Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Saad Dahleb Blida 1

Faculté des Sciences

Département d'Informatique

Mémoire de Fin d'étude

En vue d'obtention de Master 2 en Informatique

Spécialité : Génie des Systèmes Informatiques

Thème

Système de mapping du schéma des bases de données relationnelles au schéma des bases de données NoSQL

Présenté par :

CHOUCHA Chems-eddine

AMRAOUI Farouk

Promotrice : M^me^ Arkam Meriem

Année Universitaire 2016/2017

Remerciement

Nos remerciements vont en premier lieu à ALLAH
le tout puissant de m'avoir donné la foi
et de m'avoir permis d'en arriver là.
ON tient à remercier particulièrement notre Promotrice
Mme Arkam.M pour ces conseils et son suivi durant la
réalisation d ce projet.
Je remercie très sincèrement, les membres de jury
d'avoir bien voulu accepter de faire partie de la
commission d'examination de ce sujet.
On tient à remercier
Tous mes enseignants, qui ont contribué à notre
formation tout au long de nos années d'études.

DEDICACE

*Au nom du **DIEU** clément et miséricordieux et que le salut de DIEU soit sur son prophète **MOHAMED***

Je dédie ce modeste travail aux personnes qui me sont les plus chères :

La lumière de ma vie et l'espoir de mon existence, source d'affectation de courage et d'inspiration qui a autant sacrifié pour me voir atteindre ce jour... ma chère mère.

Celui qui rend tel que je suis et tel qu'il souhaite, source de respect, en témoignage de ma profonde reconnaissance pour tout l'effort et le soutien incessant qui m'a toujours apporté… mon cher père.

A Mes chers frères.

A Mes chères sœurs.

A toute la famille CHOUCHA
A mon binôme Farouk et sa famille pour le quel je souhaite une vie pleine de joie et de réussite.

Mes très chers amis

Je remercie enfin tous ceux qui m'ont aidé de prés ou de loin dans l'accomplissement de ce travail.

Et à tous ceux qui me connaissent ...
À toute la promotion 2016/2017 des Etudiants de GSI et le département d'informatique.

Chems-eddine

DEDICACE

*Au nom du **DIEU** clément et miséricordieux et que le salut de DIEU soit sur son prophète **MOHAMED***

Je dédie ce modeste travail aux personnes qui me sont les plus chères :

La lumière de ma vie et l'espoir de mon existence, source d'affectation de courage et d'inspiration qui a autant sacrifié pour me voir atteindre ce jour... ma chère mère.

Celui qui rend tel que je suis et tel qu'il souhaite, source de respect, en témoignage de ma profonde reconnaissance pour tout l'effort et le soutien incessant qui m'a toujours apporté… mon cher père.

A Mes chers frères.

A Mes chères sœurs.

A toute la famille AMRAOUI
A mon binôme Chems-eddine et sa famille pour le quel je souhaite une vie pleine de joie et de réussite.

Mes très chers amis

Je remercie enfin tous ceux qui m'ont aidé de prés ou de loin dans l'accomplissement de ce travail.

Et à tous ceux qui me connaissent ...
À toute la promotion 2016/2017 des Etudiants de GSI et le département d'informatique.

Farouk

Sommaire

Liste des figures

Résumé

En quelques années, le volume des données brassées par les entreprises a considérablement augmenté. Avec cette augmentation exponentielle des données, plusieurs utilisateurs des systèmes de bases de données relationnelles veulent migrer vers les bases de données NoSQL. Alors une question très pertinente s'impose : y'a-t-il une solution pour la migration directe de système de gestion des bases de données relationnels vers les bases de données NoSQL ?

Vu l'intérêt de la problématique posée, l'objectif ciblé de ce PFE consistait à proposer des règles de mapping du schéma relationnel au schéma NoSQL (MongoDB). Une fois les règles de passages établies, la recherche peut également être étendue en proposant la conversion de requêtes de SQL en requêtes NoSQL.

Mots clé : SQL, NoSQL, Base de données, Moteur de mapping, MongoDB, Migration de données.

Abstract

In a few years, the volume of data brewed by companies has increased considerably. With this exponential increase in data, many users of relational database systems want to migrate to NoSQL databases. So, a very relevant question is: Is there a solution for the direct migration of relational database management system to NoSQL databases? Given the interest of the problem, the targeted objective of this project was to propose rules for mapping the relational schema to the NoSQL schema (MongoDB). Once the pass rules are established, the search can also be extended by proposing the conversion of SQL queries to NoSQL queries.

Keywords: SQL, NoSQL, Database, Mapping engine, MongoDB, Data migration.

ملخص

وفي غضون سنوات قليلة، زاد حجم بيانات الشركات زيادة كبيرة. مع هذه الزيادة الهائلة في البيانات، العديد من مستخدمي أنظمة قاعدة البيانات العلائقية تريد الهجرة إلى قواعد بيانات الغير علائقية. لذلك، سؤال مهم جدا يطرح نفسه هو: هل هناك حل للهجرة المباشرة من نظام إدارة قواعد البيانات العلائقية لقواعد بيانات الغير علائقية؟

من أجل حل لهذه المشكلة، ، كان الهدف من هذا المشروع هو اقتراح نظام لتحويل المخطط العلائقي إلى مخطط غير علائقي. يمكن أيضا توسيع البحث عن طريق اقتراح تحويل استعلامات قواعد البيانات العلائقية إلى استعلامات قواعد البيانات الغير علائقية.

الكلمات المفتاحية: قواعد البيانات الغير علائقية، محرك تنظيم قواعد البيانات، هجرة البيانات

Introduction

Introduction Générale

Le passage au XXI[e] siècle a vu une augmentation considérable des volumes de données manipulées par les entreprises et organismes, notamment ceux en rapport avec Internet, réseaux sociaux, opérateurs téléphoniques, santé, agences nationales de défense du territoire, l'informatisation croissante et en tout genre implique une multiplication exponentielle de ce volume de données qui se compte maintenant en téraoctets et même en pétaoctets, c'est ce qu'on l'appelle le 'BigData'. Cela nécessite non seulement un stockage de données très important, mais aussi des besoins de traitements sur ces volumes. En plus et devant le besoin grandissant en performance et en disponibilité des services/sites possédant un fort trafic, un point souvent bloquant est la base de données. Les bases de données relationnelles atteignent vite leurs limites et rajouter des serveurs n'augmente plus assez les performances. Suite à ce problème, de nouvelles technologies ont émergé tels que les bases de données NoSQL, celles-ci changent radicalement l'architecture des bases de données que nous avons l'habitude de voir et permettent ainsi d'augmenter les performances et la disponibilité des services, il y a bien évidemment des « mais », la réponse parfaite n'existe pas. C'est ainsi que Google a migré au NoSQL en 2004 avec le moteur Bigtable. Il fut suivi par les géants du web social à savoir Facebook, Twitter et LinkedIn. Ces entreprises ont effectué la migration partielle ou totale de leur système d'information des bases de données relationnelles vers des bases NoSQL. Aussi, cette Technologie étant relativement récente, et n'existant pas encore des processus de migration standards ou formels au même titre que les autres bases de données Objet, Objet relationnel et hiérarchique bien connus de la communauté scientifique et des développeurs.

Il nous a semblé utile de proposer un moteur de mapping automatique pour la traduction des requêtes et la migration des bases de données relationnel vers une base de données NoSQL tel-que MongoDB, Riak ou Cassandra. Pour ce faire, nous avons commencé par l'étude des limites des SGBD relationnels face aux gros volumes de données ont été présentées. Ensuite, la technologie NoSQL et ses points forts, ses enjeux et défis ont été abordés. Puis et après le choix du MongoDB comme base de données cible dans le NoSQL nous présentons ce dernier ainsi que son architecture et ces principes de fonctionnements. Enfin, une approche de migration d'une base de données relationnelle vers une base de données NoSQL orienté Documents (MongoDB) a été proposée. Cette approche se décline en trois parties à savoir : la conversion des requêtes SQL vers des requêtes en MongoDB, puis la migration des données

d'une BDD relationnel vers la base de données MongoDB a partir d'un fichier SQL, enfin la traduction du schéma source en BDD NoSQL cible la création d'une BDD en MongoDB à partir d'un MCD classique d'une BDD relationnel. Un moteur de mapping de données a de ce fait, été élaboré qui couvre les trois opérations majeures plus haut mentionnées.

Chapitre 1 : Etude bibliographique

1. Introduction

De nos jours, les bases de données sont des outils que nous utilisons quotidiennement sans vraiment nous en rendre compte, que ce soit au travers d'internet, en allant visiter divers sites, ou alors au sein de notre ordinateur. Actuellement elles sont utilisées dans la plupart des projets informatiques. Leurs constructions ainsi que leur administration en font un métier à part entière qui nécessite un bon apprentissage.

Bien que le modèle relationnel soit arrivé depuis 50 ans et qu'il ait su s'imposer par sa facilité d'utilisation ainsi que ses contraintes qui garantissent l'intégrité des données, de nouvelles bases ont fait leur apparition, les bases de données NoSQL. Elles sont venues se placer non pas en voulant se substituer, mais en devenant complémentaires.

Dans ce premier chapitre, nous présentons les bases de données relationnelles ainsi que leurs limites atteintes dans les environnements à grande échelle. Il ne s'agit pas de détailler exhaustivement ce type de bases de données et les notions liées à ce modèle, mais juste faire un survol global sur les principes et les concepts les plus importants, puis nous établirons un état de l'art sur les nouveaux systèmes dites NoSQL d'où nous allons voir leur émergence, leurs différents types et nous allons montrer leur utilité face aux nouvelles exigences.

2. Définitions et concepts de base

Dans cette section nous allons essayer de détailles toutes les notions nécessaires concernant les bases de données relationnelle et non relationnelle, la différence entre elles, et les travaux déjà effectués concernant le passage des bases de données SQL vers les bases de données NoSQL.

2.1 Définition d'une base de données

De façon informelle, on peut considérer une base de données comme un ensemble structuré de données, centralisées ou non, servant pour les besoins d'une ou plusieurs applications, interrogeables et modifiables par un groupe d'utilisateurs en un temps opportun.

Plus formellement, une base de données (BDD) est un ensemble d'informations exhaustives, non redondantes, structurées et persistantes, concernant un sujet. Une donnée est dite "persistante" si sa durée de vie excède la durée d'exécution du programme qui l'a créée.

Une donnée persistante peut être rechargée en mémoire principale à tout instant. En pratique la persistance de données est assurée par leur stockage sur un support permanent (disque…) [1].

2.2 Modèle relationnel

Le modèle relationnel a été conçu par Codd au début des années 1970. Les premiers systèmes de bases de données ont été bâtis dans des centres de recherche et prennent en charge le langage SQL ou d'autres langages de requête similaires. Avec le temps les produits issus de la technologie relationnelle ont atteint un haut degré de maturité et conquis le monde de la pratique.

2.3 Un Système de gestion de base de données

Un Système de Gestion de Bases de Données **(SGBD)** est un langage (logiciel) qui sert à interagir avec une BDD et qui permet à l'utilisateur de définir les données de la base, de les consulter et de les mettre à jour. Autrement dit, il permet de décrire, modifier, interroger et administrer les données, en garantissant la qualité, la pérennité et la confidentialité des informations, tout en cachant la complexité des opérations. C'est, en fait, l'interface entre la base de données et les utilisateurs [2].

2.4 Un Système de gestion de base de données Relationnels

Un système de gestion de bases de données relationnelle (relational database management system, en anglais), SGBDR en abrégé, souvent appelé simplement système de bases de données relationnelles, est un système intégré pour la gestion unifiée des bases de données relationnelles, comme le montre la figure 1. Un SGBDR dispose de fonctions utilitaires d'une part, et d'un langage descriptif pour la définition et la manipulation de données d'autre part.

Un système de bases de données relationnelles est constitué d'un composant de stockage et d'un composant de gestion de données (voir figure 1): le composant de stockage a pour but de réunir dans des tables l'ensemble des données et tous les liens qui les unissent. On distingue d'une part les tables qui contiennent des données appartenant aux applications des utilisateurs, d'autre part les tables systèmes indispensables au fonctionnement d'une base de données. Les

tables systèmes contiennent les définitions de données que les utilisateurs peuvent consulter à tout moment sans être autorisés à les modifier. Le composant de gestion comporte essentiellement un langage relationnel de définition et de manipulation des données. Ce composant englobe aussi des fonctions utilitaires telles que la restauration de la base de données en cas de panne, la protection et la sécurité des données [2].

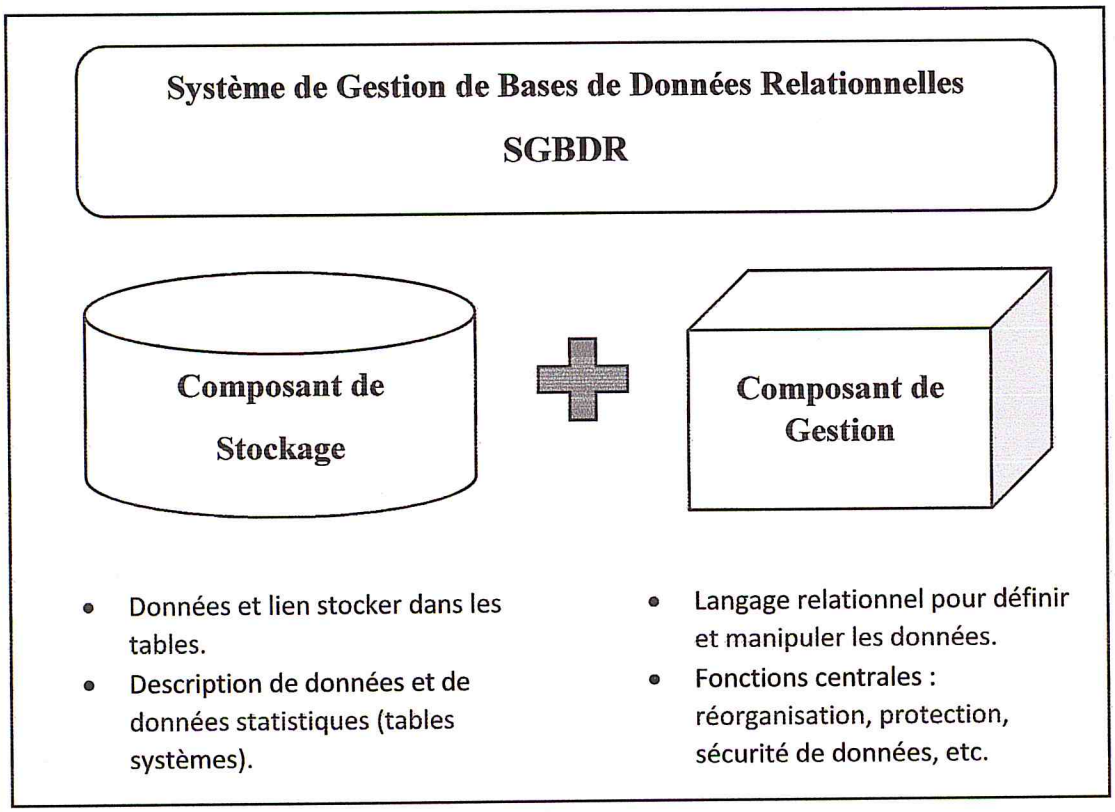


Figure 1 : Les deux composants d'un système de bases de données relationnelles[2]

2.5 Propriétés d'un SGBD relationnel

Un système de bases de données relationnelles se caractérise par les propriétés suivantes :

- Base formelle : Les SGBDR reposent sur une base formelle qui permettent une conception cohérente des bases de données et garantissent des structures de données propres.
- Les langages de requête relationnels sont de nature ensembliste : Implémentation d'un langage relationnel ensembliste permettant à l'utilisateur de décrire aisément les interrogations et manipulation qu'il souhaite effectuer sur les données.

- Organisation structurée des données dans des tables interconnectées (d'où le qualificatif relationnel), pour pouvoir détecter les dépendances et redondances des informations.
- Indépendance des données vis-à-vis des programmes applicatifs (dissociation entre la partie "stockage de données" et la partie "gestion" - ou "manipulation").
- Plusieurs utilisateurs peuvent travailler simultanément : Gestion des opérations concurrentes pour permettre un accès multi-utilisateur sans conflit.
- La cohérence et l'intégrité des données sont assurées : Ce terme englobe le stockage de données sans erreur, la protection contre les destructions, les pertes, les abus et les accès non autorisés. [2]

2.6 Limites du modèle relationnel

Les bases de données existent maintenant depuis environ 50 ans et le modèle relationnel depuis environ 40 ans. Ce modèle bien que très puissant, a des limites que certains services et sites, tels que Google, Facebook, etc. ont atteintes depuis longtemps. En effet ce genre de site possède plusieurs centaines de millions d'entrées dans leurs bases de données et tout autant de visites journalières. Par conséquent, une seule machine ne peut pas gérer la base de données, de plus pour des raisons de fiabilité, ces bases de données sont dupliquées pour que le service ne soit pas interrompu en cas de panne. La méthode consiste donc à rajouter des serveurs pour dupliquer les données et ainsi augmenter les performances et résister aux pannes. Seulement, dû aux propriétés fondamentales sur lesquelles une base de données relationnelle repose, cette approche connait quelques limites à savoir [5] :

2.6.1 Problème lié au partitionnement vertical de données

Le principal point sensible des applications est bien souvent la base de données qui lorsqu'elle est transactionnelle ou distribuée ne permet pas facilement le passage à l'échelle. En plus, pour la sécurité de fonctionnement, il faut que les données soient répliquées, ce qui complique encore un peu plus les choses. Dans les architectures modernes (orientées objet), la validation du modèle se fait désormais dans les objets, au niveau de l'applicatif. Lorsque l'on veut scaler une base de données, une première solution peut être le partitionnement :

- Vertical : une base de données pour les livres, une autre base pour les CDs, etc.

- Horizontal : une base de données pour les livres de A-M et une autre pour les livres de N-Z.

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

Les propriétés ACID, bien que nécessaires à la logique du relationnel, nuisent fortement aux performances surtout la propriété de cohérence. En effet, la cohérence est très difficile à mettre en place dans le cadre de plusieurs serveurs (environnement distribué), car pour que celle-ci, pour pouvoir satisfaire cette cohérence, il faut que tous les serveurs doivent être des miroirs les uns des autres, de ce fait deux problèmes apparaissent [1] :

- Le coût en stockage est énorme car chaque donnée est présente sur chaque serveur.
- Le coût d'insertion/modification/suppression est très grand.

2.6.3 Problème de requête non optimale dû à l'utilisation des jointures

Imaginons une table contenant toutes les personnes ayant un compte sur Facebook, soit plus de 800 millions, les données dans une base de données relationnel classique sont stockées par lignes, ainsi si on effectue une requête pour extraire tous les amis d'un utilisateur donné, il faudra effectuer la jointure entre la table des usagers (800 millions) et celle des amitiés (soit au moins 800 millions chaque usager ayant au moins un ami) puis 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. A cause de ces limitations, de nouvelles technologies ont vues le jour pour répondre à ces besoins, c'est ainsi qu'a été inventé d'autres architectures, celles-ci se nomment dans leur ensemble les bases NoSQL qui seront abordées plus loin. Malgré l'appellation NoSQL, ce n'est pas tant le langage SQL en lui-même qui est inadapté mais les grands principes sur lesquels sont construits le modèle relationnel et transactionnel. En effet, les bases de données relationnelles mettent à la disposition des développeurs un certain nombre d'opérations en d'opérations en fonction des relations entre les tables :

- Un système de jointure entre les tables permettant de construire des requêtes complexes faisant intervenir plusieurs entités (les tables en l'occurrence).
- Un système d'intégrité référentielle permettant de s'assurer que les liens entre les entités sont valides.

La mise en œuvre de tels mécanismes à un coût considérable dès lors que l'on se trouve dans le contexte d'un système distribué ou avec de gros volumes de données. Sur la plupart des SGBD relationnels, il convient de s'assurer en permanence que les données liées entre elles sont placées sur le même nœud du serveur. Lorsque le nombre de relations au sein d'une base augmente, il devient de plus en plus difficile de placer les données sur des nœuds différents du système. Nous verrons que les systèmes NoSQL règlent de différentes manières ce problème selon les besoins.

2.6.4 Problème lié au Théorème du CAP

Il existe par ailleurs, une limitation plus théorique à ce que peuvent réaliser les bases de données relationnelles distribuées qui est décrite par le théorème de CAP (Consistency, Availibility and Partition tolerence). Celui-ci énonce trois grandes propriétés pour les systèmes distribués à savoir :

- **La Cohérence (Consistency)** qui demande que tous les nœuds du système voient exactement les mêmes données au même moment.
- **La Disponibilité (Availability)** qui requiert que la perte de nœuds n'empêche pas les survivants de continuer à fonctionner correctement.
- **La Résistance au partitionnement (partition tolerance)** qui veut que aucune panne moins importante qu'une coupure totale du réseau ne doit empêcher le système de répondre correctement, c'est-à-dire qu'en cas de morcellement en sous-réseaux, chacun doit pouvoir fonctionner de manière autonome.

Le théorème de CAP stipule 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 types de systèmes à savoir : AC, AP, CP. S'agissant des systèmes AC, il s'agit des bases de données relationnelles implémentant les propriétés de cohérence et de Disponibilité (ce sont des systèmes AC). Les bases de données NoSQL sont généralement des systèmes CP (Cohérent et Résistant au partitionnement) ou AP (Disponible et Résistant au partitionnement) [3].

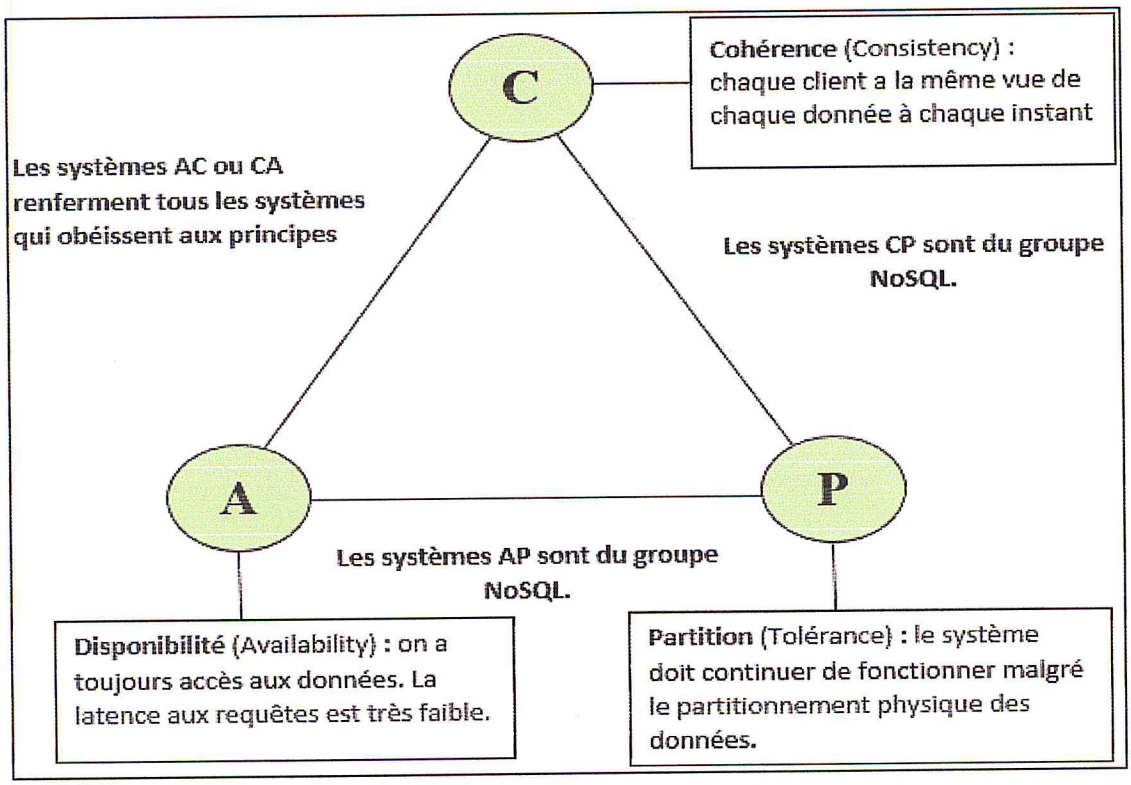


Figure 2 : Limites liées au Théorème du CAP. [5]

2.6.5 Problème lié à la gestion des objets hétérogènes

Le stockage distribué n'est pas la seule contrainte qui pèse à ce jour sur les systèmes relationnels disait « Carl STROZZI ». Au fur et à mesure du temps, les structures de données manipulées par les systèmes sont devenues de plus en plus complexes avec en contrepartie des moteurs de stockage évoluant peu. Le principal point faible des modèles relationnels est l'absence de gestion d'objets hétérogènes ainsi que le besoin de déclarer au préalable l'ensemble des champs représentant un objet. Pour répondre au besoin des objets hétérogènes non prés déclarés, on a vu apparaître des modélisations complexes sur le plan algorithmique comme le modèle Entity-Attribute-Value permettant de séparer un objet et ses champs. Ces modèles, quoique répondant au besoin de flexibilité des applications modernes, présentent un inconvénient majeur qui est leur très faible niveau de performance. La majeure partie des outils développés dans le cadre de la mouvance NoSQL permettent l'utilisation d'objet hétérogènes apportant comparativement une bien plus grande flexibilité dans les modèles de données ainsi qu'une simplification de la modélisation.

2.7 Synthèse

Les bases de données sont considérées comme les éléments les plus importants dans le monde informatique. Le modèle relationnel est imposé comme une norme et il est très répandu grâce à son efficacité et sa robustesse issues par les contraintes imposées par ce modèle.

Ces mêmes principes qui constituent la force de ce modèle, le rendent moins performant et moins réactif surtout dans un environnement distribué car elles empêchent les données d'être distribuées efficacement dans les nœuds constituants l'environnement distribué à grande échelle. De ce fait, et face à ces contraintes, des nouveaux systèmes de gestion de données nommés « NoSQL » viennent occuper leurs places en proposant des alternatives au modèle relationnel.

3. Le NoSQL, une nouvelle approche de stockage et de manipulation de données

Le NoSQL (Not only SQL) désigne une catégorie de base de données apparue en 2009 qui se différencie du modèle relationnel que l'on trouve dans des bases de données connues comme MySQL ou PostgreSQL. Ceci permet d'offrir une alternative au langage SQL.

