le monde du BigData.

5.3.2 Points forts

Parmi les avantages des bases de données orientées colonnes on site :

- **Une capacité de stockage accrue**

Comme nous l'avons vu précédemment, il n'y a aucune valeur « null » dans une base de données orientée colonne. Cela représente un gain en volume, donc en espace disque utilisé considérable. Qui dit gain en espace de stockage, dit également bien sûr gain économique important, puisque la taille de l'infrastructure nécessaire à l'accueil des données en est réduite.

- **Compression**

Dans le cas d'une base de données colonnes, les données sont compressées le plus possible. En effet, si chaque colonne contient des données qui se ressemblent trop, la compression de ces données sera très significative.
On pourrait penser que l'accès à des données compressées est lent car il nécessiterait une décompression de celles-ci. Cependant de plus en plus d'éditeurs de SGBD colonnes permettent un accès direct aux données, sans décompression lors de l'interrogation des données.

- **Rapidité d'accès aux données**

Pour le cas d'une base de données où l'on réalise plus de lectures que d'écritures et dont on veut récupérer dans de gros volumes une donnée spécifique, il est beaucoup plus performant de récupérer directement les informations de toute la colonne, plutôt que de parcourir toutes les lignes pour ne prendre que cette information.

- **Ajout de nouveaux types de données**

L'orientation en colonnes de la base permet l'ajout de nouvelles colonnes à la base avec une grande facilité, car contrairement à un SGBD relationnel, il n'est pas nécessaire de redimensionner les lignes.

- **Architecture MPP (Massively Parallel Processing)**

Souvent, les bases de données colonnes intègrent une couche de traitement MPP, basée sur des algorithmes comme « MapReduced », dont nous parlerons en détail dans la partie traitant d'Hadoop. De plus, les données sont souvent réparties sur plusieurs serveurs, et dupliquées. Les bases de données orientées colonnes se prêtent bien au clustering de base de données, permettant d'obtenir une haute disponibilité, et autorisant des traitements pouvant être répartis sur différents serveurs/clusters.

5.3.3 Points faibles

- **Efficace uniquement dans un contexte « BigData »**

L'utilisation de bases de données colonnes n'est vraiment efficace et profitable que dans le cas de grands volumes de données de mêmes types (une colonne pour chaque type), et dont les données au sens d'un type se ressemblent.

- **Interrogation des données limitée en complexité**

Le système de requêtes d'une base de données orientée colonnes est souvent minimaliste. Le stockage en colonne est plus optimisé pour récupérer de larges volumes de données qui se ressemblent, mais ce système a un prix : les opérations possibles pour interroger les données s'en voient grandement limitées par rapport à un SGBD relationnel. Seules des requêtes simples du type « select/update/delete » sont généralement possibles. Le choix d'une base colonnes se justifie dans le cas de grands volumes de données de même type dont on veut traiter massivement certains « attributs », mais pas dans le cas de données dont un traitement complexe et « ponctuel » doit être effectué lors de leur interrogation ou encore où il est nécessaire de conserver des liens comme dans un modèle relationnel.

5.3.4 Les acteurs

- **HBase**

Est une solution open source de type colonne basée sur la technologie NoSQL. Cette solution a été développée par « Apache Software Foundation ». Cette solution est utilisée par Facebook et entre autre.

- **Cassandra**

Cassandra est la solution la plus répandue. C'est une solution open source de type colonne basée sur la

technologie NoSQL. Celle-ci a été développée en java par « Apache Software Foundation »

5.4 Les bases de données orientées graphes [13]

La particularité des bases de données orientées graphe se fonde sur leur utilisation de nœuds, liens et propriétés au lieu de tables, lignes et colonnes, ceci afin d'obtenir de bien meilleures performances lors de la gestion de données associatives. Toutefois, leur implémentation varie selon les besoins et les propriétés à mettre en avant pour tel ou tel type d'utilisation. Il existe ainsi des bases orientées graphes se basant sur un enregistrement Key/Value, d'autre sur une orientation colonne ou BigTable, voire sur une combinaison de ces architectures.

5.4.1 Points forts

Parfaitement adaptées à la gestion de données relationnelles, même lorsque ces données atteignent une taille très importante, les bases de données orientées graphe sont une solution ayant l'avantage d'être basée sur le modèle mathématique des graphes. Cela permet d'y appliquer des algorithmes déjà existants et optimisés, faisant de ce type de base une architecture relativement aisée à comprendre et à prendre en main. Comme vu précédemment, il est également possible de modeler cette architecture afin d'obtenir un modèle plus proche des besoins rencontrés, en y ajoutant ou en y omettant des fonctionnalités.

Les bases de données orientées graphe font donc partie intégrante du NoSQL en proposant une alternative aux bases relationnelles pour toute problématique liée à une représentation relationnelle entre des données.

5.4.2 Points faibles

Cependant, cette architecture est très spécifique aux graphes, et la formation nœud/lien/propriété est très limitée, pour ne pas dire inadaptée, dans des cas plus classiques. Loin de remplacer les bases de données relationnelles, de même que d'autres types de bases NoSQL, les bases orientées graphe ne sont donc qu'une solution particulière à une forme particulière de BigData, et non une solution « miracle » pouvant répondre à tous les besoins de ce domaine.

5.4.3 Les acteurs

- **HyperGraphDB**

En terme d'intelligence artificielle, la solution ayant tendance à s'imposer repose sur une architecture de type neuronale, n'étant ni plus ni moins qu'un réseau de « neurones » dont les interconnexions sont évolutives. C'est dans cette optique qu'a été développé HyperGraph DB, une base orientée graphe sous licence LGPL. Une de ses particularités est de pouvoir relier des liens, et pas seulement des nœuds, par d'autres liens.
Egalement utilisée dans le cadre du web sémantique, HyperGraph DB se repose sur un modèle mathématique d'hypergraphe (chaque lien peut pointer sur plus de deux nœuds) probabiliste et pouvant s'auto-modifier, modèle se rapprochant beaucoup de la manière

dont fonctionne un cerveau. A l'instar de ce dernier, un très grand nombre de « neurones » (les nœuds, ici appelés atomes) et de liaisons sont toutefois nécessaires, d'où la nécessité de pouvoir manipuler ce très grand nombre d'éléments à travers une base de données orientée graphe.

- **Neo4j**

Neo4j est une base orientée graphe classique mais efficace et open source, respectant les conditions ACID et offrant une réplication maitre/esclave, ce qui lui vaut une certaine popularité.

6. Conclusion

Le mouvement NoSQL représente un ensemble de technologies proposant de nouvelles architectures pour les bases de données.

Ces nouveaux types de bases de données ont été stipulés pour s'aligner à l'évolution technologique actuelle liée surtout aux concepts de Big Data et Cloud Computing.

Les bases de données NoSQL permettent d'un coté de rendre le système beaucoup plus performant notamment avec le fort accroissement des volumes des données, et d'un autre coté elles permettent aussi de rendre le système plus résistants aux pannes en bénéficiant des possibilités du Cloud Computing.

Ce qu'il faut retenir, c'est que les bases de données de type NoSQL ne s'opposent pas aux bases de données de type relationnel ou autre, elles viennent plutôt combler les lacunes pour les cas qui favorisent la performance et la tolérance aux pannes.

Cependant, il est important de noter que pour les bases de données de type NoSQL étant des technologies encore très récente, il n'y a pas encore de normes qui permettent de définir une architecture type pour tel ou tel type de base de données.

.

CHAPITRE 3 :

MongoDB

1. Introduction

Ce chapitre va être consacré à un système de gestion de base de données NoSQL "orienté document" vers lequel on va migrer nos données relationnelles.

Dans la première partie, on va révéler des statistiques et des comparaisons entre les solutions SQL et NoSQL les plus populaires dans le monde.

En deuxième partie, nous allons étaler sur le modèle NoSQL qu'on a adopté à savoir MongoDB, qui fait partie des bases de type "Documentaire" présentant les données au format JSON (Java Script Object Notation). Il s'agit du système le plus populaire actuellement. MongoDB est particulièrement apprécié pour sa capacité à passer en mode distribué pour répartir le stockage et les traitements.

Ce chapitre se concentre sur MongoDB, vu comme une base distribué pour le stockage de documents JSON. MongoDB cherche à répondre aux besoins de performance, à garantir la scalabilité horizontale (réplication et sharding) en offrant de nombreuses fonctionnalités qu'on trouve dans le monde relationnel. Il intègre aussi le support de recherche full-text et les traitements de type MapReduce et aussi la recherche géo-spatiale.

2. Pourquoi MongoDB ?

Selon Julian Browne, « Lead IT architect » au sein de la célèbre compagnie de télécommunications britannique O2, *« Pour des raisons évidentes, il ne vaut vraiment pas la peine d'entreprendre un projet avec n'importe quel produit NoSQL, sauf si vous en avez besoin pour résoudre pour résoudre un problème »*. La technologie NoSQL répond à de nombreux problèmes, mais généralement elle répond à deux besoins c'est la gestion de gros volumes de données et la montée en charge.
Si aucun de ces deux problèmes n'est identifiée dans un domaine d'exploitation, il serait erroné d'adopter une solution NoSQL.
L'absence de normalisation est un aspect marquant de cette mouvance NoSQL et la panoplie de solutions existantes dans le marché (plus de 122 solutions NoSQL), mettent les différents décideurs devant un embarras dans le choix du modèle approprié par rapport à leur environnement d'exploitation.
Afin de réaliser notre projet et d'opter pour une éventuelle solution, nous avons développons une étude comparative sur les systèmes et les modèles les plus réussis pour l'adapter à notre application.
Nous avons choisi 3 bases de données de type différents (clé/valeur, orienté colonne et oriente document) et nous avons fixé un certain nombre de critères : [15]

- **Stockage de données primaire**

Apache Cassandra	Oui, sans problème
MongoDB	Oui, sans problème
Project Voldemort	Oui, sans problème

- **Utilisation de Cloud**

Apache Cassandra	Oui, il faut juste quelques configurations pour que Cassandra s'intègre facilement dans le « Cloud »
MongoDB	Oui, MongoDB peut facilement être déployé sur le Cloud.
Project Voldemort	Non, pas actuellement

- **La compatibilité avec les systèmes d'exploitation**

Apache Cassandra	Cassandra est compatible sur les plateformes Windows, Mac OS X et Linux (Ubuntu, Red Hat, CentOS)
MongoDB	MongoDB est compatible sur les plateformes Windows, Mac OS X, Linux (Debian, Ubuntu, Fedora, CentOS et Gentoo) et Solaris.
Project Voldemort	Voldemort est compatible sur les plateformes Windows et Linux (Debian)

- **Les librairies client pour les langages utilisés dans une application**

Apache Cassandra	C#,C++,Clojure,Erlang,Go,Haskell Java,JavaScript ,Perl,PHP,Python,Ruby, Scala
MongoDB	Plus de 27 librairies Actionscript , Lua ,MatLab , Perl,PHP, PowerShell ,Prolog ,Pythonn, Groovy R ,Ruby,Scala,Smalltalk ,C,C#,C++ Clojure ,ColdFusion ,D ,Dart ,Delphi Erlang,Go , Haskell,Java,javaScript, Lisp
Project Voldemort	Ruby,Node.js,PHP,Python

- **Les requêtes SQL**

Apache Cassandra	Oui, Cassandra possède un langage de requête type SQL nommé CQL.
MongoDB	Oui, MongoDB possède un équivalent de requête type SQL nommé Query Expression Objects.
Project Voldemort	Non, Voldemort ne possède pas de langage de requête type SQL ce qui est normal car Voldemort est une base de données de type clé/valeur et il n'est donc pas possible de faire des requêtes complexes sur les valeurs.

- **l'interface graphique et la gestion des tâches administratives**

Apache Cassandra	Dans la version DataStax Enterprise, une console graphique d'administration ainsi que de monitoring est disponible. La console se nomme DataStax OpsCenter.
MongoDB	10gen, la société ayant développé MongoDB, n'offre pas de console graphique de monitoring et de gestion ; cependant la communauté Open Source de MongoDB a développé divers interfaces de monitoring et d'administration.
Project Voldemort	Voldemort n'offre pas de console graphique, mais dispose d'une interface en ligne de commande permettant l'administration ainsi que le monitoring des noeuds du cluster.

- **Des communautés open source**

Apache Cassandra	Il existe une vaste communauté Open Source très active qui utilise Cassandra.
MongoDB	La communauté MongoDB est très active sur les forums/blogs ; Il existe de nombreux composants, tels que patchs, API, librairies client développés par la communauté MongoDB.
Project Voldemort	Voldemort possède une petite communauté Open Source.

- **Les documentations et les séances de formation**

Apache Cassandra	Une documentation exhaustive
MongoDB	Une documentation complète disponible en ligne sur le site de la solution
Project Voldemort	Peu de documentation disponible en ligne

2.1 Résultats de la comparaison

Après la comparaison faite, nous constatons que les deux solutions MongoDB et Cassandra ayant répondu présent sur tous les aspects ; néanmoins il faut trancher entre le modèle à adopter, Cassandra ou MongoDB ?

Il serait difficile de répondre à cette question, mais la réponse à cette question est liée au type de notre application. Si nous choisissions Cassandra, il sera plus facile à administrer (ajout de machine, montée en charge, etc.) mais il y a fort à parier que la phase de conception et la migration sera plus complexe, par contre que la souplesse des fichiers JSON nous a poussé à retenir le MongoDB.

D'un autre côté, des statistiques très récentes de Solid IT[1], publié en Mai 2015, ont révélé que MongoDB, est classé à la quatrième place de tous les SGBD, et premier de tous les SGBD NoSQL, à l'échelle mondiale.

Rank			DBMS	Database Model	Score		
May 2015	Apr 2015	May 2014			May 2015	Apr 2015	May 2014
1.	1.	1.	Oracle	Relational DBMS	1442.10	-4.03	-60.64
2.	2.	2.	MySQL	Relational DBMS	1294.26	+9.68	-14.83
3.	3.	3.	Microsoft SQL Server	Relational DBMS	1131.03	-18.09	-76.77
4.	4.	↑ 5.	MongoDB ⊞	Document store	277.32	-1.27	+52.70
5.	5.	↓ 4.	PostgreSQL	Relational DBMS	273.52	+5.20	+32.88
6.	6.	6.	DB2	Relational DBMS	201.04	+3.40	+14.57
7.	7.	7.	Microsoft Access	Relational DBMS	145.58	+3.39	+0.22
8.	8.	↑ 9.	Cassandra ⊞	Wide column store	106.55	+1.66	+24.82
9.	9.	↓ 8.	SQLite	Relational DBMS	105.16	+2.86	+15.87
10.	10.	↑ 13.	Redis	Key-value store	94.73	+0.17	+32.69

Figure 3-1 Classement MongoDB en 2015

[1] Site web : Solid IT "Classement NoSQL, Solid IT", http://db-engines.com/en/ranking

3. Présentation

MongoDB est un Système de Gestion de Base de Données tout comme MySQL, Oracle et bien d'autres, seulement, MongoDB est un SGBD orienté documents et non relationnel. En conséquence, les données ne sont pas stockées dans des tables sous forme de ligne/tuples, mais elles sont stockées dans des collections. Ces collections vont contenir des documents JSON, Ce système est classé NoSQL, ce qui veut dire que MongoDB n'utilise pas de langage Structured Query Language (SQL).

4. Caractéristiques de MongoDB

MongoDB (initialement 10gen) est l'un des rares systèmes de gestion de données NoSQL codés avec un langage qui offre de grandes performances : le C++. Les autres moteurs NoSQL populaires sont souvent codés en java ou dans des langages particuliers comme Erlang. Le terme documents ne signifie pas des documents binaires (images ou fichier son), mais une représentation structurée compréhensible par MongoDB d'une donnée complexe.

4.1 Structure des données

Le format BSON (Binary JSON) est une sérialisation binaire du format JSON. Celui-ci est un format ouvert et a été déployé afin de transmettre des données objets en JavaScript de manière à être lu par les humains. Ce format se résume à des paires clés/valeurs contenues entre accolades .BSON a été conçu pour :

- ✓ Etre léger en termes de stockage ainsi que de communication réseau.
- ✓ Etre traversable, c'est-à-dire que l'on peut accéder aux informations facilement et rapidement.
- ✓ Etre encodé et décodé de manière efficace
- ✓ Offrir plus de types de données que JSON

Un Document BSON représente donc l'unité stockée par MongoDB , qui est équivaut à une ligne dans une table relationnelle. Il est composé d'une hiérarchie de paire clé-valeur.[10]

```
{ "name" : "karesti",
  "duchess" : true,
  "hobbies" : {
              "danse" : "classique, moderne",
              "théâtre" : "improvisation, moderne",
              "autres" : "tapisserie niveau 5"
             }
}
```

Ce document sera stocké par MongoDB dans un format plus compact, plus pratique pour le stockage et le traitement.

Les documents sont contenus dans des collections, qui correspondent plus ou moins aux tables des SGBDR. Le MongoDB ne requiert pas le concept de contrainte comme le

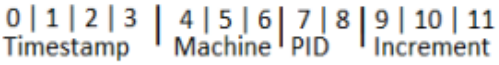
modèle relationnel. Il n'y a pas de vérification de cohérence effectuée par le serveur, toute la responsabilité est endossée à ce niveau par le code client. MongoDB stocke les documents tels qu'il reçoit. De plus, il est inutile de créer explicitement une collection avant d'y insérer des documents, car MongoDB en crée une, dés que le code client mentionne.

Il ya tout de même une contrainte nécessaire : chaque document est identifié par une clé unique dans la collection qui veut dire Chaque document JSON doit contenir une propriété _id qui identifie de manière unique votre document au sein de votre collection, pour sa valeur par défaut, MongoDB propose de générer lui-même des identifiants grâce aux ObjectId :

```
{"_id" : ObjectId("5197c6b453cce2ec3a743811")
}
```

Cela s'apparente au concept de clés techniques dans le systèmes relationnels, à la différence prés qu'il ne s'agit pas ici d'une incrémentation automatique mais de la génération d'un UUID qui est un type nommé Object ID dans MongoDB d'une taille de 96 octets :

- 4 bytes qui représentent le timestamp courant (nombre de secondes depuis epoch).
- 3 bytes pour identifier la machine.
- 2 bytes pour représenter l'identifiant du processus.
- 3 bytes qui représentent un compteur qui démarre à un numéro aléatoire.



Cet Object Id a donc une propriété très intéressante, il contient automatiquement la date de création de votre entité. Ce qui nous permet d'écrire le code suivant par exemple sur le Mongo Shell :

```
var id = ObjectId()
print(id)
ObjectId("5247019073ed0c203c79b995")
print(id.getTimestamp())
ISODate("2013-09-28T16:19:28Z")
```

4.2 Stocker des objets larges

MongoDB peut également stocker des documents binaires ou des objets larges, soit directement dans sa structure BSON, soit a l'aide de GridFS, qui est une spécification de stockage que toutes les pilotes implémentent, utilisée pour optimiser la manipulation et la sérialisation de fichier de taille importante ,à l'heure actuelle ,la taille maximale d'un document et de 16MO.

GridFS définit deux types de collections : files pour les métadonnées du fichiers, et chunks pour le contenu fichier. Si le fichier est volumineux, il sera automatiquement scindé en sous parties et chacune sera sauvée comme document dans la collection chunks représentant le contenu du fichier. Chaque document dans la collection files est caractérisé par un nom, une date d'envoi. Il contient aussi un champ unique _id qui peut être utilisé pour requêter la collection chunks et obtenir le contenu du fichier. Chaque document dans la collection chunks contient, un champ files_id identifiant le document de la collection, un attribut n numéro de segment qui est la position de ce document dans le fichier et aussi un attribut data qui correspond au segment.

4.3 Autres caractéristiques

Parmi les caractéristiques la démarquant des bases de données relationnelles, on retiendra :

- Le concept de « schéma » n'existe pas. Chaque document est libre de suivre sa propre structure.
- Jointures : les données sont généralement embarquées dans le même « document ». Cette forme de stockage peut ainsi être vue comme des jointures déjà exécutées (même si la possibilité de linkage entre documents existe surtout pour modéliser des relations N-M)
- L'atomicité des transactions n'est garantie que sur un seul document
- La modification concurrente doit être gérée au niveau de l'application
- La flexibilité, la performance et la scalabilité se font au détriment de la capacité de gérer des transactions complexes
- Terminologie : Table = Collection, Ligne = Document, Index = Index, Jointure = Données embarquées.

5. Les opérations CRUD[16]

Nous allons aborder maintenant les différentes opérations CRUD dont l'acronyme est **CREATE, READ, UPDATE, DELETE** (comme INSERT, SELECT, UPDATE et DELETE en SQL normalisé). Ces opérations vont nous permettre de réaliser la majorité des opérations sur les données (les documents).

Dans un premier temps, nous exécutons notre Shell mongo afin de nous connecter à la base de données, puis sélectionnons la collection appropriée et insérons nos premières données.

Ensuite, tout comme dans un Shell de MySQL ou tout autre SGBD, nous pouvons nous connecter à notre base de données. Pour cela, dans notre Shell mongo, on va saisir la commande suivante :

```
use maBDD
```

Le message ‘’switched to db maBDD ‘’ s'affiche .

Continuons, ici, par la création d'une collection dans laquelle nous allons, par la suite, stocker nos documents, exactement comme nous le ferions pour stocker nos données dans des tables, pour ceux qui sont habitués au SQL.
La commande suivante permet de créer une collection :

```
db.createCollection("maCollection")
```

Le message "{ "ok" : 1 " }" est visualisé, si celle-ci a bien été créée.

5.1 CREATE

La méthode permettant de créer des documents BSON est

```
db.maCollection.insert( { "_id" : "1", "nom" : "nom1" } )
```

Cette méthode va simplement insérer le document BSON que vous lui passez en paramètre. Par exemple, si vous souhaitez insérer le document ci-dessous :

```
{
"_id" : "1",
"nom" : "nom1"
}
```

Pour visualiser le contenu de notre collection nous utilisons la methode find( ) qui va permettre d'interroger et d'afficher les documents contenus dans la collection.
db.maCollection.find().
Ensuite pour visualiser la liste des documents contenus dans la collection, on exécute la commande Select similaire à celle du langage SQL :

```
SELECT * FROM maCommande;
```

5.2 READ

La méthode db.maCollection.find() prend en compte deux arguments : les critères et la projection. L'exemple suivant va illustrer le principe.

```
db.maCollection.find(critères, projection)
```

C'est à ce niveau que ça diverge avec le SQL avec la façon dont nous modélisons cette requête. Les critères correspondent aux champs contenus dans les documents que nous recherchons (en général ce qui se trouve après la clause WHERE en langage SQL). La projection, elle, va correspondre aux champs que nous souhaitons retourner dans notre résultat (en général ce qui se trouve après la clause SELECT en SQL).

```
db.maCollection.find( { } )
```

Permet d'afficher tous les documents contenus dans la collection maCollection.

