BIG DATA



Par  : Abdelmoumni Ouadie

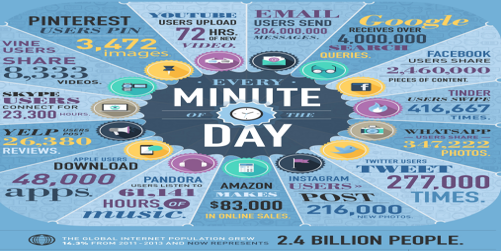
Filière  : Mastère Qualité du logiciel

Année :2017/2018

**Contexte :**

Les volumes de données à gérer sont sans précédent, ce qui implique :

* Données hétérogènes, complexes et souvent liées :
  + Produites par des applications parfois différentes,
  + Par des utilisateurs différents,
  + Avec des liens explicites (par exemple citations, ancres url, etc) ou implicites (à extraire ou à apprendre)
* Besoin de distribuer les calculs et les données :
  + Comme plusieurs serveurs/clusters, besoin d'algorithmes permettant le calcul et la distribution des données à large échelle
* Nombreux serveurs/clusters :
  + Un serveur unique ne peut stocker cette quantité d'information, garantir des temps d'accès pour grand nombre d'utilisateur, faire des calculs rapides, etc
* La croissance de la quantité des données a augmenté de façon fulgurante
* La quantité de données digitales produites double tous les 2 ans.
* En d'autres termes, on a produit autant de données digitales ces 2 dernières années que tout ce qui a été produit auparavant
* Le développement du marché des objets connectés et les pratiques de géolocalisation accroissent  considérablement le volume de données collectées disponibles.



### Exemple des énormes quantités de données :

Data centers de quelques grands acteurs du Big Data :

* Google DataCenter : 70000 servers/data center et 16 data centers, ~1M de serveurs
* Facebook : 5 data centers
* Amazon : 7 data centers, 450 000 severs
* Microsoft : environ 1M serveurs

**Problématique :**

La notion de big data est un concept qui devient plus popularisé, traduisant ainsi le fait que le monde a à faire à des volumes de données (data) à traiter de plus en plus Grands et présentant de forts enjeux (commerciaux ,marketing…).

Aujourd'hui, avec le Web 2.0, nous sommes confrontés à une croissance incomparable de la quantité de données que nous traitons, et qui augmente très rapidement comme on peut le voir sur des sites comme Facebook ,ebay…etc)

**C’est quoi le ‘Big Data’:**

Les big data c’est les grosses données, méga donnée ou encore données massives, désignent l’ensemble de données qui deviennent tellement volumineuses qu’ils en devient difficiles de travailler avec des outils classiques de gestion de base de données ou de gestion de l’information ce qui mène à l'utilisation d'outils informatiques de plus en plus performants pour les gérer

Le big data peut-être défini par ses trois caractéristiques majeures, toutes commençant par la lettre V : le Volume, la Vélocité, la Variété,

### **Volume :**



La première caractéristique majeure du big data est l'explosion du Volume de données, les entrepôts sont conçus pour gérer des Go ou To de données alors que la croissance exponentielle des données nous conduit aux Po ou Eo

Ce développement exponentiel oblige les entreprises à changer leurs outils classiques de stockage de données. Pour avoir une idée de l'ampleur du phénomène

Les avancées technologiques telles que l’arrivée des objets connectés et la multiplication des infrastructures d'échanges de données permettent en partie d'expliquer ce phénomène.

Aussi la multiplication de l'accès à internet partout dans le monde, l'automatisation des systèmes (administratifs entre autres), ainsi que la tendance grandissante du recours aux outils informatiques dans la société sont des facteurs d'accélération du développement du big data.