Le NoSQL est apparu afin de contrer la dominance des bases de données relationnelles dans le domaine de l'internet. En effet, un des problèmes récurrents des bases de données relationnelles est la perte de performance lorsque l'on doit traiter un très gros volume de données. De plus, la multiplication des architectures distribués a apporté le besoin de disposer de solution s'adaptant nativement aux mécanismes de réplication des données et de gestion de la charge.

Dans ce qui suit, nous allons définir les bases de données No-SQL, puis l'historique de leur apparition avant de passer vers le pourquoi du No-SQL et comment il fonctionne.

3.1 Définition de NoSQL

NoSQL ou « Not Only SQL » est un aspect très récent (2009), qui concerne les bases de données. L'idée de cet aspect 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 actuelles 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. 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 [3].

Donc Le NoSQL est un type de base de données, c'est une manière de stocker et de récupérer des données de façon rapide, un peu comme une base de données relationnelle, sauf qu'il n'est pas basé sur des relations mathématiques entre les tables comme dans une base de données relationnelle traditionnelle.

3.2 Historique

Bien que le terme NoSQL soit entendu pour la première fois en 1988 par **Carlo Strozzi** qui présentait ça comme un modèle de base de données plus léger et sans interface SQL, c'est en 2009, lors de la rencontre NoSQL de San Francisco, qu'il prend un essor important. Durant cette conférence, les développeurs ont présenté des projets tels que Voldemort, Cassandra Project, Dynomite, HBase, Hypertable, CouchDB, MongoDB. Cette conférence était considérée comme l'inauguration de la communauté des développeurs de logiciel NoSQL.

NoSQL est une combinaison de deux mots : No et SQL. Qui pourrait être mal interprétée car l'on pourrait penser que cela signifie la fin du langage SQL et qu'on ne devrait donc plus l'utiliser. En fait, le « No » est un acronyme qui signifie « Not only », c'est-à-dire en français « non seulement », c'est donc une manière de dire qu'il y a autre chose que les bases de données relationnelles.

La figure suivant illustre l'évolution et les différentes bases de données depuis leur apparition la première fois dans les années 1950 jusqu'à maintenant :

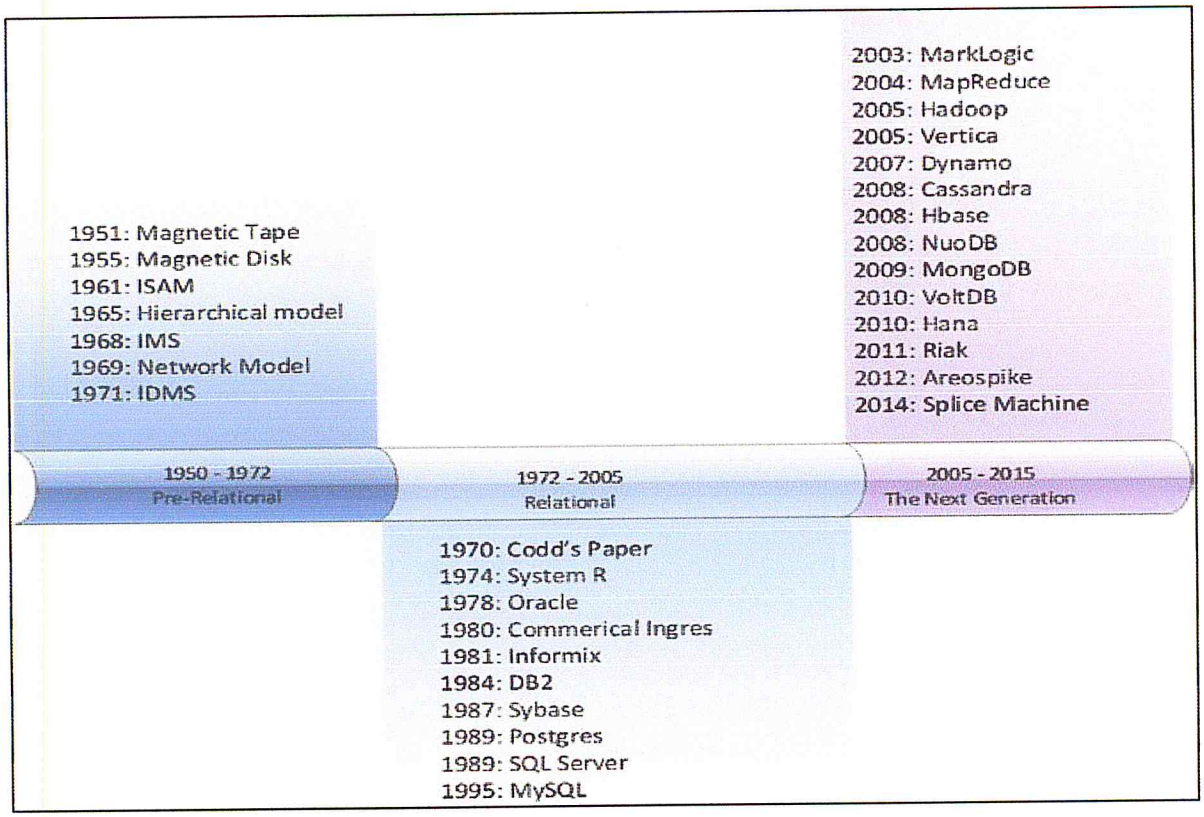


Figure 3 évolution des bases de données (1950-2015)

3.3 Nécessité du NoSQL

Le premier besoin auquel les bases NoSQL répondent est la performance. En effet, certaines entreprises, comme Google et Yahoo, ont vu leurs besoins changer au fil des années. La charge ainsi que le volume de données à gérer ont crû de façon exponentielle, ce qui a décidé ces dernières à changer leurs façons de stocker les données. Dans un premier temps, elles ont développé chacune de leur côté des solutions alternatives, en faisant des compromis sur les propriétés ACID des SGBDR.

Ces compromis, qui ne peuvent être concevables pour une base de données relationnelle, ont permis d'arriver à des solutions qui répondent mieux à la montée en charge et permettent ainsi une évolutivité plus simple. D'autres entreprises, comme Facebook, Twitter ou LinkedIn, sont allées sur le même créneau en migrant une partie de leur base de données sous une solution NoSQL. Cette migration croissante engendrée par ces dernières a donné du poids à la mouvance NoSQL ce qui a eu pour effet de multiplier et améliorer les solutions Open Source basées sur le NoSQL.

3.4 Fonctionnement du NoSQL

Les bases de données NoSQL reposent essentiellement sur plusieurs aspects qui font leurs forces et justifient leur usage aujourd'hui des géants du web. Il s'agit principalement du partitionnement horizontal des données, du fait que ces bases sont sans schéma et donc une grande flexibilité du schéma de données.

3.4.1 Scalabilité maîtrisée à travers le partitionnement horizontal

Le principe de la scalabilité horizontale consiste à simplement rajouter des serveurs en parallèle, c'est la raison pour laquelle on parle de croissance externe. On part d'un serveur basique et on rajoute des nouveaux serveurs pas chers « low cost » pour répondre à l'augmentation de la charge. Le but des systèmes NoSQL est de renforcer la scalabilité horizontale. [1]

- **Les avantages :**
  - Achat de serveurs quand le besoin s'en fait sentir.
  - Une panne de serveur ne pénalise pas le système.
  - Flexibilité du système.
  - Rendu financier dû à l'achat de matériel pas cher.

3.4.2 Les Propriétés de BASE

Le théorème CAP ne contredit en rien les propriétés ACID, mais il définit que les bases de données respectant ces dernières ne peuvent être situées qu'en C et A. En contrepartie, les bases de données NoSQL qui ne répondent pas à l'ACID ont décidé de répondre au théorème BASE, aussi appelé éventuellement consistent.

- Basically Availlable : Le système reste disponible, même en cas en de dysfonctionnement d'un des serveurs.
- Soft-state (cohérence faible) : qui permet d'être plus flexible au niveau des contraintes.
- Eventually consistent : La cohérence ne se fait pas forcément après chaque opération, mais après un certain temps.

BASE se veut comme une alternative à ACID. Ces dernières sont faites pour garantir l'intégrité des données entre elles à tout instant, alors que BASE se dit garant de l'intégrité dans le temps. C'est-à-dire qu'une base de données deviendra intègre après un certain moment, le

temps que l'information se propage entre les différents nœuds. Le théorème CAP ne se veut pas complètement détaillé, car il n'a pas l'utilité d'une marche à suivre pour arriver à un but. Il définit, avec une analyse de l'architecture, comment se situe une base de données vis-à-vis de ces trois attributs. [2]

4. Différents types de base de données NoSQL

On entend souvent parler du NoSQL comme étant un type de base de données à part entière, mais le NoSQL, venant de "Not only SQL", comprend une multitude de bases de données de type et d'utilité différentes. Il en existe 4 types distincts qui s'utilisent différemment et qui se prêtent mieux selon les types de données que l'on souhaite y stocker.

4.1 Les bases de données clé-valeur

La base de données de type clé-valeur est considérée comme la plus élémentaire. Son principe est très simple, chaque valeur stockée est associée à une clé unique. C'est uniquement par cette clé qu'il sera possible d'exécuter des requêtes sur la valeur.

La structure de l'objet stocké est libre et donc à la charge du développeur de l'application. 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.

En ce qui concerne le besoin de la scalabilité verticale, celle-ci est très fortement réduite par la simplicité des requêtes qui se résument à PUT, GET, DELETE, UPDATE. Ce type de base de données affiche donc de très bonnes performances en lecture/écriture et tendent à diminuer le nombre de requêtes effectuées, leur choix étant restreint.

Néanmoins, les requêtes ne peuvent être exécutées uniquement sur les clés à cause de la simplicité du principe. Permettant un stockage de données sans schéma, le client est contraint à récupérer tout le contenu du blob et ne peut obtenir un résultat de requêtes plus fin tel que des résultats de requêtes SQL pourraient permettre.

Ce type de base de données orienté clé-valeur implique donc des cas d'utilisation très spécifiques, à cause de leur simplicité de modèle et de la fine palette de choix de requêtes proposées. Ces systèmes sont donc principalement utilisés comme dépôt de données à

condition que les types de requêtes nécessitées soient très simples. On les retrouve comme système de stockage de cache ou de sessions distribuées, particulièrement là où l'intégrité des données est non significative. Aujourd'hui, les solutions les plus connues ayant adoptées le système de couple clé-valeur sont Voldemort (LinkedIn), Redis et Riak [4].

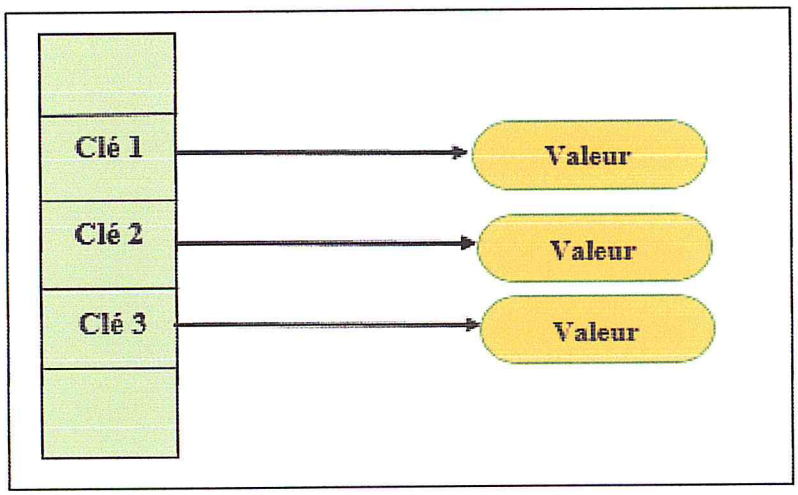


Figure 4 : illustration du modèle Clé/Valeur. [4]

4.2 Les bases de données orientées documents

Les bases fonctionnant 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.

```
{
titre : "Mon travail" ,
auteur : "AMRAOUI CHOUCHA" ,
dernier_Modif : new Date ("08/05/2017") ,
body : "NoSQL and CAP..." ,
tag : ["NoSQL", "MongoDB"] , }
```

Figure 5 : Un example d'un unretirement des données sous format JSON

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, même dans les cas où elles se retrouvent de façon hétérogène. On retrouve aussi une gestion plus simple au sein d'un cluster. Bien qu'il doit être configuré manuellement, il n'en reste pas moins que cette gestion ne se retrouve plus au sein de l'application et la base de données peut automatiquement répartir la charge entre les différents nœuds.

Les bases de données documentaires ressemblent aux bases de données hiérarchiques de par la structure des 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 est, avec les fonctions MapReduce, il est plus simple de parcourir et de rechercher les données.

On retrouve principalement MongoDB et CouchDB comme solutions basées sur le concept de base documentaire [5].

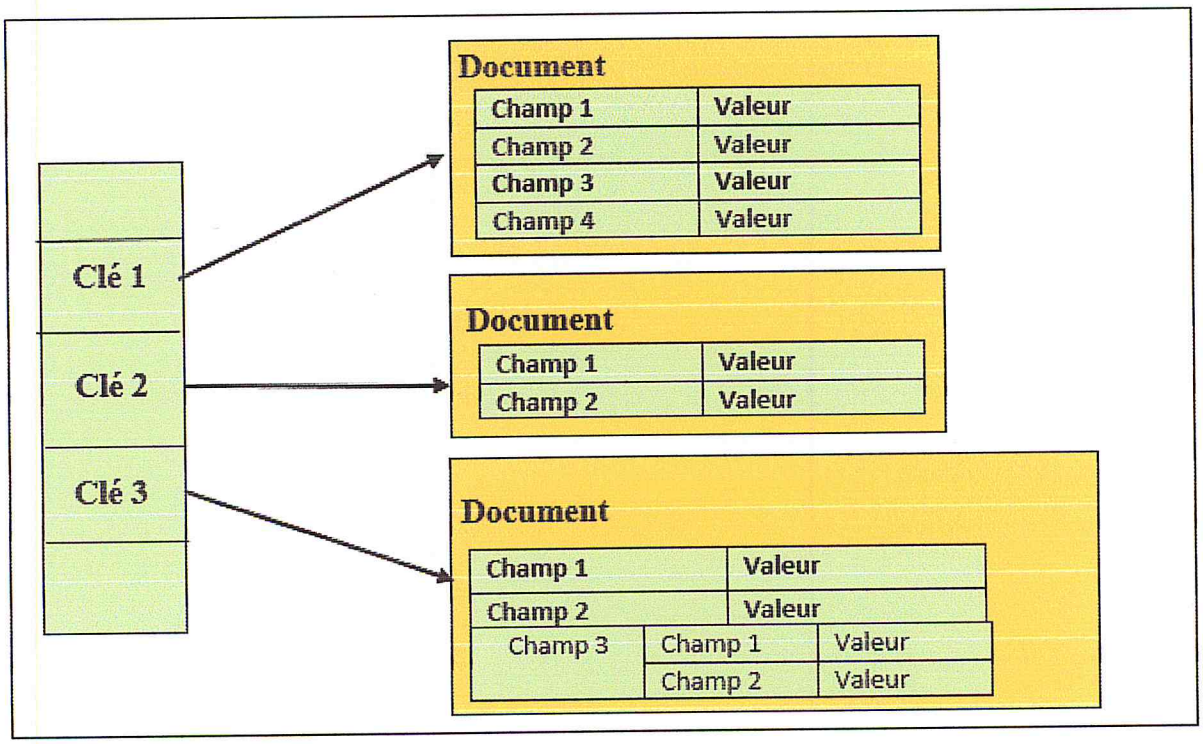


Figure 6 : Structure d'une BD orientée document. [5]

4.3 Les bases de données orientées colonnes

Les bases de données orientées colonnes ont été conçues par les géants du web afin de faire face à la gestion et au traitement de gros volumes de données s'amplifiant rapidement de jours en jours. Elles intègrent souvent un système de requêtes minimalistes proche du SQL.

Bien qu'elles soient abondamment utilisées, il n'existe pas encore de méthode officielle ni de règles définies pour qu'une base de données orientée colonnes soit qualifiée de qualité ou non.

Le principe d'une base de données colonnes consiste dans leur stockage par colonne et non par ligne. C'est-à-dire que dans une base de données orientée ligne (SGBD classique) on stocke les données de façon à favoriser les lignes en regroupant toutes les colonnes d'une même ligne ensemble. Les bases de données orientée colonnes quant à elles vont stocker les données de façon à ce que toute les données d'une même colonne soient stockées ensemble. 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 donc 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 tout simplement pas. Les bases de données colonnes ont été pensées pour pouvoir stocker plusieurs millions de colonnes et se relèvent donc parfaites pour gérer le stockage dit « one-to-many »

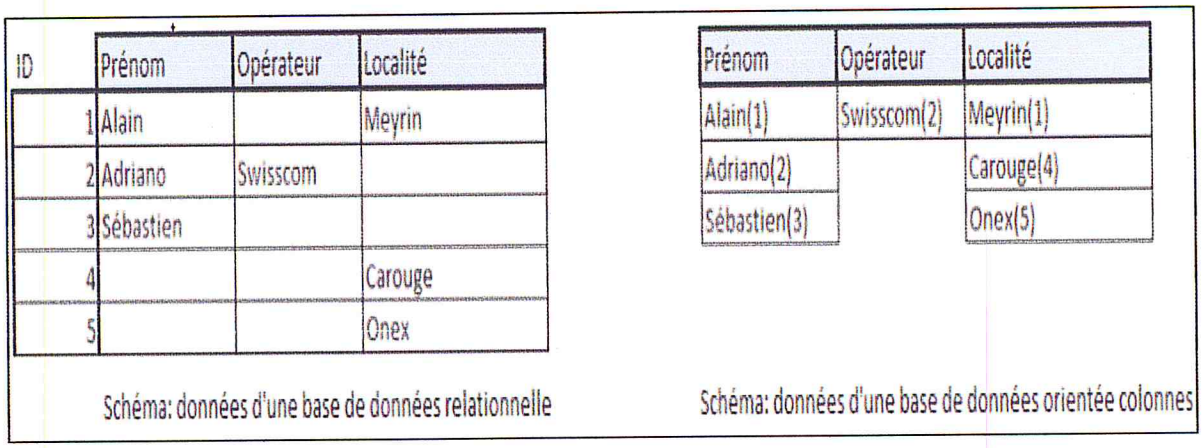


Figure 7 : BD relationnelles vs BD orientées colonnes. [4]

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 pour commencer les familles de colonnes, qui est 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 [4].

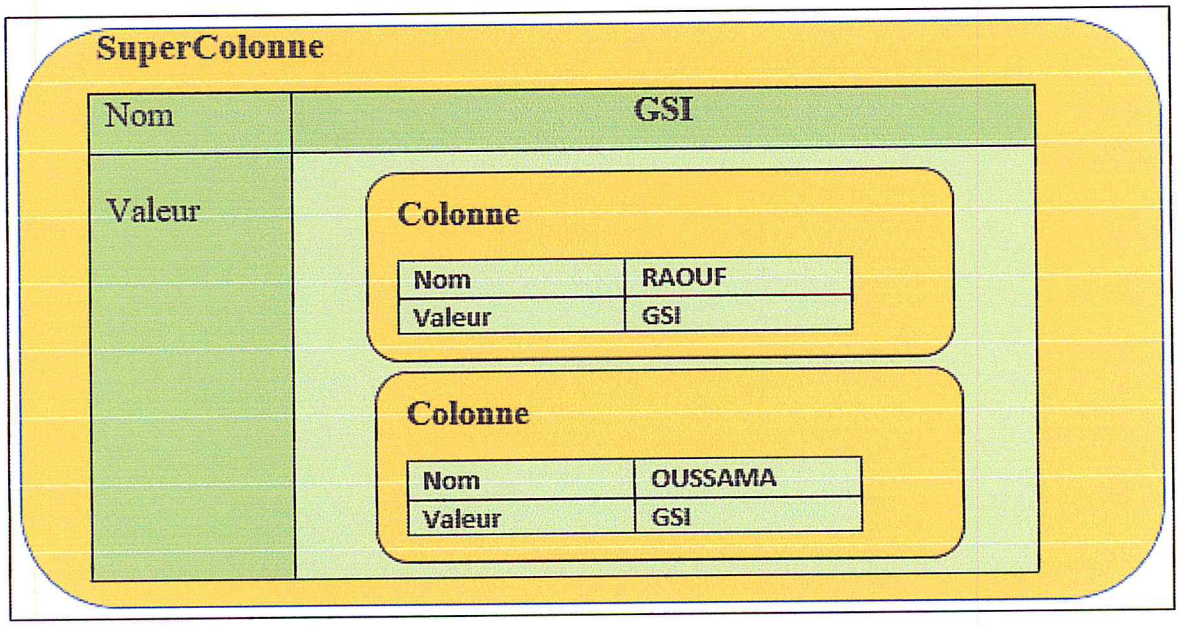


Figure 8 : Structure d'une BD orientée colonnes. [4]

Les bases de données orientées colonnes comportent des avantages considérables. Leur capacité de stockage est accrue, en partie grâce à l'espace économisé sur les valeurs « null »,

comme déjà vu plus haut. Leur compression est significative pour autant que les données des colonnes se ressemblent fortement. Afin d'éviter la décompression à chaque interrogation des données, de plus en plus de SGBD orientées.

Elles assurent également une rapidité remarquable d'accès aux données. Si l'on souhaite récupérer un gros volume de donnés d'un attribut spécifique d'un enregistrement, il est préférable de récupérer les informations de toute cette colonne plutôt que de parcourir les lignes une à une et de prendre uniquement cette information. Cependant, l'utilisation de ces bases de données n'est réellement qu'efficace dans un contexte « BigData », pour les grands volumes de données de même type, et dont les données se ressemblent. De plus, le système de requêtes minimalistes comme cité plus haut a son prix et les différentes solutions possibles pour interroger les données sont très limitées.

4.4 Les bases de données orientées graphe

Bien que les bases de données de type clé-valeur, colonne, ou document tirent leur principal avantage de la performance du traitement de données, les bases de données orientées graphe permettent de résoudre des problèmes très complexes qu'une base de données relationnelle serait incapable de faire. Les réseaux sociaux (Facebook, Twitter,etc), où des millions d'utilisateurs sont reliés de différentes manières, constituent un bon exemple : amis, fans, famille etc. Le défi ici n'est pas le nombre d'élément à gérer, mais le nombre de relations qu'il peut y avoir entre tous ces éléments. En effet, il y a potentiellement n^2 relations à stocker pour n éléments. Même s'il existe des solutions comme les jointures, les bases de données relationnelles se confrontent très vite à des problèmes de performances ainsi que des problèmes de complexité dans l'élaboration des requêtes. L'approche par graphes devient donc inévitable pour les réseaux sociaux tels que Facebook ou Twitter.

Les bases de données graphe sont également utilisées pour la gestion d'importantes structures informatiques. Elles permettent également l'élaboration de liens entre les divers intérêts que pourraient avoir un internaute, afin de pouvoir lui proposer des produits susceptibles de l'intéresser. Ainsi, les pubs s'affichant sur Facebook sont très souvent en relation avec les recherches effectuées sur Google, et les propositions d'achats de sites de vente en ligne tels que Ebay et Amazon sont en relation avec des achats déjà effectués (par exemple : les personnes ayant acheté ce produit ont également acheté …).

Comme son nom l'indique, ces bases de données reposent sur la théorie des graphes, avec trois éléments à retenir :

- Un objet (dans le contexte de Facebook nous allons dire que c'est un utilisateur) sera appelé un Nœud.
- Deux objets peuvent être reliés entre eux (comme une relation d'amitié).
- Chaque objet peut avoir un certain nombre d'attributs (statut social, prénom, nom etc.)

Les données sont donc stockées sur chaque nœud, lui-même organisé par des relations. A partir de là, il sera nettement plus aisé d'effectuer des opérations qui auraient été très complexes et lourdes dans un univers relationnel [4].

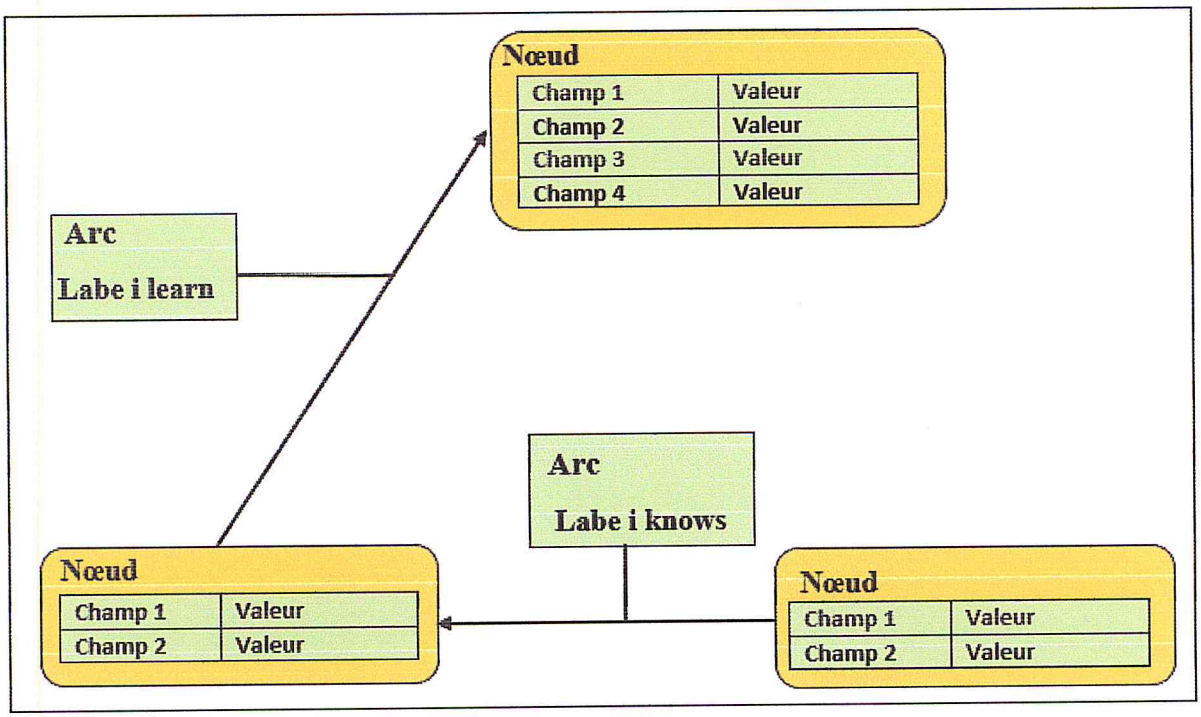


Figure 9 : Structure d'une BD orientée graphe. [4]

L'implémentation d'une base de données graphe peut varier selon les besoins, ou les propriétés à mettre en avant suivant l'utilisation. Certaines se basent sur des orientations colonne, d'autres sur l'enregistrement clé-valeur ou sur une combinaison de plusieurs d'entre elles. HyperGraphDB, Neo4j, FlockDB, BigData sont quelques noms de bases de données orientées graphe.