5.3 UPDATE

Premier exemple avec la fonction update() qui va être la fonction principale pour mettre à jour nos documents MongoDB. La structure de la fonction update() est la suivante :

```
db.maCollection.update(sélection, modification, options)
```

Autre exemple :

```
db.universite.update(

{ "type" : "etudiant", "prenom" : "IMANE" },
{ $set : { "note" : 13 } },
{ upsert : true,
  multi: false
}
)
```

Supposons ici que nous travaillons sur une collection nommée "Université", et nous voulons attribuer une note de 13/20 à tous les étudiantes s'appelant Imane, la requête ci-dessus va permettre de le faire. On commence par sélectionner tous les étudiantes appelés Imane dans la collection, puis modifier la note de chaque étudiante nommée Imane à 13/20. Concernant les options Upsert et Multi : un Upsert est un mélange entre le mot Update et Insert. En fait, lors de notre mise à jour, si 'upsert' est définit à true, MongoDB va automatiquement créer le/les document(s) si ceux-ci n'existent pas déjà. Si dans ce cas, la mise à jour ne trouve aucune étudiante nommé Imane, alors un document sera créé pour cette étudiante avec la note spécifiée à 13/20. L'option Multi, va permettre de spécifier si l'on souhaite mettre à jour/modifier un seul document ou plusieurs. Si 'Multi' est définit à False, alors le premier document trouvé par l'opération Update sera mit à jour, par contre, si 'Multi' est définit à True, tous les documents concernés seront modifiés.

5.4 DELETE

La méthode remove() va être utilisée pour supprimer des documents dans une collection, tout comme la commande Delete du langage SQL. En voici sa structure :

```
Db.maCollection.remove(sélection, unique)
```

La sélection va, bien sûr, correspondre aux documents que nous cherchons à supprimer. Le second paramètre, "unique", indique si un seul document doit être supprimé ou plusieurs. Ce paramètre est similaire au paramètre "Multi" de la fonction Update() . Ce paramètre n'étant pas spécifié par défaut, la requête de suppression va supprimer tous les documents.

6. Indexation

Un index permet d'organiser des informations sur les valeurs de certains champs d'une collection. Il est nécessaire de bien définir le périmètre et le type des requêtes qui seront effectuées pour définir les index de la bonne manière. Dans un SGBD, qu'il soit relationnel ou NoSQL, l'indexation est souvent la clé de la performance et dans ces deux cas l'approche est assez similaire. Il est très simple de créer un index sur un champ choisi, cela permet une recherche par clé ou par séquence ordonnée performante. Sans cet index, le système devrait parcourir touts les documents d'une collection pour vérifier les valeurs de toutes les clés ; les index sont obligatoires pour le sharding (expliqué dan la section qui sui)t. Initialement tout document MongoDB possède un identifiant unique : le champ _id, un index est toujours créé sur ce champ et il ne peut être supprimé.

Pour mieux comprendre on va prendre un document JSON dont notre collection nommée profile, en premier temps on va visualiser le contenu de cette collection :

```
db.profile.find()
 { "_id" : ObjectId("5315c3437fec2f001e01c882"), "firstName" : "Hugo", "lastName" : "Lassiege" }
 { "_id" : ObjectId("5315c3567fec2f001e01c883"), "firstName" : "Jean-Baptiste", "lastName" : "Lemée" }
 { "_id" : ObjectId("5315c3647fec2f001e01c884"), "firstName" : "Olivier", "lastName" : "Girardot" }
 { "_id" : ObjectId("5315c36e7fec2f001e01c885"), "firstName" : "Florent", "lastName" : "Biville" }
 { "_id" : ObjectId("5315c3777fec2f001e01c886"), "firstName" : "Nicolas", "lastName" : "Rey" }
 { "_id" : ObjectId("5315c37f7fec2f001e01c887"), "firstName" : "Vincent", "lastName" : "Doba" }
 { "_id" : ObjectId("5315c38a7fec2f001e01c888"), "firstName" : "Jonathan", "lastName" : "Dray" }
 { "_id" : ObjectId("5315c3937fec2f001e01c889"), "firstName" : "Stuart", "lastName" : "Corring" }
```

Faisons une première recherche simple pour trouver une personne par son nom de famille :

```
db.profile.find({lastName:"Lassiege"})
 { "_id" : ObjectId("5315c3437fec2f001e01c882"), "firstName" : "Hugo", "lastName" : "Lassiege" }
```

Nous utilisons maintenant la méthode explain () pour voir le nombre des objets scannés pour avoir le résultat :

```
db.profile.find({lastName:"Lassiege"}).explain()
 {
 "cursor" : "BasicCursor",
 "isMultiKey" : false,
 "n" : 1,
 "nscannedObjects" : 8,
 "nscanned" : 8,
 "nscannedObjectsAllPlans" : 8,
 "nscannedAllPlans" : 8,
 "scanAndOrder" : false,
 "indexOnly" : false,
 "nYields" : 0,
 "nChunkSkips" : 0,
 "millis" : 0,
 "indexBounds" : {},
 "server" : "HUGO:27017"
 }
```

Cet explain plan nous indique que nous avons scanné nos 8 éléments pour trouver notre résultat. Dans cet exemple, le résultat est retourné très rapidement, mais parcourir l'ensemble des éléments d'une collection sur une grosse volumétrie est une problématique.

Pour remédier à ce problème nous allons ajouter un index pour améliorer notre recherche :

```
db.profile.ensureIndex({lastName : 1})
db.profile.find({lastName : {$in :["Lassiege"]}}).explain()
 {
 "cursor" : "BtreeCursor lastName_1",
 "isMultiKey" : false,
 "n" : 1,
 "nscannedObjects" : 1,
 "nscanned" : 1,
 "nscannedObjectsAllPlans" : 1,
 "nscannedAllPlans" : 1,
 "scanAndOrder" : false,
 "indexOnly" : false,
```

Le résultat s'est améliorée, nos deux requêtes passent désormais par cet index et effectuent une seule lecture par l'index. MongoDB est adapté à des besoins modestes de stockage de données, mais il est aussi un système de gestion de données conçu au départ pour être distribué, cette distribution pourra être réalisé avec deux méthodes la réplication et le partitionnement (Sharding).

7. Réplication

C'est le point fondamental de toute architecture clustérisée. La réplication se doit garantir la redondance des données, ainsi le basculement automatique des applications clientes est invisible en cas de panne matérielle ou logicielle.

Une grappe de serveurs partageant des copies d'un même ensemble de documents est appelée un Replicat Set (RS) dans MongoDB. Dans un RS, un des nœuds joue le rôle de

maître (primary*);* les autres esclaves (secondary). Nous allons nous en tenir à la terminologie maître-esclave pour rester en état cohérent. Un RS contient typiquement trois nœuds, un maître et deux esclaves. C'est un niveau de réplication suffisant pour assurer une sécurité presque totale des données. Dans une grappe MongoDB, on peut trouver plusieurs RS, chacun contenant un sous-ensemble d'une très grande collection. Le principe sera éclairci quand nous expliquons le partitionnement. Pour l'instant, on se limite à un seul Replica Set. [3]

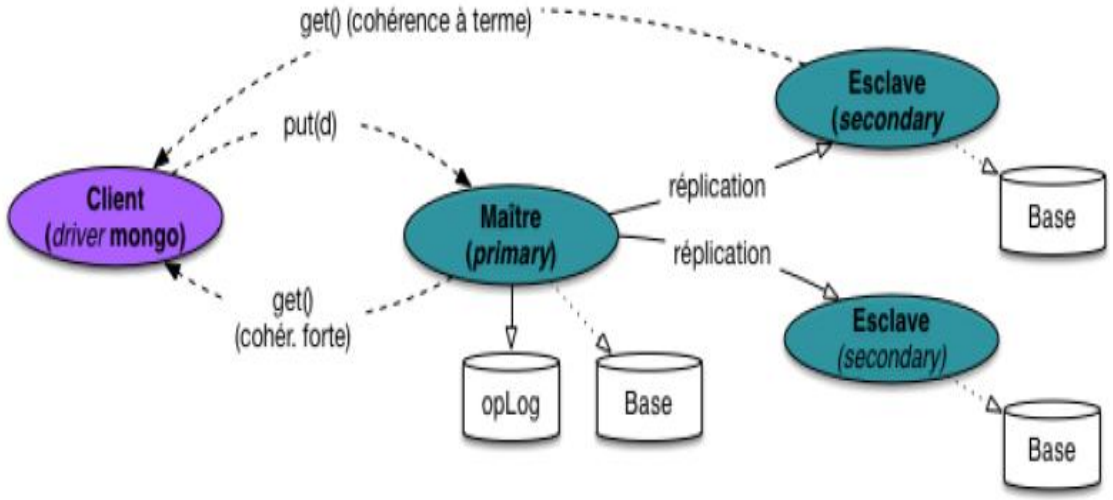


Figure 3-2 Replica-Set dans mongodb

La figure montre le fonctionnement de MongoDB. Le maître est ici chargé du stockage de la donnée principale. L'écriture dans la base est paresseuse, et un journal des transactions est maintenu par le maître (c'est une collection spéciale nommée opLog). La réplication vers les deux esclaves se fait en mode asynchrone qui veut dire Les membres secondaires du replica set copient les opérations du membre primaire, depuis l'opLog, de manière asynchrone, peut importe le temps que prendra la copie d'une opération sur un des ensemble de données, le membre primaire et le/les autre(s) secondaires continuerons d'exécuter leurs tâches. En conséquence, un membre secondaire ne retourne pas forcément les données les plus récentes immédiatement.

MongoDB propose deux types de cohérence :

- **Cohérence forte** : La cohérence forte est obtenue en imposant au client d'effectuer toujours les lectures via le maître. Les esclaves ne servent pas à répartir la charge, mais jouent le rôle restreint d'une sauvegarde/réplication continue, avec remplacement automatique du maître si celui-ci subit une panne. On ne constatera aucune différence dans les performances avec un système constitué d'un seul nœud.
- **Cohérence à terme :** La cohérence à terme est obtenue en autorisant les clients à effectuer des lectures sur les esclaves.

7.1 La reprise sur panne dans MongoDB

Une situation de failover fait référence au fait qu'un serveur primaire échoue et ne réponde plus, on va donc transférer les opérations vers d'autres serveurs. Ici, c'est ce qu'il se passe avec MongoDB, si le membre primaire ne répond plus pendant 10 secondes, une élection a lieue pour désigner un des membres secondaires en tant que nouveau primaire. Le premier membre secondaire qui obtient la majorité de votes devient donc le membre primaire du replica set. Les battements de coeur ou"heartbeats", comme indiqués sur les images vont vérifier continuellement si un chaque membre répond ou non.

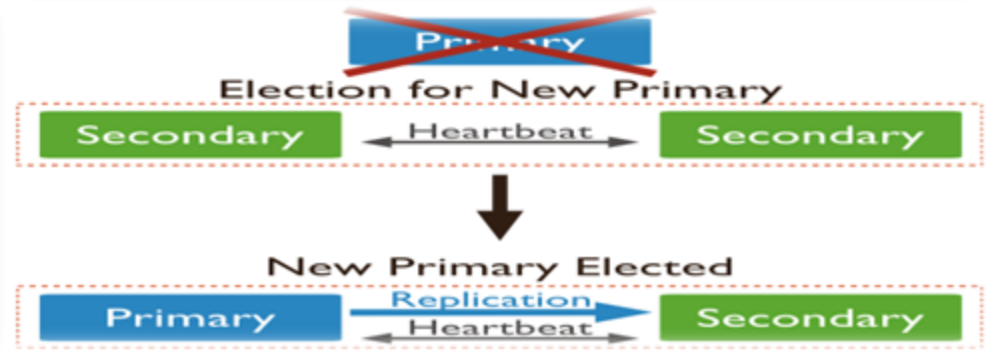


Figure 3-3 Election d'un primaire en cas de panne

Dans l'exemple ci-dessus l'application cliente communique avec le nœud maitre pour ses transactions d'écriture et de lecture. Sur le nœud Esclave, la lecture seule est autorisée. Le nœud Arbitre n'a qu'un rôle de supervision, si le nœud maitre connait un arrêt de service, le nœud arbitre désigne alors l'esclave comme nouveau maitre. L'opération est invisible aux applications clientes et aucune perte de données n'est à déplorer.

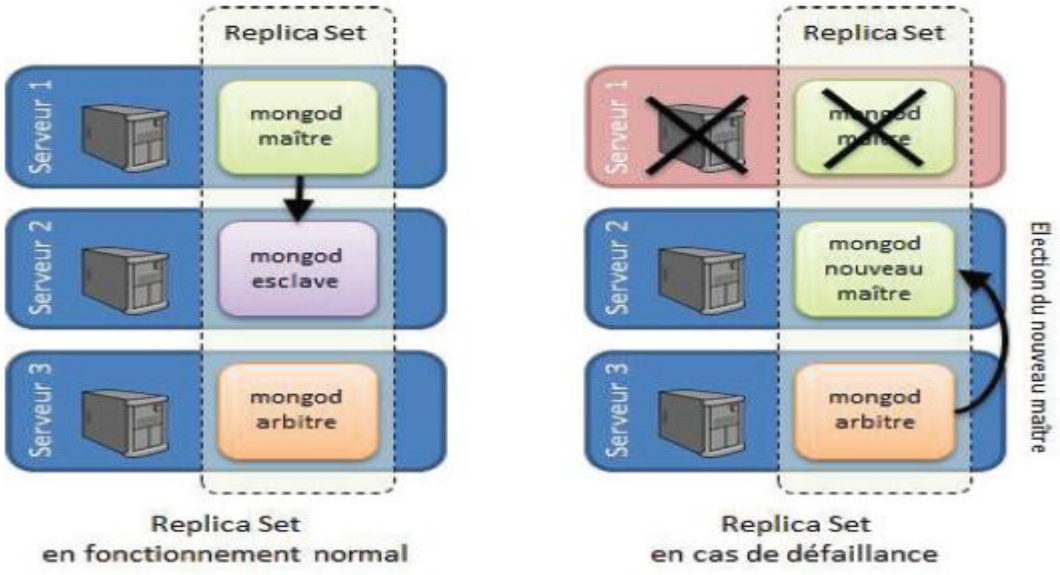


Figure 3-4 Situation fail-over 1

La réplication est destinée à résoudre le problème de panne en dupliquant les données sur plusieurs serveurs et en permettant donc qu'un serveur prenne la relève quand une autre tombe en panne. Le fait d'avoir des mêmes données sur plusieurs serveurs par réplication ouvre la voie à la distribution de la charge et donc la Scalabilité. C'est pour cette raison en nous passons à une autre technique, le partitionnement.

8. Repartitionnement (Sharding)

Fonctionnalité phare de MongoDB, le sharding définit les partitions horizontales d'une collection donc c'est la collection et non pas une base de donnée qu'on répartit. Les documents de la collection seront distribués équitablement entre les différents nœuds pour que le traitement soit équilibre. Supposons qu'on a une collection de taille 1TO et on a 4 Shard alors que chaque Shard a un sous document de la collection de taille 256 GB.[3]

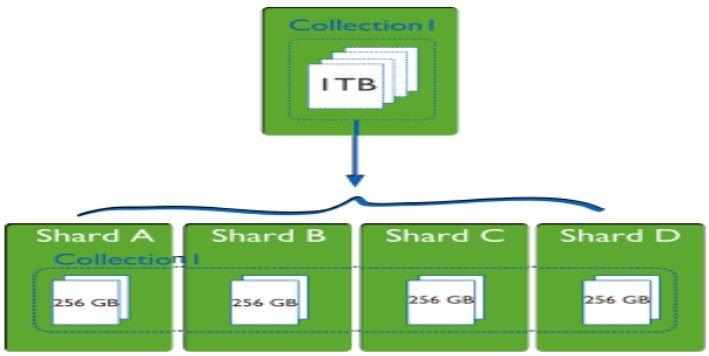


Figure 3-5 partitionnements d'une collection

Dans l'architecture de MongoDB avec réplication et partitionnement, les nœuds du système sont classés en trois catégories :

- Les routeurs (processus Mongos) communiquent avec les applications clientes, se chargent de diriger les requêtes vers le serveur de stockage concerné, et transmettent les résultats.
- Les replicat set (processus Mongod) sont en charge d'un ou plusieurs fragments (Shard) et gèrent localement la reprise d'une panne par réplication.
- Des config servers (processus Mongod avec l'option configsrv) stockent les données de routage et plus généralement la configuration complète du système : la liste des replica set, les maîtres et esclave pour chaque replica set, liste des fragments et allocation des fragments à chaque replica set.

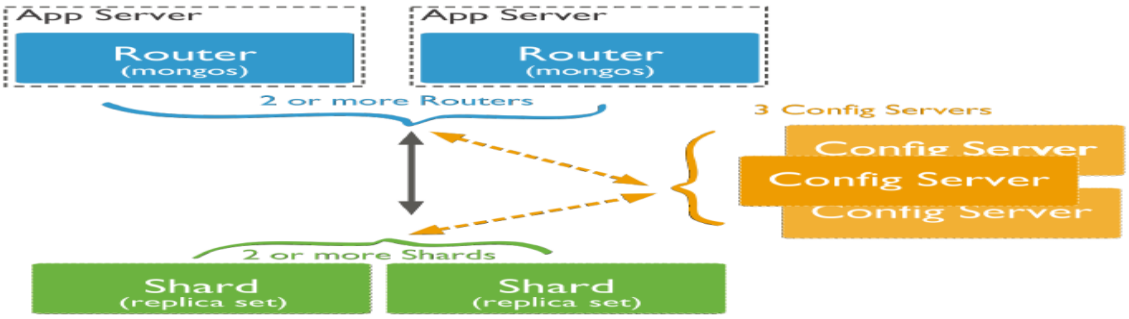


Figure 3-6 Architecture du MongoDB

8.1 Clé de partitionnement

Un partitionnement se fait toujours en fonction d'une clé, donc la première décision à faire c'est le choix de la clé.

Il existe deux méthodes pour déterminer la clé de partitionnement : par intervalle et par hachage.

8.1.1 Partitionnement par intervalle

On considère le domaine de valeur de la clé de partition et on le devise en n intervalles définissant n fragments.

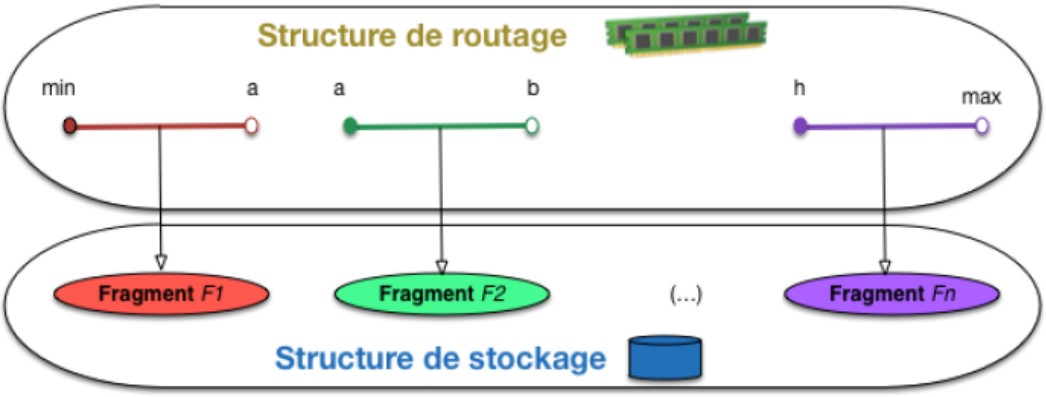


Figure 3-6 Partitionnement par intervalle

8.1.2 Partitionnement par hachage [17]

Le partitionnement par hachage dans un environnement distribué repose globalement sur la même organisation que pour le partitionnement par intervalle. Un routeur maintient une structure qui guide l'affectation des documents aux serveurs de stockage, Chaque serveur étant localement en charge de gérer le fragment qui lui est alloué. Cette structure au niveau du routage est la table de hachage établissant une correspondance entre les valeurs des clés et les adresses des fragments.

Le hachage repose sur la fonction de hachage qui distribue les valeurs de clé vers un intervalle [0,n-1]. (n correspondant au nombre de fragments)

Le principe du hachage cohérent est de considérer dès le départ un intervalle immuable D = [0, n-1] pour le domaine d'arrivée de la fonction de hachage, où n est choisi assez grand pour réduire le nombre de collisions.

On interprète ce domaine comme un anneau parcouru dans le sens des aiguilles d'une montre, de telle sorte que le successeur de 2^{64} est 0. La fonction de hachage associe donc chaque serveur de la grappe à une position sur l'anneau ; on peut par exemple prendre l'adresse IP d'un serveur, la convertir en entier et appliquer mod 2^{64}, ou tout autre.

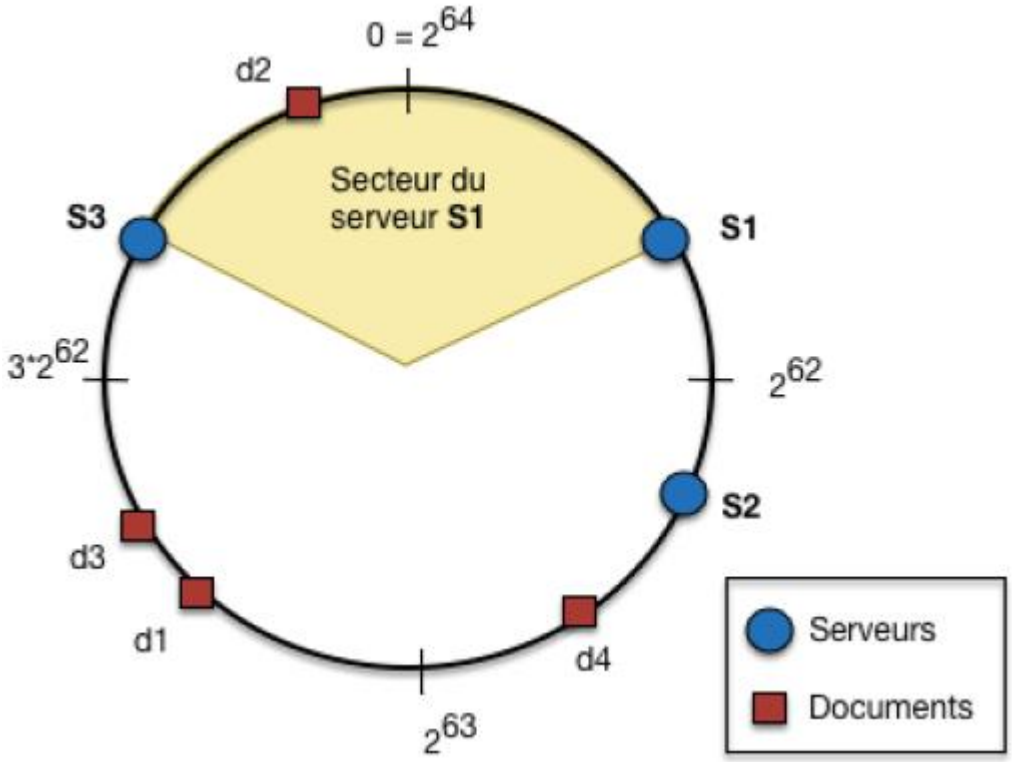


Figure 3-7 L'anneau du hachage consistent et la règle d'affectation

La règle d'affectation est alors définie de la manière suivante : chaque serveur est en charge de l'arc cercle qui le précède sur l'anneau. Par exemple le serveur S2 est positionné par la fonction de hachage en $h(S2) = b$, quelque part entre 2^{62} et 2^{63}, chaque serveur doit stocker le fragment de la collection correspondant aux objets positionnés sur l'arc de cercle dont il est responsable.
Dans cette figure on remarque que le serveur S1 stockera les le document d2,
S3 stockera les documents d3 et d1 par contre que le serveur S2 aucun document n'est stocké.
Un premier problème pratique apparaît immédiatement : les positions des serveurs étant déterminées par la fonction de hachage indépendamment de la distribution des données, certains serveurs se voient affecter un tout petit secteur, et d'autres un très grand.
On remarque que cette figure pose un problème, c'est le déséquilibre entre S2 et S3 donc pour résoudre ce problème, il faut placer les serveurs sur plusieurs positions sur l'anneau, ce qui tend à multiplier les arcs de cercles, et par un effet d'uniformisation rendre leurs tailles comparables.