### **Vélocité :**

La vélocité c’est la fréquence à laquelle les données ont pu être générées, capturées, partagées et mises à jour

Les données sont créées de plus en plus vite et nécessitent des traitements en temps-réel. La vélocité du big data est une caractéristique que l'on retrouve notamment dans le phénomène d'enchère en temps réel, ou *Real-Time Bidding (RTB)* (qui nécessite un traitement en temps réel : traitement dynamique de données)

Des flux croissants de données doivent être analysés en quasi-temps réel pour répondre aux besoins des processus chrono-sensibles (comme les systèmes mis en place par les bourses ,ils doivent être à la fois performants et rapide pour pouvoir traiter la donnée en temps réel sinon une perte énorme de liquidité s’en suivrait

### **Variété :**



Le volume énorme de données implique une variété de données Il ne s'agit pas de **données relationnelles** traditionnelles, ces données sont brutes, semi-structurées voire non structurées

Les données étant le plus souvent reçues de façon hétérogène et non structurée, elles doivent être traitées et catégorisées avant d'être analysées et utilisées dans la prise de décision.

Ce sont des données complexes provenant du web (Web-Mining),elles peuvent être de tous les types : données de localisation, données démographiques, données relatives aux réseaux sociaux, historiques de navigation, achats en ligne



### **Et pas que… :**

Un quatrième et cinquième aspect peuvent être ajouté aux trois caractéristiques précédentes :

* **La Véracité :**Il est nécessaire de vérifier l'exactitude des données reçues. Même si elles paraissent homogènes et cohérentes entre elles, ces données peuvent être incomplètes et/ou inexactes. Fait référence à la fiabilité de la donnée. Avec autant de formes de grosse donnée, la qualité et la précision sont moins vérifiables
* **La Valeur** :il s’agit d’être capable de se concentrer sur les données ayant une réelle valeur et étant actionnables. C’est bien beau d’avoir accès aux grosses données mais encore faut-il les transformer en valeur, sinon c’est inutile vu le prix que ça va coûter si la valeur de ces données n’en vaut pas la peine

L’incapacité à gérer de **très grands volumes** de données à des débits extrêmes font que les Les SGBD relationnels montrent leur limite avec de très hauts débits et des données de types qui ne sont pas compatibles avec les schémas rigides du modèle relationnel (**données non structurés** )

🡺 En Big Data, non seulement on gère des quantités très importantes de données, mais des données qui peuvent aussi avoir une taille très importante, et souvent on veut un résultat très rapidement ! Les SGBDR traditionnels atteignent ici leurs limites.

**ACID VS BASE :**

**Propriétés ACID :**

Le concept ACID fonctionne bien pour les bases de données relationnelles

– **Atomicity** : tout ou rien, une modification des données doit être réalisée dans son intégralité ou pas du tout.

– **Consistency**: les données doivent toujours être cohérentes entre elles, même en cas d'erreur. Dans ce cas là, on effectuera un RollBack

– **Isolation** :  Pas d'interférences entre les transactions. Utilisation des verrous et des points de synchronisation

– **Durability** : Lorsqu'une transaction s'est achevée, avec succès (Commit) ou en erreur (Rollback), les données doivent être dans un état stable et cohérent.

**Propriétés Base :**

Cependant, avec le développement du Cloud computing et des systèmes distribués, de nouvelles bases de données ont été conçues pour répondre à des contraintes différentes.

– **Basically Available** : le système doit toujours être accessible (ou indisponible sur de courtes périodes)

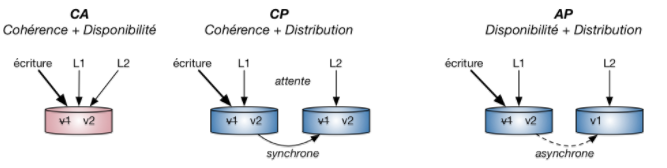
– **Soft state** : l’état de la BD n’est pas garanti à un instant donné (les mises à jour ne sont pas immédiates : cf. cohérence à terme)

– **Eventual consistency** : la cohérence des données à un instant donné n’est pas primordiale (mais assurée à terme : verrouillage optimiste en reportant à plus tard la vérification de l’intégrité)

**Théorème de cap :**

Le théorème CAP affirme que tout système à état partagé en réseau ne peut avoir que deux des trois propriétés désirables. Néanmoins, en gérant explicitement les partitions, les concepteurs peuvent optimiser la cohérence et la disponibilité, atteignant ainsi un compromis des trois.