Le point positif de ce type de bases est qu'elles sont parfaitement adaptées à la gestion des données relationnelles même dans un contexte de « BigData ». De plus, leur architecture étant

modelable, elles peuvent être adaptées selon les besoins rencontrés. Cependant, cette architecture est très limitée dans d'autre cas plus classiques car très spécifique aux graphes.

4.5 Comparaison Entre les Types de Bases de Données NOSQL

Il en existe 4 types distincts des bases de données NoSQL qui s'utilisent différemment et qui se prêtent mieux selon le type de données que l'on souhaite y stocker. Le tableau suivant illustre entre ces quatre types.

Modèle	Performance	Évolutivité	Flexibilité	Complexité	Fonctionnalités
Clef/Valeur	+++	+++	+++	+++	Variable
Orienté Colonnes	+++	+++	+	+	Minimale
Orienté Document	+++	+	+++	+	Variable
Orienté Graphes	+	+	+++	-	Théorie des Graphes

Figure 10 Comparaison entre les types de base de données NoSQL

5. Quelques bases de données NoSQL

A ce jour-là, on trouve une grande variété des BD NoSQL, qui sont open-source ou privée, gratuites ou payantes et qui appartiennent aux différents modèles de données, ils sont disponibles sous différents environnements (Windows ou linux …), facile à installer et à utiliser. Le tableau suivant illustre quelques bases de données NoSQL :

Nom	Type de licence	Modèle de données	Langage d'interrogation	Description
Redis	Licence BSD, supporté commercialement par Redis labs	Clé/Valeur	API pour la plupart des langages de programmation	Redis est un système de gestion de base de données clé-valeur scalable, très hautes performances, écrit avec le langage de programmation C ANSI et distribué sous licence BSD. Il fait partie de la mouvance NoSQL et vise à fournir les performances les plus élevées possibles.
Riak	Apache open-source project, sponsorisé par Basho Technologies	Clé/Valeur	API pour Java, Ruby, Python.	Riak est une base de données distribué clé/valeur NoSQL qui offre une haute disponibilité, tolérance aux pannes, simplicité opérationnelle et évolutivité. Riak met en œuvre les principes du Dynamo d'Amazon avec une forte influence du théorème CAP.
Cassandra	Open-source under Apache license	Orientée colonne	Cassandra Query Language (CQL)	Apache Cassandra est un système de gestion de base de données (SGBD) de type NoSQL conçu pour gérer des quantités massives de données sur un grand nombre de serveurs, assurant une haute disponibilité en éliminant

				les points individuels de défaillance. Il permet une répartition robuste sur plusieurs centres de données, avec une réplication asynchrone sans master et une faible latence pour les opérations de tous les clients
CouchDB	Apache licensed, Couchbase, Inc.	Orientée document	N1QL, API Java et 8 autres langages	Apache CouchDB est un système de gestion de base de données orienté documents, écrit en langage Erlang et distribué sous licence Apache.
Neo4J	GPL/AGPL, par Neo Technology	Orientée graphe	Cypher	Neo4j est un système de gestion de base de données au code source libre orienté graphes, développé en Java par la société suédo-américaine Neo technology.
DynamoDB	Propriété de Amazon	Clé/Valeur	API java, .NET, PHP	Amazon DynamoDB est un service de base de données NoSQL propriétaire entièrement géré qui est offert par Amazon.com dans le cadre du portefeuille Amazon Web Services.
HBase	Apache license, sponsorisé par Cloudera, MapR, Hortonworks, et plus d'autres	Orientée colonnes	API Java	HBase est un système de gestion de base de données non-relationnelles distribué, écrit en Java, disposant d'un

				stockage structuré pour les grandes tables.
MongoDB	GNU AGPL, Apache licensed drivers. Supporté commercialement par MongoDB, Inc.	Orientée document	JavaScript	MongoDB est un système de gestion de base de données orientée documents, répartissable sur un nombre quelconque d'ordinateurs et ne nécessitant pas de schéma prédéfini des données. Il est écrit en C++. Le serveur et les outils sont distribués sous licence AGPL, les pilotes sous licence Apache et la documentation sous licence Creative Commons. Il fait partie de la mouvance NoSQL.

Figure 11 les bases de données nosql les plus répandus.

6. NoSQL vs BDR

Choix de NOSQL en opposition aux bases de données relationnelles conduits par les contraintes du marché et les besoins techniques :

- **BigData** : Adaptation des BD NOSQL au BigData
  - Vitesse (Velocity) : beaucoup de données qui arrivent rapidement, à partir de plusieurs sources.
  - Variété (Variety) : données structurées, semi-structurées ou non-structurées.
  - Volume : données très nombreuses (To et Po).
  - Complexité (Complexity) : données stockées et gérées à plusieurs endroits et data centers.
- **Disponibilité continue des données :**

Le manque de disponibilité peut être fatal pour une entreprise, les BD NOSQL utilisent une architecture hautement distribuée à travers les data centers, et le cloud qui facilitent la mise à jour ou la modification sans déconnecter la base et offrent une disponibilité continue.

- **Indépendance de l'emplacement :** Possibilité de consulter et modifier une BD sans savoir où est-ce que ces opérations ont réellement lieu.
- **Modèles de données flexibles :**
  - Dans le modèle relationnel, les relations entre les tables sont prédéfinies, fixes et organisées dans un schéma strict et uniforme.
    - Problèmes d'évolutivité et de performance en gérant de grands volumes de données.
  - Les BD NOSQL peuvent accepter tout type de données (structurées, semi-structurées ou non-structurées) plus facilement.
- **Business Intelligence et Analyse :**
  - Capacité des bases NOSQL à exploiter les données collectées pour dériver des idées.
  - Extraction d'informations décisionnelles à partir d'un grand volume de données, difficile à obtenir avec des bases relationnelles.
  - Bases NoSQL permettent le stockage et la gestion des données des applications métier, et fournissent une possibilité de comprendre les données complexes, et de prendre des décisions.

Tableau Récapitulatif de Comparaison entre NoSQL et BDR :

BDR	NoSQL
Consistance forte	Consistance éventuelle
Grande quantité de données	Enorme quantité de données
Évolutivité possible	Evolutivité facile
SQL	Map-Reduce
Bonne disponibilité	Très haute disponibilité

Figure 12 comparaison entre les BDR et les BD NoSQL

7. Travaux Antérieurs

La dernière décennie a enregistré une énorme quantité de transition de données qui se déroule sur Internet. Le monde d'aujourd'hui est relié par des sites de réseaux sociaux. De tels sites ont une grande taille de bases de données. Notre recherche dans ce domaine montre que de nombreuses approches fournissent des outils ou des utilitaires pour la conversion de base de données ont ete propose :

Mital Potey et d'autres chercheurs du département Computer Engg de l'inde en [6] ils ont proposé un utilitaire en ligne qui aidera à convertir les bases de données relationnelles de type MySQL a des bases de données NoSQL telles que MongoDB, et qui fonctionnera sur le nuage d'Amazon, tout en utilisant Amazon Cloud Services qui offre d'autres services tels que l'évolutivité, l'élasticité et la haute disponibilité. Toutefois cette solution n'est pas encore intégrée dans les services d'amazon.

Mohamed Hanine et d'autres chercheurs du département Computer Science du Maroc en [7], dans leur article ils ont fait une étude pour la réalisation d'une méthodologie qui facilite la migration d'une base de données relationnel (MySQL) vers une base de données NoSQL (MongoDB), en utilisant un processus appelée ETL (Extract, transform and load). À titre d'illustration, ils ont implémenté un prototype de logiciel développée sur JAVA qui repose sur la méthodologie proposée, mais jusqu'à encore une autre fois cette solution n'est pas disponible.

Gansen Zhao et d'autres chercheurs de l'université School of Computer Science en [8] dans leur article ils ont proposé un modèle général de conversion de schéma pour transformer les bases de données SQL en bases de données NoSQL, en utilisant un algorithme de transformation de graphe, afin d'assurer l'exactitude de la conversion de schéma.

JackHare [9] est un framework pour la traduction du SQL vers le NoSQL avec l'utilisation de l'architecture MapReduce. Il nous donne une approche pour exploiter HBase (système de gestion de base de données non-relationnelles distribué, disposant d'un stockage structuré pour les grandes tables). Parmi ces solutions il transforme les données depuis les BDD relationnelle vers HBase. Il fournit un modèle de conversion concret qui stocke toutes les tables d'une base de données relationnelle dans une seule table HBase. En utilisant trois règles de transformation, JackHare transforme le schéma des BDD relationelle vers un schéma HBase par la génération un mappage de schéma.

8. Classement de quelques SGBD (selon DB-Engine)

DB-Engine ne prend pas en compte le nombre d'installations des bases de données. Il ne s'agit donc pas de mesure de bases installées mais il se présente comme un indicateur de tendance. À partir d'une formule interne combinant ces paramètres, le BD-Engines Ranking a pu attribuer des scores aux différents systèmes de gestion de base de données. À l'issue du classement, les SGBDR dominent le podium : Oracle se positionne comme le système le plus populaire de l'année 2016. Il est suivi par MySQL en 2-ème position, Microsoft SQL Server est en 3-ème position, MongoDB en 4eme position est le premier SGBD NoSQL, cassandra en 7eme position et Redis en 10-ème position.

Le tableau suivant illustre le classement des dix premières bases de données les plus utilisée sen 2016 [w1] :

299 systems in ranking, March 2016

Rank			DBMS	Database Model	Score		
Mar 2016	Feb 2016	Mar 2015			Mar 2016	Feb 2016	Mar 2015
1.	1.	1.	Oracle	Relational DBMS	1472.01	-4.13	+2.93
2.	2.	2.	MySQL	Relational DBMS	1347.71	+26.59	+86.62
3.	3.	3.	Microsoft SQL Server	Relational DBMS	1136.49	-13.73	-28.31
4.	4.	4.	MongoDB	Document store	305.33	-0.27	+30.32
5.	5.	5.	PostgreSQL	Relational DBMS	299.62	+10.97	+35.19
6.	6.	6.	DB2	Relational DBMS	187.94	-6.55	-10.91
7.	7.	7.	Microsoft Access	Relational DBMS	135.03	+1.95	-6.66
8.	8.	8.	Cassandra	Wide column store	130.33	-1.43	+23.02
9.	↑ 10.	↑ 10.	Redis	Key-value store	106.22	+4.14	+9.17
10.	↓ 9.	↓ 9.	SQLite	Relational DBMS	105.77	-1.01	+4.06

Figure 13 Classement des dix meilleures bases de données par DB-Engines (Mars 2016)

8.1 Synthèse

En se basant sur les SGBD NoSQL qui convient à notre projet, deux entre eux sortent du lot : MongoDB et Redis, mais en ajoutant d'autres critères comme la maturité de la solution, la popularité, la documentation fournie, la grandeur de la communauté pour garantir la durabilité, la facilité d'implémentation et d'utilisation, le rythme d'évolution, le dynamisme, notre choix sera très clairement et sans aucun doute pour Mongodb.

Notre choix de MongoDB comme une Base de Données cible repose sur plusieurs critères parmi ces derniers :

- Il est classé le quatrième des BD les plus populaires et le premier dans les BD NoSQL.
- Il est utilisé par les plus grand société tel que : Facebook, CISCO, Adobe, eBay …etc.
- Il Support plusieurs langage de programmation tel que Java, PHP, Python … etc.

Il possède plusieurs autres caractéristiques que nous allons les voir dans le prochain chapitre, et qui ont fait de lui notre base de données cible dans le monde de NoSQL.

9. Conclusion

Les bases de données sont considérées comme les éléments les plus importants dans le monde informatique. Le modèle relationnel est imposé comme une norme et il est très répandu grâce à son efficacité et sa robustesse issues par les contraintes imposées par ce modèle. Ces mêmes principes qui constituent la force de ce modèle, le rendent moins performant et moins réactif dans un environnement distribué car elles empêchent les données d'être distribuées efficacement dans les nœuds constituants l'environnement distribué à grande échelle. De ce fait, et face à ces contraintes, des nouvelles systèmes de gestion de données nommés «NoSQL» viennent occuper leurs places en proposant des alternatives au modèle relationnel. Le NoSQL est une nouvelle approche de stockage et de manipulation de données, dont la plupart ont vu le jour pendant les 4 ou 5 dernières années. Elles ont su se démocratiser grâce leur large utilisation par les grandes entreprises du domaine informatique, qui ont adopté ces nouvelles solutions pour répondre à leurs besoins d'évolutivité. Un administrateur doit par conséquent examiner soigneusement le type de bases de données le mieux adapté à ses besoins, car un mauvais choix pourrait avoir de très lourdes conséquences. D'où notre choix d'une base de données cible dans le NoSQL pour la conception de notre moteur de mapping qui nous allons voir dans le chapitre suivant, c'est imposer sur le MongoDB. Ce choix a été fait après les études et les critères illustre précédemment.

Chapitre 2 : Conception du Système de Mapping

1. Introduction

Les bases de données NoSQL ont gagné en popularité ces dernières années parce qu'elles ont pu venir au bout des nouveaux problèmes que les bases de données relationnelles étaient incapables de résoudre. On assiste ici à une explosion des types des systèmes NoSQL disponibles sur le marché, avec chacune ses propres propriétés et cas d'utilisations.

Après le choix du MongoDB comme base de données cible dans le NoSQL pour notre solution proposée en se basant sur plusieurs critères cités dans le chapitre précédent. Nous allons dans ce chapitre présenter la base de données MongoDB et ses différentes caractéristiques et architectures puis nous illustrons une table de mapping entre le SQL et le MongoDB par la suite nous allons détailler l'architecture adoptée pour notre solution proposée ainsi que les différents scenarii de traduction des requêtes SQL vers MongoDB.

2. MongoDB

MongoDB est l'un des systèmes de gestion de données NoSQL les plus populaires. Développé par la société new-yorkaise MongoDB Inc., il est disponible depuis 2009, la dernière version disponible jusqu'à aujourd'hui est la 3.4.3

2.1 Présentation de MongoDB

Développé depuis 2007 par 10gen (une société de logiciel), MongoDB est un système de gestion de base de données orientée document. Ecrit en C++ et distribué sous licence AGPL (licence libre), elle est très adaptée aux applications web.

MongoDB a été adopté par plusieurs grands noms de l'informatique, tels que Foursquare, SAP, ou bien même GitHub.

Manipulant des documents au format BSON (Binary JSON), un dérivé de JSON binaire plus axé sur la performance, l'utilisation d'un pilote est donc nécessaire pour communiquer avec une base MongoDB. Fonctionnant comme une architecture

Distribuée centralisée, il réplique les données sur plusieurs serveurs avec le principe de maître-esclave, permettant ainsi une plus grande tolérance aux pannes. La répartition et la duplication

de document est faite de sorte que les documents les plus demandés soient sur le même serveur et que celui-ci soit dupliqué un nombre de fois suffisant.

Par sa simplicité d'utilisation du point de vue de développement client, ainsi que ces performances remarquables, MongoDB est l'une de base de données orientées document la plus utilisée. [4]

2.2 Caractéristiques du MongoDB

MongoDB 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.

MongoDB Inc. le définit comme un SGBD orienté documents. Le terme « documents » ne signifie pas des documents binaires (image ou fichier son, par exemple), mais une représentation structurée, compréhensible par MongoDB, d'une donnée complexe. Plus simplement, on stocke la représentation des données comprise par MongoDB. Ce format est appelé BSON (Binary Serialized Document Notation ou Binary JSON). Pour l'utilisateur, la structure visible est du JSON (JavaScript Object Notation, un format de données textuel dérivé de la syntaxe d'expression des objets en JavaScript).

Voici quelques caractéristiques qui avantage MongoDB des autres BDD NoSQL :

- Stockage/récupération des données sous forme de documents Document (format BSON : Binary JSON).
  - JSON partout, y compris dans les requêtes.
- Gestion des Index riche et complète.
  - Tout comme dans un SGBDR.
  - Très efficace (B-Tree).
- Puissant langage de requête.
- Réplication et haute disponibilité.
- Architecture MapReduce intégrée.
- Auto-Sharding (scaling horizontal).
  - Les collections volumineuses peuvent être divisées et distribuées sur plusieurs "shards" (eux-mêmes situés sur plusieurs machines). "Auto-balancing" intégré.
- Mises à jour des données très rapides.
  - Opérations de mises à jour atomiques.

- Mises à jour "in place".

- Et aussi : facile à installer, très bonne documentation, large communauté d'utilisateurs.

2.3 Documents sous MongoDB

MongoDB est une base documentaire dans laquelle les documents sont regroupés sous forme de collections, les collections étant l'équivalent des tables du SQL. Il est possible de représenter chaque document au format JSON (MongoDB utilise une variante binaire plus compacte de JSON nommé BSON pour son stockage interne). Chaque document dispose d'une clé unique permettant de l'identifier dans la collection. [2]

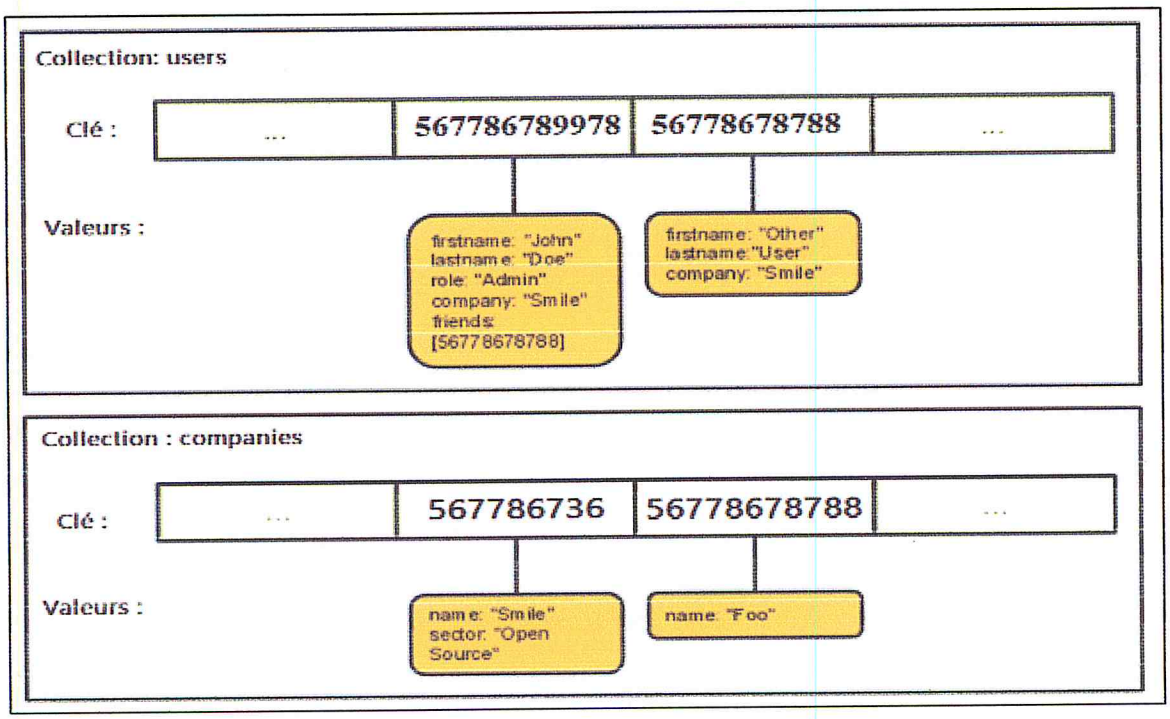


Figure 14 : Example d'un Documents sous MongoDB. [2]

2.4 Architecture

MongoDB possède différents modes de fonctionnement :

- Single.
- Replication
  - Master / Slave.
  - Replica Set
- Sharding.

2.4.1 Mode Single

Le mode Single (Serveur seul), qui ne sert à faire fonctionner une base de données que sur un seul serveur. Dans ce mode mettant en jeu un seul serveur, un seul processus nommé mongod est utilisé et traite directement les données issues des requêtes du client. [5]

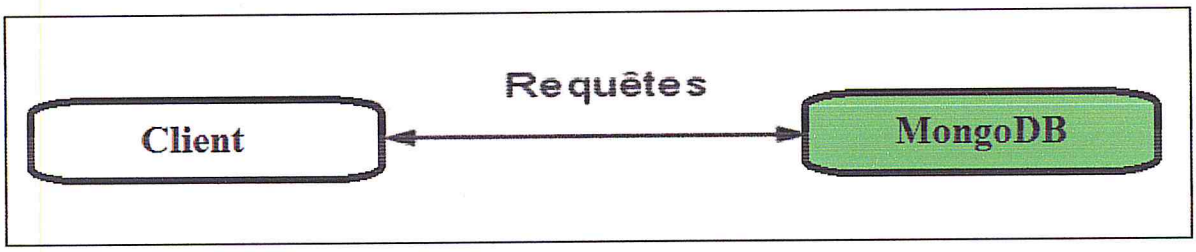


Figure 15 : MongoDB Serveur seul

2.4.2 Réplication

La réplication est découpée en deux modes, alors que le Sharding se montre plus comme un méta-mode car il utilise l'une des deux méthodes de réplication comme couche sous-jacente.

2.4.2.1 Réplication Maître/ Esclave

Le mode Master / Slave bien connu dans l'informatique, n'apporte rien de plus au sein de MongoDB. Un serveur officie en tant que maître et s'occupe des demandes des clients. A côté de cela, il s'occupe de répliquer les données sur le serveur esclave de façon asynchrone. L'esclave est présent pour prendre la place du maître si ce dernier venait à tomber. L'avantage premier de cette structure permet de garantir une forte cohérence des données, car seul le maître s'occupe des clients. Aucune opération n'est faite sur l'esclave, hormis quand le maître envoie la mise à jour. [2]

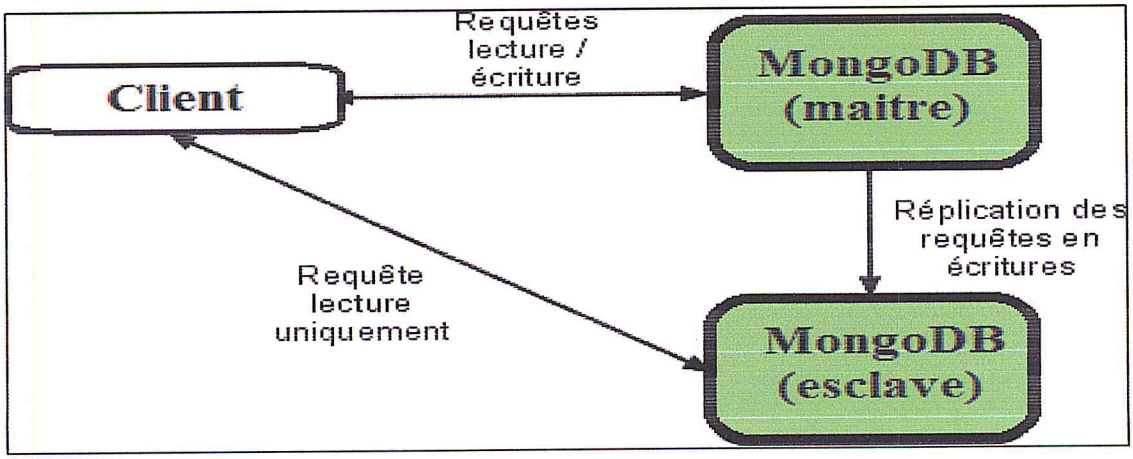


Figure 16 : MongoDB maitre /esclave

2.4.2.2 Réplication par Replica Set

Il s'agit d'un mode de réplication plus avancé que le mode maître / esclave présenté ci-dessus, ce mode permet d'éviter la présence d'un point de défaillance unique au sein de l'infrastructure par la méthode suivante :

- On ajoute n serveurs au Replica Set : Chaque serveur dispose d'une priorité.
- Un des serveurs est considéré comme le nœud primaire : On peut voir le rôle de nœud primaire comme celui du serveur maître dans le modèle maître / esclave.
- En cas de défaillance du nœud primaire, un nouveau nœud au sein du Replica Set est choisi et devient nœud primaire. [4]

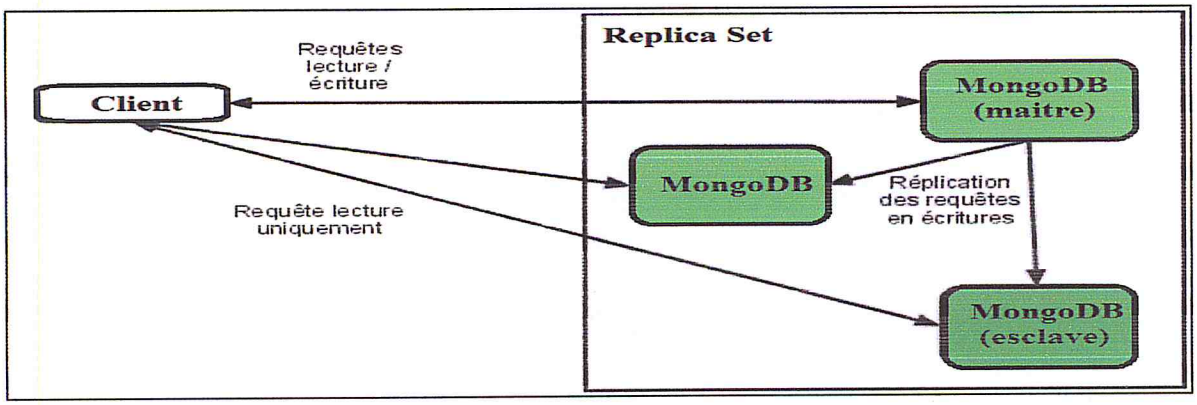


Figure 17 : Réplication par Replica Set

2.4.3 Sharding (Partitionnement des donnés via sharding)

Le Sharding est une surcouche qui est basée sur du Master / Slave ou du Replica Sets. Sharding sert à partager les données entre plusieurs Shard, chaque Shard devant stocker une partie des données.