Après avoir détaillé les différents éléments d'une architecture MongoDB, nous allons voir maintenant comment organiser un cluster de machines afin d'avoir une répartition de charge cohérente et une haute disponibilité avec le minimum de serveurs. Nous allons prendre comme exemple une base MongoDB seulement avec trois machines pour la base, plus deux serveurs PHP y accédant. Alors, nous avons 3 serveurs dédiés à la base de données qui nous amène à créer trois Shards pour répartir les données sur ces trois machines, le but c'est garantir la haute disponibilité, chaque Shard sera donc lui

même un replica-set. Un replica set a besoin de 3 machines: un maître, un esclave et un arbitre. Afin de ne pas poser toute la charge sur une même machine, nous allons alterner les rôles.
Ainsi, le Shard A sera hébergé sur le serveur 1 (maître), avec un backup sur le serveur 2 (esclave) alors que le serveur 3 servira d'arbitre. Pour le Shard B, on décale tout d'un serveur: le serveur 2 devient le maître, le serveur 3, l'esclave et le 1 l'arbitre. Et ainsi de suite.
Enfin, nous créons 3 instances de config servers sur les trois serveurs. Ces instances ne consomment quasiment pas de puissance CPU ou de disque et sont indispensables au fonctionnement de la base.
Imaginons qu'un serveur, par exemple le serveur 2, tombe en panne. Le Shard A perd donc son backup, mais il peut toujours fonctionner car le serveur maître est le serveur 1. Le Shard B perd quant à lui son serveur maître. Le serveur 1, qui est l'arbitre du Shard B va détecter la défaillance du serveur 2 et demander au serveur 3 de devenir maître du Shard B. Quant au Shard C, il perd l'arbitre, ce qui n'aura pas d'impact tant que le serveur 1 et le serveur 3 communiquent correctement. Ces 3 Shard se comportent alors normalement et la défaillance du serveur 2 n'aura donc pas d'impact sur la disponibilité du service. Dans cet exemple, Il faudrait qu'un incident survienne sur 2 machines simultanément pour que le service soit interrompu. Or MongoDB ne limite pas le nombre d'esclaves présents dans un Replica Set. Il est donc possible de se prémunir de la perte de deux serveurs en rajoutant des serveurs esclaves.

9. Conclusion

Malgré son jeune âge, MongoDB brille par ses fonctionnalités et ses performances, dans la mesure où la modélisation en document est peu contraignante et correspond à de nombreux cas d'utilisation. Elle dispose d'une capacité à évoluer en environnement distribué via des mécanismes de réplication et de sharding. Son intégration est particulièrement réussie avec la plupart des langages de programmation ainsi que sa documentation de qualité lui confère une popularité importante. MongoDB profite du fort regain d'intérêts pour les bases documentaires qui permettent de mieux coller aux environnements modernes qui se doivent de manipuler des données fortement hétérogènes et pour lesquels les SGBD relationnels ne sont pas nécessairement les plus adaptés. A noter que de nombreux projets open source tendent à considérer l'intégration de MongoDB en tant que moteur de stockage.

CHAPITRE 4 :

Implémentation et mise en œuvre

1. INTRODUCTION

De nombreuses organisations stockent leurs données dans des SGBDR et espèrent aujourd'hui profiter des bases de données NoSQL qui ont émergé récemment, alors leur tendance actuelle est de migrer leurs bases de données classiques vers des bases de données qui garantissent le passage à l'échelle et une meilleure évolutivité afin de répondre aux besoins de performance et de disponibilité de données face à des volumes de données en croissance permanente.

Pour réussir un basculement pareil, nous avons besoin de définir une certaine démarche. L'objectif principal de notre projet est la migration d'une BDD SQL (Relationnelle) vers une BDD NoSQL orientée document. Cette migration consiste à la traduction du schéma source en un schéma cible ainsi que la conversion des données.

2. ENVIRONNEMENT DE TRAVAIL

Dans cette section, nous allons recenser les outils Soft et Hard utilisés pour réaliser notre application ainsi que ses principales interfaces.

2.1 Environnement hard

- Hôte : Acer ASPIRE 5749
- Microprocesseur : intel ® Core ™ i3-2350M
- RAM : 4GO

2.2 Langage de programmation

Java comme langage de programmation orienté objet, développé par Sun Microsystems et destiné à fonctionner dans une machine virtuelle, il permet de créer des logiciels compatibles avec des nombreux systèmes d'exploitation.

2.3 Outil de développement

NetBeans version 8.0, est un environnement de développement intégré (EDI), placé en open source par Sun en juin 2000 sous licence CDDL et GPLv2 (Common Development and Distribution License).

2.4 Système de gestion de base de données

Nous avons utilisé deux types de système de gestion de bases de données. Pour le relationnel on a opté pour le MySQL et pour le NoSQL c'est MongoDB.

2.4.1 MySQL

Nous avons utilisé MySQL Workbench, version 6.2 CE qui fait partie des logiciels de gestion de base de données relationnelles les plus utilisés au monde.

MySQL est un serveur de bases de données relationnelles développé dans un souci de performances élevées en lecture, ce qui signifie qu'il est davantage orienté vers le service de données déjà en place que vers celui de mises à jour fréquentes et fortement sécurisées. Il est multithread et multi-utilisateur.

2.4.2 MongoDB

Pour la solution NoSQL, nous avons opté pour la version 2.6 de MongoDB comme un modèle orienté document.

2.5 Format de données communiquées

Le format de données communiquées c'est JSON, qui est un format de données textuel, générique, Il permet de représenter des informations structurées.
Le principal avantage de l'utilisation de JSON, est sa simplicité de mise en œuvre. Au rang des avantages, nous pouvons également citer :

- Facile à apprendre, car sa syntaxe est réduite et non-extensible.
- Ses types de données sont connus et simples à décrire.

La démarche que nous allons suivre pour aboutir notre projet est la suivante :

1. Création base de données sous MySQL Workbench
2. Génération des fichiers JSON contient les enregistrements des tables
3. Installation MongoDB et l'importation des fichiers JSON

3. 1ere PHASE : Création de la base de données et insertion des données sous MySQL Workbench

La base de données sur laquelle on va appliquer la migration vers un système NoSQL orienté document, est celle d'un réseau social gérée, par le moteur MySQL. Voici une présentation globale de son contenu :

3.1 Concept de réseau social

Le terme désigne une plateforme web permettant à l'internaute de s'inscrire et d'y créer une carte d'identité virtuelle appelée le plus souvent « profil ». Le réseau est dit social en ce qu'il permet d'échanger avec les autres membres inscrits sur le même réseau : des messages publics ou privés, des liens hypertextes, des vidéos, des photos, des jeux… L'ingrédient fondamental du réseau social reste cependant la possibilité d'ajouter des «amis», et de gérer ainsi une liste de contacts.
Le modèle conceptuel de données d'un réseau social est le suivant :

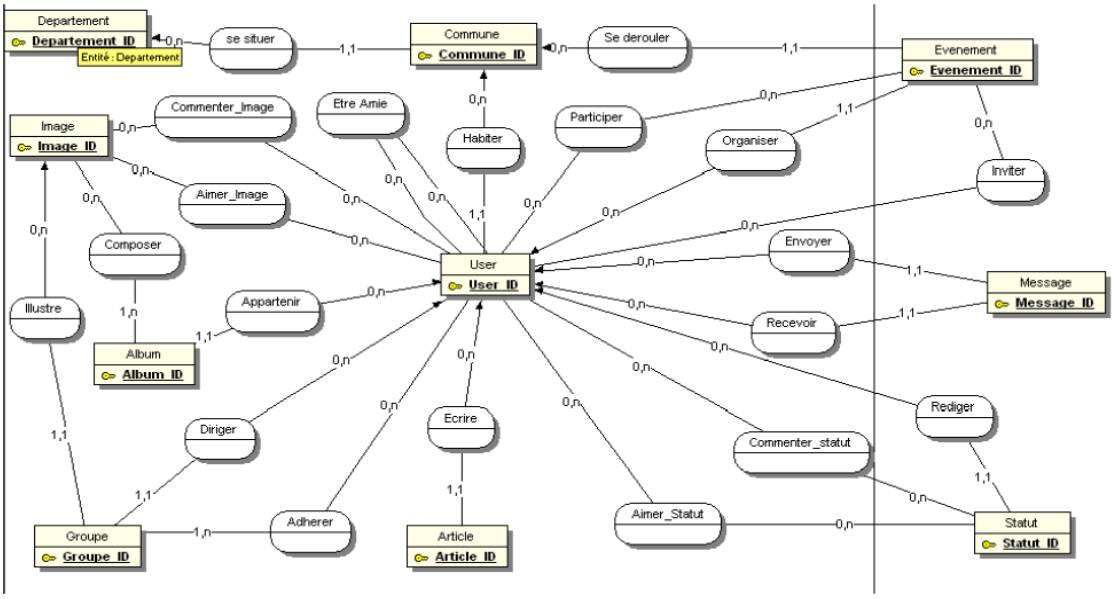


Figure 4-1 MCD réseau social

Après la transformation, on obtient le modèle logique de données ci-dessous qui compte dix huits (18) relations. Le MLD ci-dessous fera l'objet d'une traduction afin de

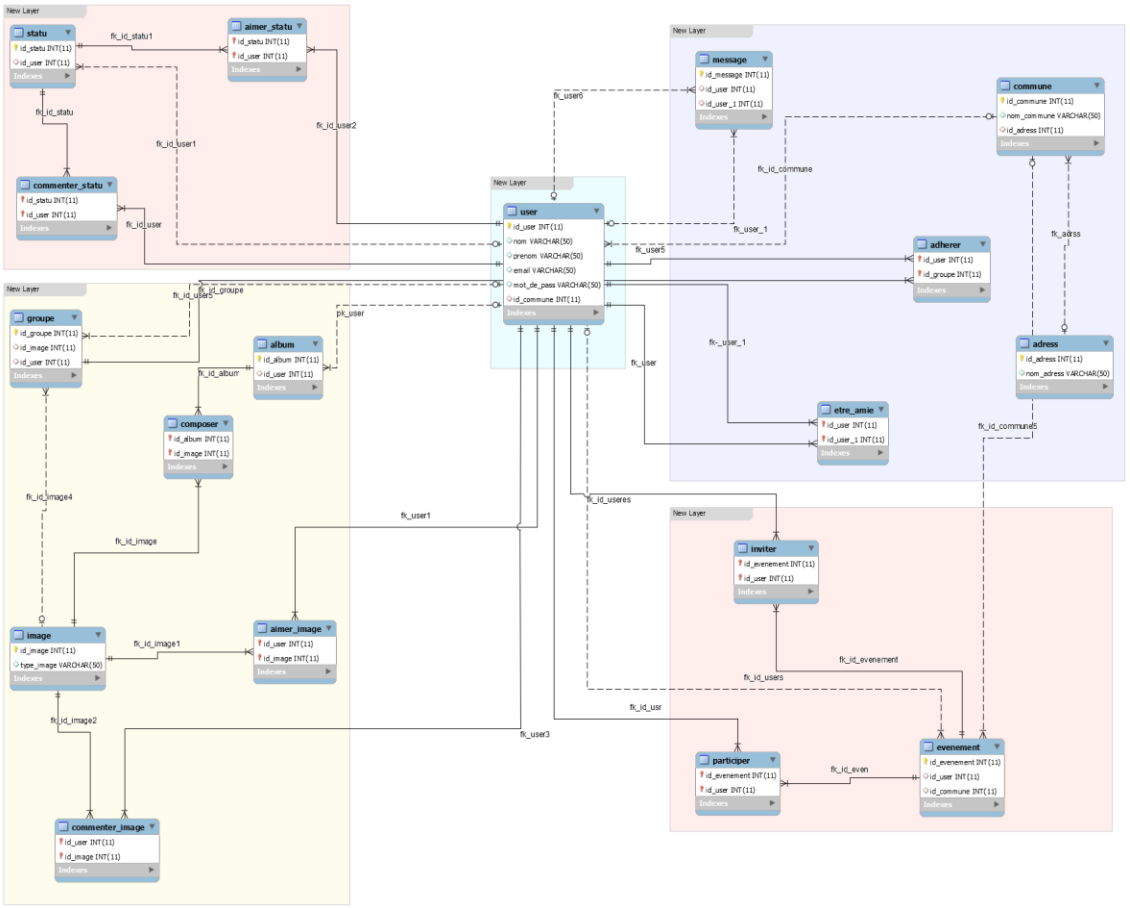
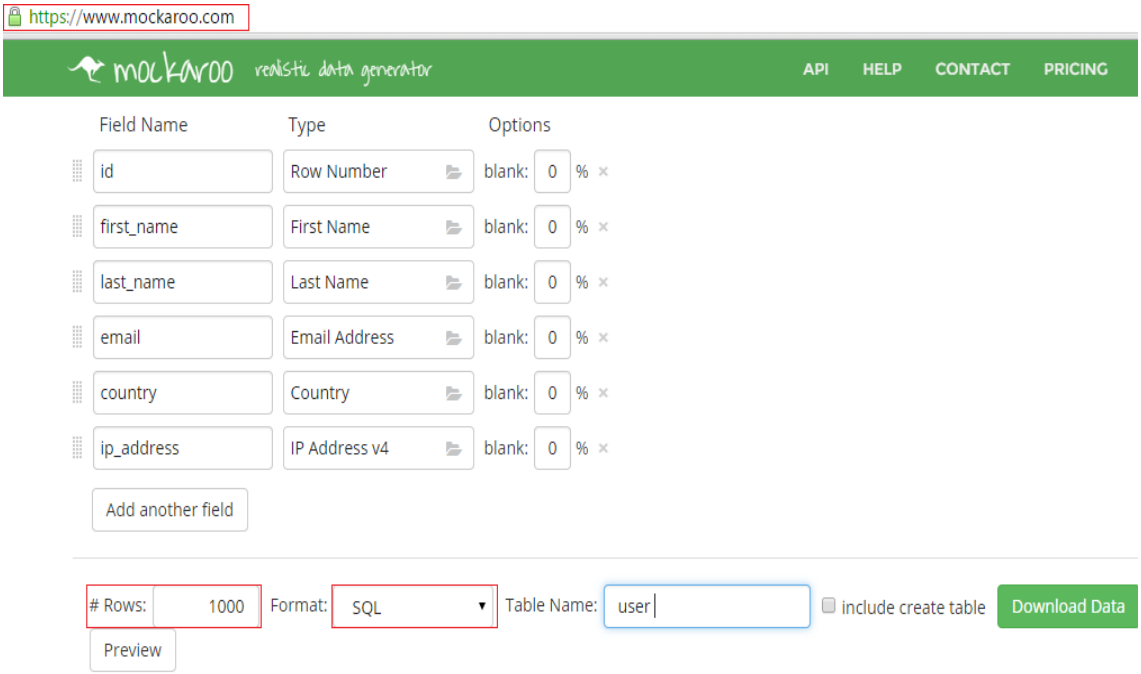


Figure 4-2 MLD reseau_social

déterminer son équivalent en base de données NoSQL Orientée document.

Après la création des tables de la base de données, nous allons utiliser un générateur de données pour remplir toutes les tables avec les données correspondantes.
Mockaroo offre un service gratuit pour générer des données qui peut être personnalisé pour émuler nos besoins de conception et les données des tables. Les données peuvent être exportées vers un certain nombre de formats mais dans notre cas, on va s'intéresser au format SQL.

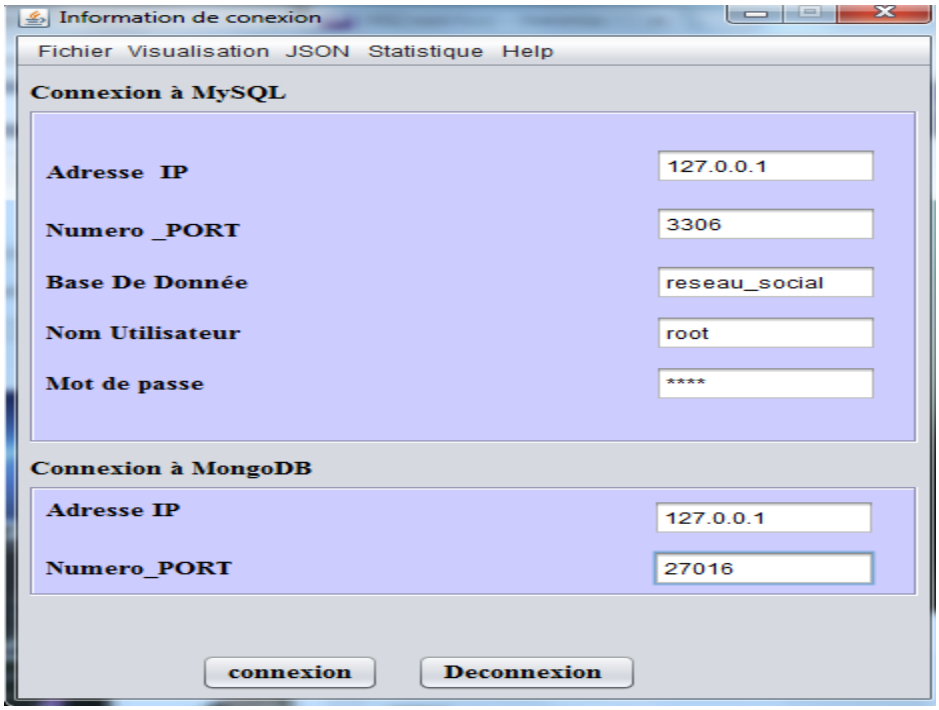


L'image ci-dessous explique comment enregistrer un script « .SQL » d'INSERT qui contient 1000 enregistrements de la table « User ». Après la génération du script, nous importons ce fichier à MySQL en utilisant l'option « importer » puis on exécute ce script pour remplir la table concernée.

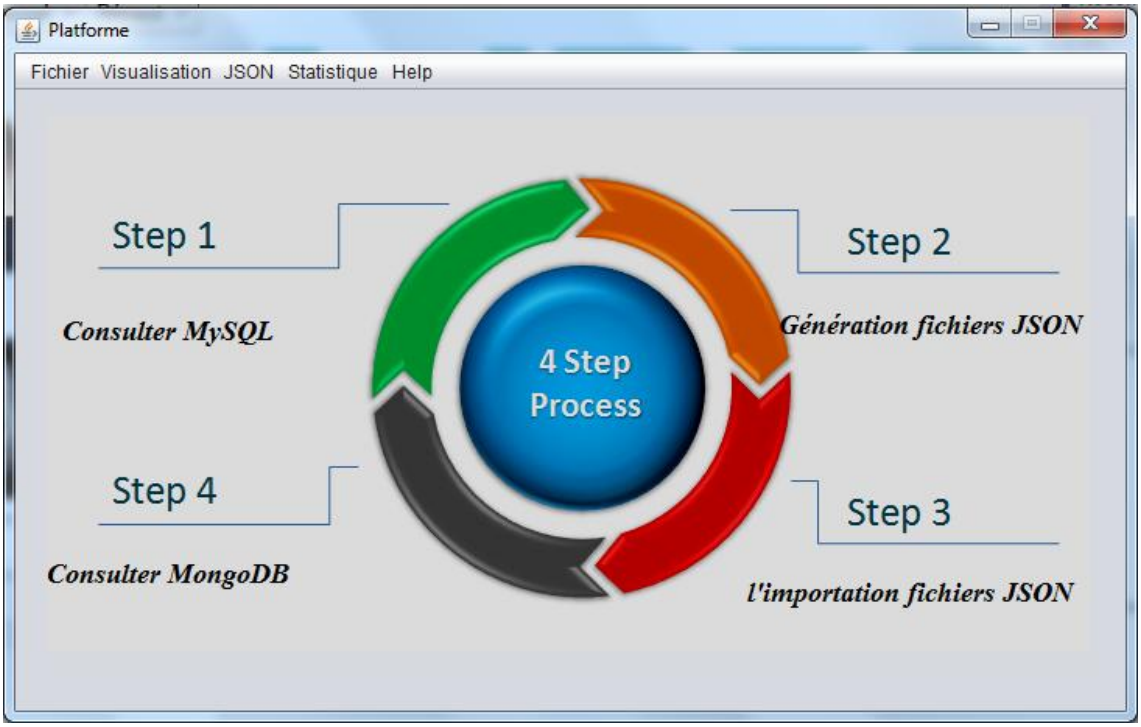
4. 2éme PHASE : Génération des fichiers JSON

Cette phase va consister à la connexion à MySQL, visualisation du contenu de la base de données et génération des fichiers JSON.

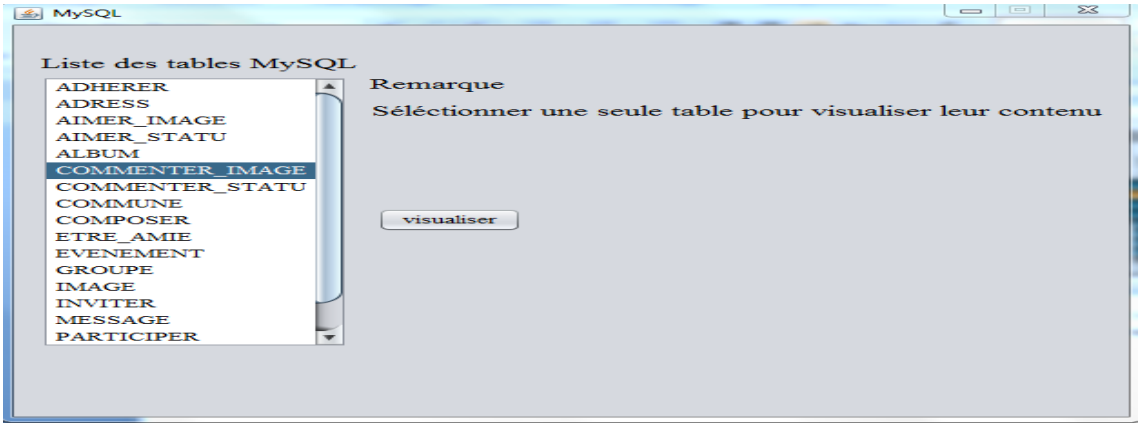
Nous commençons par se connecter à MySQL en utilisant comme adresse IP Localhost ou 127.0.0.1, le numéro de port du service MySQL par default c'est 3306, le nom de notre base de donnée c'est « reseau_social » et « root » comme nom d'utilisateur et mot de passe.