* Cohérence (C), équivalent à n'avoir qu'une seule copie à jour des données ;
* Haute disponibilité (A, NdT: Availability) des données (pour les mises à jour) ;
* Tolérance au partitionnement en sous-réseaux (P, NdT: network Partition)



Prenons le couple **CA** (Consistency-Availability), il représente le fait que lors d'opérations concurrentes sur une même donnée, les requêtes L1 et L2 retournent la nouvelle version (v2) et sans délai d'attente. Cette combinaison n'est possible que dans le cadre de bases de données [transactionnelles](http://sys.bdpedia.fr/conc.html) telles que les SGBDR.

Le couple **CP** (Consistency-Partition Tolerance) propose maintenant de distribuer les données sur plusieurs serveurs en garantissant la tolérance aux pannes (réplication). En même temps, il est nécessaire de vérifier la cohérence des données en garantissant la valeur retournée malgré des mises à jour concurrentielles. La gestion de cette cohérence  nécessite un protocole de synchronisation des réplicas, introduisant des délais de latence dans les temps de réponse (L1 et L2 attendent la synchronisation pour voir v2). C'est le cas de la base NoSQL *MongoDB*.

Le couple **AP** (Availability-Partition Tolerance) à contrario s'intéresse à fournir un temps de réponse rapide tout en distribuant les données et les réplicas. De fait, les mises à jour sont asynchrones sur le réseau, et la donnée est "*Eventually Consistent*" (L1 voit la version v2, tandis que L2 voit la version v1). C'est le cas de *Cassandra* dont les temps de réponses sont appréciables, mais le résultat n'est pas garanti à 100% lorsque le nombre de mises à jour simultanées devient important.

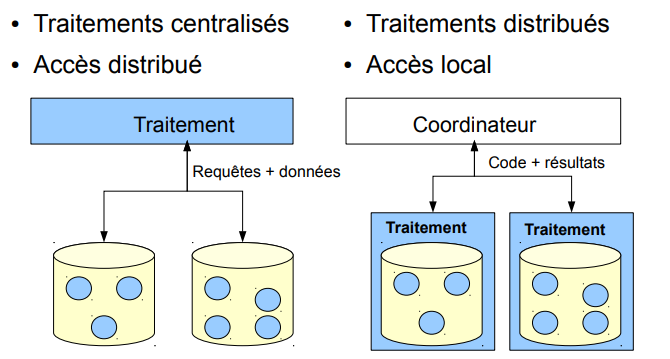
**La solution : Le NoSQL**

## **Définition :**

C’est une Catégorie de SGBD s'affranchissant du modèle relationnel des SGBDR,non basé sur leur architecture , L'explicitation du terme la plus populaire de l'acronyme est *Not only SQL*

Ces bases de données visent d’énormes quantités de données avec une structuration de données faible ou quasi inexistante et avec une possibilité de redondance de données

## **Différences entre les SGDB et les bases de données NoSQL :**



## **Types de base de données NoSQL :**

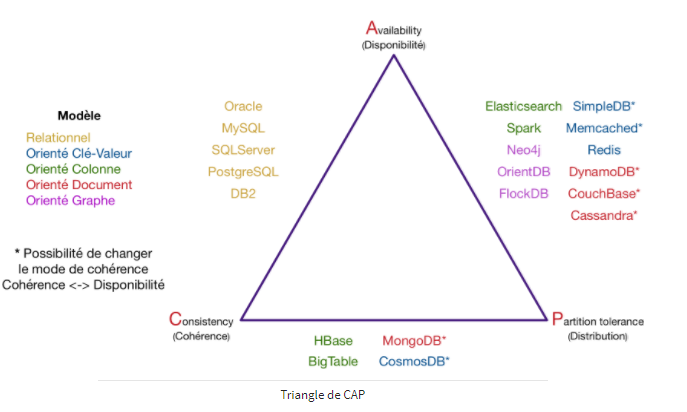
Il existe quatre types de SGBD NoSQL :

1. Orienté document (MongoDB, coucheDB)
2. Clé / valeur (Redis,MemBase,Kyoto)
3. Orienté colonne (Cassandra, Hbase,cloudData...)
4. Orienté graphe (Neo4J,FlockDB ...)

Durant les Chapitres à venir on traitera une base de données de chaque type pour bien comprendre leur architecture et leur apport.