- **Les Shards :** Ce sont un groupe de serveurs en mode Master / Slave ou Réplica Sets.
- **Les mongos :** Ce sont des serveurs qui savent quelles données se trouvent dans quel Shard et leur donnent des ordres (lecture, écriture).
- **Les Config Servers :** Ils connaissent l'emplacement de chaque donnée et en informent les mongos. De plus, ils organisent la structure des données entre les Shards.

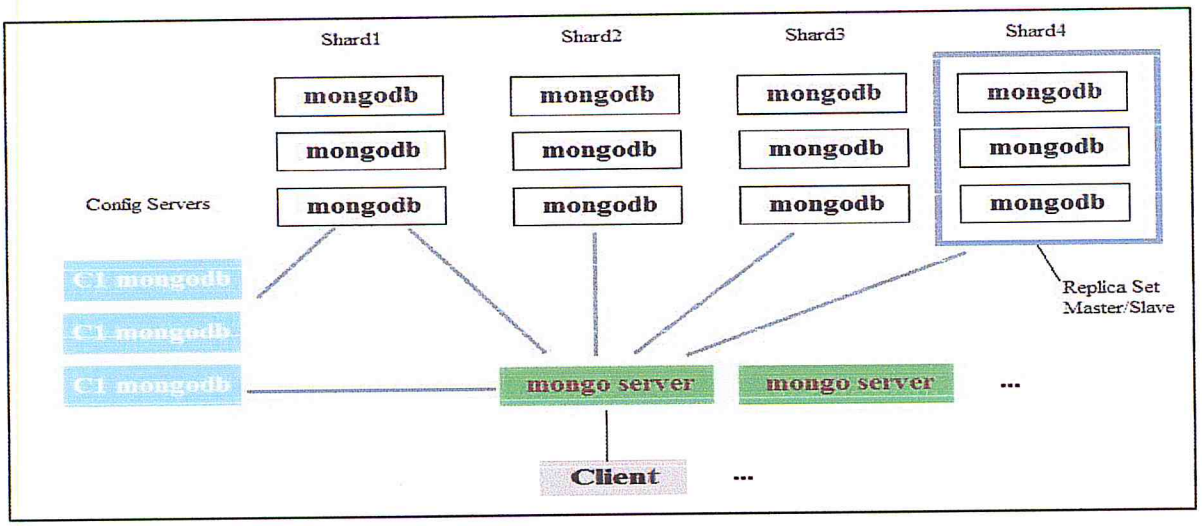


Figure 18 : Partitionnement des donnés via sharding.

2.5 Outils de développement

2.5.1 Stabilité de l'API

MongoDB est un outil mature dont l'API évolue peu à ce jour. Deux modes d'accès principaux peuvent être utilisés :

- Driver spécifique au langage de programmation permettant une communication directement en BSON.
- Une API REST disponible sous forme d'un serveur HTTP embarqué optionnel.

2.5.2 Langages supportés

Mongo supporte officiellement un grand nombre de langage au travers de drivers. Pour les autres langages, on pourra compter sur un support communautaire assez vaste. Enfin, certains frameworks commencent à implémenter un support de Mongo (sous forme de plugins).

2.5.3 Cohérence des données

Les écritures étant toujours effectuées sur la même machine, la cohérence en écriture est assurée sans aucun problème.

En lecture, la cohérence des données est assurée en partie par le développeur qui peut choisir d'accepter de lire sur un serveur esclave, quitte à ce que celuici n'ait pas pu répliquer l'ensemble des données écrites depuis le serveur maître, ou sur le serveur maître.

2.5.4 Transactions

On pourra néanmoins s'appuyer sur un certain nombre d'opérations atomiques et donc transactionnelles par définition dont on peut trouver la liste sur le site de mongodb.org10.

2.5.5 Autres considérations

A l'image de la plupart des solutions NoSQL, MongoDB propose une implémentation de MapReduce, utilisable en Javascript.

Par ailleurs, Mongo implémente un système de fichiers distribués nommé GridFS permettant le stockage de fichiers volumineux (pièce jointes, …).

2.6 Outil de production

2.6.1 Scalabilité

De par sa conception, Mongo est scalable selon deux modes :

- Scalabilité en lecture : celle -ci peut être obtenue simplement en utilisant les replica set.
- Scalabilité en écriture : celle- ci peut être obtenue via les mécanismes de sharding.

2.6.2 Tolérance à la panne

Les mécanismes de replica set permettent au système de résister face à la disparition de l'un de ces nœuds de stockage. Par ailleurs il est possible de placer les nœuds d'un replica set dans plusieurs datacenters (par exemple en assignant au nœud du datacenter de secours, une priorité inférieure) pour obtenir un système qui supporte la perte complète d'un datacenter.

Le système est légèrement plus centralisé que certains autres systèmes SQL lorsqu'on utilise le sharding (présence des nœuds de routage) mais ceux-ci peuvent être dupliqués sans problème, même sur plusieurs datacenters. Cela ne constitue donc pas non plus un point de défaillance unique.

2.6.3 Outils de monitoring / Administration

Outre le fait que MongoDB dispose de capacités de monitoring basiques embarquées, de nombreux plugins existent permettant l'intégration de MongoDB dans les outils d'analyse les plus populaires du marché comme Munin, Ganglia, Cacti, Zabbix ou Nagios.

Il existe par ailleurs un certain nombre d'outils SaaS de monitoring intégrant des modules spécifiques pour MongoDB (Server Density, CloudKick, AppFirst, …).

2.6.4 Gestion des sauvegardes

MongoDB dispose d'un outil de sauvegarde nommé mongodump. Celui-ci procède à un export de la base de données qui peut, par la suite, être restauré via mongorestore.

Cet outil n'est pas adapté à des configurations traitant, un grand volume de données. En général, celles-ci utilisent des shards contenant un replica set. La technique habituelle pour sauvegarder de telles données est la suivante :

- On stoppe les fonctionnalités de répartition du sharding durant l'opération de sauvegarde.
- On stoppe un des serveurs de configuration (les autres continuent à fonctionner pendant ce temps) puis on sauvegarde ces fichiers.
- Pour chaque shard, on stoppe l'un des serveurs du replica set que l'on sauvegarde.
- On redémarre le serveur de configuration ainsi que les nœuds éteints dans les shards.
- On redémarre les fonctionnalités de répartition du sharding.

3. Terminologie SQL versus NoSQL

Si on veut apprendre ce qu'une base de données NoSQL (MongoDB) on devrait apprendre aussi les concepts de base fondamentaux de mongodb tel que les documents, collections, bases de données, ici nous introduisons un par un.

Le tableau suivant aidera facilement à comprendre certains concepts de mongodb [w1,w2] :([1])

[1] Certaine opérateurs et fonctions cité dans MongoDB n'ont pas la même application dans le SQL mais elles ont le même principe car dans le SQL les opérations s'appliquent sur les tables, les colonnes et les lignes mais en MongoDB elles s'appliquent sur des collections et les documents.

SQL Terminologie / Concepts	Terme MongoDB / notion	Explication / Description
Base de données	Base de données	Base de données
Table	Collection	Table de base de données / Collections
Ligne	Document	L'enregistrement des données de ligne / Documentation
Colonne	Champ	Champ de données / Domain
Index	Index	Index
Jointures de table		Le tableau joint, MongoDB ne fonctionne pas
Clé primaire	Clé primaire	Clé primaire, MongoDB champ automatiquement _id comme la clé primaire

Figure 19 terminologie SQL vs MongoDB

Le tableau suivant illustre les différents types de données couramment utilisés en MongoDB.

Types de données	Description
Chaîne	String. Stockage de données couramment utilisés types de données. En MongoDB, chaîne UTF-8 codé est légal.
Entier	Les valeurs entières. Il est utilisé pour stocker des valeurs. Selon le serveur que vous utilisez, il peut être divisé en 32 bits ou 64 bits.
Boolean	Valeur booléenne. Pour stocker des valeurs booléennes (vrai / faux).
Double	Double-precision valeur à virgule flottante. Pour le stockage de valeurs à virgule flottante.
Clés Min / Max	La valeur minimale et un élément valeur BSON (binaire JSON) et la valeur la plus élevée du rapport relatif.
Arrays	Pour tableau ou liste ou stocker plusieurs valeurs pour une clé.
Timestamp	Timestamp. Modifier ou ajouter des enregistrements documentent le moment précis.
Objet	Pour les documents incorporés.
Null	Il crée une valeur vide.

Symbole	Ce type de données est sensiblement égal au type de chaîne, mais la différence est qu'il est généralement pour l'utilisation de symboles spéciaux langue dactylographiées.
Date	Date Heure. Format de l'heure UNIX utilisé pour stocker la date ou l'heure actuelle. Vous pouvez spécifier votre propre date et l'heure : Date objet est créé, les informations de date d'arrivée.
Object ID	ID d'objet. ID utilisé pour créer le document.
Données binaires	Les données binaires. Pour le stockage des données binaires.
Code	Type de Tag. Code JavaScript est utilisé pour stocker des documents.
Expression régulière	Régulier Type d'expression. Pour le stockage de l'expression régulière.

Figure 20 types de données en MongoDB

A. MongoDB CRUD Opérations

Pour assurer les opérations *CRUD* (Create, Read, Update, Delete) MongoDB dispose de quatre commandes principales :

SQL	MongoDB	Explication / Description
CREATE TABLE	**db.createCollection()**	Création d'une nouvelle table (collection)
INSERT INTO **insert**	**db.collection.insert()** **insert()**	Insérer un document dans une collection (insérer une valeur dans table)
SELECT FROM **select**	**db.collection.find()** **find()**	L'opération de lecture d'un document d'une collection
UPDATE SET **update**	**db.collection.update()** **update()**	L'opération de modification d'un document existant d'une collection
DELETE FROM **delete**	**db.collection.delete()** **delete()**	L'opération de suppression d'un document d'une collection

Figure 21 Opérations CRUD en SQL et en MongoDB

B. Les Opérateur (Fonction) d'agrégation MongoDB

Les opérations d'agrégation permettent d'effectuer des opérations sur un ensemble d'enregistrement. Les opérations d'agrégation groupent des valeurs à partir de plusieurs

documents et peuvent effectuer diverses opérations sur les données groupées pour obtenir un résultat unique. MongoDB fournit trois façons d'effectuer l'agrégation : le pipeline d'agrégation, la fonction de map-reduce et les méthodes de single purpose aggregation.

Dans le tableau suivant nous illustrons certain mots clé en SQL et leurs correspondances dans MongoDB :

SQL	MongoDB	Explication / Description
WHERE	**$match**	La commande d'agrégation **$match** dans ce cas permet de compter le nombre d'enregistrement dans un document.
GROUP BY	**$group**	La commande **$group** est utilisée pour regrouper plusieurs résultats et utiliser une fonction de totaux sur un groupe de résultat.
HAVING	**$match**	La condition **$match** est presque similaire à WHERE à la seule différence celle-là est utiliser avec d'autres fonctions pour permettre le filtrage des documents.
SELECT	**$project**	Consiste à lire des données issues de la base de données.
ORDER BY	**$sort**	La commande **$sort** permet de trier les lignes dans un résultat d'une requête.
LIMIT	**$limit**	La clause **$limit** est à utiliser dans une requête pour spécifier le nombre maximum de résultats que l'ont souhaite obtenir.
SUM ()	**$sum**	La commande d'agrégation **$sum** permet de calculer la somme totale d'un document contenant des valeurs.
COUNT ()	**$sum**	La commande d'agrégation **$sum** elle peut être utiliser pour compter le nombre d'enregistrement dans un document.

join	**$lookup**	Effectue une jointure externe dans une collection de la même base de données pour filtrer dans les documents de la collection pour le traitement.
AVG()	**$avg**	Retourne la moyenne de "champ" sur l'ensemble des enregistrements.
MIN()	**$min**	Retourne la plus petite valeur parmis la liste des enregistrements sélectionnés.
MAX()	**$max**	Retourne la plus grande valeur de parmis la liste des enregistrement sélectionnés.
RAND()	**$sample**	Sélectionne de manière aléatoire le nombre de documents spécifié.

Figure 22 table de mapping (mots réservés)

C. Les Opérateur de Comparaison

Les opérateurs de comparaison testent si deux expressions sont identiques.

SQL	MongoDB	Explication / Description
=	**$eq**	Egal(e) à.
>	**$gt**	Supérieur à.
>=	**$gte**	Supérieur à ou égal à.
<	**$lt**	Inférieur à (Less than).
<=	**$lte**	Inférieur à ou égal à.
!=	**$ne**	Différent de.
/	**$cmp**	Retourne : 0 si les deux valeurs sont équivalentes, 1 si la première valeur est supérieure à la seconde et -1 si la première valeur est inférieure à la seconde.

Figure 23 : table de mapping (opérateurs de comparaison)

D. Les Opérateur Logique

Les opérateurs logiques testent le caractère vrai ou faux d'une condition.

SQL	MongoDB	Explication / Description
OR	$or	TRUE si l'une ou l'autre expression booléenne est TRUE.
AND	$and	TRUE lorsque les deux expressions sont TRUE.
NOT	$not	NOT inverse une expression.
NOR	$nor	C'est l'opération logique « non ou ».
IN	$in	Détermine si une valeur donnée correspond à la valeur d'une liste ou d'une sous-requête.
NOT IN	$nin	Détermine si une valeur donnée ne correspond pas à la valeur d'une liste ou d'une sous-requête.

Figure 24 : table de mapping (opérateurs logique)

E. Les Opérateurs Arithmétique

Les opérateurs arithmétiques effectuent des opérations mathématiques sur deux expressions d'au moins l'un de ces types est de catégorie numérique.

SQL	MongoDB	Explication / Description
abs()	$abs	Valeur absolue.
+	$add	Addition.
-	$subtract	Soustraction.
/	$divide	Division (la division entière tronque les résultats).
*	$multiply	Multiplication.
ceil()	$ceil	Plus petit entier supérieur à l'argument.
exp()	$exp	Exponentiel.
floor()	$floor	Plus grand entier inférieur à l'argument.
ln()	$ln	Logarithme.
log()	$log	Logarithme base 10
mod(y, x)	$mod	Reste de y/x.
power(a , b)	$pow	a élevé à la puissance b.
sqrt()	$sqrt	Racine carré.
trunc()	$trunc	Tronque vers zéro ou sur un decimal

Figure 25 : table de table de mapping (opérateurs arithmétique)

F. Les Opérateurs de chaînes de caractères

Retourne une chaîne qui est le résultat de la concaténation de plusieurs valeurs de chaîne.

SQL	MongoDB	Explication / Description
CONCAT	$concat	Permis la concaténation de plusieurs valeurs de chaîne.
INSTRB()	$indexOfBytes	Addition.
INSTR()	$indexOfCP	Renvoie la position de la première apparition de la sous-chaîne.
SUBSTRING_INDEX()	$split	Renvoie une sous-chaîne d'une chaîne avant le nombre spécifié d'occurrences du délimiteur.
octet_length()	$strLenBytes	Nombre d'octets en chaîne.
length()	$strLenCP	Nombre de caractères en chaîne.
STRCMP()	$strcasecmp	Est utilisé pour comparer deux chaines de caracteres.
SUBSTRING()	$substr	Permet d'extraire une sous-chaîne de caractères à partir d'une chaîne selon la position des caractères.
SUBSTRB()	$substrBytes	Permet d'extraire une sous-chaîne de caractères à partir d'une chaîne selon la position des caractères en Byte.
SUBSTR()	$substrCP	Permet d'extraire une sous-chaîne de caractères à partir d'une chaîne selon la position des caractères.
LOWER()	$toLower	Permet la conversion d'une chaine de caractère en minuscules.
UPPER()	$toUpper	Permet la conversion d'une chaine de caractère en majuscules.

Figure 26 : table de mapping (opérateurs de chaines de caractères)

G. Les Opérateurs de Date

Il existe une multitude de fonction qui concerne les éléments temporels pour pouvoir lire ou écrire plus facilement des données à une date précise ou à un intervalle de date.

SQL	MongoDB	Explication / Description
DAYOFYEAR()	$dayOfYear	Retourner le jour dans l'année (de 1 à 366).
DAYOFMONTH()	$dayOfMonth	Retourner le jour dans le mois (de 1 à 31).
WEEKDAY()	$dayOfWeek	Déterminer le jour de la semaine à partir d'une date.
YEAR()	$year	Retourne l'année pour une date en tant que numéro.
MONTH()	$month	Extraire le numéro du mois à partir d'une date.

WEEK()	**$week**	Déterminer le numéro de la semaine dans une année, à partir d'une date.
HOUR()	**$hour**	Retourne l'heure d'une date comme un nombre entre 0 et 23.
MINUTE()	**$minute**	Extraire le nombre de minutes d'une heure.
SECOND()	**$second**	Extraire le nombre de secondes d'une minute.
/	**$millisecond**	Retourne les millisecondes d'une date comme un nombre compris entre 0 et 999.
DATE_FORMAT()	**$dateToString**	Retourne la date en tant que chaîne de caractère.
DAYOFWEEK()	**$isoDayOfWeek**	Retourner le jour dans la semaine.
WEEK()	**$isoWeek**	Déterminer le numéro de la semaine dans une année, à partir d'une date
YEARWEEK()	**$isoWeekYear**	Retourne l'année et la semaine à partir d'une date.

Figure 27 table de mapping (opérateurs de date)

H. Les Opérateurs de tableau

SQL	MongoDB	Explication / Description
IFNULL()	**$ifNull**	Si expr1 n'est pas NULL, IFNULL () renvoie expr1; Sinon il renvoie expr2.
CASE	**$switch**	Permet d'utiliser des conditions de type « si / sinon » pour retourner un résultat disponible entre plusieurs possibilités.
IF()	**$cond**	Un opérateur ternaire qui évalue une expression et, en fonction du résultat, renvoie la valeur de l'une des deux autres expressions.
/	**$type**	Retourne le type de données du fichier BSON.
/	**$first**	Retourne une valeur du premier document pour chaque groupe. L'ordre n'est défini que si les documents sont dans un ordre défini.
/	**$last**	Retourne une valeur du dernier document pour chaque groupe. L'ordre n'est défini que si les documents sont dans un ordre défini.

/	**$push**	Retourne un tableau de valeurs d'expression pour chaque groupe.
/	**$literal**	Permet d'extraire une sous-chaîne de caractères à partir d'une chaîne selon la position des caractères.
EXPLAIN	**explain()**	Cette méthode renvoie un document avec le plan de requête et, éventuellement, les statistiques d'exécution.
SET	**$set**	Il est utilisé pour spécifier, mettre à jour ou créer la clé si elle n'existe pas.
/	**$zip**	Pour fusionnez deux listes ensemble.
@var_name	**$let**	Définit les variables à utiliser dans le cadre d'une sous-expression et renvoie le résultat de la sous-expression.
/	**$collStats**	Retourne des statistiques concernant une collection ou un view.
/	**$redact**	Réforme chaque document dans le flux en limitant le contenu de chaque document en fonction des informations stockées dans les documents eux-mêmes.
/	**$bucket**	Classer les documents entrants en groupes, appelés seaux, en fonction d'une expression spécifiée et des limites du seau.
/	**$meta**	Recherche de texte dans métadonnées.
array_cat()	**$concatArrays**	Permit la concaténation des tableaux pour renvoyer le tableau concaténé.

Figure 28 table de mapping (opérateurs de tableaux)

3.1 Les obstacles de passage du SQL vers le NoSQL

Contrairement aux systèmes de gestion de bases de données relationnelles, MongoDB ne se base pas sur les propriétés ACID mais il respecte les propriétés BASE, ce non-respect des propriétés ACID rend plusieurs étapes pas reproductibles dans le passage du SQL vers le NoSQL.

➢ Pas de Jointure

En NoSQL 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). Exemple :

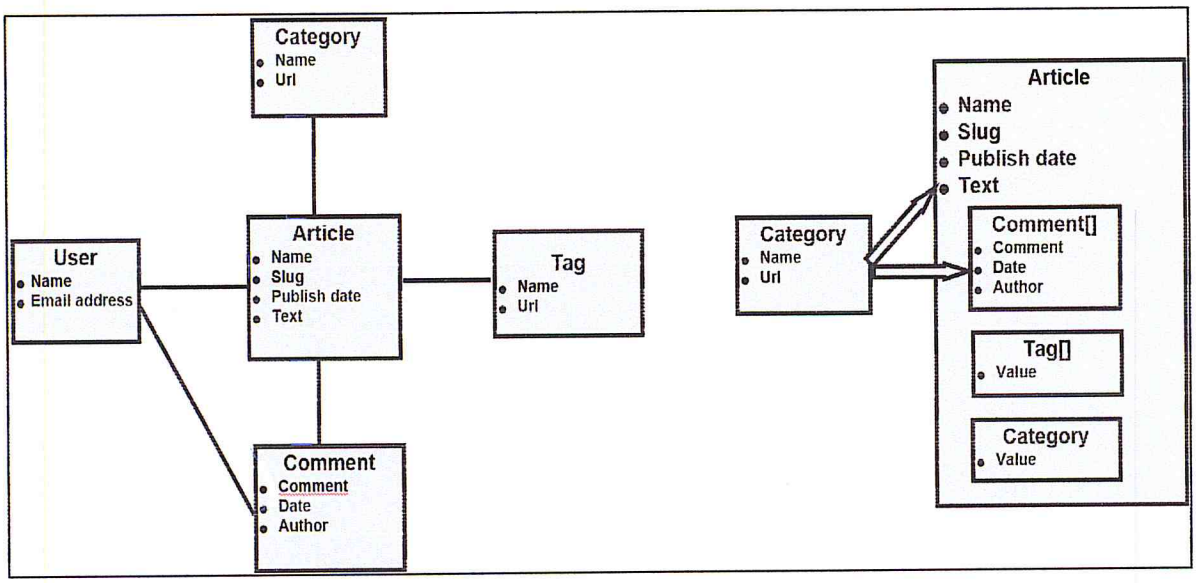


Figure 29 : Les jointures en SQL et leurs représentations dans le NoSQL

On peut voir que le choix a été d'imbriquer les documents de comment, tag et category pour un article, dans chaque document article respectif. Si votre application va devoir interroger, très souvent, à la fois le comment, tag et category alors le mieux sera de regrouper les documents en un seul, afin de réduire le nombre de requêtes à effectuer. Ce modèle, quant à lui, est dénormalisé, on ne peut pas réaliser cela en SQL [w2].

➢ Pas de schéma

Schéma dynamique/flexible : Commençons par une légère introduction sur la gestion des relations de données. Il est essentiel de considérer plusieurs principes lorsque vous choisissez le type de modélisation qui vous concerne, en passant par l'accessibilité des informations jusqu'à l'optimisation des requêtes. MongoDB a un schéma flexible, ce qui signifie que vous pouvez ajouter plusieurs documents ayant une structure différente dans une même collection [w2].

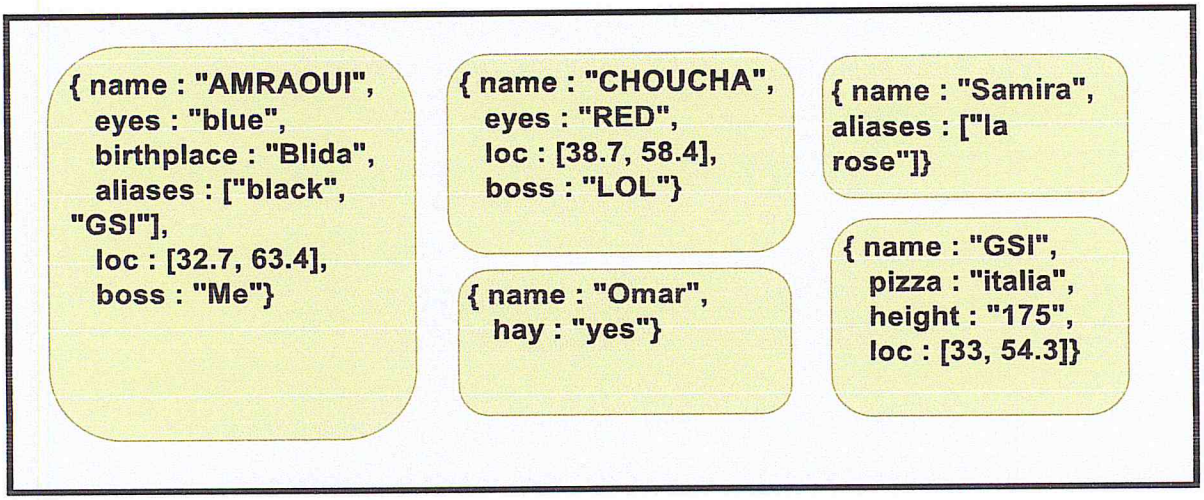


Figure 30 : Les Documents d'une Collection dans MongoDB.

➢ Pas de transactions

Il n'y a pas de transaction dans mongo, cependant les mises à jour d'un document se font de manière atomique.

Aucune écriture ne peut affecter plus qu'un seul et unique document. Le modèle dénormalisé (documents imbriqués) combine toute les données pour une entité, reprenons l'exemple sur l'article dans lequel on a regroupé ses commentaires, son tag associé et sa catégorie. Ce type de modélisation va faciliter les opérations d'écriture atomiques vu qu'une seule opération d'écriture permet l'insertion ou la modification de données pour une entité. En revanche, adopter la modélisation normalisée (par référence) va vous forcer à effectuer plusieurs requêtes qui ne seront pas atomiques collectivement [w2].

4. Les Requêtes agrégées

MongoDB propose un système de requêtage avancé, le framework d'agrégation, afin d'obtenir l'information désirée. Celui-ci permet notamment de filtrer les résultats, de les compiler ou même de les transformer.

Il y a trois façons d'utiliser le framework d'agrégation :

- Le mapReduce.

- Les fonctions dédiées.
- Le pipeline d'agrégation.