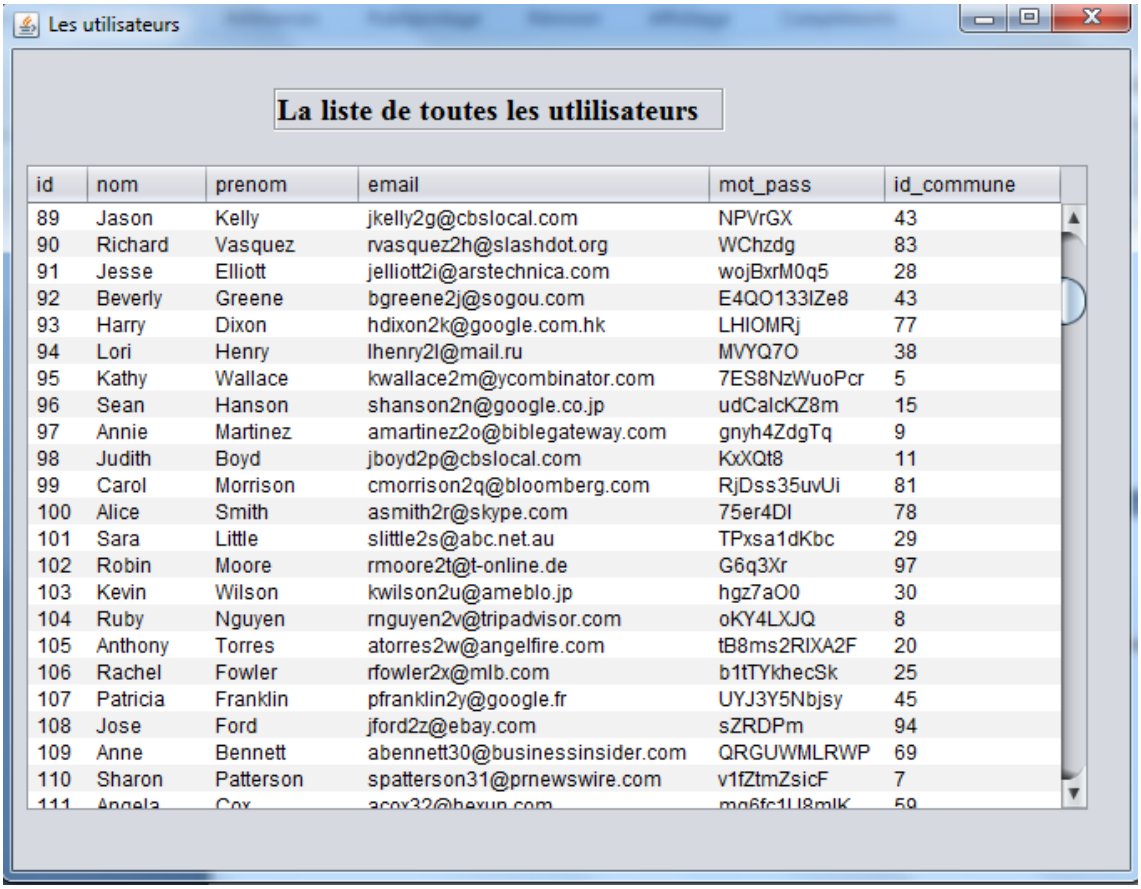
Après la connexion la plateforme de l'application s'affichera



Une fois connecté, le menu visualisation/BDD_MySQL permet de visualiser toutes les tables de la base de données. Pour consulter le contenu d'une table, il faut juste la sélectionner et cliquer sur visualiser.

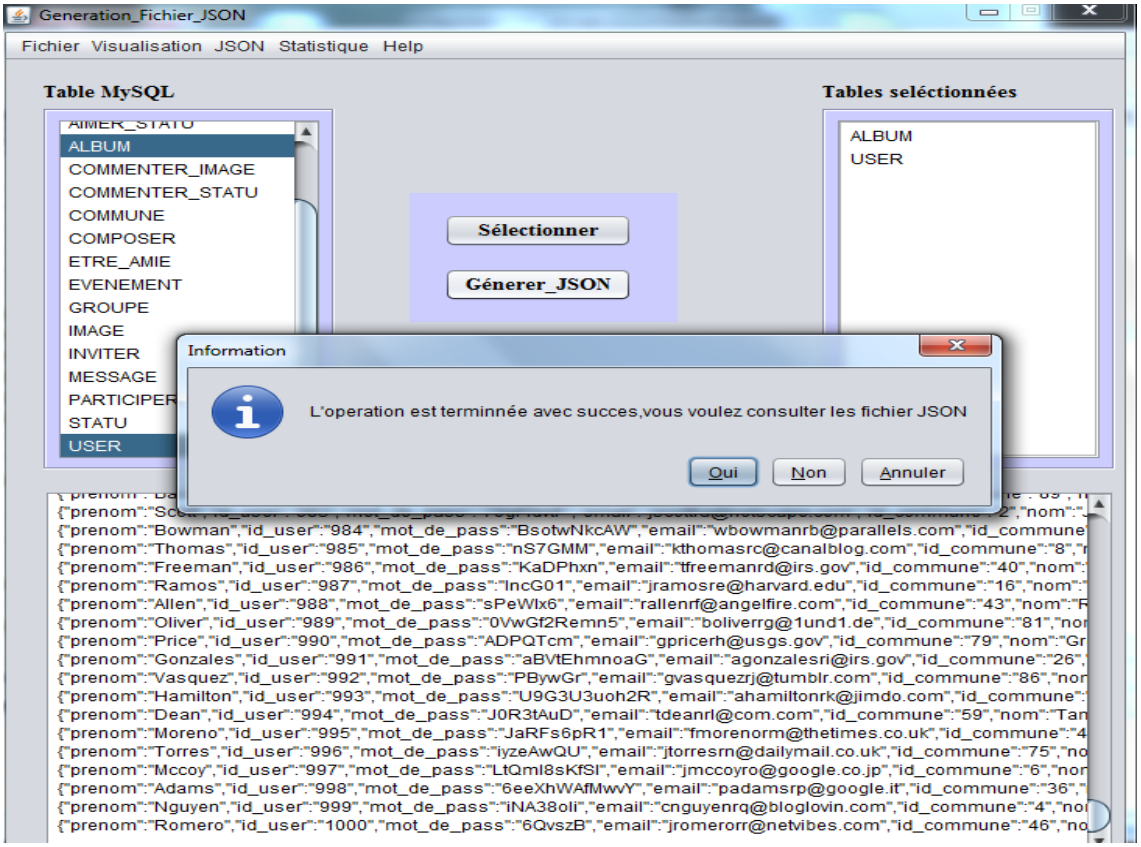


Le contenu de la table « user »

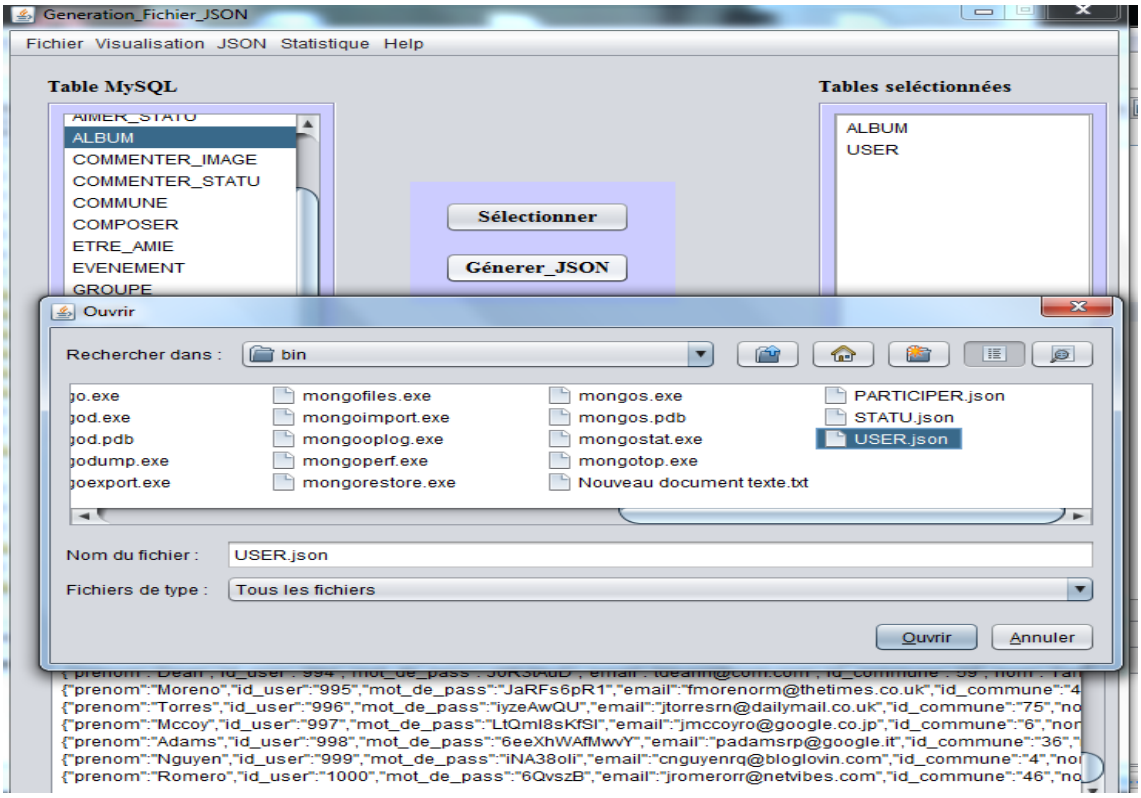


id	nom	prenom	email	mot_pass	id_commune
89	Jason	Kelly	jkelly2g@cbslocal.com	NPVrGX	43
90	Richard	Vasquez	rvasquez2h@slashdot.org	WChzdg	83
91	Jesse	Elliott	jelliott2i@arstechnica.com	wojBxrM0q5	28
92	Beverly	Greene	bgreene2j@sogou.com	E4QO133IZe8	43
93	Harry	Dixon	hdixon2k@google.com.hk	LHIOMRj	77
94	Lori	Henry	lhenry2l@mail.ru	MVYQ7O	38
95	Kathy	Wallace	kwallace2m@ycombinator.com	7ES8NzWuoPcr	5
96	Sean	Hanson	shanson2n@google.co.jp	udCalcKZ8m	15
97	Annie	Martinez	amartinez2o@biblegateway.com	gnyh4ZdgTq	9
98	Judith	Boyd	jboyd2p@cbslocal.com	KxXQt8	11
99	Carol	Morrison	cmorrison2q@bloomberg.com	RjDss35uvUi	81
100	Alice	Smith	asmith2r@skype.com	75er4DI	78
101	Sara	Little	slittle2s@abc.net.au	TPxsa1dKbc	29
102	Robin	Moore	rmoore2t@t-online.de	G6q3Xr	97
103	Kevin	Wilson	kwilson2u@ameblo.jp	hgz7aO0	30
104	Ruby	Nguyen	rnguyen2v@tripadvisor.com	oKY4LXJQ	8
105	Anthony	Torres	atorres2w@angelfire.com	tB8ms2RIXA2F	20
106	Rachel	Fowler	rfowler2x@mlb.com	b1tTYkhecSk	25
107	Patricia	Franklin	pfranklin2y@google.fr	UYJ3Y5Nbjsy	45
108	Jose	Ford	jford2z@ebay.com	sZRDPm	94
109	Anne	Bennett	abennett30@businessinsider.com	QRGUWMLRWP	69
110	Sharon	Patterson	spatterson31@prnewswire.com	v1fZtmZsicF	7
111	Angela	Cox	acox32@hexun.com	mq6fc1U8mlK	59

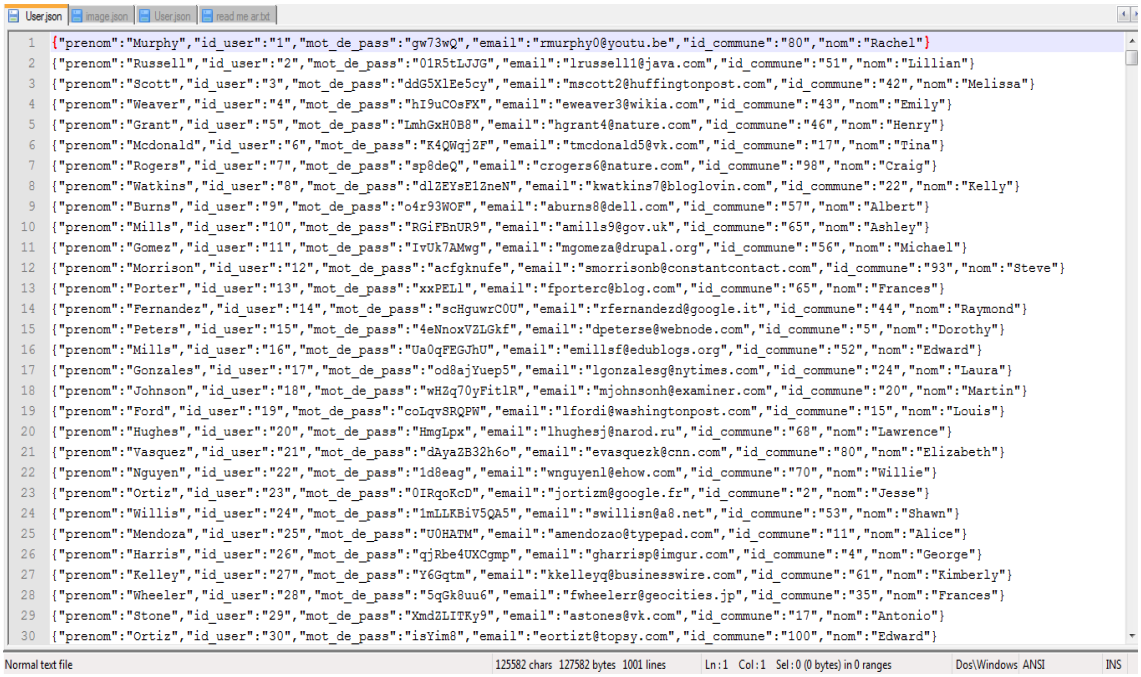
Le menu JSON/GENERER_JSON permet de générer les fichiers JSON d'une table ou plusieurs tables sélectionnées en même temps.



Dés que l'opération se termine avec succès, on peut consulter les fichiers JSON crées.



Un extrait du contenu du fichier USER.JSON généré :



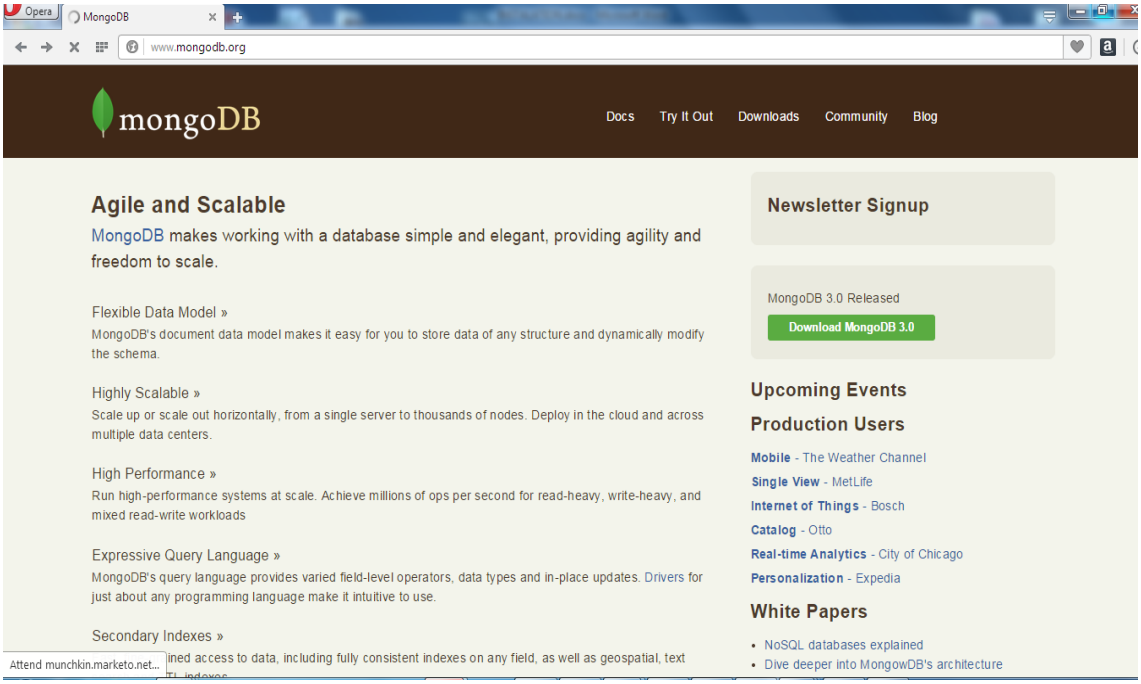

```
{"prenom":"Murphy","id_user":"1","mot_de_pass":"gw73wQ","email":"rmurphy0@youtu.be","id_commune":"80","nom":"Rachel"}
{"prenom":"Russell","id_user":"2","mot_de_pass":"01R5tLJJG","email":"lrussell1@java.com","id_commune":"51","nom":"Lillian"}
{"prenom":"Scott","id_user":"3","mot_de_pass":"ddG5XlEe5cy","email":"mscott2@huffingtonpost.com","id_commune":"42","nom":"Melissa"}
{"prenom":"Weaver","id_user":"4","mot_de_pass":"hI9uCOsFX","email":"eweaver3@wikia.com","id_commune":"43","nom":"Emily"}
{"prenom":"Grant","id_user":"5","mot_de_pass":"LmhGxH0B8","email":"hgrant4@nature.com","id_commune":"46","nom":"Henry"}
{"prenom":"Mcdonald","id_user":"6","mot_de_pass":"K4QWqjZF","email":"tmcdonald5@vk.com","id_commune":"17","nom":"Tina"}
{"prenom":"Rogers","id_user":"7","mot_de_pass":"sp8deQ","email":"crogers6@nature.com","id_commune":"98","nom":"Craig"}
{"prenom":"Watkins","id_user":"8","mot_de_pass":"dlZEYsE1ZneN","email":"kwatkins7@bloglovin.com","id_commune":"22","nom":"Kelly"}
{"prenom":"Burns","id_user":"9","mot_de_pass":"o4r93WOF","email":"aburns8@dell.com","id_commune":"57","nom":"Albert"}
{"prenom":"Mills","id_user":"10","mot_de_pass":"RGiFBnUR9","email":"amills9@gov.uk","id_commune":"65","nom":"Ashley"}
{"prenom":"Gomez","id_user":"11","mot_de_pass":"IvUk7AMwg","email":"mgomeza@drupal.org","id_commune":"56","nom":"Michael"}
{"prenom":"Morrison","id_user":"12","mot_de_pass":"acfgknufe","email":"smorrisonb@constantcontact.com","id_commune":"93","nom":"Steve"}
{"prenom":"Porter","id_user":"13","mot_de_pass":"xxPELl","email":"fporterc@blog.com","id_commune":"65","nom":"Frances"}
{"prenom":"Fernandez","id_user":"14","mot_de_pass":"scHguwrC0U","email":"rfernandezd@google.it","id_commune":"44","nom":"Raymond"}
{"prenom":"Peters","id_user":"15","mot_de_pass":"4eNnoxVZLGkf","email":"dpeterse@webnode.com","id_commune":"5","nom":"Dorothy"}
{"prenom":"Mills","id_user":"16","mot_de_pass":"Ua0qFEGJhU","email":"emillsf@edublogs.org","id_commune":"52","nom":"Edward"}
{"prenom":"Gonzales","id_user":"17","mot_de_pass":"od8ajYuep5","email":"lgonzalesg@nytimes.com","id_commune":"24","nom":"Laura"}
{"prenom":"Johnson","id_user":"18","mot_de_pass":"wHZq70yFitlR","email":"mjohnsonh@examiner.com","id_commune":"20","nom":"Martin"}
{"prenom":"Ford","id_user":"19","mot_de_pass":"coLqvSRQPW","email":"lfordi@washingtonpost.com","id_commune":"15","nom":"Louis"}
{"prenom":"Hughes","id_user":"20","mot_de_pass":"HmgLpx","email":"lhughesj@narod.ru","id_commune":"68","nom":"Lawrence"}
{"prenom":"Vasquez","id_user":"21","mot_de_pass":"dAyaZB32h6o","email":"evasquezk@cnn.com","id_commune":"80","nom":"Elizabeth"}
{"prenom":"Nguyen","id_user":"22","mot_de_pass":"1d8eag","email":"wnguyenl@ehow.com","id_commune":"70","nom":"Willie"}
{"prenom":"Ortiz","id_user":"23","mot_de_pass":"0IRqoKcD","email":"jortizm@google.fr","id_commune":"2","nom":"Jesse"}
{"prenom":"Willis","id_user":"24","mot_de_pass":"1mLLKBiV5QA5","email":"swillisn@a8.net","id_commune":"53","nom":"Shawn"}
{"prenom":"Mendoza","id_user":"25","mot_de_pass":"U0HATM","email":"amendozao@typepad.com","id_commune":"11","nom":"Alice"}
{"prenom":"Harris","id_user":"26","mot_de_pass":"qjRbe4UXCgmp","email":"gharrisp@imgur.com","id_commune":"4","nom":"George"}
{"prenom":"Kelley","id_user":"27","mot_de_pass":"Y6Gqtm","email":"kkelleyq@businesswire.com","id_commune":"61","nom":"Kimberly"}
{"prenom":"Wheeler","id_user":"28","mot_de_pass":"5qGk8uu6","email":"fwheelerr@geocities.jp","id_commune":"35","nom":"Frances"}
{"prenom":"Stone","id_user":"29","mot_de_pass":"XmdZLITKy9","email":"astones@vk.com","id_commune":"17","nom":"Antonio"}
{"prenom":"Ortiz","id_user":"30","mot_de_pass":"isYim8","email":"eortizt@topsy.com","id_commune":"100","nom":"Edward"}
```


5. 3iéme PHASE : Installation MongoDB et l'importation des fichiers JSON

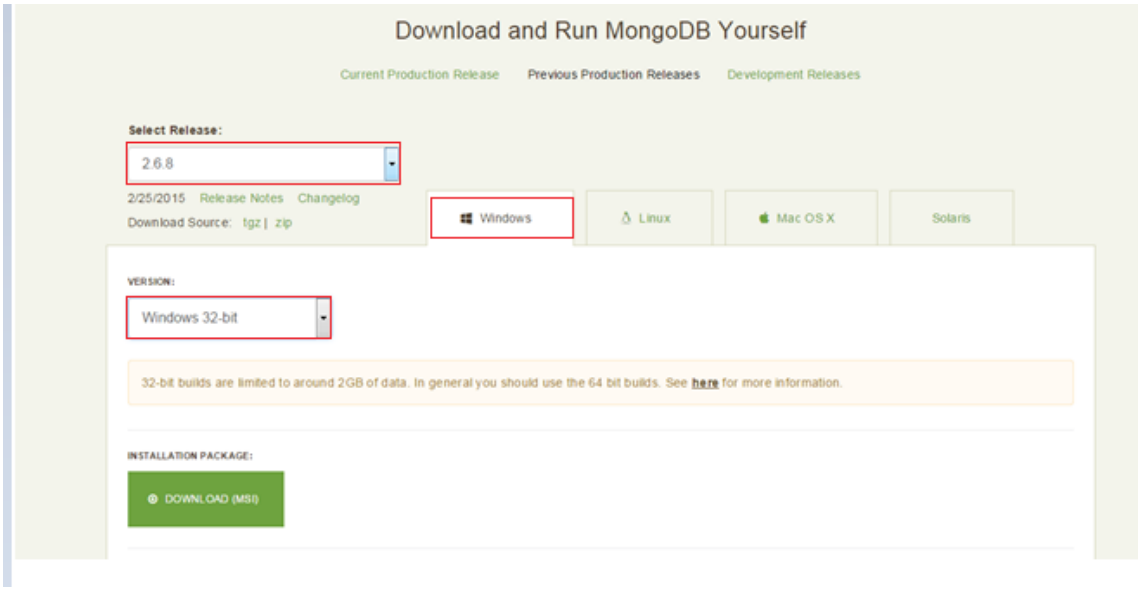
Dans cette partie, on va effectuer l'installation de MongoDB, configurer le ReplicaSet pour que le système n'ait pas un point de défaillance unique, ensuite nous allons procéder à la migration de la base de données.