**Classement des bases de données sur le triangle de CAP :**

🡺Grâce à ce théorème de CAP, il est alors possible de classer toutes les bases de données en les plaçant sur le "triangle de CAP", tout en ajoutant des codes couleurs pour chaque modèle de stockage présenté dans le chapitre précédent



Nous pouvons constater que les bases de données relationnelles se retrouvent sur la face CA du triangle, combinant disponibilité et cohérence. Nous retrouvons bien MongoDB pour le couple CP (cohérence et distribution) mais également les solutions orientées colonnes comme HBase ou BigTable.

Le couple AP (Disponibilité et distribution) regroupe le plus grand nombre de solutions NoSQL. En effet, la plupart cherchent les performances en relâchant volontairement la cohérence. Nous y retrouvons principalement des solutions orientées clé-valeur et graphes, mais également orientées documents (clé-valeur étendu).

Certaines solutions proposent également de modifier la politique de gestion de la concurrence (DynamoDB, CouchBase, Cassandra, MongoDB, CosmosDB...), dans ce cas, ils changent simplement de face sur ce triangle en passant de CP à AP.

L'avantage de ce triangle CAP est de fournir un critère de choix pour votre application. Vous pouvez ainsi vous reposer sur vos besoins en terme de fonctionnalité, de calculs et de cohérence. Le triangle servira de filtre sur les myriades de solutions proposées.

# Cassandra



## **Base de données orientées colonnes :**

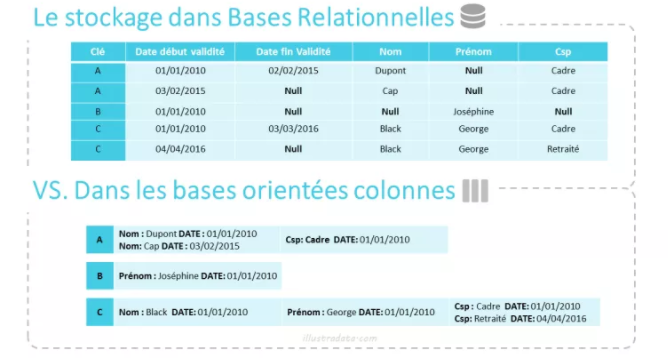
Ces bases de données NoSQL sont celles se rapprochant le plus des bases de données classiques (SGBDR). En effet, on y retrouve le principe de “table” avec des lignes et des colonnes, mais elles ont deux principales grosses différences :

1. **Les colonnes sont dynamiques.** Au sein d’une même table deux individus peuvent ne pas avoir le même nombre de colonnes car**les valeurs nulles ne sont pas stockées** (ce qui est le cas dans les SGBDR relationnels).

* Cette propriété permet de libérer de la place de stockage et d’améliorer les performances de traitement car la volumétrie de données à traiter est plus faible.
* Avec cette propriété, on a plus tendance également à ne créer qu’une seule table contenant toutes les données (et donc colonnes) dont on a besoin et non plus une multitude de tables comme c’est le cas dans les modèles relationnels. Cette absence de ‘jointure’ entre les tables améliore également les performances.

1. **L’historisation des données se fait à la valeur et non pas à la ligne comme dans les SGBDR** cela empêche le stockage d’informations en doublon et de ce fait allège considérablement la base de données et les temps de calcul.

🡺Pour mieux comprendre, ci-dessous un exemple explicatif des différences entre un SGBR et une base de données orientée colonnes



🡺On constate qu’il n’y a pas de redondance côté base de données orienté colonnes puisque Black et George sont stockés qu’une seule fois et c’est les colonnes qui changent qui sont restockées contrairement aux SGBDR qui recalquent la même ligne en changeant que les champs à changer.

**Récapitulatif**

* Les données sont stockées par colonne
* Facile d'ajouter une colonne, il est couteux d'ajouter une ligne
* Possibilité de compresser les données d'une colonne
* Proche d'un SGBDR mais avec un nb de colonnes dynamique
* Nb de colonnes variables par enregistrement
* Ex : Cassandra, HBase

### Avantages des bases de données orientées colonnes :

* Flexibilité
* Temps de traitement
* Non-stockage des valeurs null
* Historisation à la valeur
* Requêtage en temps Réel
* Données semi-structurés
* Indexation incluses (colonnes)
* Scalabilité horizontale

### Inconvénients des bases de données orientées colonnes :

* Peu adapté aux données interconnectés ou complexes (Le travail des bases de données orienté graphes).
* Non-adaptée pour les données non-structurées.