Le framework d'agrégation est un des points forts de MongoDB.

➢ **MapReduce**

Paradigme de traitement de données pour condenser de grands volumes de données en résultats agrégés utiles. En termes très simples, la commande MapReduce prend deux entrées primaires, la fonction de mappage et la fonction de réducteur.

Un Mapper débutera par la lecture d'une collection de données pour construire une map avec les champs que nous souhaitons traiter. Puis cette paire clé, valeur est introduite dans un réducteur, qui traitera les valeurs.

Exemple d'un prototype explicatif d'une requête sous MapReduce :

```
>db.collection.mapReduce(
   function() {emit(key,value);},  //map function
   function(key,values) {return reduceFunction},   //reduce
function
   {
      out: collection,
      query: document,
      sort: document,
      limit: number
   }
```

- **Map :** Fonction javascript qui mappe une valeur à une clé et emet la paire clé/valeur.
- **Reduce :** Fonction javascript qui réduit ou regroupe tout les documents ayant la même clé.
- **Out :** Spécifie l'emplacement du résultat de la spécifies the de la map-reduce.
- **Query :** Précise les critères de sélection en option pour sélectionner les documents.
- **Sort :** Précise les critères de tri en option.
- **Limit :** Indique le nombre maximal option de documents à retourner.

➢ Les Fonctions dédiées

Ce sont les méthodes permettent de répondre spécifiquement à des besoins d'agrégation :

- **Count :** Pour compter le nombre d'éléments respectant les paramètres fournis. La commande utilisée est : db.collection.count(query)

Exemple : `> db.movies.count({year: 1997})`

Les autres fonctions ont été cite dans la partie dictionnaire de données.

➢ Le pipeline

Le pipeline est un framework qui permet - comme son nom l'indique - d'enchaîner des actions pour transformer et traiter les résultats de requêtes spécifiques.

La commande utilisée est : `db.collection.aggregate(pipeline, options)`

Parmi les options possibles :

- explain : obtenir des méta-informations sur la requête.
- allowDiskUse : permettre l'utilisation d'un cache sur disque.
- cursor : définir la taille du lot pour le curseur.

5. Architecture de l'application

Le but de notre projet de fin d'étude est de développer une application qui permettre à l'utilisateur de :

- ➢ Traduire des requêtes SQL vers des requêtes MongoDB.
- ➢ La Conversion d'un fichier SQL vers une base de données MongoDB
- ➢ La Création d'une base de données MongoDB à partir d'un diagramme MCD (Modèle Conceptuel de Données). L'architecture de l'application est présentée par la figure 18 :

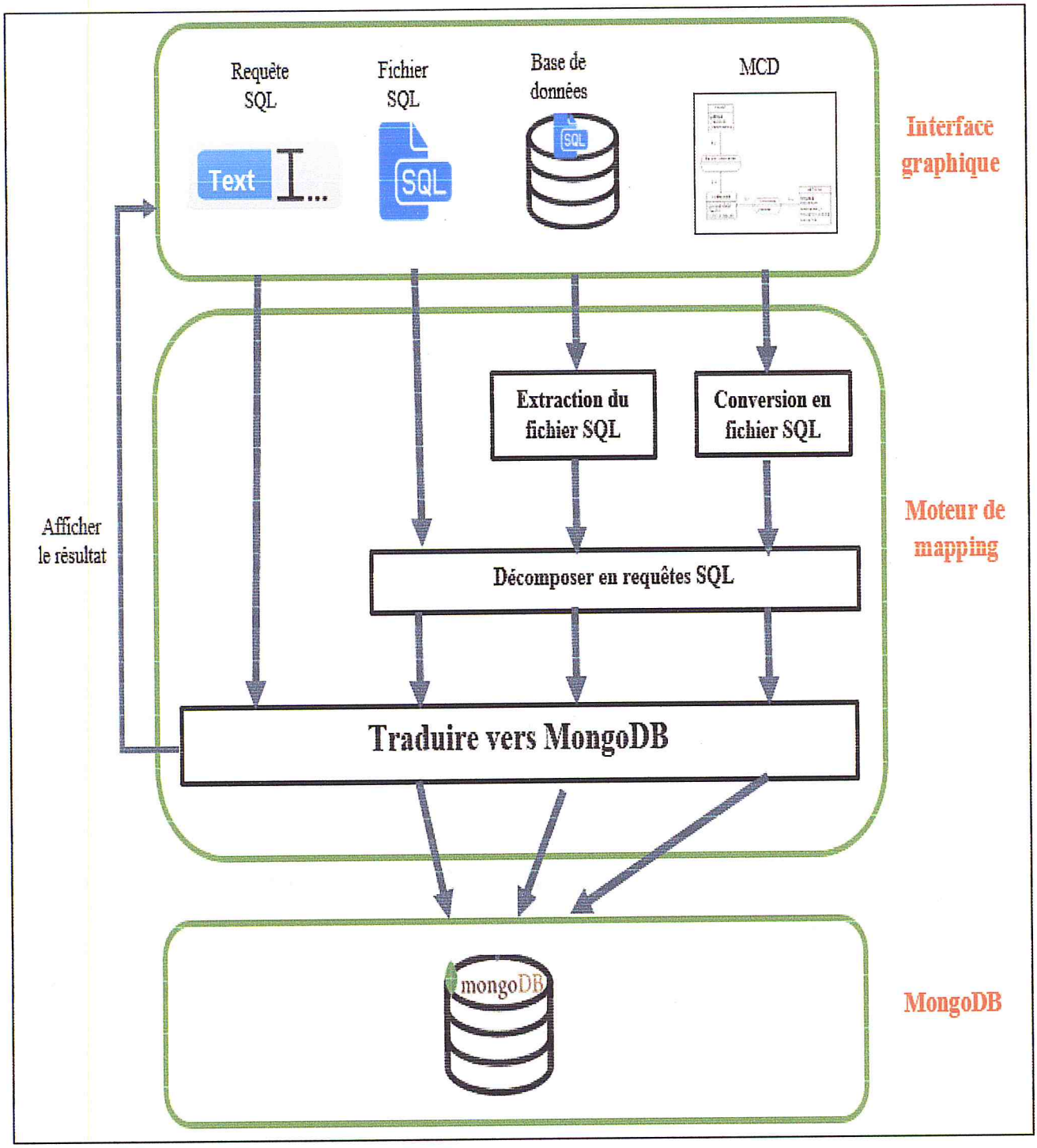


Figure 31 : l'architecture générale de l'application

a. Interface graphique

L'interface graphique permet de faire l'intégration entre l'utilisateur et le moteur de mapping, elle regroupe trois composantes à savoir :

- Lecteur des requêtes SQL
- Lecteur du fichier SQL.
- Editeur du diagramme MCD.

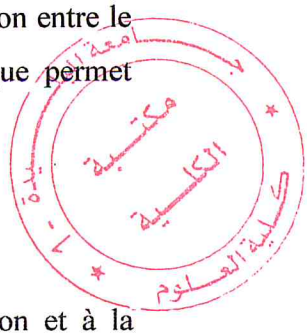
La traduction de la requête se fait en saisissant la requête SQL dans le champ de saisir, le programme de traduction va convertir la requête et l'afficher par la suite. L'application permet aussi d'importer un fichier SQL pour faire la conversion, une autre possibilité, c'est la création d'un MCD.

b. Moteur de mapping

Le passage de SQL vers MongoDB se fait dans notre application par un programme de mapping qui contient toutes les fonctions de conversion, ces fonctions permet la conversion des opérations CRUD avec les requêtes de jointure, ce programme est utilisé ainsi pour convertir les fichiers SQL.

c. Moteur d'interaction avec MongoDB

Un autre composant dans notre application qui est très important, c'est le programme d'exécution des requêtes MongoDB, ce programme ce base sur l'API JAVA qui permet la manipulation des bases de données MongoDB. Cette bibliothèque offre la connexion entre le programme écrit en java et le serveur de MongoDB, de plus, cette bibliothèque permet d'exécuter les requêtes MongoDB.



6. Les Différents Scénariis de mapping

Dans cette section, nous discuterons des opérations relatives à la création et à la modification des collections (table en SQL), d'insertion, de lecture, de mise à jour et de suppression des documents (lignes en SQL).

L'algorithme consiste à remplacer les requêtes SQL par leurs équivalents en MongoDB tout en respectant la syntaxe de ce dernier. Pour chaque opération en MongoDB.

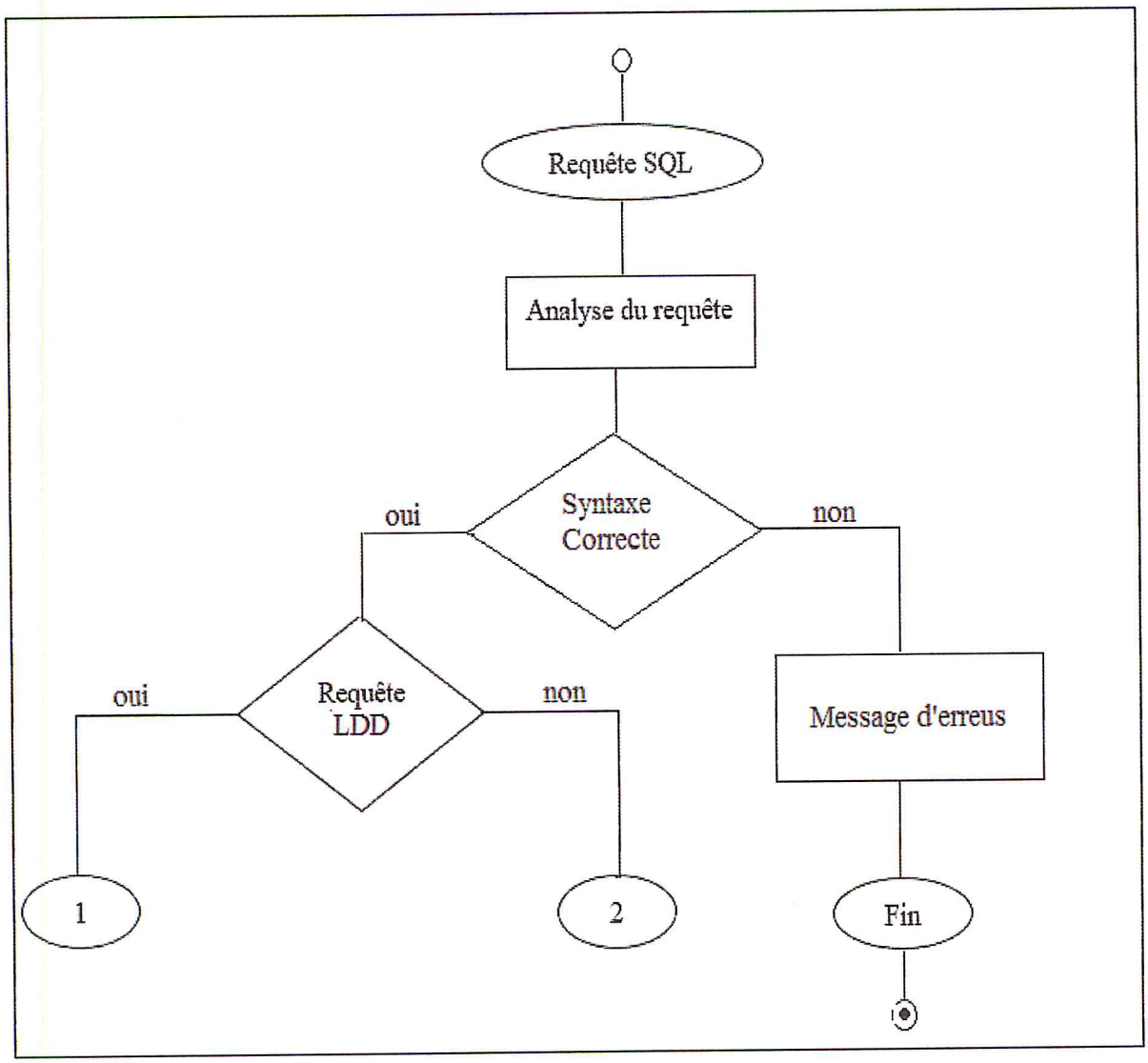


Figure 32 : Organigramme général de mapping.

6.1. Scénarii de Création

Dans MongoDB, on trouve 3 opérations de création : création de la base de données, création de la collection et la création des indexes, il n'est pas nécessaire de créer explicitement la structure de la collection (comme nous le faisons pour les tables à l'aide d'une requête

CREATE TABLE). La structure du document est automatiquement créée lorsque le premier insert se produit dans la collection.

L'algorithme détecte la première clause de la requête SQL en entrée, il doit déterminer la nature de l'opération, dans le cas d'une création, l'algorithme remplace la requête par son équivalant en MongoDB : db.createDatabase('nom') pour la création d'une base de données, db.createCollection('nom') en cas où c'est une création d'une collection (table en SQL) et db.ensureIndex('paramètre') pour la création des indexes.

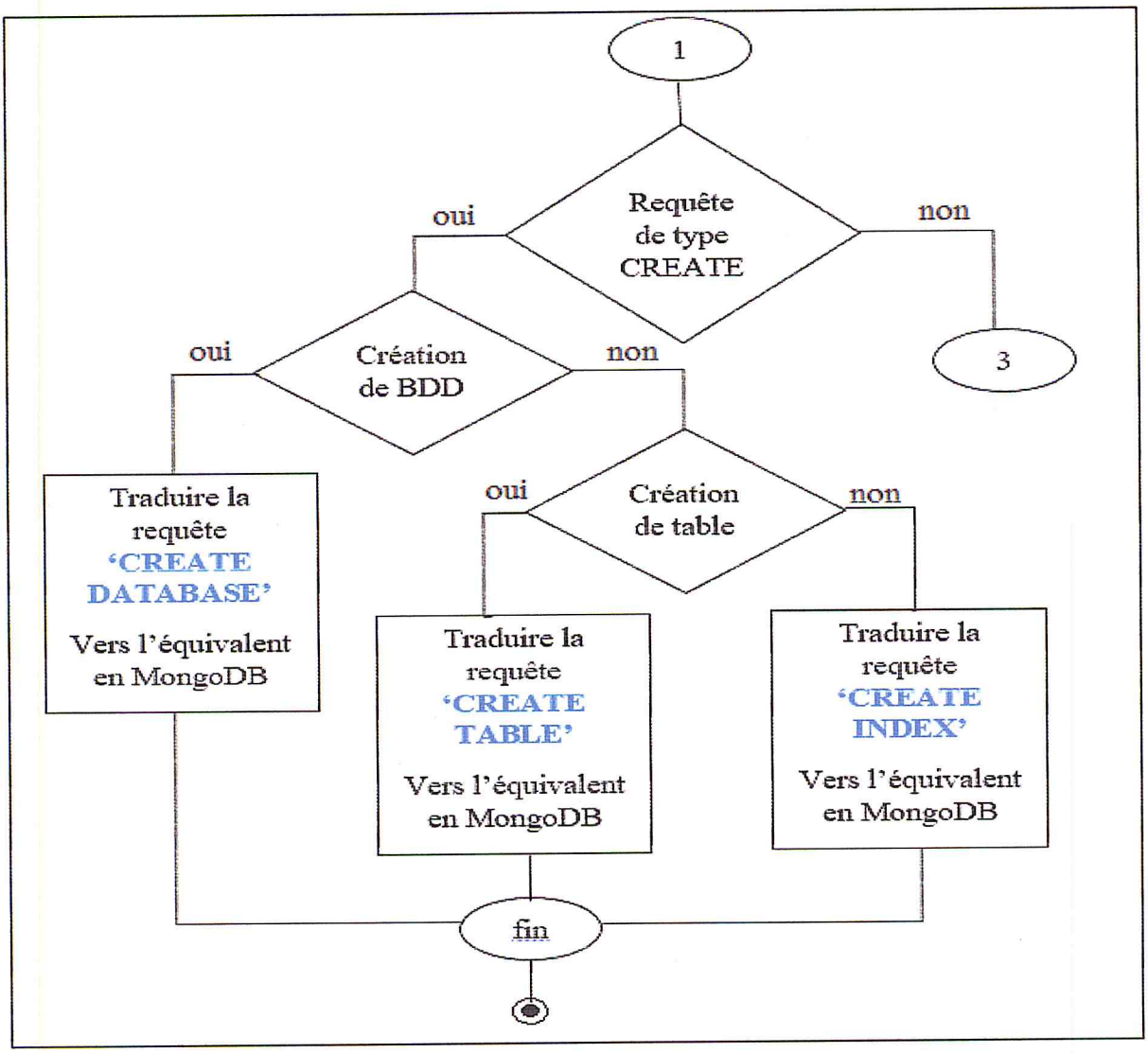


Figure 33 : Organigramme de Création.

6.2 Scénarii de Suppression de base de données ou de tables

En MongoDB, on a deux Operations de suppression de la structure : la suppression complète de la base de données actuelle et la suppression de la collection.

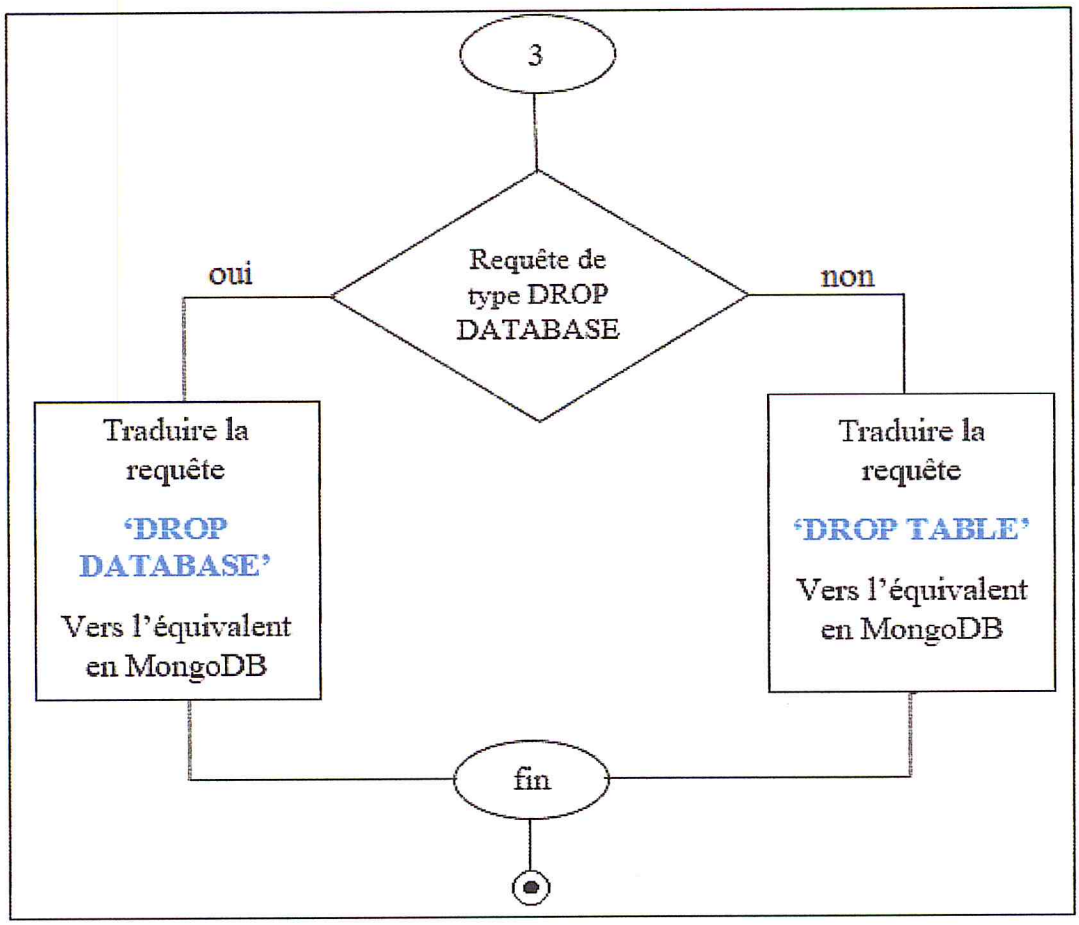


Figure 34: Organigramme de Suppression de BD ou de tables.

6.3 Scénarii de Insertion

La conversion de la requête d'insertion du SQL vers MongoDB se fait d'une façon très souple il suffit seulement la détection du mot clé INSERT puis exécuté sa fonction équivalente en MongoDB. L'algorithme d'insersion nous donne la sortie suivante : db.collection.insert('paramètres'), où collection est le nom de la collection et 'paramètres' sont les données à insérer exprimer sous la forme {'clé' :'valeur'} où 'clé' est l'emplacement d'insertion et 'valeur' est la donnée à insérer.

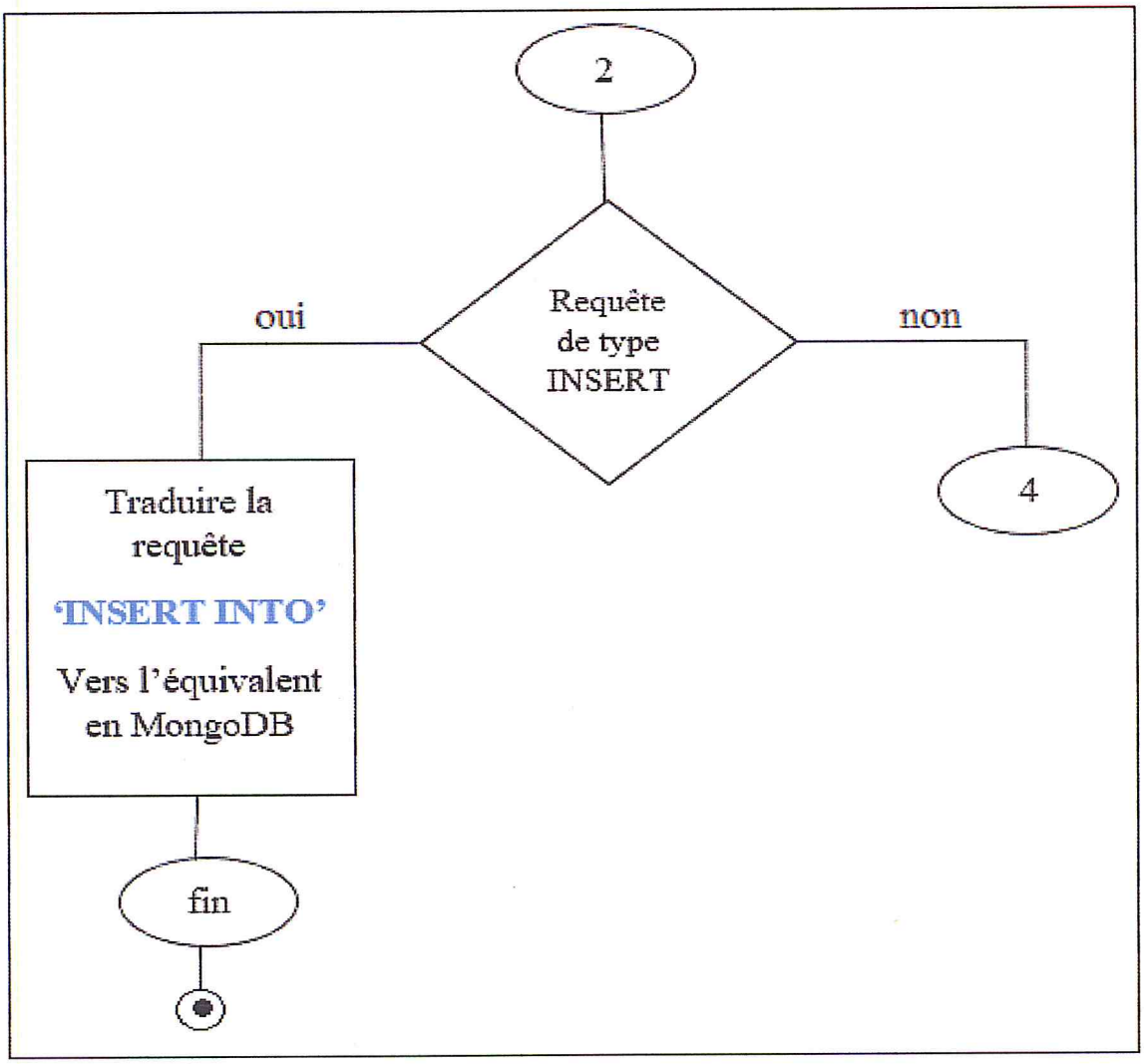


Figure 35: Organigramme d'Insertion.

6.4 Scénarii de mise à jour

L'Algorithme de conversion de la requête de mise à jour du SQL vers MongoDB fonctionne d'une manière plus souple d'où il suffit la détection du mot clé **UPDATE** puis retournée sa fonction équivalente en MongoDB : db.collection.update(paramètre1, paramètre2)

Paramètre1 spécifie les critères pour sélectionner les documents a modifier, paramètre2 spécifie l'opération de mise à jour à effectuer.