5.1 Configuration l'environnement de travail

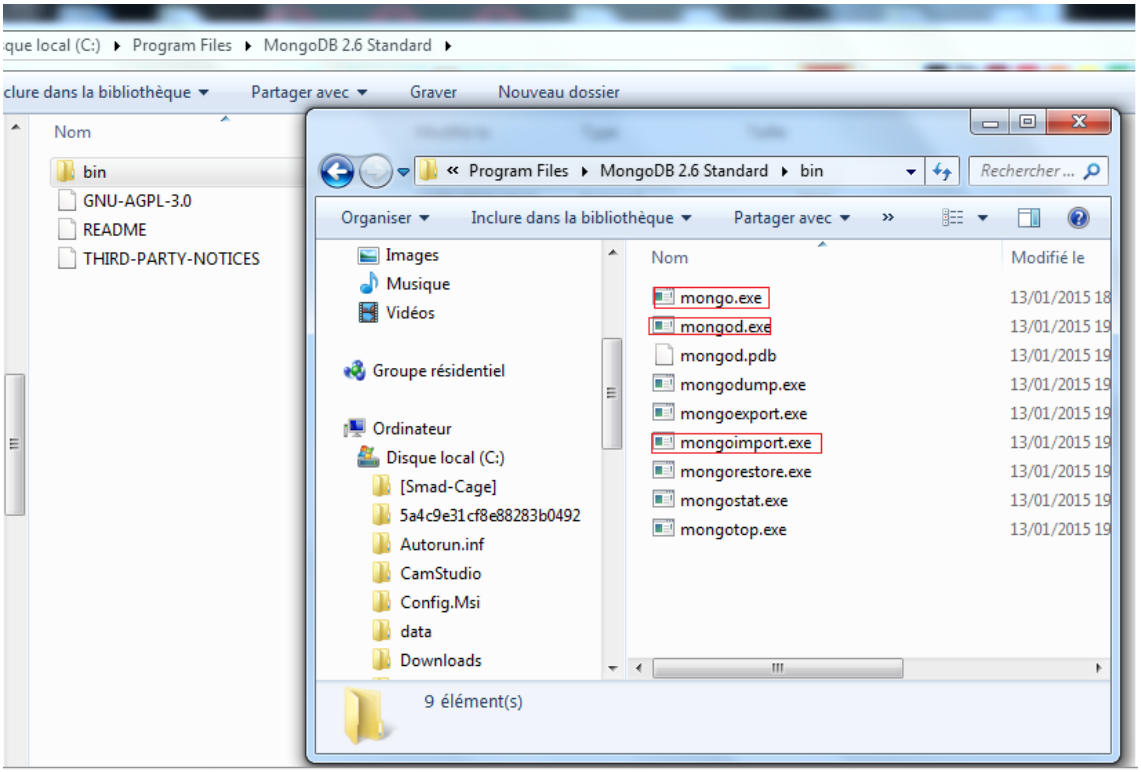
Il faut commencer par télécharger MongoDB disponibles à l'adresse suivante : http://www.mongodb.org .



La version Windows 32 bits va être été sélectionnée.



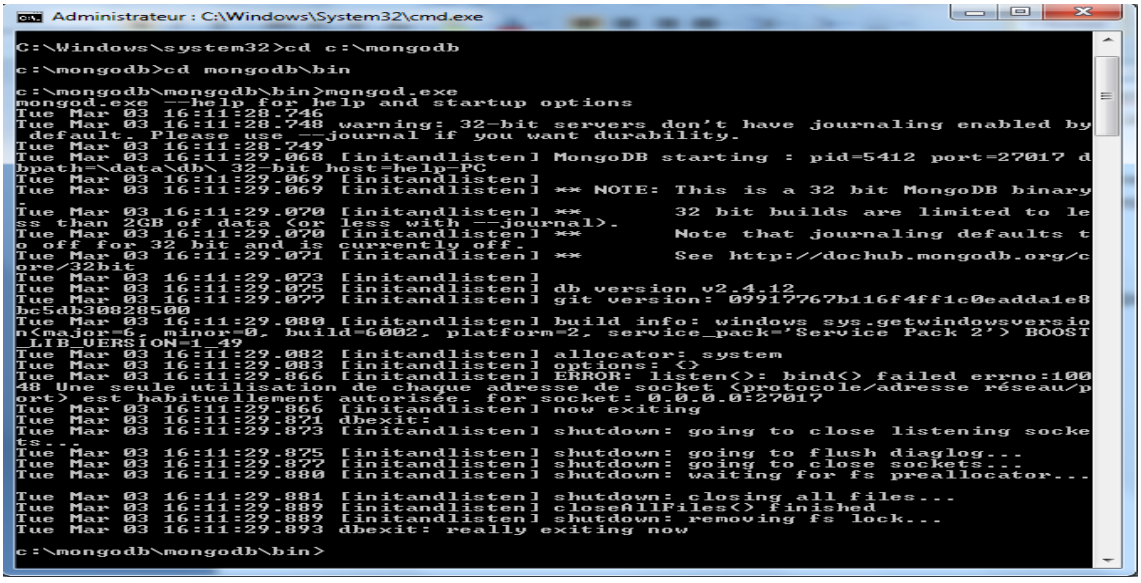
Une fois le téléchargement est terminé il est recommandé de dézipper l'archive et de placer le contenu dans un répertoire nommé **mongodb** à la racine du disque dur : **« c:\mongodb»**.
Pour stocker les informations, MongoDB utilise le répertoire par default c:\monogdb\data\db. Il faut donc créer ce répertoire à partir de l'explorateur Windows.



- Le Mongod est responsable de créer et gérer la base de données MongoDB ;
- Mongo est l'interpréteur de la ligne de commande MongoDB, utilisé pour manipuler la base de données MongoDB et ses objets ;
- Mongodump est utilisé pour sauvegarder toutes les bases ainsi que les collections au format BSON et JSON ;
- Mongoexport Permet d'exporter la BDD, dans les fichiers JSON associé à chaque table ;
- Mongorestore permet de restaurer l'ensemble des fichiers sauvegardés avec Mongodump ;
- Mongoimport permet d'importer des données d'une collection dans une base de données existante ou non existante.

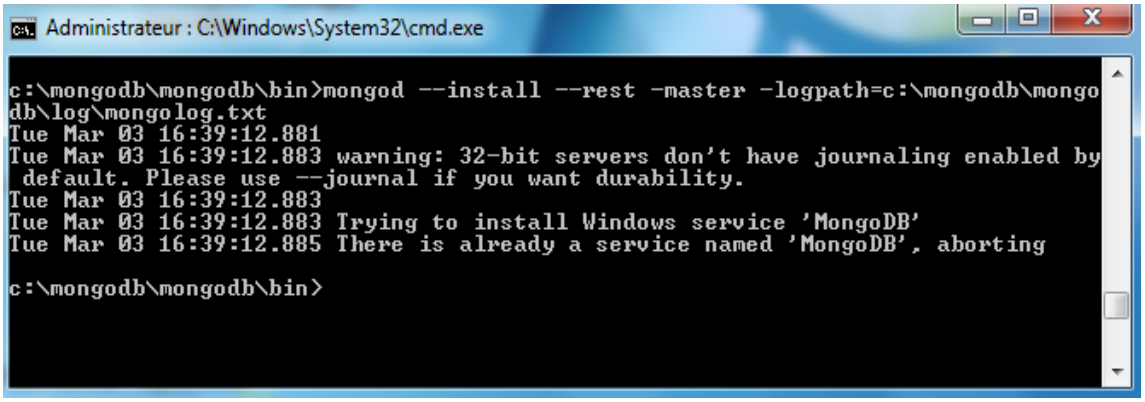
5.1.1 Vérification de l'environnement

Pour faire les vérifications, on lance **mongod.exe** à partir du répertoire **c:\mongodb\mongodb\bin**.



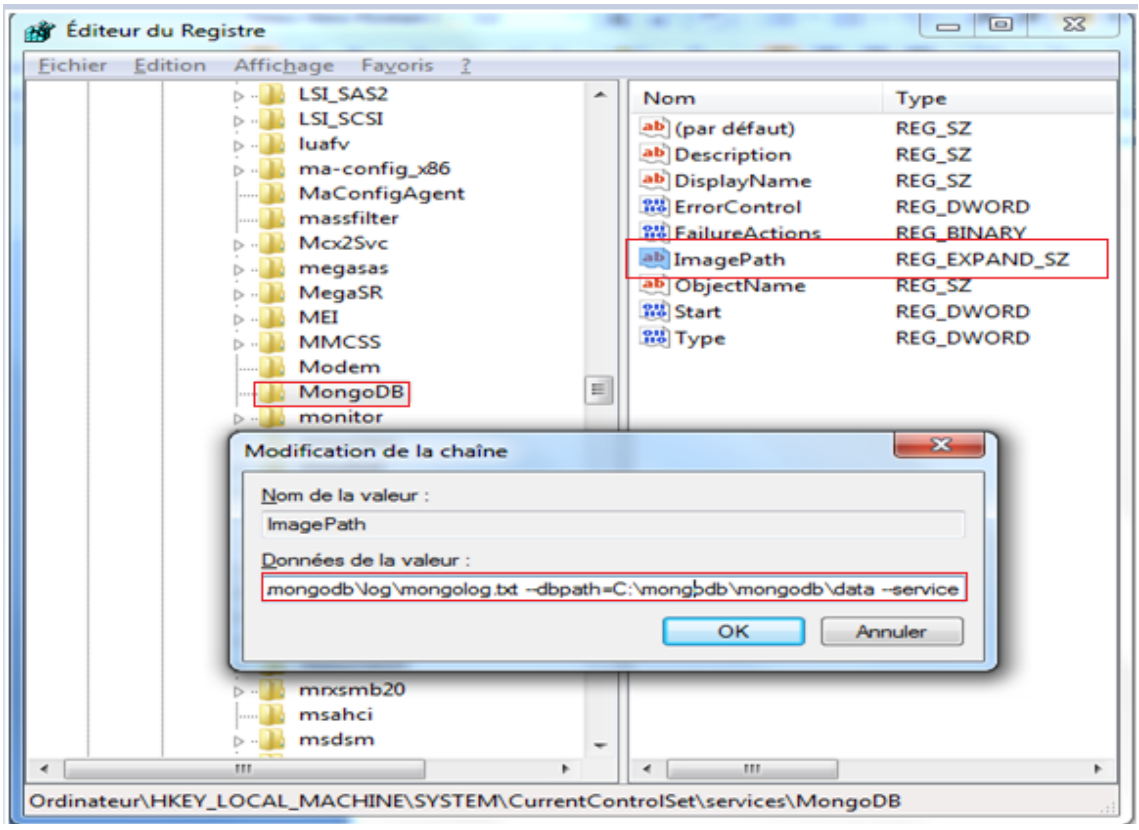
5.1.2 Installation de service MongoDB

1. Il faut créer un fichier log dans le répertoire :
   c:\mongodb\mongodb\log\mongolog.txt
2. On va exécuter la commande suivante : `mongod --install --rest -master -logpath=C:\mongodb\mongodb\log\mongolog.txt`

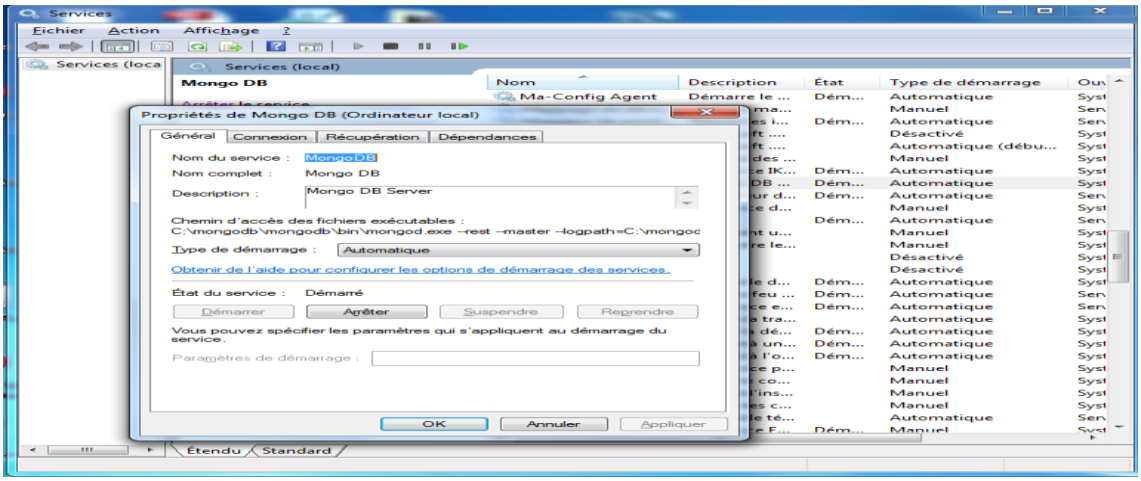


Ce message est obtenu parce que on déjà installer le service avec cette commande

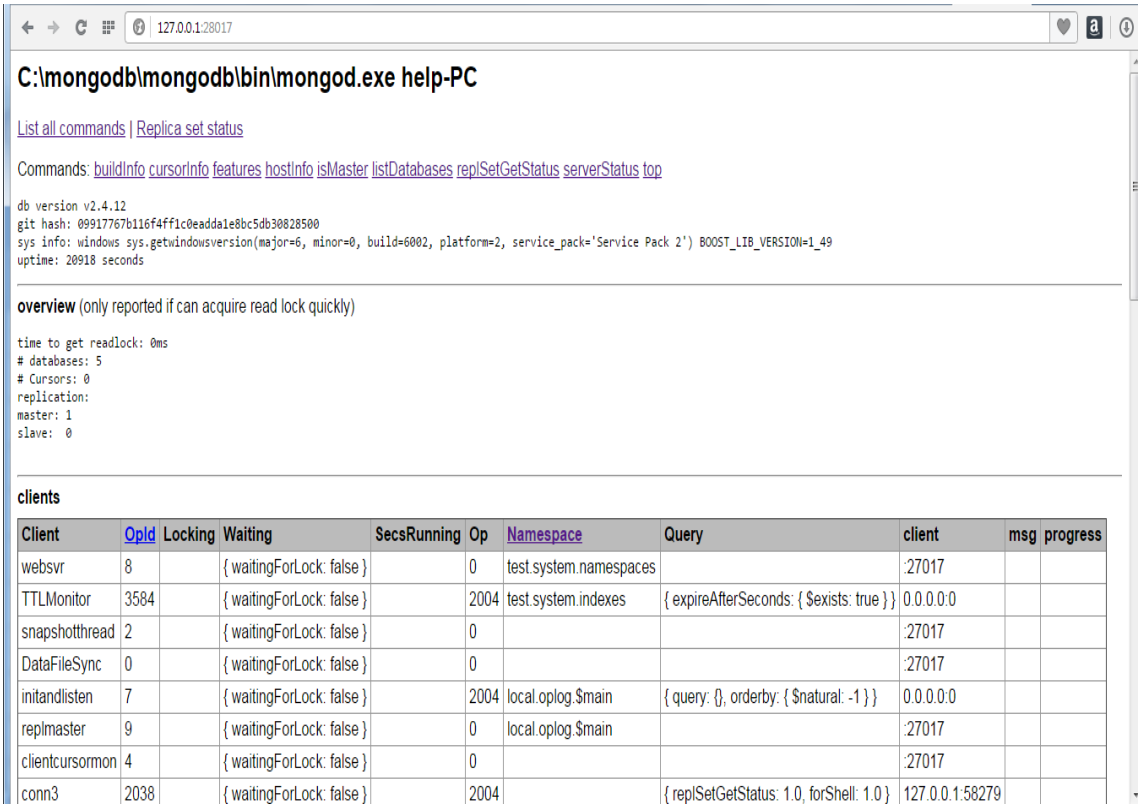
3. Lancer l'éditeur de registres
4. Chercher HKEY_LOCAL_MACHINE >> SYSTEM >> CurrentControlSet >> services>>MongoDB
5. Cliquer sur ImagePath ,changer sa valeur avec cette valeur :
   C:\mongodb\mongodb\bin\mongod.exe --rest --master -logpath=C:\mongodb\mongodb\log\mongolog.txt --dbpath=C:\mongodb\mongodb\data –service



6. Lancer maintenant le service MongoDB

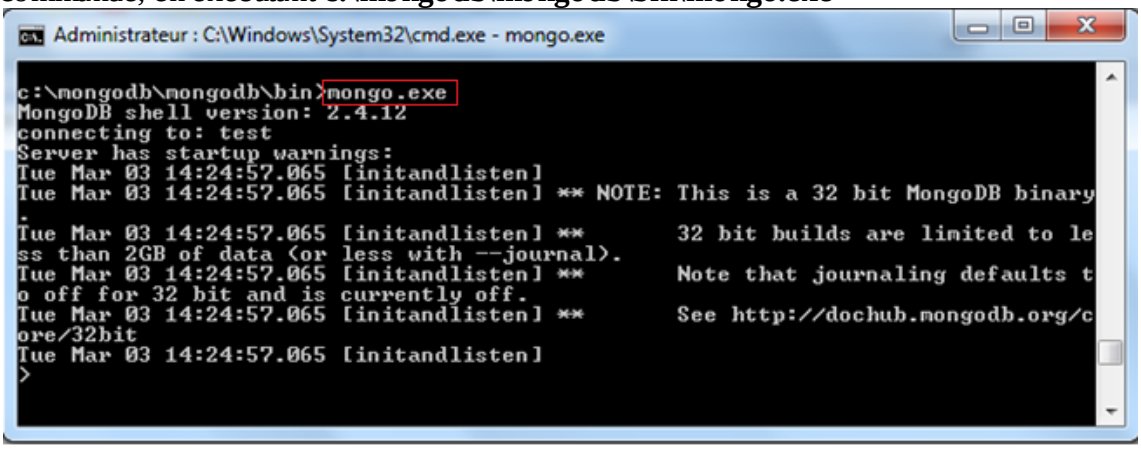


Le serveur MongoDB doit être démarré, le port utilisé par default est 27017 pour le serveur et 28017 pour la console d'administration de MongoDB.
À l'aide d'un navigateur internet, on peut se connecter sur la console d'administration à l'adresse : http://localhost:28017 .Cet écran illustre la console d'administration



5.1.3 Démarrage d'un client MongoDB

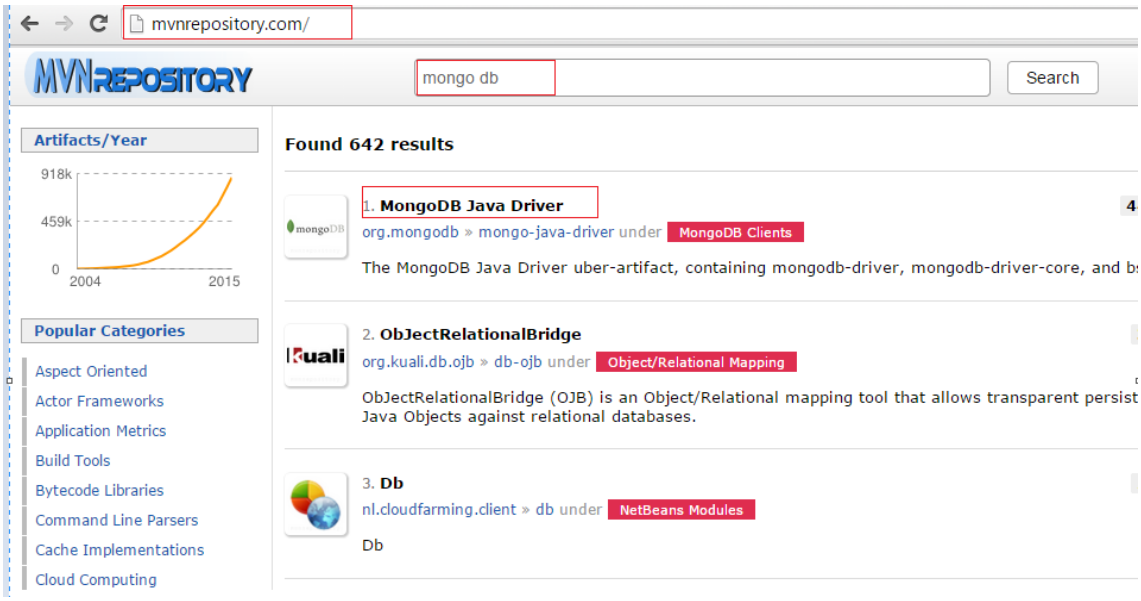
Il est possible de réaliser des manipulations sur une base MongoDB avec la ligne de commande, en exécutant **c:\mongodb\mongodb\bin\mongo.exe**



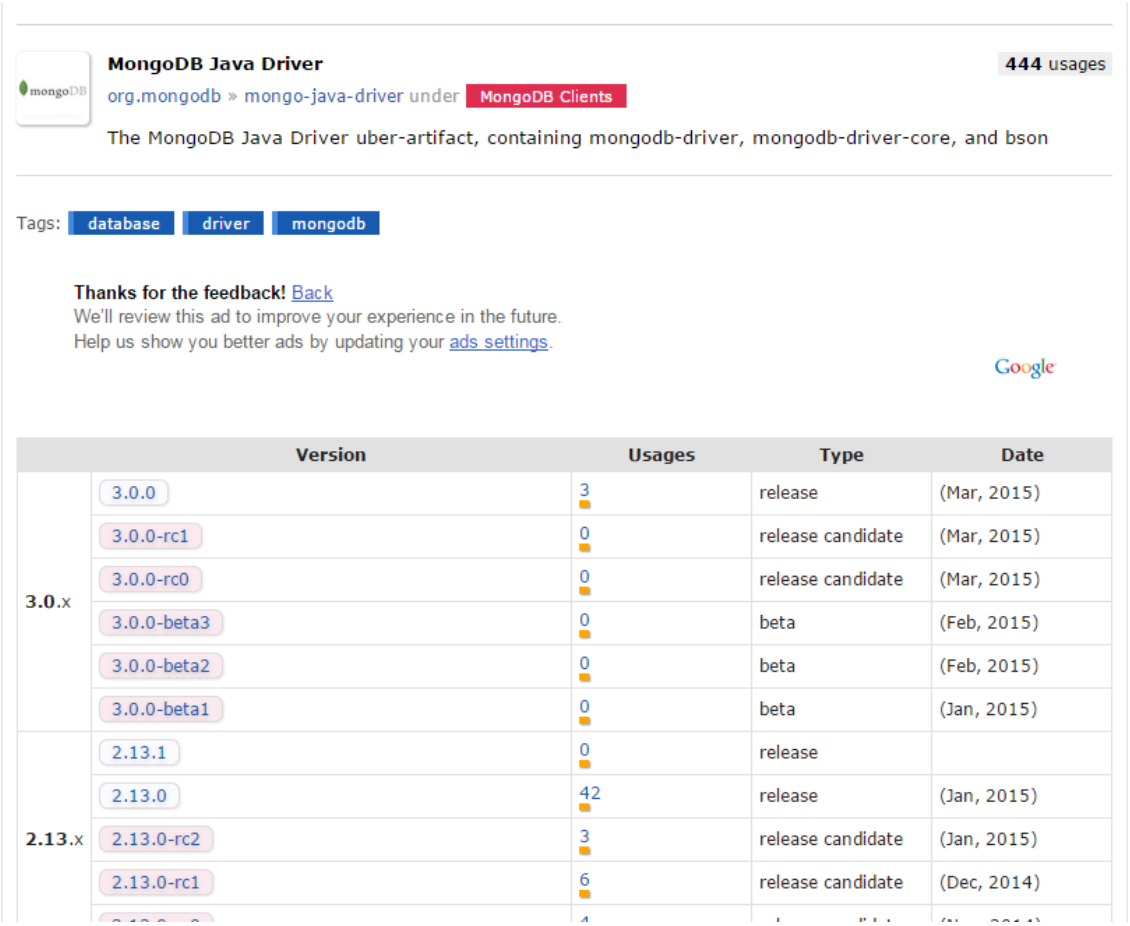
L'accès à la base de données se fait par la ligne de commande et la connexion est faite par default sur la base test.