## **Cassandra :**

Cassandra est une base de données NOSQL (Not Only SQL) orientée colonne, développée par Facebook en 2007 la messagerie interne. En 2008, le projet est cédé à la fondation Apache et devient "top-level-project" à partir de 2010. Cassandra est alors enrichie et de nouvelles fonctionnalités y sont ajoutées (CQL).

Le projet Cassandra est optimisé pour gérer de grandes quantités de données et connais une utilisation dans certains gros projets web tels que Facebook, Twitter ou Digg.com.

C’est une base de données distribuée qui permet de stocker une grande quantité de données grâce à sa scalabilité horizontale.

Scalabilité horizontale : la possibilité offerte par l'architecture Cassandra d'ajouter de nouvelles machines qui sont appelées des nœuds.

### Avantages :

* **Évolutivité élastique - Cassandra est hautement évolutive;** il permet d'ajouter plus de matériel pour accueillir plus de clients et plus de données selon l'exigence.
* **Toujours sur l’architecture -** Cassandra n'a pas de point de défaillance unique et il est disponible en permanence pour les applications critiques de l' entreprise qui ne peut pas se permettre un échec.
* **Rapide des performances linéaires échelle -** Cassandra est linéairement évolutive, à savoir, il augmente votre débit que vous augmentez le nombre de nœuds dans le cluster. Par conséquent, il maintient un temps de réponse rapide.
* **Stockage de données flexible -** Cassandra accueille tous les formats de données possibles, y compris: structurée, semi-structurée, et non structurées. Il peut accueillir dynamiquement des modifications à vos structures de données selon vos besoins.
* **Distribution de données facile -** Cassandra fournit la flexibilité pour distribuer des données où vous avez besoin par la réplication des données sur plusieurs centres de données.
* **Écriture rapide - Cassandra a été conçu pour fonctionner sur du matériel de base pas cher.** Il effectue des écritures fulgurantes et peut stocker des centaines de téraoctets de données, sans pour autant sacrifier l'efficacité de lecture.

### Inconvénients :

* Difficulté d’utilisation
* Pas d’interface graphique robuste

## **L’architecture de Cassandra :**

### Cluster :

Un cluster est un regroupement de plusieurs nœuds (serveur physique) qui communiquent entre eux pour la gestion de données.

Cassandra est une base de données contenue dans un cluster.

Comme de nombreuse base NoSQL, les données sont réparties sur plusieurs nœuds et peuvent être répliquée sur 1 à N nœuds.

Un utilisateur peut se connecter sur n'importe quel nœud et accéder à l'ensemble des données.

### Œil sur l’architecture De Cassandra :

Une instance Cassandra est une collection de nœuds indépendants qui sont configurés ensemble pour former un cluster.

Dans un cluster, tous les nœuds sont égaux, ce qui signifie qu'il n'y a pas de nœud maître ou un processus centralisant leur gestion.

En fait, Cassandra utilise un protocole appelé Gossip afin de découvrir la localisation et les informations sur l'état des autres nœuds du cluster. Le protocole Gossip est un protocole de communication de type « *peer-to-peer »* dans lequel les nœuds échangent périodiquement des informations sur leur état mais également sur ce qu'ils savent des autres nœuds.

Pour être plus précis, le processus s'exécute toutes les secondes afin d'échanger les messages avec au plus trois autres nœuds du cluster. De plus, une version est associée à ces messages afin de permettre d'écraser les informations plus anciennes.

Ainsi, quand un nœud démarre, il regarde dans son fichier de configuration les points de ralliement (SEED) qui devront au moins être contactés une fois.

Cependant, afin d'éviter un partitionnement, tous les nœuds du cluster doivent disposer de la même liste de nœuds dans leur fichier de configuration.

La détection des échecs est une méthode pour déterminer localement si un autre nœud est accessible ou pas. En outre, les informations récoltées par ce mécanisme permettent à Cassandra d'éviter d'émettre des requêtes aux nœuds qui ne sont plus accessibles.