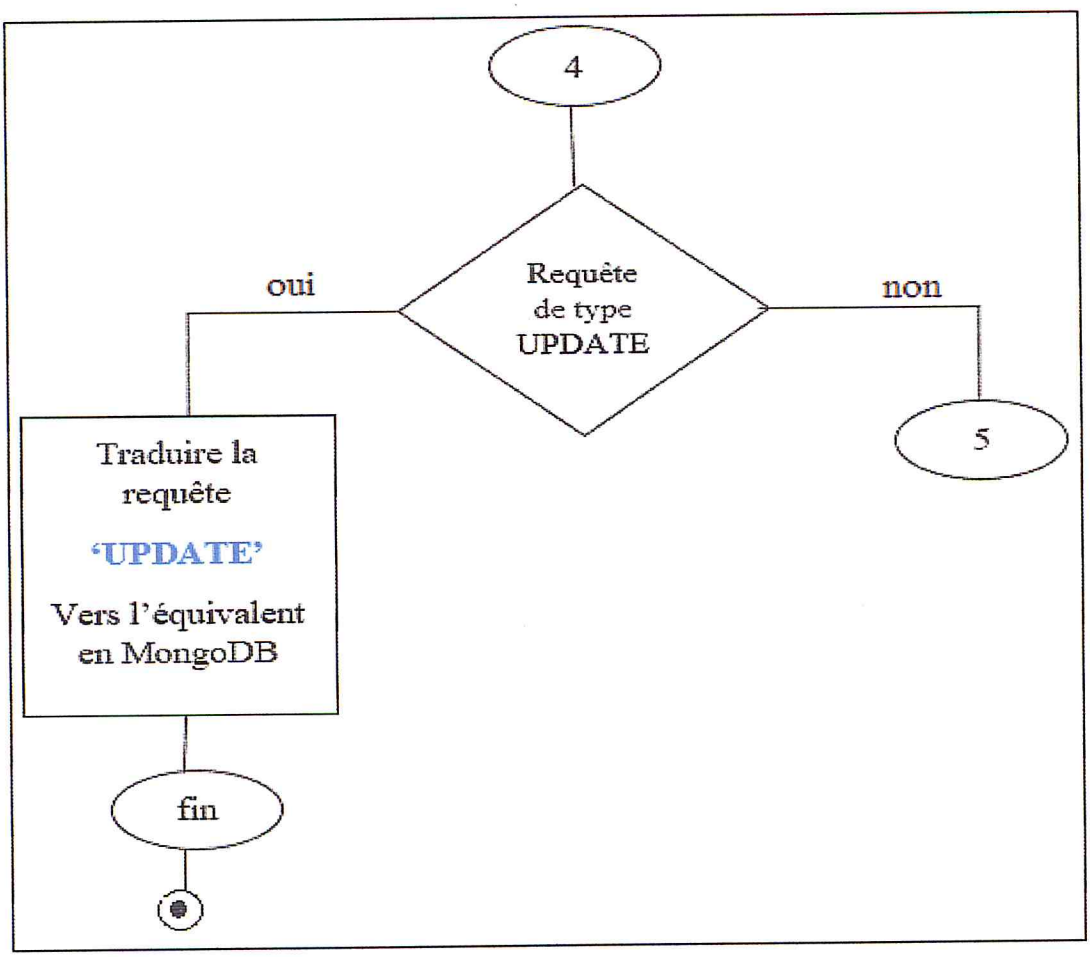


Figure 36 : Organigramme de mise à jour.

6.5 Scénarii de Suppression des enregistrements

La conversion de la requête de suppression des enregistrements du SQL vers MongoDB se fait d'une manière très souple il suffit seulement la détection du mot clé DELETE puis l'algorithme retourne sa fonction équivalente en MongoDB. La sortie est : db.collection.remove(paramètres), où collection est le nom de la collection et paramètres sont les conditions de la suppression.

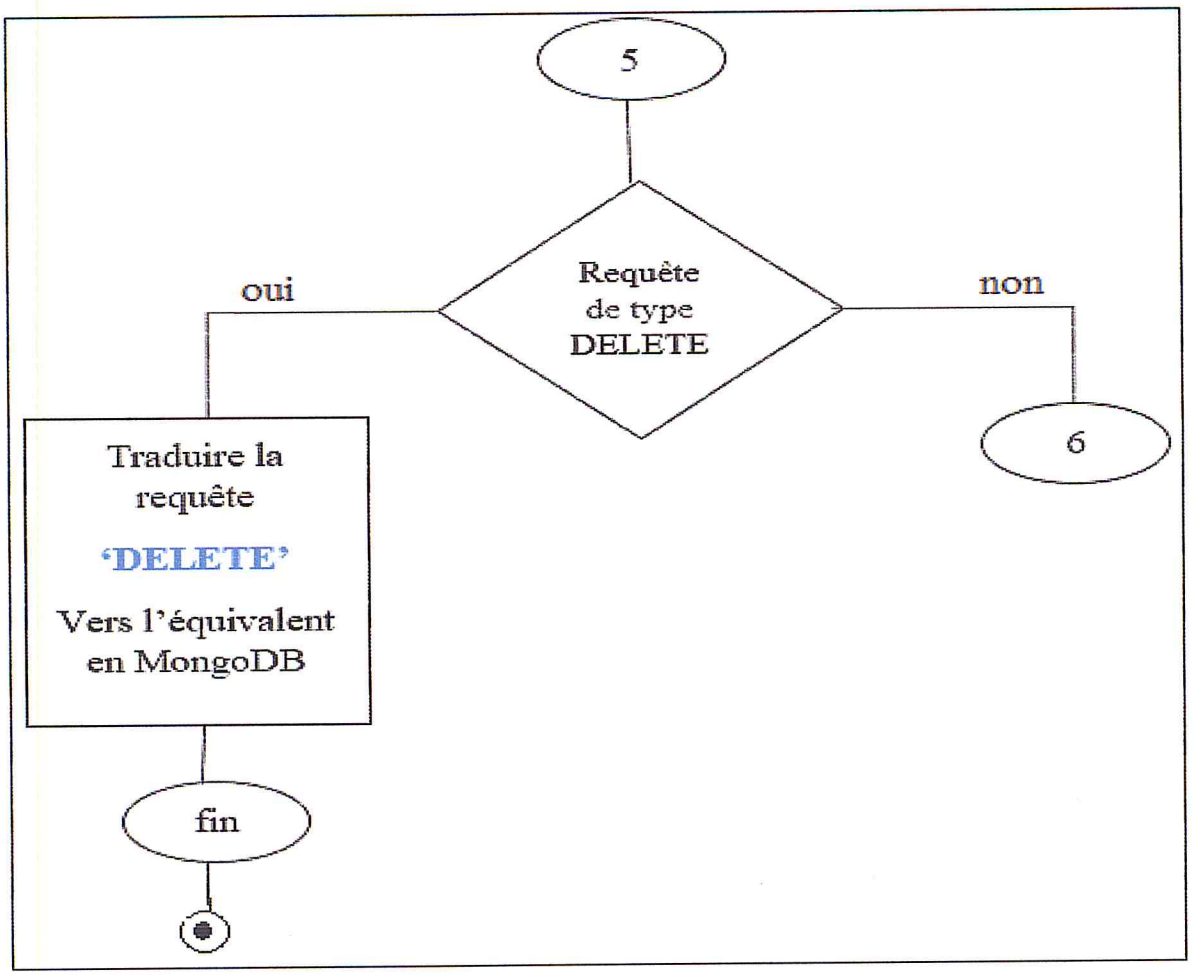


Figure 37 : Organigramme de Suppression des enregistrements.

6.6 Scénarii des Requêtes de recherche et de jointure

Pour une requêtes simple, la requête SQL est remplacé par :db.collection.find(p1,p2), p1 exprime les conditions qui se trouve après la clause WHERE en SQL, p2 sont les champs à rechercher, la syntaxe est bien définie dans la section précédente.

Si l'algorithme détecte une jointure ou une requête imbriquée, il remplacera la requête par : db.collection.aggregate(paramètres) , où dans paramètres $group est utilisée pour les requêtes imbriquée et le mot clé $lookup est utilisé pour les jointures, la syntaxe de $group et $lookup est bien exprimé précédamment dans ce chapitre.

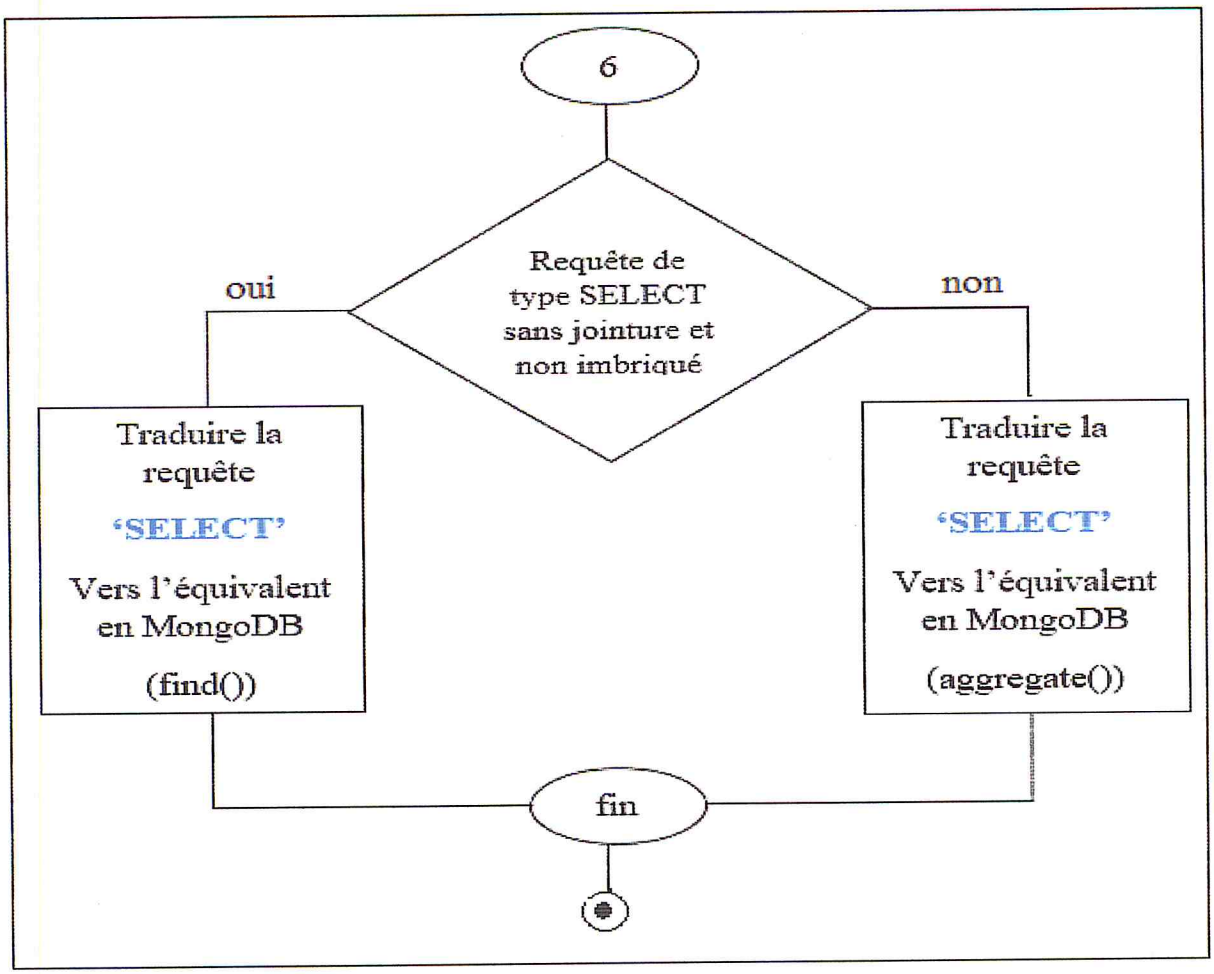


Figure 38 : Organigramme de de recherche et de jointure.

7. Conclusion :

Dans ce chapitre nous avons présenté la base de données MongoDB, ainsi nous avons décrit les architectures, les principes de fonctionnement, les caractéristiques techniques et les composants de notre système. Le chapitre a fait l'objet d'une présentation générale du MongoDB et notre application.

Enfin, le vrai intérêt de MongoDB, surtout par rapport à d'autres bases de données NoSQL, c'est sa gestion de requêtages. On peut requêter de façon très fine grâce à une grande quantité de mots clés. On peut donc faire des requêtes aussi riches qu'en SQL, mais tout ça sur une base de données orientée Documents.

Chapitre 3 : Implémentation, Test et validation

1. Introduction

Après avoir décrit notre solution, nous aborderons dans ce chapitre la partie test et validation de notre application. En première partie, nous présenterons l'environnement de travail et les outils de développement utilisés. Ensuite, nous présentons notre jeu d'essai. Enfin, nous présenterons notre application ainsi que son fonctionnement et les résultats.

2. Présentation de l'environnement de travail

2.1 Systèmes d'exploitation

Nous utilisons le système Windows 10 son type est. 64-bit, sa RAM et de 8GO, son Disque dure et de 1000 GO Le processeur est Intel(R) Core(TM) i7 CPU 4700MQ @ 2.40GHz 2.40 GHz.

2.2 Langage de Programmation 'MongoDB'

–**NetBeans** est un projet open source ayant un succès et une base d'utilisateur très large. Sun Microsystems a fondé le projet open source NetBeans en Juin 2000 et continue d'être le sponsor principal du projet. Conçu en Java, NetBeans est disponible sous Windows, Linux, Solaris (sur x86 et SPARC), Mac OS X. Aujourd'hui, deux projets existent :

– **EDI NetBeans :** c'est un environnement de développement, un outil pour les programmeurs pour écrire, compiler, déboguer et déployer des programmes. Il est écrit en Java, mais peut supporter n'importe quel langage de programmation. Il y a également un grand nombre de modules pour étendre l'EDI NetBeans.

– **Plateforme NetBeans :** c'est une fondation modulable et extensible utilisée comme brique logicielle pour la création d'applications bureautiques. Les partenaires privilégiés fournissent des modules à valeurs ajoutées qui s'intègrent facilement à la plateforme et peuvent être utilisés pour développer ses propres outils et solutions.

2.3 SGBD No-SQL

MongoDB 3.4 est la dernière version du leader des bases de données NoSQL pour les applications modernes, Un point culminant des fonctionnalités et des améliorations natives ce dernier qui permettre l'évolution facile des solutions pour relever les nouveaux défis et les cas d'utilisation.

MongoDB 3.4 introduit un équilibrage automatique des données entre les nœuds et une synchronisation plus rapide des réglages de répliques, et la compression réseau intra-cluster qui aider à répondre efficacement aux demandes dynamiques d'application.

3. Tests et validation

Dans l'univers relationnel, on utilise un langage particulier, SQL, pour ces opérations et ce quel que soit le langage utilisé pour le reste de l'application. Avec MongoDB, on reste dans le langage hôte et on utilise le driver approprié qui fournit les APIs nécessaires.
En illustre l'exemple des requêtes par la supposition d'un Modèle Conceptuel des Données (MCD) de ventes qui sera traduit a la suite a des table en SQL et des collections en MongoDB tout en utilisant des requêtes en SQL et leur équivalent en MongoDB.

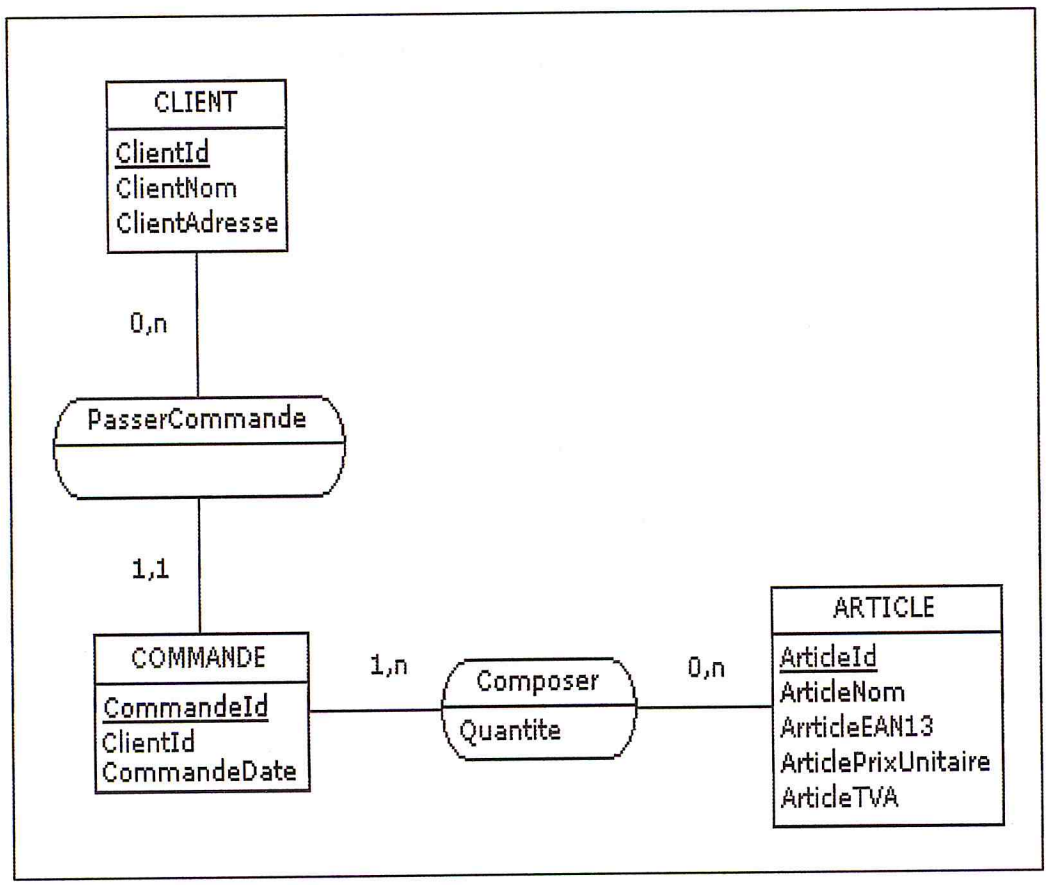


Figure 39 : Exemple de BD 'VENTES' pour le jeu de test.

- **Table CLIENT**

ClientId	ClientNom	ClientAdresse
001	AMINE	BLIDA
002	FARID	ALGER
003	KARIM	TIPAZA

Figure 40 : la table CLIENT

- **Table COMMANDE**

CommandeId	ClientNum	CommandeDate
256	001	26-06-2017
257	002	17-08-2017
258	003	14-09-2017

Figure 41 : La table COMMANDE

- **Table ARTICLE**

ArticleId	ArticleNom	ArticlePrixU	ArticleTVA	Quantite
A1	Disque 1TO	6000	19%	112
A2	RAM 8GO	2000	19%	45
A3	CABLE ETHERNET 10M	600	19%	97

Figure 42 : La table ARTICLE

- **Afficher les bases de données existantes :**

Pour afficher les bases de données existantes, on utilise **show dbs.** Normalement on devra avoir une base local propre à mongo et une base test :

Dans SQL :

```
SQL> SHOW DATABASES ;
```

Dans Mongo :

```
show dbs
local   0.03125GB
test    (empty)
```

Capture d'écran pour le test de validité de la requête dans notre application :

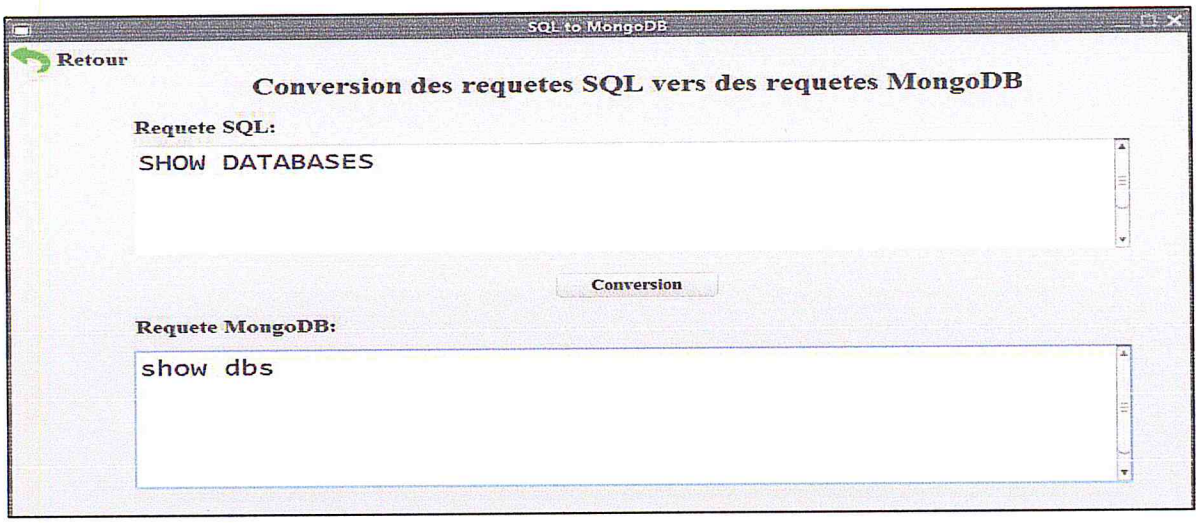


Figure 43 : Capture d'écran n°01

- **Création d'une base de données factures :**

Syntaxe de création d'une base de données ventes en SQL et en MongoDB sont les suivantes :

Dans SQL : **SQL> CREATE DATABASE VENTES ;**

Dans Mongo : **use VENTES**

Capture d'écran pour le test de validité de la requête dans notre application :

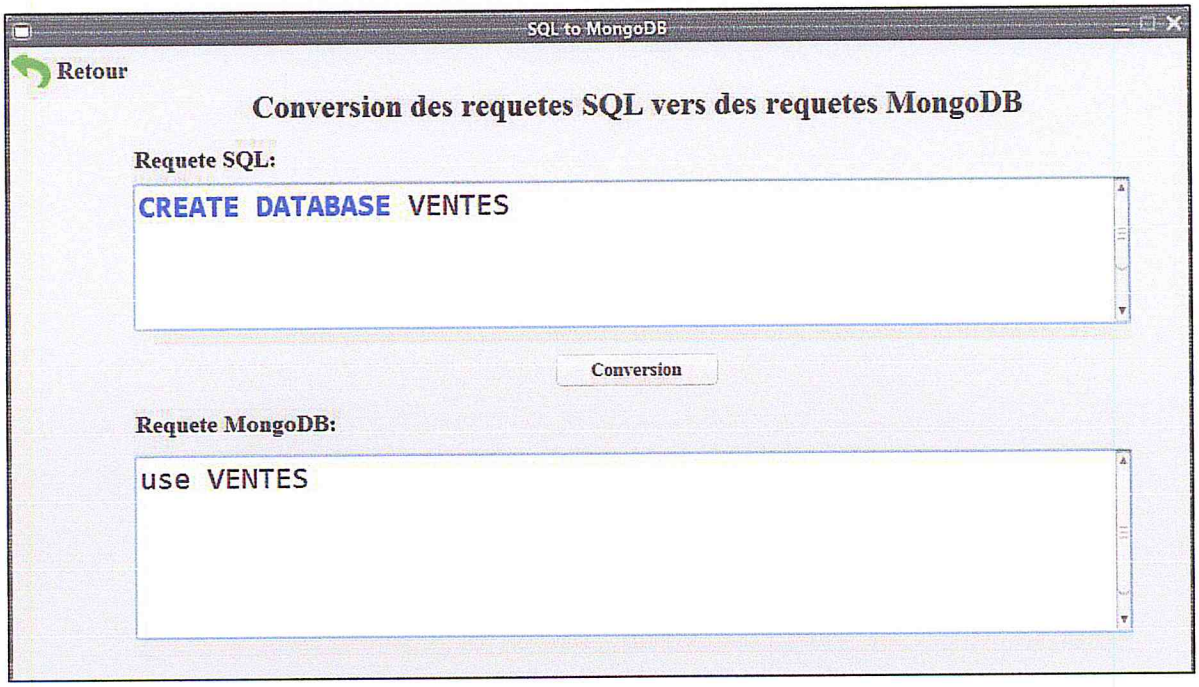


Figure 44 : Capture d'écran n°02

- **Création des Collections (Tables) :**

Une collection est tous simplement un ensemble de document. On peut la comparer à une table. Par exemple, une collection de 50 clients contiendra 50 documents.

➢ **Création de la Collections (Tables) CLIENT :**

Dans SQL :

```
CREATE TABLE `CLIENT`(
`ClientId` INT PRIMARY KEY NOT NULL,
`ClientNom` VARCHAR(100),
`ClientAdresse` VARCHAR(255), )
```

En MongoDB on peut créer une Collection de deux manieres :

- Soit d'une façon implicite lorsque de l'insertion du premier client comme suit :

```
db.CLIENT.insert ( {
ClientId : "abc123",
ClientNom : "Zahra",
ClientAdresse : "Blida" } )
```

- Soit d'une façon explicite car La collection ne décrit pas et ne force pas la structure de ses documents :

```
db.createCollection("CLIENT")
```

Capture d'écran pour le test de validité de la requête dans notre application :

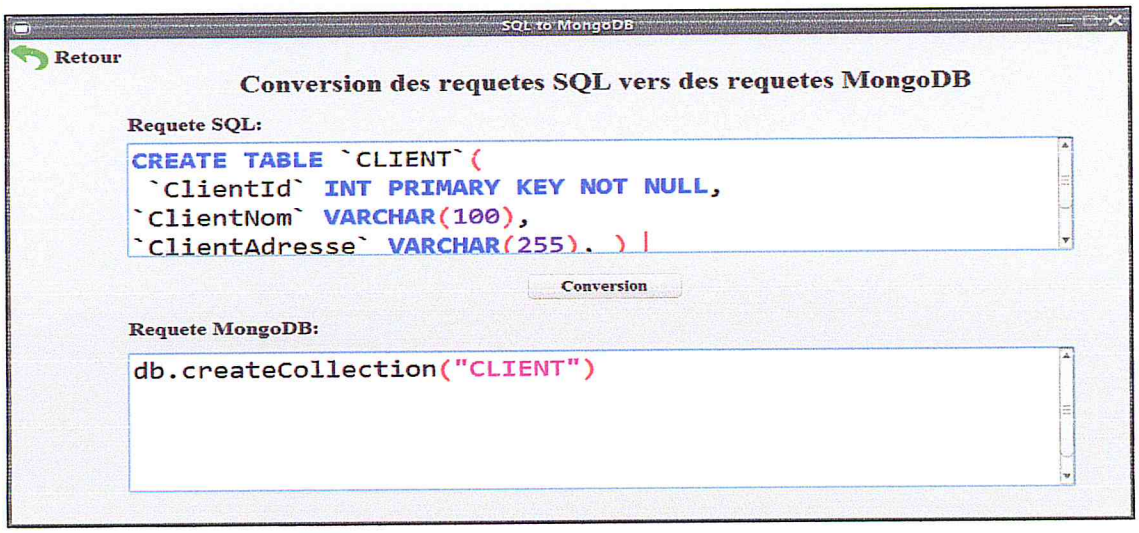


Figure 45: Capture d'écran n°03

On suppose que les deux autres tables ont été créer de la même façon.

- **Opération Insérer :**

L'insertion de données dans une collection s'effectue à l'aide de la fonction **insert().** Cette fonction permet au choix d'inclure un seul document à la base existante ou plusieurs documents (**insertMany()**) d'un coup.