5.2 La connexion avec JAVA

En premier temps nous installons les drivers pour MongoDB. Le dépôt Maven contient de nombreux drivers dont un parmi eux, pour utiliser MongoDB à partir de JAVA. Le dépôt central est disponible à l'adresse suivante : http://mvnrepository.com/, il faut chercher le driver mongo db comme indique la figure :

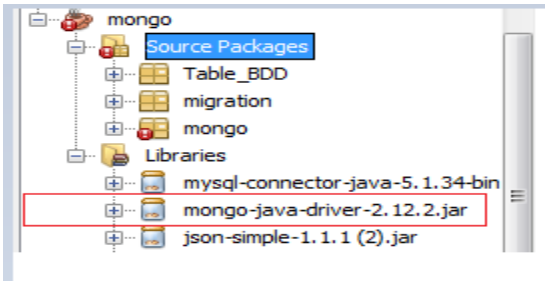


La figure montre les différentes versions du driver sur le repository Maven, on choisit de télécharger la version 2.11.3 en Binary.jar correspondant au driver mongo pour java



	Version	Usages	Type	Date
3.0.x	3.0.0	3	release	(Mar, 2015)
	3.0.0-rc1	0	release candidate	(Mar, 2015)
	3.0.0-rc0	0	release candidate	(Mar, 2015)
	3.0.0-beta3	0	beta	(Feb, 2015)
	3.0.0-beta2	0	beta	(Feb, 2015)
	3.0.0-beta1	0	beta	(Jan, 2015)
2.13.x	2.13.1	0	release	
	2.13.0	42	release	(Jan, 2015)
	2.13.0-rc2	3	release candidate	(Jan, 2015)
	2.13.0-rc1	6	release candidate	(Dec, 2014)

Après le téléchargement du pilote, on crée notre projet sur le NetBeans ensuite faire un clic droit sur **Librairies** et choisir **Add JAR/Floder** pour ajouter au JAVA le driver MongoDB téléchargé : **mongo-java-driver-2.11.3.jar**



Ce script est utilisé pour connecter à MongoDB par JAVA

```
MongoClient mongoClient =new MongoClient();
mongoClient = new MongoClient( "localhost" , 27016 );
System.out.println("-----------la lste de toute BDD--------------");
DefaultListModel<String> model=new DefaultListModel<>();
 DefaultListModel<String> model1=new DefaultListModel<>();
List <String> liste_de_nom =mongoClient.getDatabaseNames();
```

5.3 Configuration de la réplication --ReplicaSet

La réplication des bases de données permet la redondance et la sauvegarde des données d'une base. Pour MongoDB, elle consiste à la mise en place d'un cluster d'instances de serveur de base de données (Mongod).

Une organisation est nécessaire à la bonne réplication des données au sein d'un Replica Set ainsi chaque Replica Set possèdera une et une seule instance de Mongod désignée comme primaire, les autres instances étant alors secondaires.

Dans cette configuration, seule l'instance primaire va recevoir les requêtes d'écriture (insert et update) pour les répliquer ensuite sur les autres instances du Replica Set.

Pour notre cas, nous créons un Replica Set, qui possède un nœud primaire et deux instances secondaires.

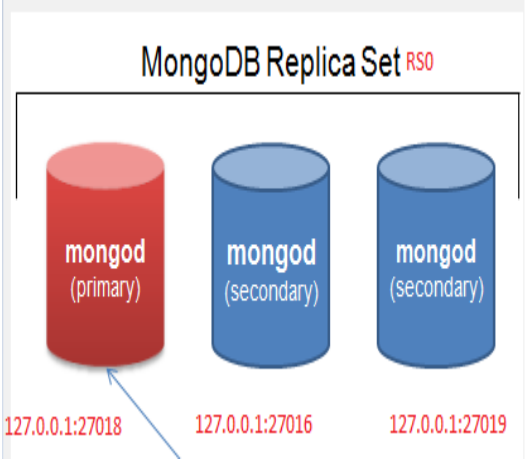


Figure 4-3 Replica-Set

Passons à la configuration du notre Replica Set : nous créons en premier temps un dossier rs0-0 dans le répertoire c:\data qui va contenir la liste des bases de données,

nous lançons le processus mongod en indiquant le numéro de port 27016, le nom de replica set rs0, le chemin de la base de données c:\data\rs0-0 .

- Nœud primaire :

```
C:\mongo\bin>mongod.exe --port 27016 --replSet rs0 --dbpathc:\data\rs0-0 --small
files --oplogsize 128
```

- Premier nœud secondaire :

```
C:\mongo\bin>mongod.exe --port 27018 --replSet rs0 --dbpathc:\data\rs0-1 --small
files --oplogsize 128
```

- Deuxième nœud secondaire :

```
C:\mongo\bin>mongo --port 27016
```

- Ensuite on va se connecter sur la machine maitre

- L'étape suivante c'est l'initiation de la réplication

```
> rs.initiate()
{
        "info2" : "no configuration explicitly specified -- making one",
        "me" : "help-PC:27016",
        "info" : "Config now saved locally.  Should come online in about a minut
e.",
        "ok" : 1
}
```

- Création du fichier de configuration

```
> rsconf = {
... _id: "rs0",
... members: [
... {
... _id: 0,
... host: "localhost:27016"
... }
... ]
... }
{
        "_id" : "rs0",
        "members" : [
                {
                        "_id" : 0,
                        "host" : "localhost:27016"
                }
        ]
```

Nous utilisons la commande rs.conf() pour visualiser le fichier de configuration de la réplication

```
rs0:PRIMARY> rs.conf()
```

```
{
        "_id" : "rs0",
        "version" : 1,
        "members" : [
                {
                        "_id" : 0,
                        "host" : "help-PC:27016"
                }
        ]
}
```

- Après la configuration nous pouvons maintenant ajouter les nœuds secondaires

```
rs0:PRIMARY> rs.add("help-PC:27018")
rs0:PRIMARY> rs.add("help-PC:27019")
```

- Visualisation du statut de la réplication

Une attente de l'ordre de quelques secondes est nécessaire pour que l'élection se fasse. Nous voyons que help-PC : 27018 et help-PC : 27019 se synchronisent sur help-PC : 27016, car c'est le primaire.

Nous allons maintenant se connecter sur les nœuds secondaires :

```
C:\mongo\bin>mongo --port 27018
```

```
rs0:SECONDARY>
```

```
C:\mongo\bin>mongo --port 27019
MongoDB shell version: 2.6.8
connecting to: 127.0.0.1:27019/test
Server has startup warnings:
2015-04-21T12:36:37.785+0200 [initandlisten]
2015-04-21T12:36:37.786+0200 [initandlisten] ** NOTE: This is a 32 bit MongoDB b
inary.
2015-04-21T12:36:37.786+0200 [initandlisten] **       32 bit builds are limited
to less than 2GB of data (or less with --journal).
2015-04-21T12:36:37.787+0200 [initandlisten] **       Note that journaling defau
lts to off for 32 bit and is currently off.
2015-04-21T12:36:37.787+0200 [initandlisten] **       See http://dochub.mongodb.
org/core/32bit
2015-04-21T12:36:37.788+0200 [initandlisten]
rs0:SECONDARY>
```

Si un problème ou une panne surgit au niveau du nœud primaire, il aura une sélection d'un nouveau nœud primaire parmi les nœuds secondaires :

5.4 Configuration de repartitionnement –sharding

Dans cette partie nous allons implémenter le Sharding MongoDB (Distribution) conformément au schéma de la figure :

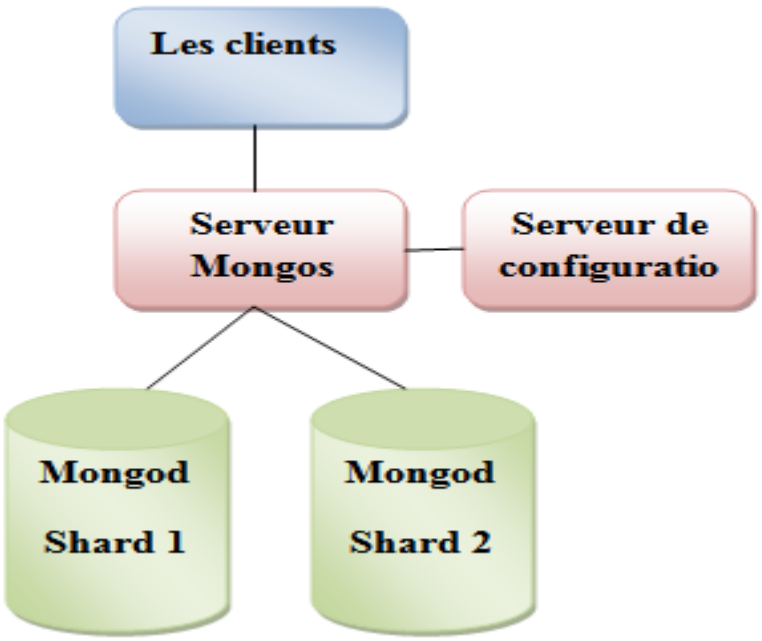


Figure 4-4 Principe sharding

La configuration requis quatre serveurs :

- Serveur Mongos qui gère la répartition et distribue les données dans 2 serveurs Shard (Mongod)
- Un serveur de configuration utilisé par Mongos pour gérer la configuration des différents Shards,
- Deux serveurs Shards, chacun exécute le service Mongod et contient des partitions des données distribuées par Mongos.

Pour configurer le sharding, il faut suivre la démarche suivante :

a. **Démarrage des instances des serveurs de configuration**

Les processus des serveurs de configuration, sont des instances Mongod, qui stockent les méta-informations du cluster. Nous désignons un Mongod en tant que serveur de configuration en utilisant l'option –configsrv.
Nous créons le répertoire de données pour l'instance de serveur de configuration, le serveur stocke ses données dans le répertoire "data\configdb"

```
C:\mongo\bin>mkdir c:\data\configdb
```

Maintenant nous démarrons les instances avec la commande suivante :

```
C:\mongo\bin>mongod --configsvr --port 27010
```

b. Démarrage des instances Mongos

Les instances mongos sont peu gourmandes en ressources et ne nécessitent pas de dossier de données. Vous pouvez exécuter une instance mongos en parallèle à d'autres composants du cluster sur un même système.Nous démarrons l'instance Mongos par la commande suivante :

```
C:\mongo\bin>mongos --configdb localhost:27010 --port 27011
```

c. L'ajout des Shards au cluster

Un Shard peut être un Mongod en mode standalone ou alors un replica Set mais dans notre cas nous utilisons le mode standalone.
Avant d'ajouter les Shard nous lançons les instances Mongod en indiquant le numéro de port, le répertoire qui stocke les métadonnées.

```
C:\mongo\bin>mongod --port 27012 --dbpath c:\data\shard
```

```
C:\mongo\bin>mongod --port 27013 --dbpath c:\data\shard1
```

Nous communiquons avec le cluster par l'intermédiaire d'une instance mongos

```
C:\mongo\bin>mongo --port 27011 --host localhost
MongoDB shell version: 2.6.8
connecting to: localhost:27011/test
mongos>
```

Maintenant nous pouvons ajouter les Shard à l'aide de la commande sh.addShard() avec des paramètres adresse IP et le numéro de port

```
mongos> sh.addShard("localhost:27012")
{ "shardAdded" : "shard0000", "ok" : 1 }
mongos> sh.addShard("localhost:27013")
{ "shardAdded" : "shard0001", "ok" : 1 }
mongos>
```

d. Activer le sharding pour une base de données et la collection

Avant que nous puissions sharder une collection, nous devons activer le Shard pour la base de données reseau_social contenant la collection User ensuite et pour la collection User nous avons activé le sharding en utilisant la commande sh.shardCollection() en indiquant la clé de Shard id_user :1.

```
mongos> sh.enableSharding("reseau_social")
{ "ok" : 1 }
mongos> sh.shardCollection("reseau_social.User", {id_user":1})
2015-05-04T18:31:33.886+0200 SyntaxError: Unexpected token ILLEGAL
mongos> sh.shardCollection("reseau_social.User", {"id_user":1})
{ "collectionsharded" : "reseau_social.User", "ok" : 1 }
mongos>
```

Nous utilisons maintenant la commande sh.Status () pour voir le statut de sharding, la figure nous indique qu'on a deux Shard, la base de données reseau_social est partitionnée ainsi que la clé de sharding pour la collection User c'est id_user :1

```
mongos> sh.status()
--- Sharding Status ---
  sharding version: {
        "_id" : 1,
        "version" : 4,
        "minCompatibleVersion" : 4,
        "currentVersion" : 5,
        "clusterId" : ObjectId("55479bc185c7bc5deedf5f06")
}
  shards:
        {  "_id" : "shard0000",  "host" : "localhost:27012" }
        {  "_id" : "shard0001",  "host" : "localhost:27013" }
  databases:
        {  "_id" : "admin",  "partitioned" : false,  "primary" : "config" }
        {  "_id" : "test",  "partitioned" : false,  "primary" : "shard0000" }
        {  "_id" : "reseau_social",  "partitioned" : true,  "primary" : "shard00
00" }
                reseau_social.User
                        shard key: { "id_user" : 1 }
                        chunks:
                                shard0000       1
                        { "id_user" : { "$minKey" : 1 } } -->> { "id_user" : { "
$maxKey" : 1 } } on : shard0000 Timestamp(1, 0)
```

6. Migration des données MySQL vers MongoDB

Pour réaliser cette migration, il faut suivre les trois étapes suivantes :

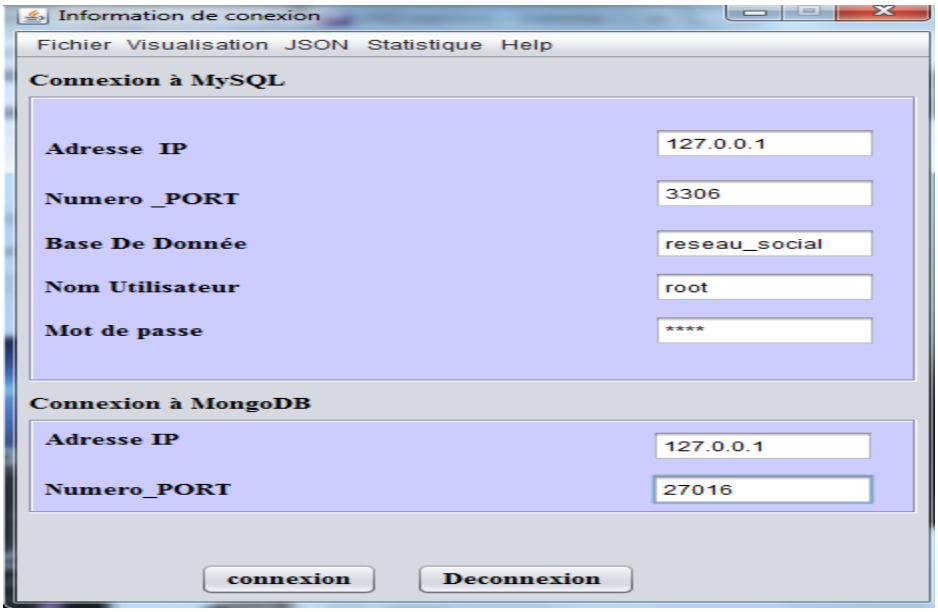
- Génération des fichiers Json
- Import par l'outil Mongoimpor
- Conversion en BSON.

Figure 4-5 les étapes de migration

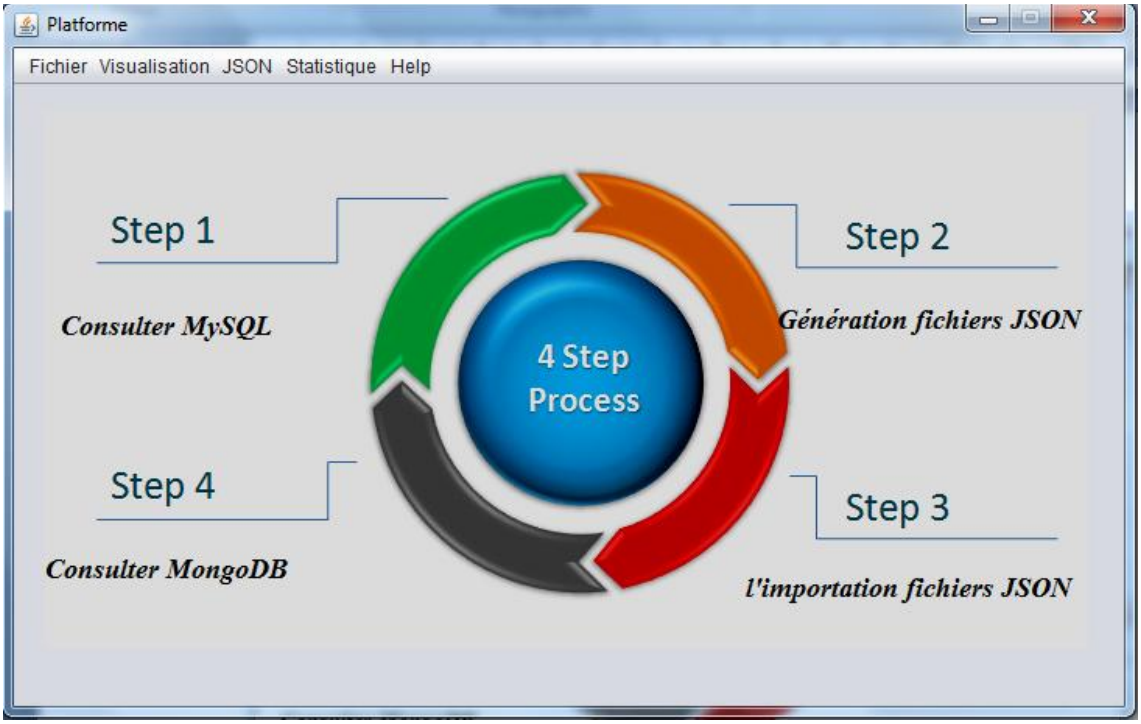
6.1 Génération des fichiers JSON

Pour la génération des fichiers JSON, nous allons développer une application JAVA sous NetBeans, cette application permet d'extraire les données d'une table MySQL ou plusieurs tables et de générer automatiquement les données en format JSON vers des fichiers spécifiques.