En fait, ce mécanisme fonctionne sur le principe de *heartbeat* soit de manière directe (en récoltant les informations directement des nœuds) soit de manière indirecte (en récoltant les informations par l'intermédiaire de la connaissance des autres nœuds).

Lorsqu'un nœud est déclaré comme inaccessible, les autres nœuds stockent les messages susceptibles d'avoir été manqués par ce nœud. Cependant, il peut arriver qu'entre le moment où le nœud devient inaccessible et le moment où sa perte est détectée, un laps de temps s'écoule et qu'ainsi, les réplicas ne soient pas conservés. En outre, si le nœud vient à être indisponible pendant une période trop importante (par défaut, une heure), alors les messages ne sont plus stockés. C'est pour cette raison qu'il est conseillé d'exécuter régulièrement l'outil de réparation des données (NODE\_REPAIR).

### Partitionnement des données dans Cassandra :

Il est possible de configurer le partitionnement pour une famille de colonnes en précisant que l'on veut que cela géré avec une stratégie de type Ordered Partitioners.

Ce mode peut, en effet, avoir un intérêt si l'on souhaite récupérer une plage de lignes comprises entre deux valeurs (chose qui n'est pas possible si le hash MD5 des clés des lignes est utilisé).

Cependant, il est conseillé d'utiliser plutôt une deuxième clé d'indexation positionnée sur la colonne contenant les informations voulues. En effet, utiliser la stratégie Ordered Partitionners a les conséquences suivantes :

* L’écriture séquentielle peut entraîner des *hotspots :* si l'application tente d'écrire ou de mettre à jour un ensemble séquentiel de lignes, alors l'écriture ne sera pas distribuée dans le cluster ;
* Un *overhead* accru pour l'administration du *load balancer* dans le cluster : les administrateurs doivent calculer manuellement les plages de jetons afin de les répartir dans le cluster ;
* Répartition inégale de charge pour des familles de colonnes multiples.

### La répartition des données dans Cassandra :

La réplication est le processus permettant de stocker des copies des données sur de multiples nœuds afin de permettre leur fiabilité et la tolérance à la panne. Quand un *keyspace* est créé dans Cassandra, il lui est affecté la stratégie de distribution des réplicas, c'est-à-dire le nombre de réplicas et la manière dont ils sont répliqués dans le cluster.

La stratégie de réplication repose sur la configuration du cluster Snitch afin de déterminer la localisation physique des nœuds ainsi que leur proximité par rapport aux autres.

Il est souvent fait référence au facteur de réplication (réplication factor que nous nommerons RF par la suite) pour parler du nombre total de réplicas dans le cluster.

Aussi :

* Un facteur de réplication de 1 signifie qu'il n'y a qu'une seule copie de chaque ligne .
* Un facteur de réplication de 2 signifie qu'il existe deux copies de chaque ligne ;
* ...

Tous les réplicas possèdent la même importance : il n'y a pas de réplicas principaux ou maîtres du point de vue de la lecture et de l'écriture.

Ainsi, la règle générale est que le facteur de réplication ne doit pas être supérieur au nombre de nœuds dans le cluster. Cependant, il est possible d'augmenter le facteur de réplication puis d'ajouter le nombre de nœuds désirés à postériori. À noter toutefois que lorsque le facteur de réplication est supérieur au nombre de nœuds, alors les écritures ne sont plus prises en compte tandis que l'opération de lecture reste maintenue tant que le degré de consistance est respecté.

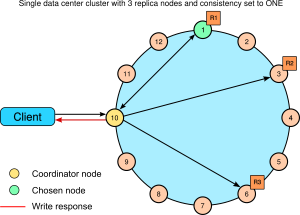
Concernant les stratégies de distribution des réplicas, cela permet de jouer sur la façon dont les réplicas sont répartis dans le cluster pour un keyspace. Cette stratégie est à renseigner lors de la création de la keyspace.

De plus, il est possible de configurer la manière dont les nœuds sont groupés ensemble dans la topology du réseau global. Cet élément de configuration correspond au snitch et est associé au cluster. Cassandra utilise alors cette information pour router les requêtes entre les nœuds.