En SQL :

```
INSERT INTO CLIENT (`ClientId`, `ClientNom`, `ClientAdresse`)
VALUES ('001', 'AMINE', 'BLIDA');
```

En MongoDB :

```
Db.CLIENT.insert ( {
 ClientId : "001",
ClientNom : "AMINE",
ClientAdresse : "BLIDA" } )
```

Capture d'écran pour le test de validité de la requête dans notre application :

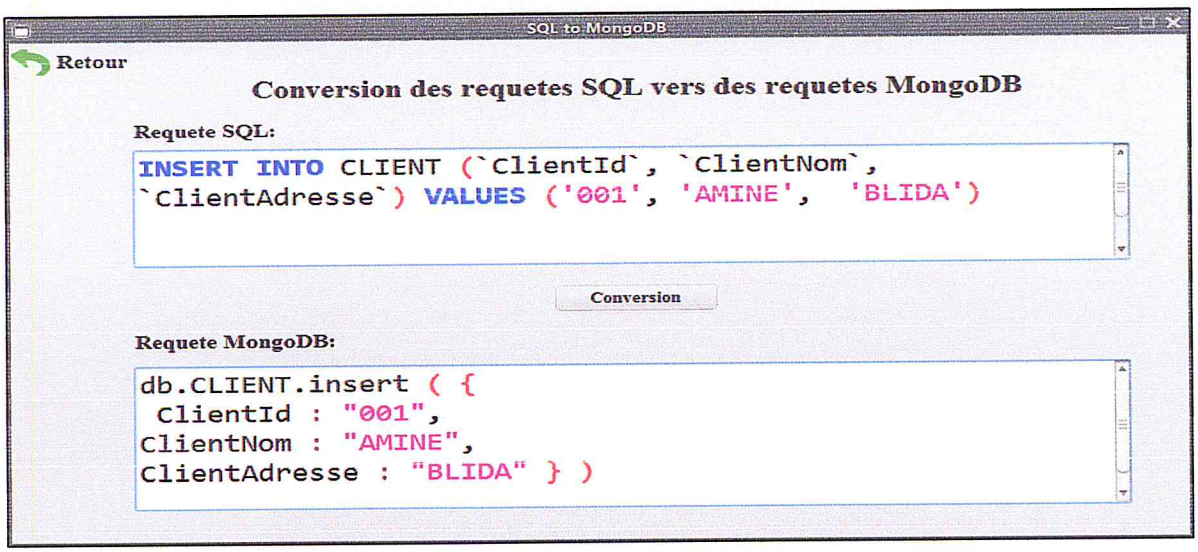


Figure 46 : Capture d'écran n°04

- **Opération de Lecture :**

L'utilisation la plus courante de SQL et NoSQL consiste à lire des données issues de la base de données. Cela s'effectue grâce à la fonction find(), qui retourne des enregistrements dans une collection. Cette commande peut sélectionner une ou plusieurs documents d'une collection.

➢ Retourner toute la liste de la collection (Table) ARTICLE :

En SQL : **SELECT * FROM ARTICLE ;**

En Mongo : **db.ARTICLE.find();**

Capture d'écran pour le test de validité de la requête dans notre application :

➢ Retourner que les ArticleId de la Collection (Table) ARTICLE :

En SQL : **SELECT ArticleId FROM ARTICLE;**

En Mongo : **db.ARTICLE.find({}, {ArticleId:1});**

Capture d'écran pour le test de validité de la requête dans notre application :

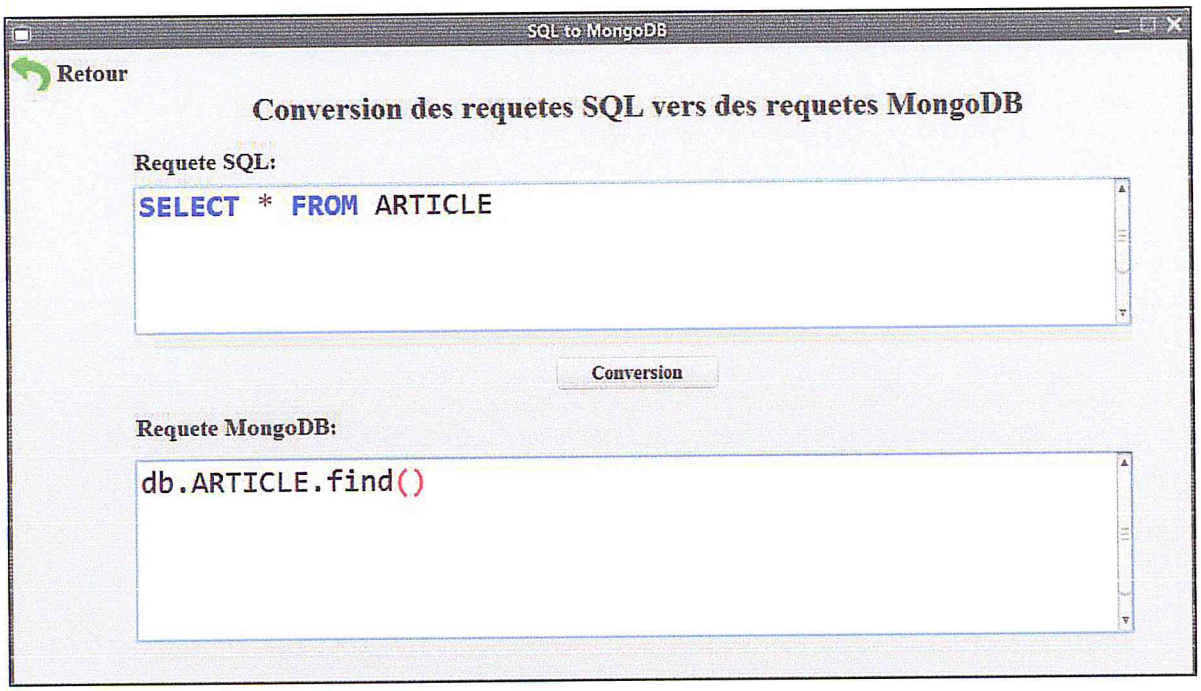


Figure 47 : Capture d'écran n°05

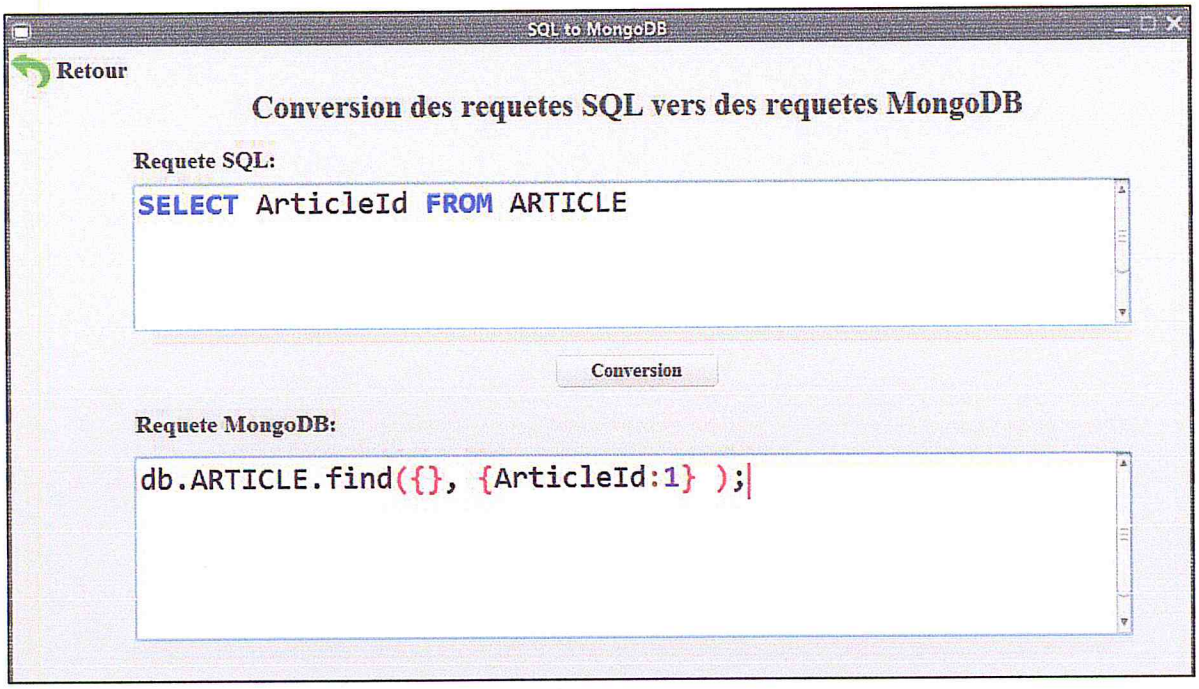


Figure 48 : Capture d'écran n°06

- Retourner les enregistrements dont leurs ArticlePrixU est supérieur a 1000 de la Collection (Table) ARTICLE :

En SQL : **SELECT * FROM ARTICLE WHERE ArticlePrixU > 1000 ;**

En Mongo : **db.ARTICLE.find({ "ArtId" : { "$gt" : 1000 } });**

Capture d'écran pour le test de validité de la requête dans notre application :

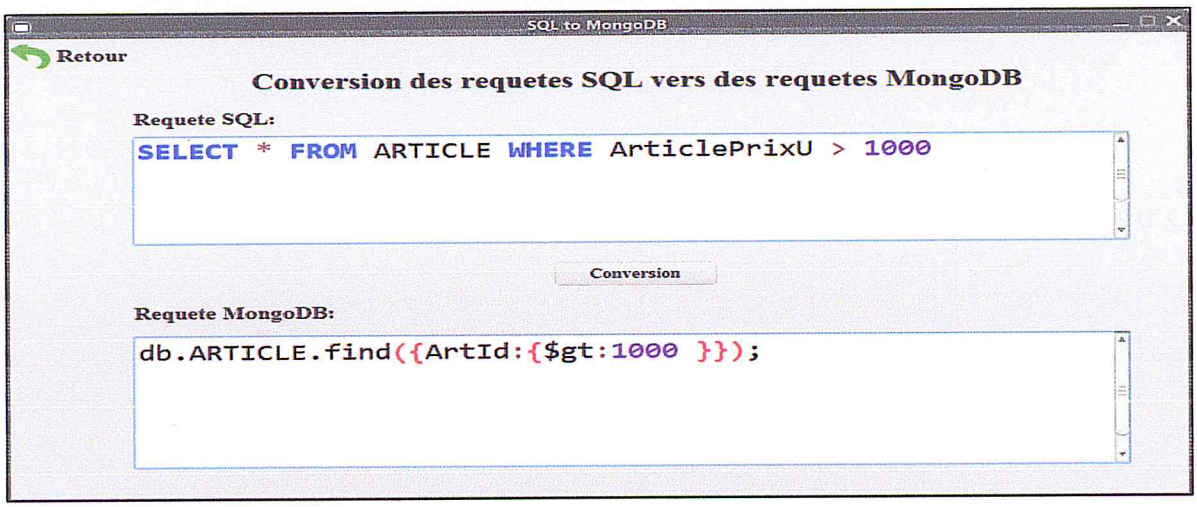


Figure 49 : Capture d'écran n°07

- Retourner les enregistrements dont leurs ArticlePrixU est inférieur a 1000 de la Collection (Table) ARTICLE :

En SQL :

```
SELECT * FROM ARTICLE WHERE ArticlePrixU < 1000 ;
```

En Mongo :

```
db.ARTICLE.find( { "ArtId" : { "$lt" : 1000 } });
```

Capture d'écran pour le test de validité de la requête dans notre application :

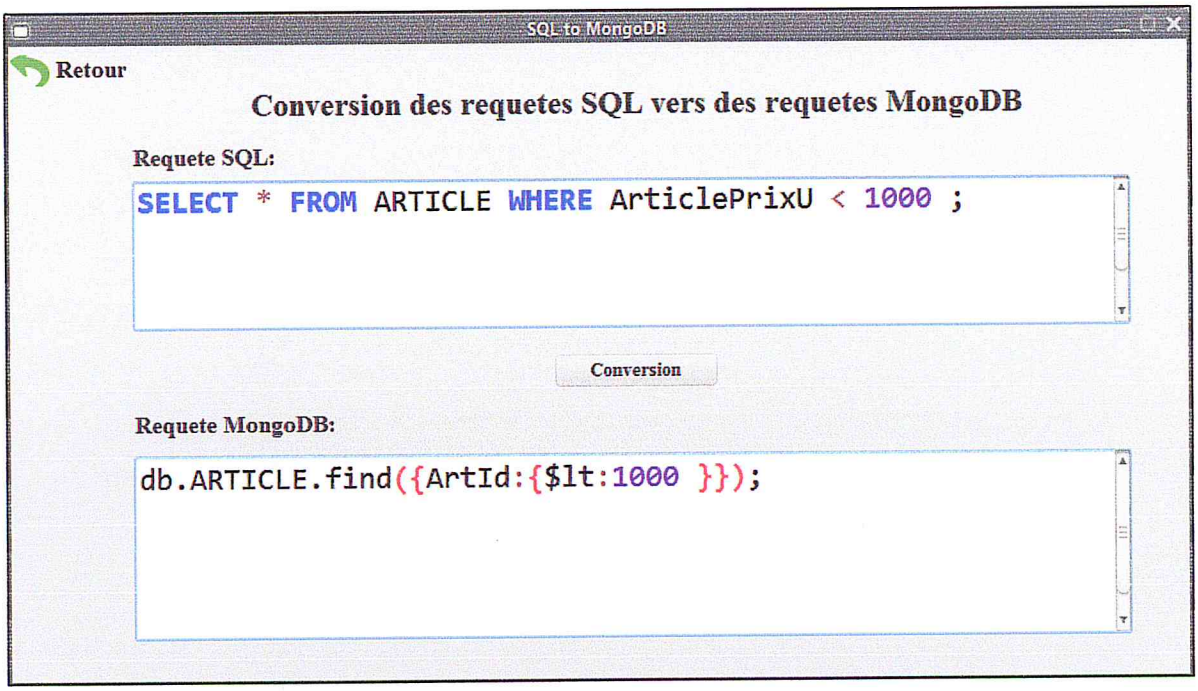


Figure 50 : Capture d'écran n°08

- Récupérer les Clients dont l'Adresse du Client (ClientAdresse) est "BLIDA" ou "ALGER" :

En SQL :

```
SELECT * FROM CLIENT WHERE ClientAdresse = 'BLIDA'
OR ClientAdresse = 'ALGER' ;
```

En Mongo :

```
db.CLIENT.find({ $or:[{ ClientAdresse: "BLIDA"}, {
ClientAdresse: "ALGER" }] })
```

Capture d'écran pour le test de validité de la requête dans notre application :

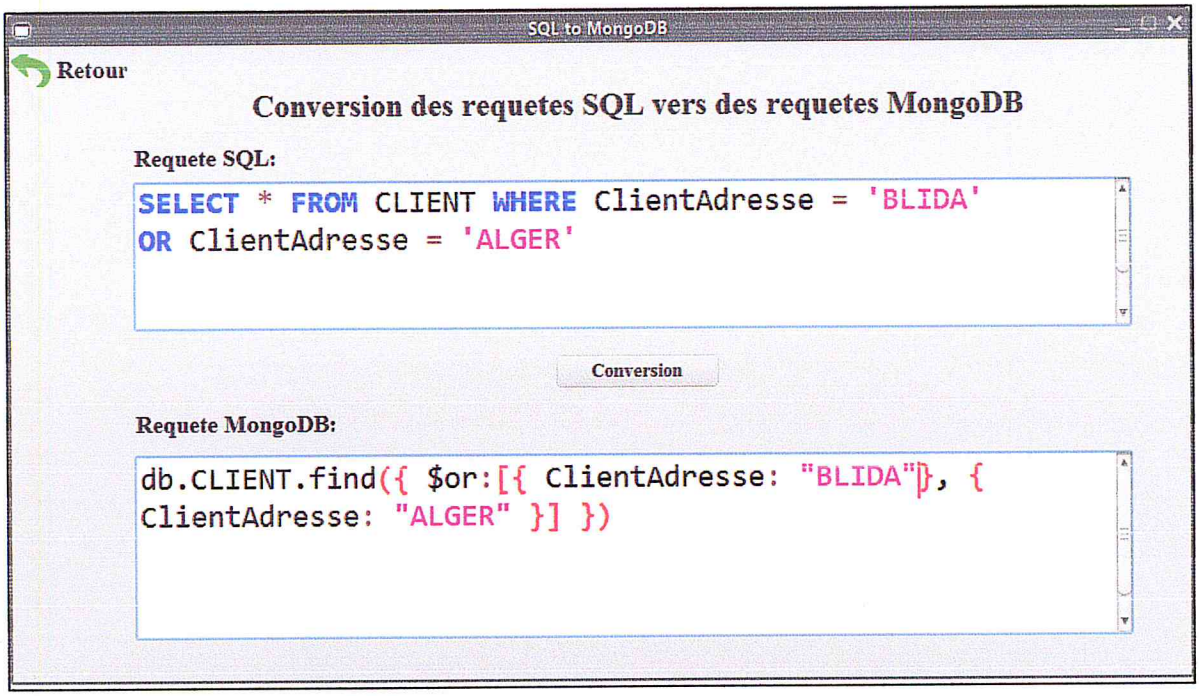


Figure 51 : Capture d'écran n°09

- Pour récupérer uniquement certaine clé, on utilise l'argument projection de find(). Par exemple, récupérer uniquement les noms des Article (ArticleNom) dont lA Quantité est supérieur à 40 :

En SQL : **SELECT ArticleNom FROM ARTICLE WHERE Quantite > 40 ;**

En Mongo : **db.ARTICLE.find({ Quantite :{ $gt : 40 }},{ "ArticleNom" : true })**

Capture d'écran pour le test de validité de la requête dans notre application :

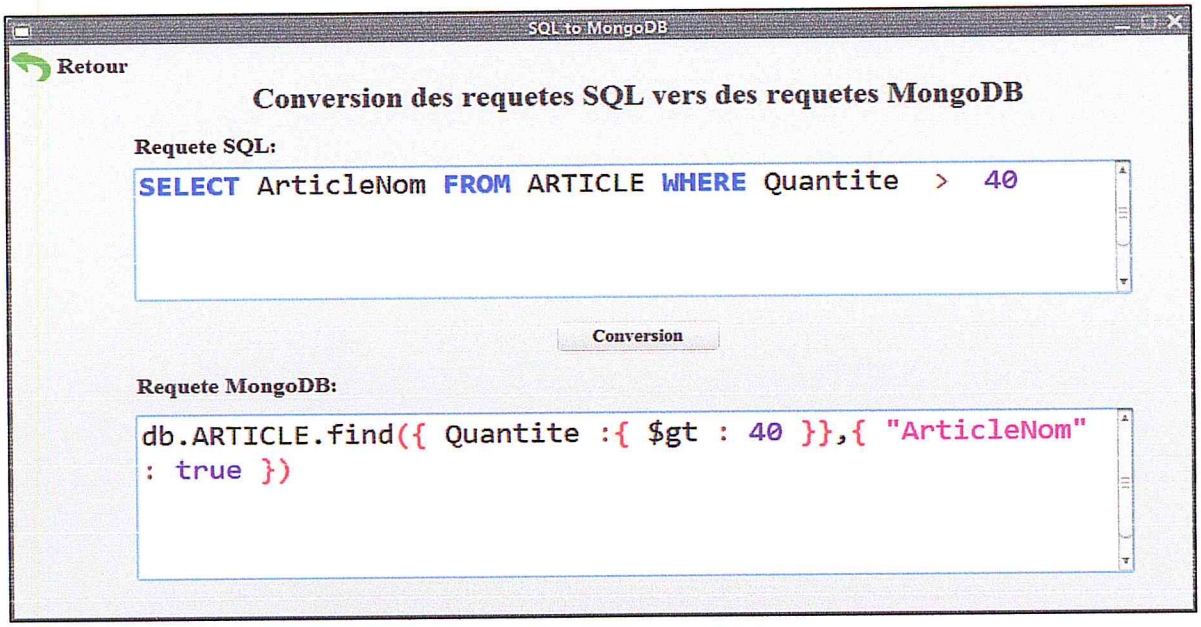


Figure 52: Capture d'écran n°10

- Pour limiter le nombre de résultat d'une requête en spécifiant le nombre maximum de résultats que l'on souhaite obtenir.

En SQL : **SELECT * FROM CLIENT LIMIT 10 ;**

En Mongo : **db.CLIENT.find().limit(10);**

Capture d'écran pour le test de validité de la requête dans notre application :

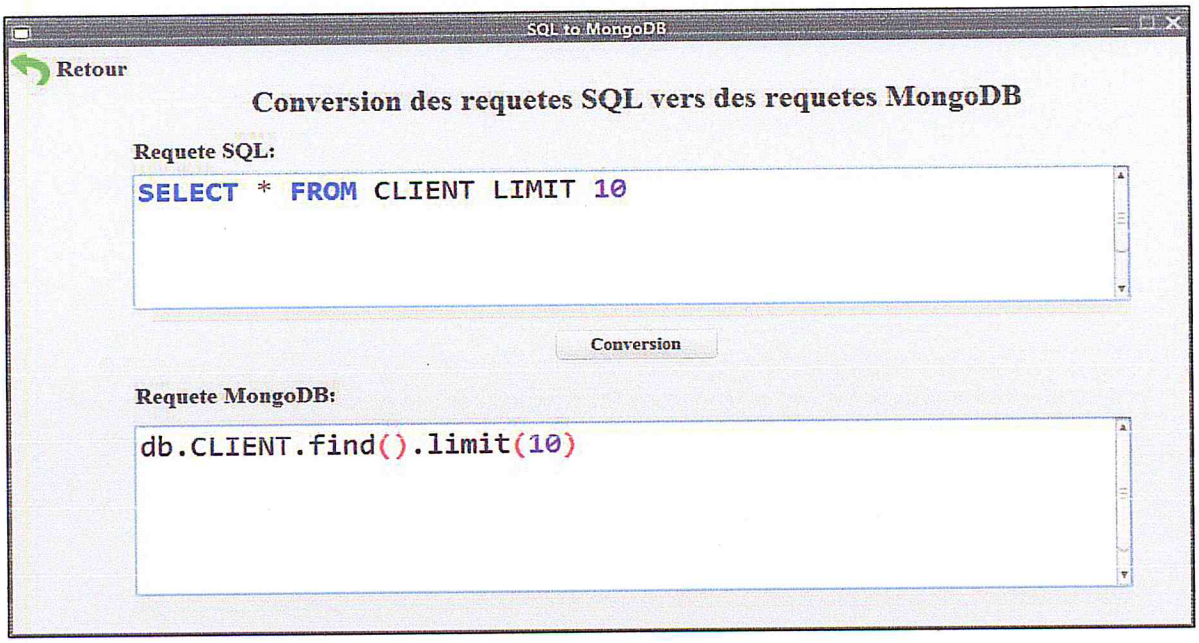


Figure 53 : Capture d'écran n°11

➢ Pour ordonner la liste par Quantite décroissant -1 pour décroissant et 1 pour croissant. :

En SQL :

```
SELECT * FROM ARTICLE ORDER BY Quantite DESC;
```

En Mongo :

```
db.ARTICLE.find().sort({ "Quantite" : -1 });
```

Capture d'écran pour le test de validité de la requête dans notre application :

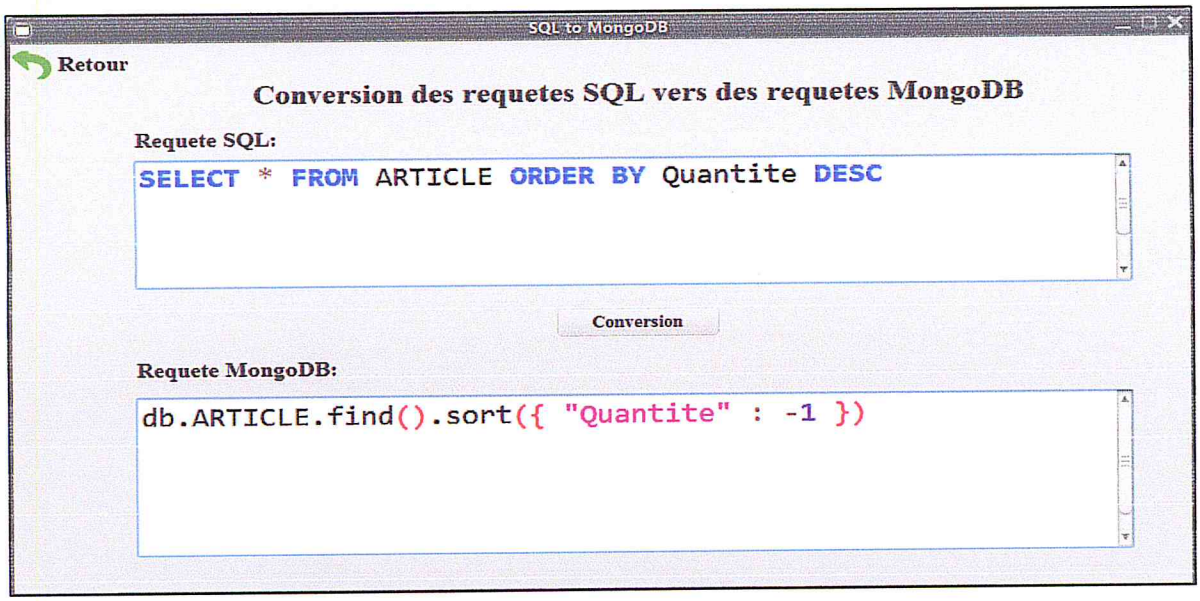


Figure 54 : Capture d'écran n°12

➢ Requête de jointure :

En SQL :

```
SELECT ClientNom FROM COMMANDE JOIN CLIENT
WHERE CLIENT.ClientId = COMMANDE.ClientNum ;
```

En Mongo :

```
db.COMMANDE.aggregate([{
 $lookup:{
form : CLIENT,
localField : ClientNum,
foreignField : ClientId,
as : newCLIENT } }
```

Capture d'écran pour le test de validité de la requête dans notre application :

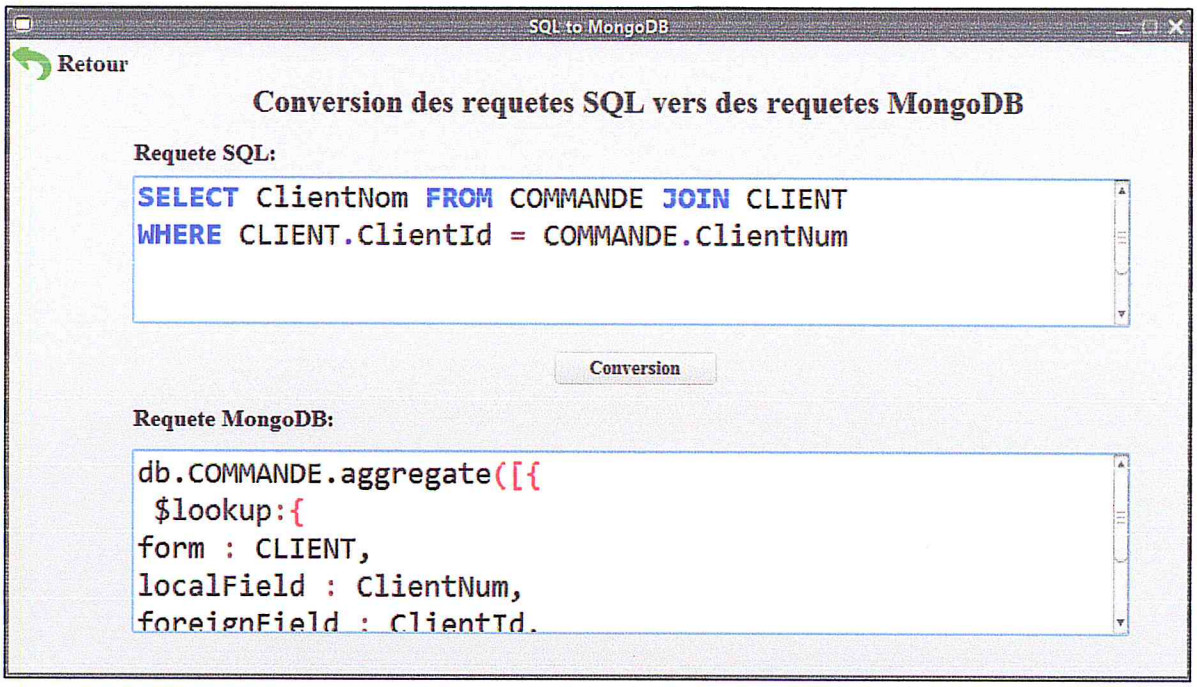


Figure 55 : Capture d'écran n°13

- **Opération UPDATE :**

La commande **update** permet d'effectuer des modifications sur des documents existants. Très souvent cette commande est utilisée avec des opérateurs pour spécifier les documents qui doivent porter la ou les modifications.