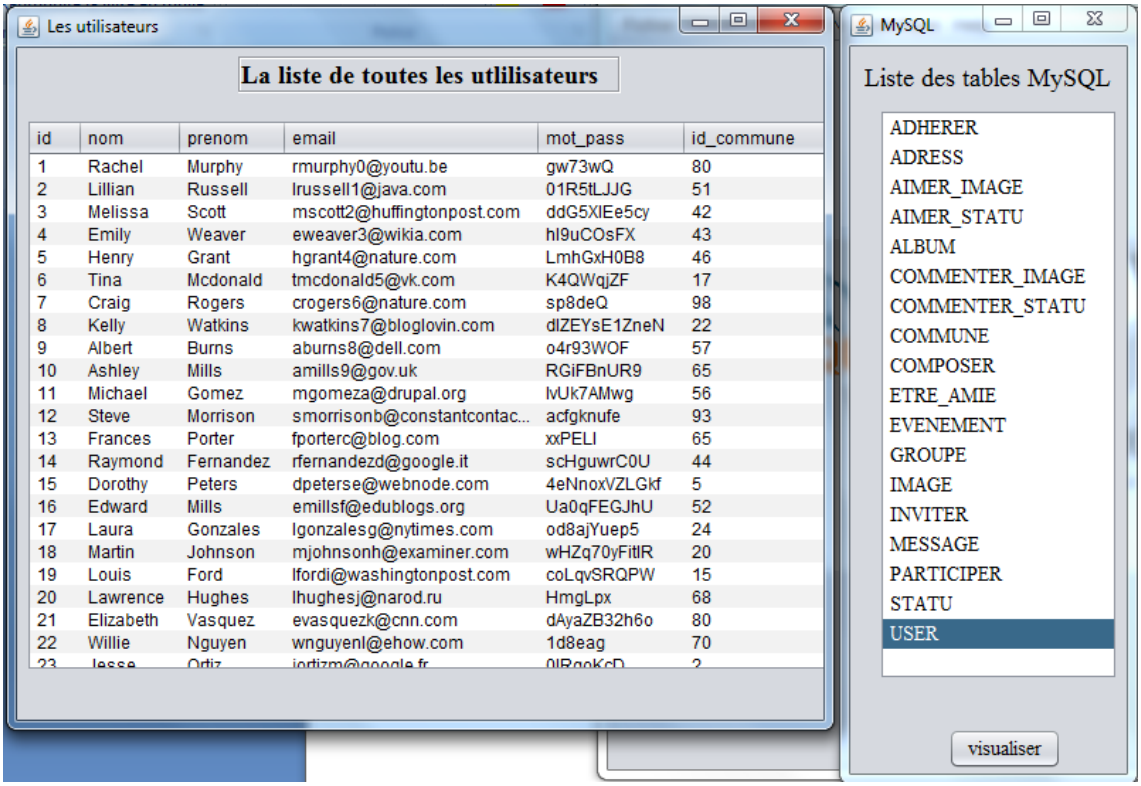
Le bouton d'accueil donne l'accès à la fenêtre suivante dans laquelle on va renseigner les paramètres de connexion à la base MySQL et MongoDB .Une fois connecté la



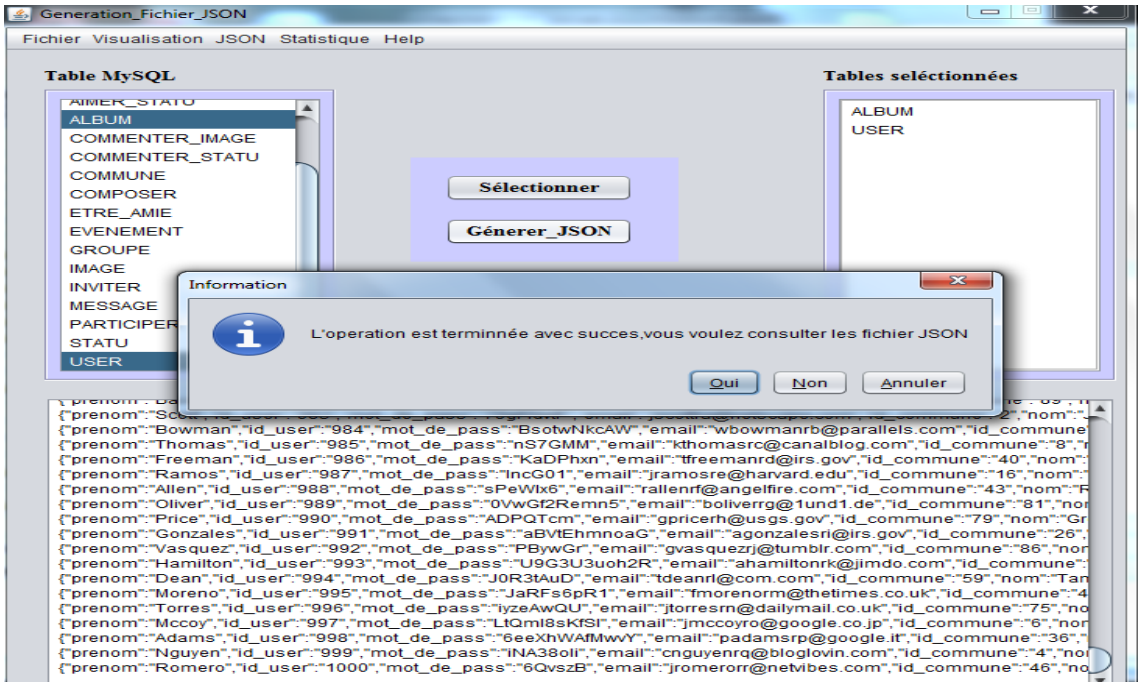
démarche est illustrée dans l'écran suivant :



Bouton Consulter MySQL permet d'afficher la liste des tables de la base de données reseau_social. Pour visualiser le contenu d'une seule table, il faut juste la sélectionner.

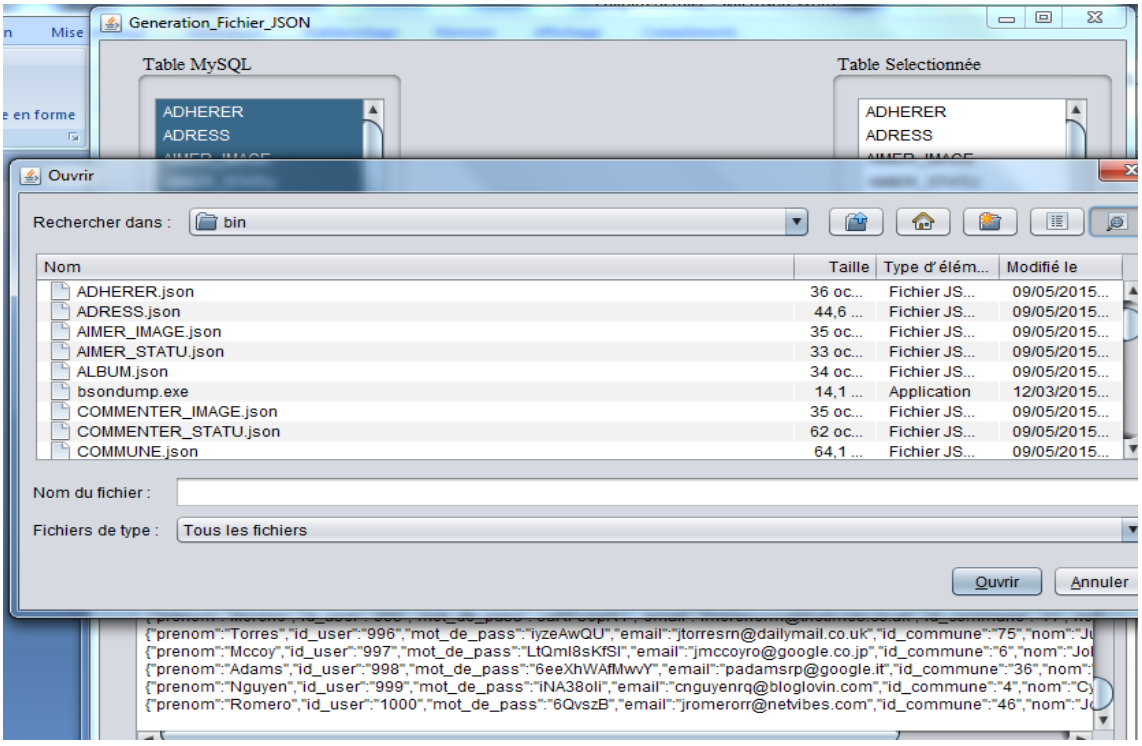


id	nom	prenom	email	mot_pass	id_commune
1	Rachel	Murphy	rmurphy0@youtu.be	gw73wQ	80
2	Lillian	Russell	lrussell1@java.com	01R5tLJJG	51
3	Melissa	Scott	mscott2@huffingtonpost.com	ddG5XlEe5cy	42
4	Emily	Weaver	eweaver3@wikia.com	hl9uCOsFX	43
5	Henry	Grant	hgrant4@nature.com	LmhGxH0B8	46
6	Tina	Mcdonald	tmcdonald5@vk.com	K4QWqjZF	17
7	Craig	Rogers	crogers6@nature.com	sp8deQ	98
8	Kelly	Watkins	kwatkins7@bloglovin.com	dlZEYsE1ZneN	22
9	Albert	Burns	aburns8@dell.com	o4r93WOF	57
10	Ashley	Mills	amills9@gov.uk	RGiFBnUR9	65
11	Michael	Gomez	mgomeza@drupal.org	lvUk7AMwg	56
12	Steve	Morrison	smorrisonb@constantcontac...	acfgknufe	93
13	Frances	Porter	fporterc@blog.com	xxPELI	65
14	Raymond	Fernandez	rfernandezd@google.it	scHguwrC0U	44
15	Dorothy	Peters	dpeterse@webnode.com	4eNnoxVZLGkf	5
16	Edward	Mills	emillsf@edublogs.org	Ua0qFEGJhU	52
17	Laura	Gonzales	lgonzalesg@nytimes.com	od8ajYuep5	24
18	Martin	Johnson	mjohnsonh@examiner.com	wHZq70yFitlR	20
19	Louis	Ford	lfordi@washingtonpost.com	coLqvSRQPW	15
20	Lawrence	Hughes	lhughesj@narod.ru	HmgLpx	68
21	Elizabeth	Vasquez	evasquezk@cnn.com	dAyaZB32h6o	80
22	Willie	Nguyen	wnguyenl@ehow.com	1d8eag	70
23	Jesse	Ortiz	jortizm@google.fr	0lRqoKcD	2



Passons à la génération des fichiers JSON qui se fait en cliquant sur génération fichier JSON ou bien choisir le menu (JSON/Génerer_fichier_JSON), afficher la liste des tables MySQL, choisir un ou plusieurs tables et cliquer sur générer Json.

Après la fin de la génération des fichiers, on peut visualiser le contenu du répertoire :



La troisième étape consiste à importer les fichiers JSON vers MongoDB.

Avant de faire l'importation ou bien la migration des données nous connectons au replica-Set pour vérifier les bases de données existantes dans le replica-Set rs0 alors nous nous connectons aux instances Mongod puis aux instances mongo.

```
C:\mongo\bin>mongod.exe --port 27016 --replSet rs0 --dbpathc:\data\rs0-0 --small
files --oplogsize 128
```

```
C:\mongo\bin>mongod.exe --port 27018 --replSet rs0 --dbpathc:\data\rs0-1 --small
files --oplogsize 128
```

```
C:\mongo\bin>mongod.exe --port 27019 --replSet rs0 --dbpathc:\data\rs0-2 --small
files --oplogsize 128
```

```
C:\mongo\bin>mongo --port 27016
MongoDB shell version: 2.6.8
connecting to: 127.0.0.1:27016/test
Server has startup warnings:
2015-05-10T12:01:47.140+0200 [initandlisten]
2015-05-10T12:01:47.141+0200 [initandlisten] ** NOTE: This is a 32 bit MongoDB b
inary.
2015-05-10T12:01:47.141+0200 [initandlisten] **       32 bit builds are limited
to less than 2GB of data (or less with --journal).
2015-05-10T12:01:47.142+0200 [initandlisten] **       Note that journaling defau
lts to off for 32 bit and is currently off.
2015-05-10T12:01:47.142+0200 [initandlisten] **       See http://dochub.mongodb.
org/core/32bit
2015-05-10T12:01:47.143+0200 [initandlisten]
rs0:PRIMARY> show dbs
admin  (empty)
local  0.078GB
rs0:PRIMARY>
```

```
C:\mongo\bin>mongo --port 27018
MongoDB shell version: 2.6.8
connecting to: 127.0.0.1:27018/test
Server has startup warnings:
2015-05-10T12:03:34.196+0200 [initandlisten]
2015-05-10T12:03:34.196+0200 [initandlisten] ** NOTE: This is a 32 bit MongoDB b
inary.
2015-05-10T12:03:34.196+0200 [initandlisten] **       32 bit builds are limited
to less than 2GB of data (or less with --journal).
2015-05-10T12:03:34.196+0200 [initandlisten] **       Note that journaling defau
lts to off for 32 bit and is currently off.
2015-05-10T12:03:34.197+0200 [initandlisten] **       See http://dochub.mongodb.
org/core/32bit
2015-05-10T12:03:34.197+0200 [initandlisten]
rs0:SECONDARY> show dbs
admin  (empty)
local  0.078GB
rs0:SECONDARY>
```

```
C:\mongo\bin>mongo --port 27019
MongoDB shell version: 2.6.8
connecting to: 127.0.0.1:27019/test
Server has startup warnings:
2015-05-10T12:04:17.800+0200 [initandlisten]
2015-05-10T12:04:17.800+0200 [initandlisten] ** NOTE: This is a 32 bit MongoDB b
inary.
2015-05-10T12:04:17.800+0200 [initandlisten] **       32 bit builds are limited
to less than 2GB of data (or less with --journal).
2015-05-10T12:04:17.801+0200 [initandlisten] **       Note that journaling defau
lts to off for 32 bit and is currently off.
2015-05-10T12:04:17.801+0200 [initandlisten] **       See http://dochub.mongodb.
org/core/32bit
2015-05-10T12:04:17.802+0200 [initandlisten]
rs0:SECONDARY> show dbs
admin  (empty)
local  0.078GB
rs0:SECONDARY>
```

Nous remarquons que dans le replica-Set rs0, le nœud primaire contient juste deux bases de données qui sont par default admin et local qui sont répliquées sur les deux nœuds secondaires.

Nous passons maintenant à la phase de migration des données nous utilisons Mongoimport qui permet d'importer les données vers MongoDB, donc nous importons

les données de MySQL au nœud primaire de replica-Set rs0 en indiquant le numéro de port ainsi que le nom de la collection et la base de donnée et le fichier JSON à importer.

```
C:\mongo\bin>mongoimport  --db reseau_social -c user < user.json
```

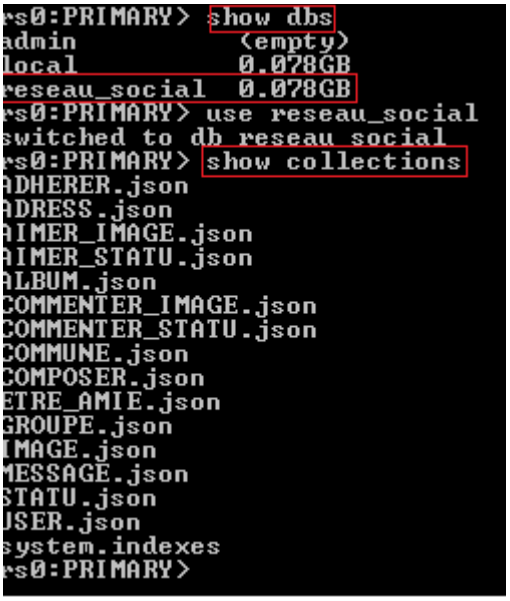
Mais si nous devrons importer plusieurs fichiers JSON, on doit utiliser la commande suivante

```
C:\mongo\bin>For %i IN (*.json) DO mongoimport --host 127.0.0.1:27016 --db resea
u_social --collection %i --type json --file %i
```

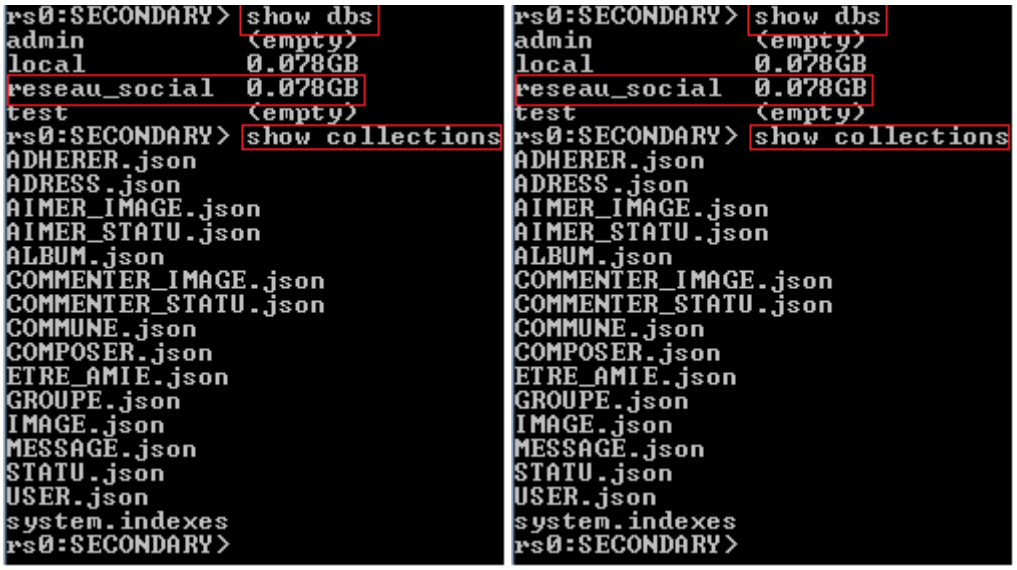
```
C:\mongo\bin>For 27016 --db reseau_social --collection i
C:\mongo\bin>For %i IN (*.json) DO mongoimport --host 127.0.0.1:27016 --db resea
u_social --collection %i --type json  --file %i

C:\mongo\bin>mongoimport --host 127.0.0.1:27016 --db reseau_social --collection
ADHERER.json --type json  --file ADHERER.json
connected to: 127.0.0.1:27016
2015-05-10T14:14:09.952+0200 imported 1 objects

C:\mongo\bin>mongoimport --host 127.0.0.1:27016 --db reseau_social --collection
ADRESS.json --type json  --file ADRESS.json
connected to: 127.0.0.1:27016
2015-05-10T14:14:10.564+0200 check 9 1000
2015-05-10T14:14:10.681+0200 imported 1000 objects

C:\mongo\bin>mongoimport --host 127.0.0.1:27016 --db reseau_social --collection
AIMER_IMAGE.json --type json  --file AIMER_IMAGE.json
connected to: 127.0.0.1:27016
2015-05-10T14:14:11.091+0200 imported 1 objects

C:\mongo\bin>mongoimport --host 127.0.0.1:27016 --db reseau_social --collection
AIMER_STATU.json --type json  --file AIMER_STATU.json
connected to: 127.0.0.1:27016
2015-05-10T14:14:11.763+0200 imported 1 objects

C:\mongo\bin>mongoimport --host 127.0.0.1:27016 --db reseau_social --collection
ALBUM.json --type json  --file ALBUM.json
connected to: 127.0.0.1:27016
2015-05-10T14:14:12.249+0200 imported 1 objects

C:\mongo\bin>mongoimport --host 127.0.0.1:27016 --db reseau_social --collection
COMMENTER_IMAGE.json --type json  --file COMMENTER_IMAGE.json
connected to: 127.0.0.1:27016
2015-05-10T14:14:12.601+0200 imported 1 objects

C:\mongo\bin>mongoimport --host 127.0.0.1:27016 --db reseau_social --collection
COMMENTER_STATU.json --type json  --file COMMENTER_STATU.json
connected to: 127.0.0.1:27016
2015-05-10T14:14:12.730+0200 imported 1 objects

C:\mongo\bin>mongoimport --host 127.0.0.1:27016 --db reseau_social --collection
COMMUNE.json --type json  --file COMMUNE.json
```

Après l'importation, nous allons vérifier si les données sont vraiment importées dans le nœud primaire ? Et est ce que sont-ils répliquées sur les nœuds secondaires ?

- Nœud primaire :

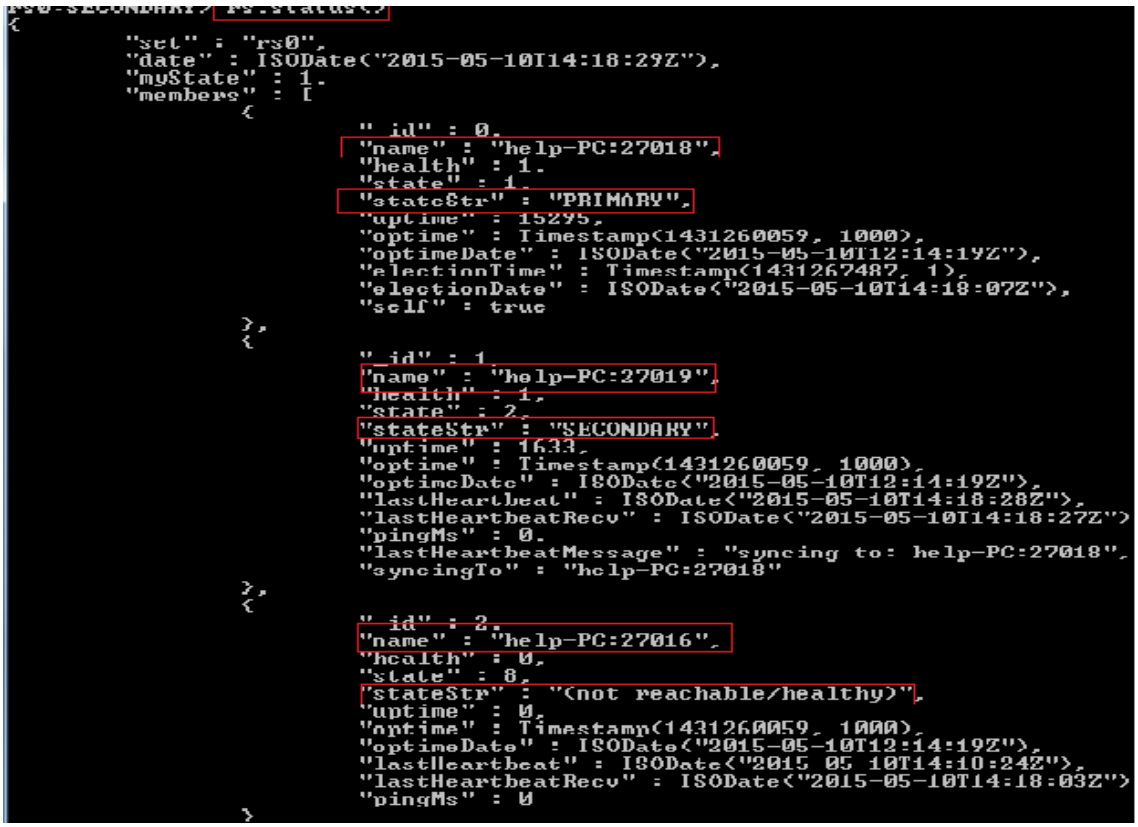


- Les deux nœuds secondaires :



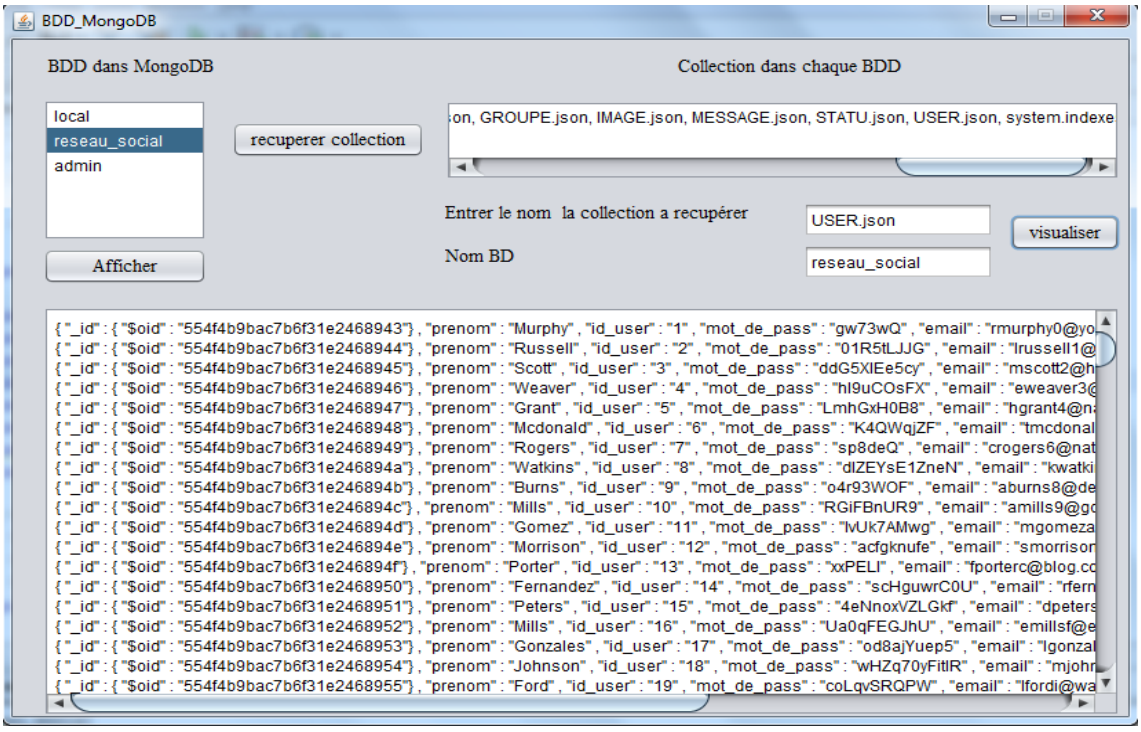
Donc les données sont répliquées sur les deux nœuds.