➢ Remplacer tous les noms des clients (ClientNom) "AMINE" par "FAYCEL" :

En SQL :

```
UPDATE CLIENT SET ClientNom = 'FAYCEL' WHERE
ClientNom = 'AMINE';
```

En Mongo :

```
db.CLIENT.update({"prenom":"FAYCEL"},{$set:{"prenom":"
AMINE"}},{multi:true})
```

Capture d'écran pour le test de validité de la requête dans notre application :

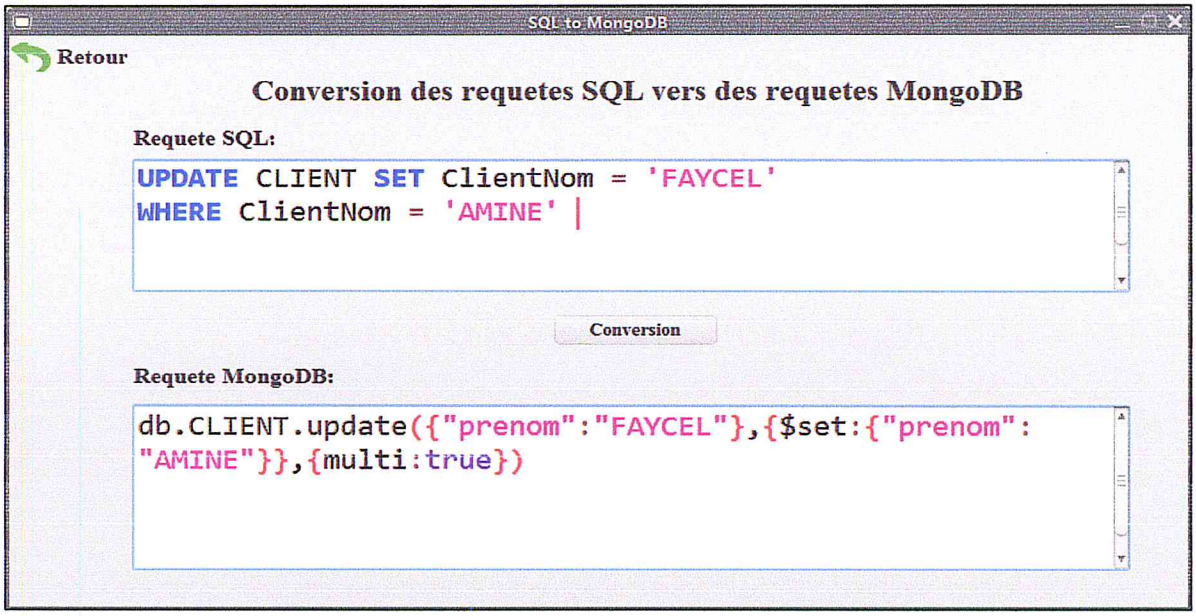


Figure 56 : Capture d'écran n°14

- Remplacer tous les valeurs de l'ArticleTVA de la collection ARTICLE dont l'ArticleNom corresponde à "RAM 8GO" :

En SQL :

```
UPDATE ARTICLE SET ArticleTVA = '25' WHERE
ArticleNom = 'RAM 8GO' ;
```

En Mongo :

```
db.ARTICLE.update({ ArticleNom : "RAM 8 GO" },
{ $set: {ArticleTVA:25 } })
```

Capture d'écran pour le test de validité de la requête dans notre application :

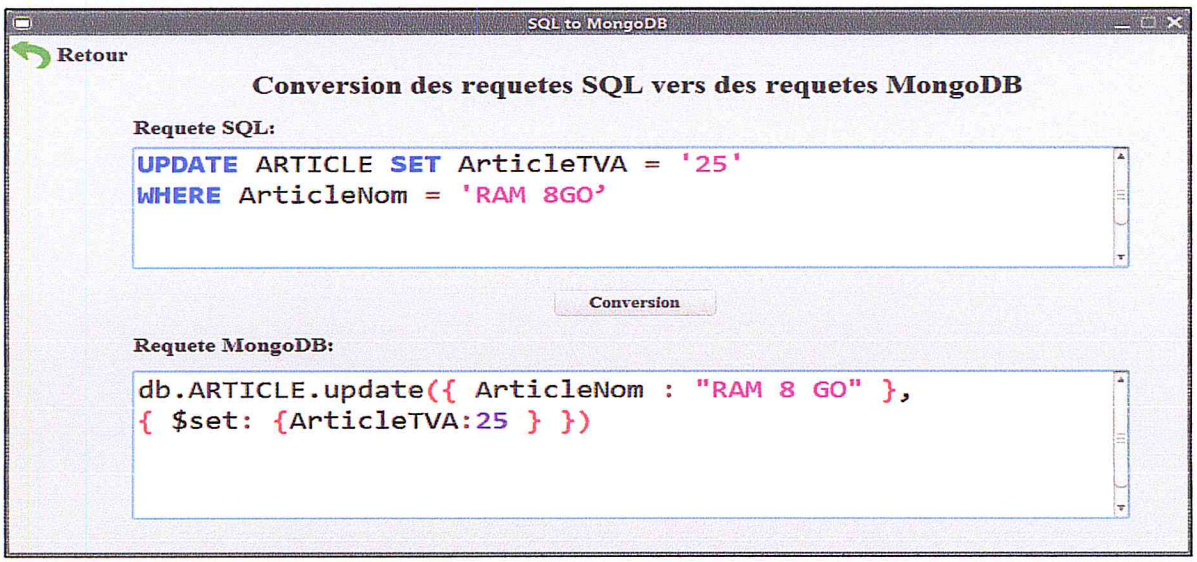


Figure 57 : Capture d'écran n°15

- **Opération de Suppression :**

La Fonction remove() en MongoDB permet de supprimer des documents dans une collection. En utilisant cette fonction associée à d'autres operateurs il est possible de sélectionner les documents concernés qui seront supprimées.

➢ Supprimer tous les articles dont l'ArticleNom corresponde à "Disque 1TO" :

En SQL : **DELETE FROM ARTICLE WHERE `ArticleNom` = 'Disque 1TO' ;**

En Mongo : **db.ARTICLE.remove({ ArticleNom : "Disque 1TO" })**

Capture d'écran pour le test de validité de la requête dans notre application :

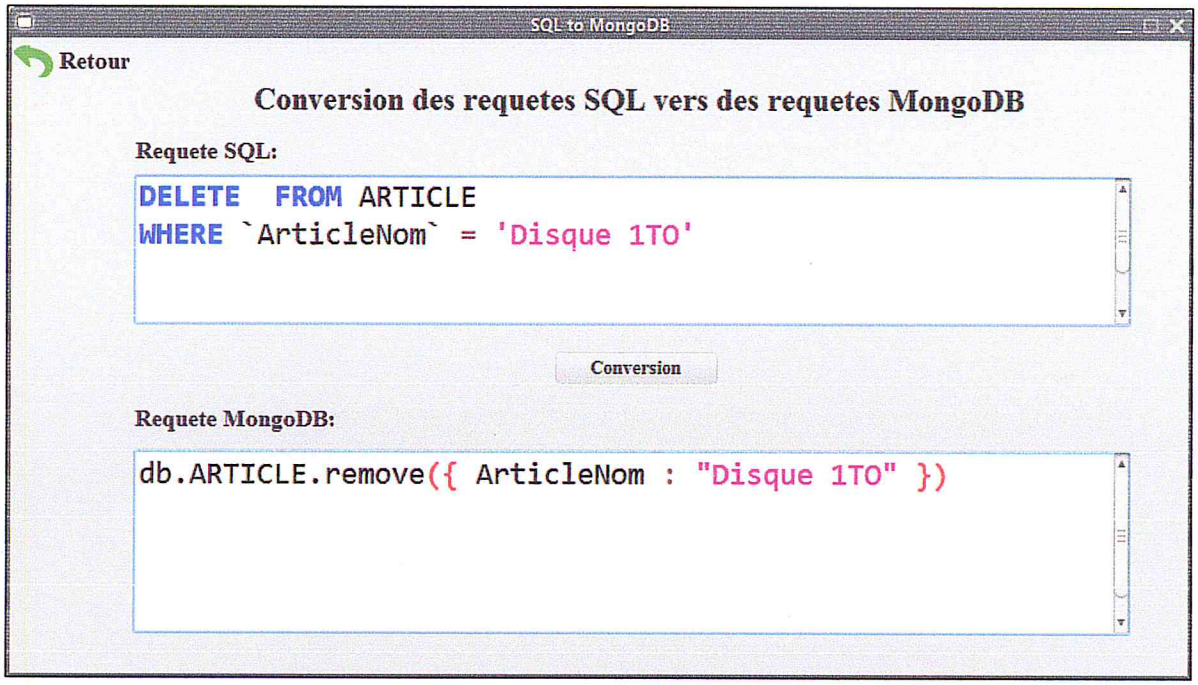


Figure 58 : Capture d'écran n°16

➢ En souhaite supprimer les articles dont leurs Quantité est inférieur ou égale a 2 :

En SQL : **DELETE FROM ARTICLE WHERE ` Quantite ` <= 2 ;**

En Mongo : **db.ARTICLE.remove({ Quantite : { $lte : 2 } }) ;**

Capture d'écran pour le test de validité de la requête dans notre application :

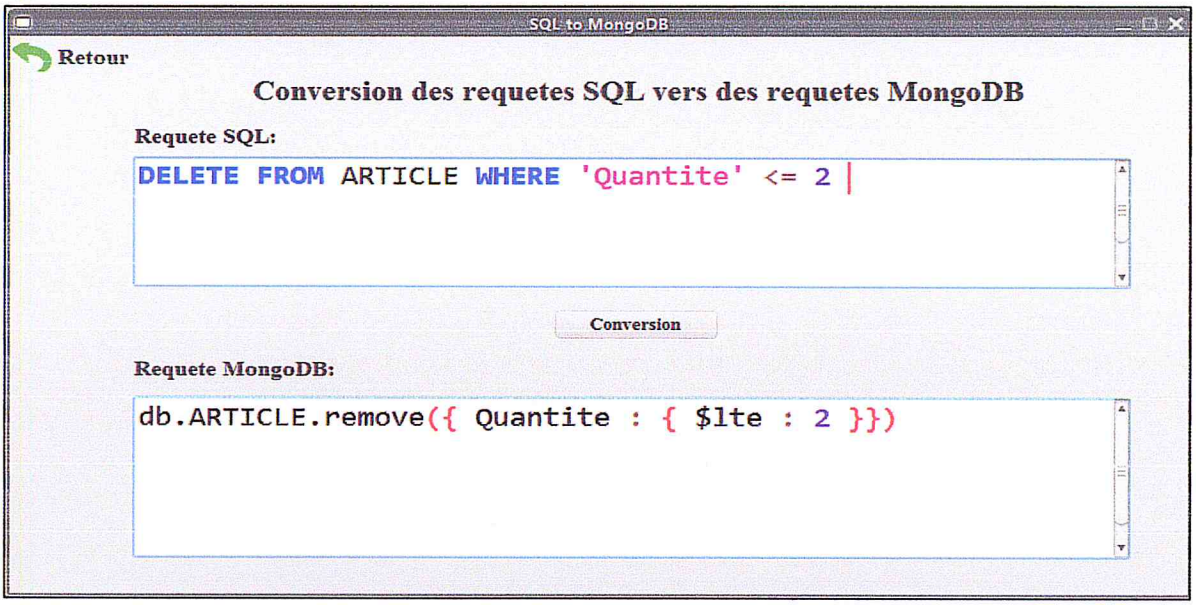


Figure 59 : Capture d'écran n°17

- En souhaite supprimer les articles dont leurs ArticleNom corresponde à "Disque 1TO" et leur quantité est inferieurs a 10 :

En SQL :

```
DELETE FROM ARTICLE WHERE
`ArticleNom` = 'Disque 1TO' AND `Quantite`< 10 ;
```

En Mongo :

```
db.ARTICLE.remove( { "ArticleNom" : "Disque 1TO" ,
"Quantite" : { $lt : 10 } } );
```

Capture d'écran pour le test de validité de la requête dans notre application :

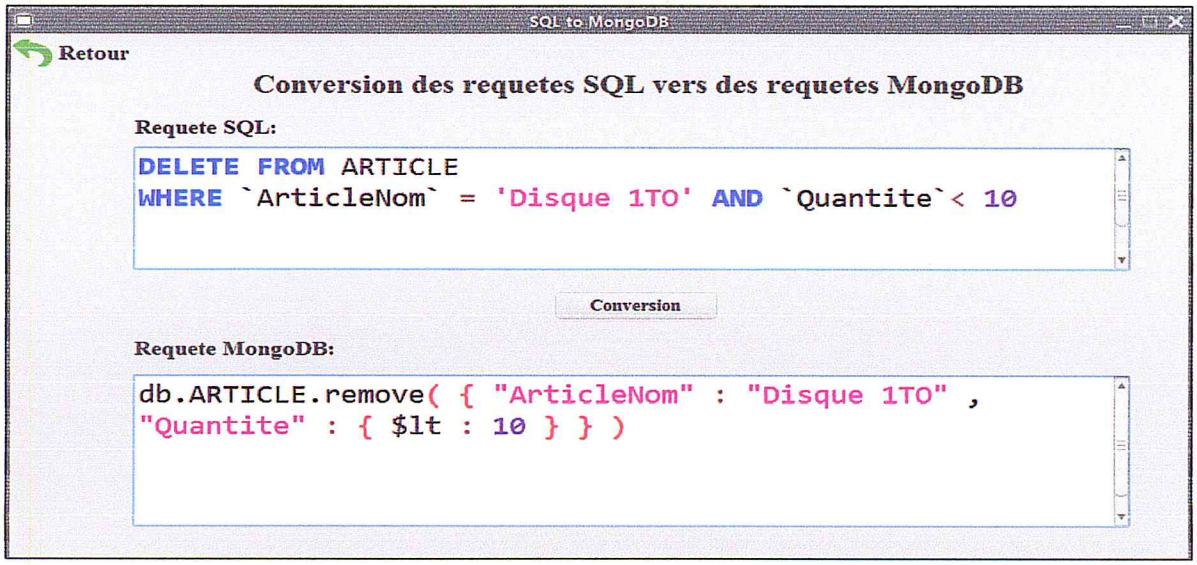


Figure 60 : Capture d'écran n°18

- Supprimer la collection (Table) :

En SQL :

```
DROP TABLE `COMMANDE` ;
```

En Mongo :

```
db.COMMANDE.drop()
```

Capture d'écran pour le test de validité de la requête dans notre application :

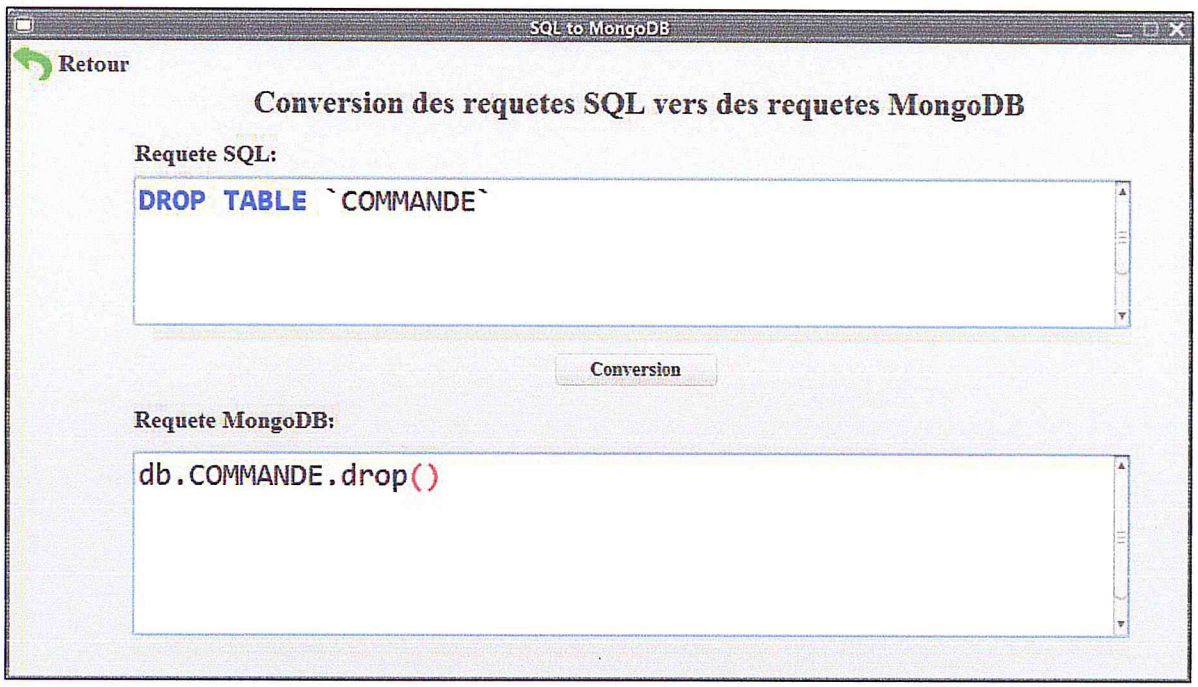


Figure 61 : Capture d'écran n°19

- Supprimer la base de données :

En SQL :

```
DROP DATABASE `VENTES` ;
```

En Mongo :

```
use VENTES
db.runCommand({dropDatabase: 1});
```

Capture d'écran pour le test de validité de la requête dans notre application :

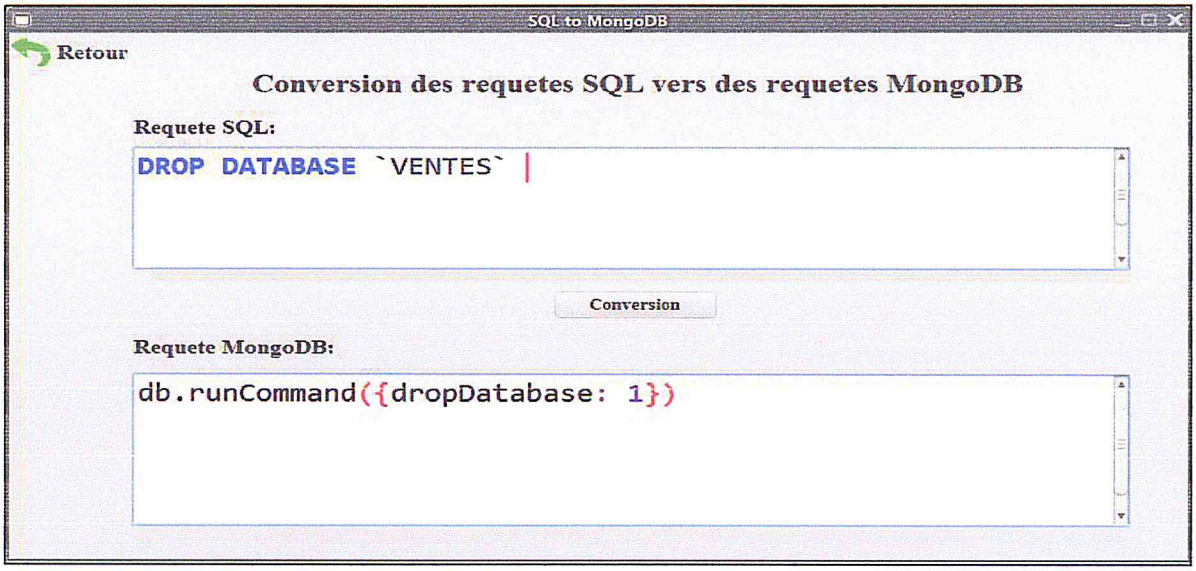


Figure 62 : Capture d'écran n°20

4. Conclusion

Nous avons présenté dans ce chapitre l'environnement et les outils de développement utilisée dans notre travail, par la suite nous avons testé notre moteur de mapping en prenant un exemple d'une base de données relationnelle sur laquelle nous avons fait des opérations de mise-à-jour et de recherche après sa création et son remplissage. Notre système a pu convertir la base de données ainsi que toutes les opérations SQL en MongoDB, ce qui valide le travail effectué.

Conclusion

Conclusion

Actuellement, le NoSQL est une technologie qui émerge en puissance, il est mis en 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.

Plusieurs solutions NoSQL basées sur des architectures différentes sont développées dans tous les secteurs. La tendance vers une favorisation d'une solution NoSQL précise est loin d'être réalisée à cause du nombre important des solutions existantes et l'absence de normalisation. Actuellement, plusieurs solutions open-source et payantes sont présentées aux acteurs des différents secteurs liés au Big Data, qui mettent les utilisateurs devant un embarras dans le choix du modèle approprié par rapport à leur environnement d'exploitation.

L'objectif ciblé de ce PFE consistait à proposer des règles de mapping du schéma relationnel au schéma NoSQL (MongoDB). Une fois les règles de passages établies, la recherche peut également être étendue en proposant la réécriture de requêtes de SQL dans RDB en requêtes NoSQL.

Enfin, il s'agira là, d'une démarche puissante qui aidera non seulement, à faire connaître la technologie NoSQL, mais aussi, à vulgariser l'utilisation du NoSQL qui reste malgré tout peu connu. On peut estimer que l'objectif tracé au préalable a été atteint, néanmoins plusieurs axes de recherches peuvent être développés dans le futur, comme les transactions et les triggers qui n'a pas été traité dans ce travail car il n'y a pas des solutions existantes, aussi on veut généraliser la migration vers les bases de données NoSQL orientées document et pourquoi pas vers tout les bases de données NoSQL au plus tard. En plus de ça notre application est open source et elle sera posée aux grand publique pour a portée des nouvelles modifications et aide à la contribution et l'évolution des bases de données NoSQL.

Références

REFERENCES BIBLIOGRAPHIQUES

[1] A. BACHIR et M. BOUDEFLA, Migration de la base de données des passeports biométriques sous Oracle vers une base de données NoSQL MongoDB, Mémoire master informatique, UNIVERSITE DE Abou Bakr Belkaid– Tlemcen, Septembre 2016.

[2] AMARA Manel Warda, Etude comparative des bases de données NoSQL, Mémoire master informatique, UNIVERSITE DE Abou Bakr Belkaid– Tlemcen, Juin 2015.

[3] KOUEDI Emmanuel, Approche de migration d'une base de données relationnelle vers une base de données NoSQL orientée colonne, Mémoire master informatique, UNIVERSITE DE YAOUNDE I, mai 2012.

[4] Adriano Girolamo Piazza, NoSQL Etat de l'art et benchmark, Travail de Bachelor réalisé en vue de l'obtention du Bachelor HES, Haute École de Gestion de Genève, octobre 2013.

[5] Lionel HEINRICH, Not Only SQL, Travail de Bachelor réalisé en vue de l'obtention du Bachelor HES, Haute École de Gestion de Genève, octobre 2012.

[6] M. Potey, M. Digrase, G. Deshmukh, M. Nerkar, Database Migration from Structured Database to non-Structured Database, Journal International de Computer Applications, 0975-8887.

[7] M. Hanine, A. Bendarag, O. Boutkhoum, Data Migration Methodology from Relational to NoSQL Databases, International Journal of Computer, Electrical, Automation, Control and Information Engineering, 9:12, 2015.

[8] Z. Gansen, L. Qiaoying, L. Libo, L. Libo, Schema Conversion Model of SQL Database to NoSQL, Ninth International Conference on P2P, 2014.

[9] C. Wu-Chun, L. Hung-Pin, C. Shih-Chang, J. Mon-Fong, C. Yeh-Ching, JackHare: a framework for SQL to NoSQL translation using MapReduce, Springer Science+Business Media New York, 2013.

REFERENCES WEB

[w1] https://docs.mongodb.com Consulté le 20-03-2017

[w2] https://mongoteam.gitbooks.io Consulté le 20-03-2017