Un autre cas est prévu, est au cas où que le nœud primaire tombe en panne, alors il faut faire une sélection d'un nœud primaire parmi les nœuds secondaires. Pour vérifier ça, on se connecte à n'importe quel nœud secondaire et on exécute la commande rs.status () pour identifier quel nœud parmi les nœuds secondaires devient le primaire.



Donc le nœud secondaire 127.0.0.1 :27018 est devenu le nœud primaire, par conséquent toutes les lectures et écritures passent par ce nœud.

Pour visualiser le contenu de notre base de données MongoDB, on lance notre application :



Test et exploitation du sharding

Nous commençons par tester le sharding pour une collection MongoDB. Nous allons prendre un exemple appliqué sur la collection user.

Etant donné que nous sommes connectés par l'interpréteur mongos et nous avons activé le sharding pour la base de données et la collection (voir la section 5.4) , nous allons insérer un ensemble de données à la collection User en utilisant le localhost :27011.

```
C:\mongo\bin>mongo --port 27011 --host localhost
MongoDB shell version: 2.6.8
connecting to: localhost:27011/test
mongos> use reseau_social
switched to db reseau_social
mongos> db.User.count()
18000
mongos>
```

Pour s'assurer que la distribution à été bien faite dans les deux nœuds Shard1, Shard2, nous utilisons la commande getShardDistribution() de la collection :

```
mongos> db.User.getShardDistribution()

Shard shard0000 at localhost:27012
 data : 2.06MiB docs : 9009 chunks : 1
 estimated data per chunk : 2.06MiB
 estimated docs per chunk : 9009

Shard shard0001 at localhost:27013
 data : 2.05MiB docs : 8991 chunks : 2
 estimated data per chunk : 1.02MiB
 estimated docs per chunk : 4495

Totals
 data : 4.11MiB docs : 18000 chunks : 3
 Shard shard0000 contains 50.04% data, 50.04% docs in cluster, avg obj size on
hard : 240B
 Shard shard0001 contains 49.95% data, 49.95% docs in cluster, avg obj size on
hard : 240B
```

Nous remarquons que les données ont été bien distribuées entre les deux nœuds de notre environnement configuré : le 1er Shard contient 9009 chunks (50,04%) et le 2^{iéme} Shard contient 8991 chunks (49.95%) .

Nous pouvons aussi se connecter à l'un des Shard pour consulter la collection User, nous visualisonsseulement les chunks de ce Shard, ce qui confirme les résultats acquis.

```
C:\mongo\bin>mongo --port 27012
MongoDB shell version: 2.6.8
connecting to: 127.0.0.1:27012/test
Server has startup warnings:
2015-05-05T15:19:58.456+0200 [initandlisten]
2015-05-05T15:19:58.456+0200 [initandlisten] ** NOTE: This is a 32 bit MongoDB
inary.
2015-05-05T15:19:58.456+0200 [initandlisten] **       32 bit builds are limited
to less than 2GB of data (or less with --journal).
2015-05-05T15:19:58.456+0200 [initandlisten] **       Note that journaling defa
lts to off for 32 bit and is currently off.
2015-05-05T15:19:58.456+0200 [initandlisten] **       See http://dochub.mongodb
org/core/32bit
2015-05-05T15:19:58.456+0200 [initandlisten]
> use reseau_social
switched to db reseau_social
> db.User.count()
9009
```

```
C:\mongo\bin>mongo --port 27013
MongoDB shell version: 2.6.8
connecting to: 127.0.0.1:27013/test
Server has startup warnings:
2015-05-05T15:24:15.826+0200 [initandlisten]
2015-05-05T15:24:15.826+0200 [initandlisten] ** NOTE: This is a 32 bit MongoDB b
inary.
2015-05-05T15:24:15.826+0200 [initandlisten] **       32 bit builds are limited
to less than 2GB of data (or less with --journal).
2015-05-05T15:24:15.827+0200 [initandlisten] **       Note that journaling defau
lts to off for 32 bit and is currently off.
2015-05-05T15:24:15.827+0200 [initandlisten] **       See http://dochub.mongodb.
org/core/32bit
2015-05-05T15:24:15.827+0200 [initandlisten]
> use reseau_social
switched to db reseau_social
> db.User.count()
3991
>
```

7. Résultats et comparaison entre MySQL et MongoDB

Dans cette section, nous présentons quelques résultats importants obtenus en comparant les performances entre MySQL et MongoDB pour note base de données. Plusieurs facteurs et critères de comparaison peuvent être utilisées, nous avons choisi le temps d'exécution, l'élasticité.

7.1 Temps exécution

Pour une confrontation entre les deux SGBD, nous exécutons quelques requêtes et calculons le temps écoulé pour chaque SGBD, le temps d'exécution des requêtes dépend fortement de la nature de ces requêtes, certaines requêtes bien conçues dans le relationnel peuvent être plus rapide que celles dans le NoSQL et vice versa. Donc, la comparaison de temps d'exécution entre MongoDB et MySQL est strictement relative à la nature des requêtes et la façon dont sont modélisées les collections incluses. Dans l'exemple de la figure suivante nous avons comparé le temps d'exécution d'une recherche globale sur la collection USER de 1000 lignes : dans MySQL le temps est presque 31 millisecondes, et pour MongoDB c'est 1 milliseconde. Ce résultat positif pour MongoDB dépend justement de cette requête et ne peut être généralisé.

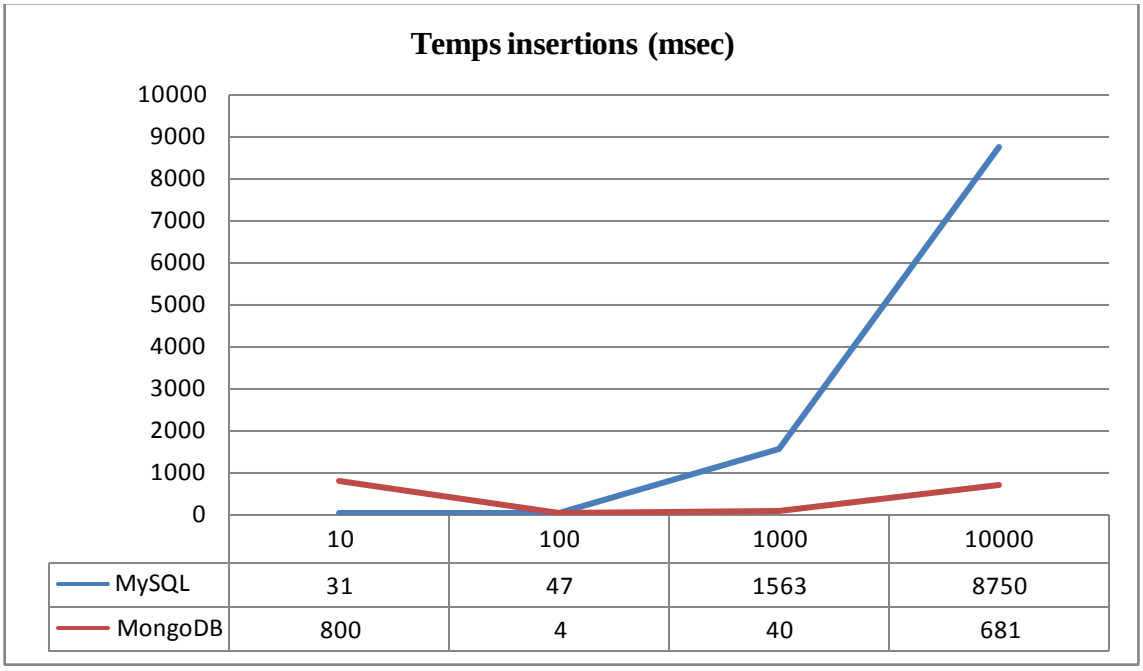
- **MySQL**

```
mysql> SELECT * from user;

1000 rows in set (0.02 sec)
```

- **MongoDB**

```
rs0:PRIMARY> use reseau_social
switched to db reseau_social
rs0:PRIMARY> db.USER.json.find().pretty().explain()
{
        "cursor" : "BasicCursor",
        "isMultiKey" : false,
        "n" : 1000,
        "nscannedObjects" : 1000,
        "nscanned" : 1000,
        "nscannedObjectsAllPlans" : 1000,
        "nscannedAllPlans" : 1000,
        "scanAndOrder" : false,
        "indexOnly" : false,
        "nYields" : 7,
        "nChunkSkips" : 0,
        "millis" : 1,
        "server" : "help-PC:27016",
        "filterSet" : false
}
rs0:PRIMARY>
```

Nous passons à l'insertion dans les deux bases de données, nous avons pris un ensemble de 10, 100,1000, 10000 insertions et nous avons calculé le temps d'exécution dans chaque SGBD MySQL et MongoDB.



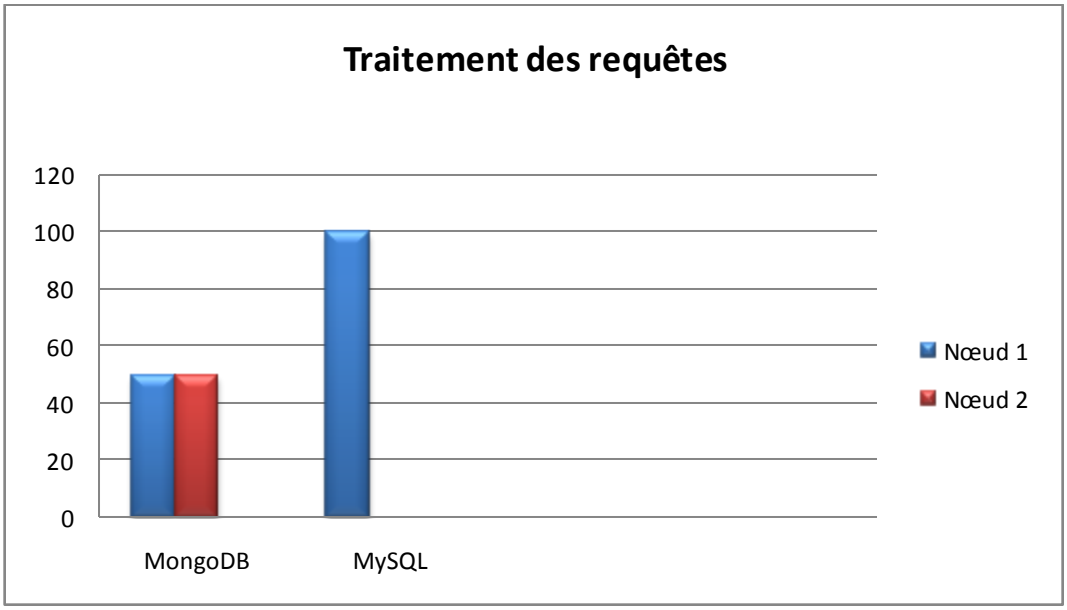
	10	100	1000	10000
MySQL	31	47	1563	8750
MongoDB	800	4	40	681

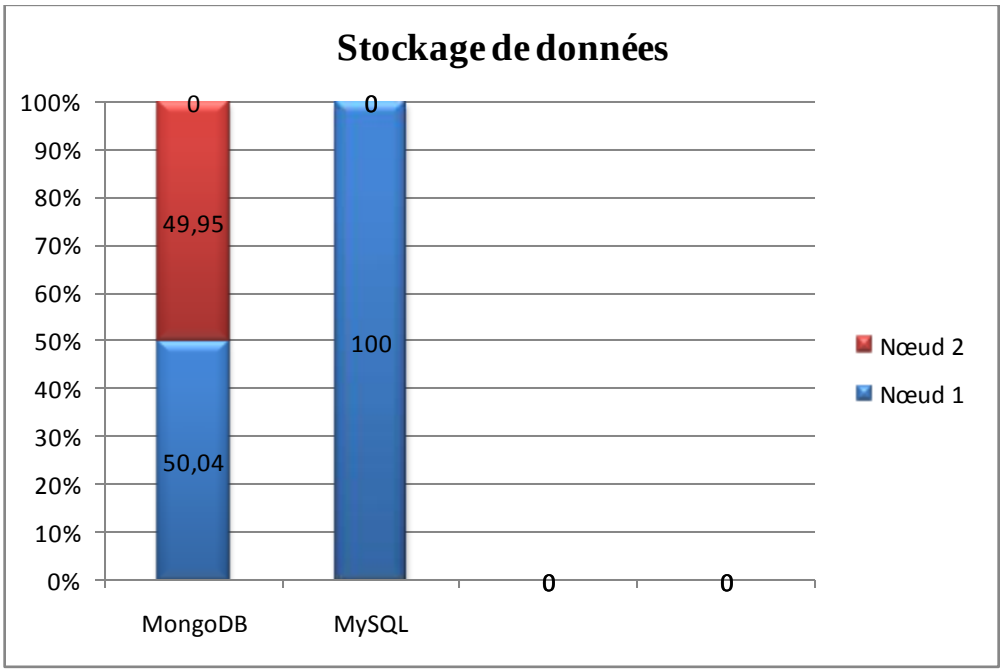
D'après le graphe et le tableau nous constatons que le MongoDB est plus efficace lorsque nous insérons une grande quantité de données d'une part, d'autre part elle a pris un peu de temps pour insérer seulement 10 objets.

7.2 Elasticité

Le gain principal de la performance obtenu est l'élasticité qui correspond à la possibilité d'utiliser plusieurs nœuds pour distribuer les données, et partager les traitements entre eux. La figure suivante compare les performances des deux bases de données MySQL et MongoDB pour le stockage et le traitement.

On peut constater dans cette figure, que dans MongoDB, les deux nœuds partagent les données et participent aux traitements des requêtes, par contre que dans MySQL un seul nœud prend en charge le stockage et le traitement.





8. Conclusion

Dans ce chapitre nous avons parcouru toutes les étapes et exécuté toutes les opérations requises pour la migration des données du système relationnel MySQL vers le système NoSQL MongoDB.

L'étude comparative montre que la solution MongoDB est très bénéfique et donne de meilleures performances par rapport à celle de MySQL, soit en temps de réponse, soit en équilibrage de charges entre les nœuds.
Ainsi, on peut conclure que MongoDB est une base de données NoSQL légère, rapide et simple à maitriser.
L'atout majeur de MongoDB et son efficacité ressentie dans les environnements à grande échelle ou le temps de réponse et la disponibilité de l'information sont plus importants que son intégrité.

Conclusion générale

Actuellement, le NoSQL est une technologie qui émerge en puissance, il est mis œuvre dans des environnements manipulant de grandes masses de données tels que Google, Yahoo, Twitter, Facebook, etc. Les moteurs de recherche sont les premiers utilisateurs de ces technologies puisqu'ils ont besoin d'une grande puissance de stockage et de traitement de ces volumes de données, de même pour les réseaux sociaux qui gère une très grosse montée en charge dû au grand nombre d'utilisateurs et de requêtes simultanées.

Le NoSQL a efficacement contribué à résoudre la problématique d'élasticité et de flexibilité des bases de données dans des contextes distribués.

Les systèmes de bases de données relationnelles répondant au besoin transactionnel grâce aux propriétés ACID (Atomicity, Consistency, Isolation, et Durability) sont entrain d'atteindre leurs limites pour la gestion de grandes masses d'informations estimées en Tera et Peta Octets, en conséquence le passage vers les systèmes distribuées NoSQL s'impose comme la meilleure alternative pour palier aux rudes contraintes liées à la performance.

L'objectif de ce mémoire étant la proposition d'une approche de migration d'une base de données relationnelle SQL vers une base de données NoSQL orientée document MongoDB dans un environnement distribué.

L'exportation des données relationnelles vers un format reconnu par les deux systèmes était la clé de réussite de notre solution. L'importation des données à partir des fichiers JSON vers MongoDB était la deuxième phase du processus de migration.

La configuration de MongoDB dans un environnement complètement distribué s'est appuyée sur la réplication et le sharding.

En fin, on peut estimer que l'objectif tracé au préalable a été atteint, néanmoins plusieurs axes de recherches peuvent être développés dans le futur, comme la migration des métadonnées, qui n'a pas été traité dans ce travail, aussi les différents liens entre les tables de la base de données relationnelles, à savoir les clés étrangères. Le travail peut être étendu à d'autres SGBD Relationnels et d'autres Solutions NoSQL comme Cassandra, HBase, Redis et autres.

Références bibliographiques

[1] FOUCRET, A. (2011). *NoSQL une nouvelle approche du stockage et la manipulation des données*. France. Available at: http://www.smile.fr/Actualites/Nos-actualites/Livre-blanc-nosql

[2] Maletras, x.(2012). Le NoSQL –Cassandra .Thèse professionnelle.

[3] Bruchez, R. (2013). *Les bases de données NoSQL comprendre et mettre en œuvre*. Paris: Eyrolles.

[4] Léonard, M. (2014). *L'AVENIR DU NoSQL*. [En ligne] disponible sur : http://www.leonardmeyer.com/wp-content/uploads/.../avenirDuNoSQL.pdf .

[5] LACOMME, P., ARIDHI, S. and PHAN, R. (2014). *Base de données NoSQL et Big Data*. France: Ellipses.

[6] PIAZZA, A. (2013). *NoSQL Etat de l'art et benchmark*. Travail de Bachelor HES. Haute Ecole de Gestion de Genève, Informatique de gestion

[7]Figuière, M. (2015). *NoSQL Europe : Bases de données clé-valeur et Riak | Blog Xebia France*. [En ligne] disponible sur: http://blog.xebia.fr/2010/04/26/nosql-europe-bases-de-donnees-cle-valeur-et-riak

[8] Cloud Computing — Les bases de données NoSQL. (2014).. [en ligne] disponible sur : http://www.heig-vd.ch

[9] HEINRICH, L. (2012). *Architecture NoSQL et réponse au théorème de CAP*. Travail de Bachelor HES. Haute Ecole de Gestion de Genève, Informatique de gestion.

[10] Morin, M. (2015). *Structure of a JSON Document*. [En ligne] disponible sur : http://ruby.about.com/od/tasks/a/Structure-Of-A-Json-Document.htm

[11] Couchdb.apache.org. *Apache CouchDB*. [En ligne] disponible sur http://couchdb.apache.org

[12] Figuière, M. (2015). *NoSQL Europe : Bases de données orientées colonnes et Cassandra | Blog Xebia France*. [En ligne] disponible sur : http://blog.xebia.fr/2010/05/04/nosql-europe-bases-de-donnees-orientees-colonnes-et-cassandra

[13] InfoQ, (2015). *Les Bases Orientées Graphes, NoSQL et Neo4j*. [En ligne] disponible sur: http://www.infoq.com/fr/articles/graph-nosql-neo4j.

[14] Brun, G. and Brun, G. (2011). *NoSQL vs relationnel ou NoSQL + relationnel*. [En ligne] disponible sur : http://blog.oxiane.com/2011/06/15/nosql-vs-relationnel-ou-nosql-relationnel .

[15] Db-engines.com, (2015). *DB-Engines - Knowledge Base of Relational and NoSQL Database Management Systems*. [En ligne] Available at: http://www.db-engines.com

[16] Copeland, R. (2013). *MongoDB applied design patterns*. Sebastopol, CA: O'Reilly Media, Inc.

[17] Tiwari, S. (2011). *Professional NoSQL*. Indianapolis, IN: Wiley.

Liste des figures

Liste des tableaux

Résumé

L'informatisation croissante de tous les domaines (Astrologie, Météorologie, E-commerce, E-Administration, E-Gouvernement, Multi-media...) a eu pour conséquence une augmentation exponentielle des volumes de données qui se compte désormais en pétaoctets. La gestion de ces volumes de données est donc devenue un problème que les bases de données relationnelles ne sont plus en mesure de gérer en raison des propriétés d'acidité. En réponse à ce passage à l'échelle accéléré, de nouveaux concepts ont émergé comme le Big Data, le NoSQL.
Notre projet consiste à concevoir et appliquer un ensemble de règles de migration automatique des données d'une solution SQL à savoir MySQL vers une solution MySQL à savoir MongoDB. La réplication (duplication des données) et le sharding (distribution de données) sont les ajouts majeurs de MongoDB pour répondre aux nouveaux besoins imposés par l'explosion de données, à savoir la performance, l'extensibilité et la disponibilité.

Mots Clés : Base de données relationnelles, Big Data, NoSQL, MongoDB, Réplication, Partitionnement.

Abstract

The increasing computerization of all the fields (Astrology, Meteorology, E-commerce, E-Administration, E-Govenrment, Multi-media) as a consequence had an exponential increase in volumes of data which amounts from now on in pétaoctets. The management of these volumes of data thus became a problem which the relational databases are not any more able to manage because of the properties of acidity. In answer tothis passage on the scale accelerated, new concepts emerged as Big Dated, NoSQL. Our project consists to conceive and apply a set of rules of automatic migration of the data of asolution SQL to knowing MySQL towards a MySQL solution with knowing MongoDB. The replication (duplication of data) and the sharding (distribution of data) are the major additions of MongoDB to meet the new needs imposed by the explosion for data, namely the performance, extensibility and the availability.

Key Words: Relational databases, Big Data, NoSQL, MongoDB, Replication, Sharding.

التلخيص

وقد أدت الحوسبة المتزايدة في جميع الميادين (علم التنجيم، والطقس، التجارة الإلكترونية، الحكومة الإلكترونية، الحكومة الإلكترونية، الوسائط المتعددة ...) إلى الزيادة الهائلة في حجم البيانات التي تتمثل الآن بيتابايت. وبالتالي فإن إدارة هذه البيانات أصبحت مشكلة. ردا على هذا التحول المتسارع للحجم، ظهرت مفاهيم جديدة من أجل حل هذه المشاكل.

الكلمات المفتاحية

قاعدة البيانات المصفوفة , قاعدة البيانات الكبيرة ,نو سكل, مونقو دب , النسخ المتماثل ,التقسيم