### Cassandra avec une application cliente :

Tous les nœuds de Cassandra sont égaux. Ainsi, une demande de lecture ou d'écrire peut interroger indifféremment n'importe quel nœud du cluster. Quand un client se connecte à un nœud et demande une opération d'écriture ou de lecture, le nœud courant sert de coordinateur du point de vue du client.

Le travail du coordinateur est de se comporter comme un proxy entre le client de l'application et les nœuds qui possèdent la donnée. C'est lui qui a la charge de déterminer quels nœuds de l'anneau devront recevoir la requête.



#### **L’écriture dans Cassandra**

Concernant les requêtes d'écriture, le coordinateur émet la requête à tous les réplicas qui possèdent la ligne à modifier. Aussi longtemps qu'ils sont disponibles, ils reçoivent les demandes d'écriture quel que soit le niveau de consistance demandé par le client. Le niveau de « consistance » d'écriture détermine le nombre de nœuds qui doivent acquitter l'écriture afin de considérer l'écriture comme ayant réussi.

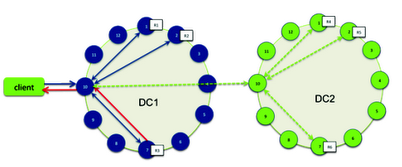
Dans le cas où il existe plusieurs data center deployments, Cassandra optimise les performances d'écriture en choisissant un nœud coordinateur dans chaque data center distant afin de traiter les requêtes des réplicas dans le data center. Le nœud coordinateur contacté par l'application cliente n'a alors qu'à transmettre les requêtes d'écriture à un seul nœud de chaque data center distant.

🡺Si le niveau de consistance choisi est **ONE** ou **LOCAL\_QUORUM**, alors seuls les nœuds du même « data center » que le nœud coordinateur doit acquitter l'écriture.

Cassandra utilise un paramètre de cohérence pour gérer la cohérence des données parce que Les données peuvent être copiées sur plusieurs nœuds au sein du cluster

Il existe plusieurs valeurs possibles pour le Consistency Level, dont les plus importantes :

* **ONE**: un seul nœud est consulté / l'écriture est validée après écriture sur un seul nœud.
* **QUORUM**: (RF/2) +1 nœuds sont consultés / l'écriture est validée après écriture sur (RF/2) +1 nœuds.
* **ALL**: RF nœuds sont consultés / l'écriture est validée après écriture sur RF nœuds



#### **La lecture dans Cassandra**

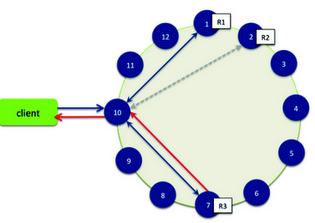
Concernant la lecture, il y a deux types de requêtes de lecture qu'un coordinateur peut émettre à un réplica :

* Une requête de lecture directe. Dans ce cas, le nombre de réplicas contactés par une demande de lecture directe est déterminé par le niveau de consistance spécifié par le client ;
* Une requête de réparation de lecture en tâche de fond. Dans ce cas, elle est envoyée à tous les réplicas additionnels qui n'ont pas reçu de requête directe. Ce type de requête permet de vérifier que la ligne est consistante par rapport aux autres réplicas.

Ainsi, dans un premier temps, le coordinateur contacte les réplicas en fonction du niveau de consistance qui a été spécifié. Ces réplicas sont choisis en fonction de leur capacité à répondre rapidement. Ces derniers répondent avec la donnée demandée et, s'il existe plusieurs réponses, elles sont comparées en mémoire afin de vérifier leur consistance.

Si ce n'est pas le cas, alors c'est le réplica qui est le plus récent (en se basant sur le timestamp) qui est utilisé par le coordinateur pour répondre au client.

En outre, afin de s'assurer que tous les réplicas ont la version la plus récente, le coordinateur les contacte en tâche de fond et compare la donnée de tous les réplicas restants qui possèdent la ligne. Il demande ensuite, éventuellement, une opération d'écriture afin de mettre à jour la donnée. C'est cette opération qui est appelée read repair et qui peut être configurée par famille de colonnes (par défaut, elle est désactivée).



🡺La base de données Cassandra est répartie sur plusieurs machines qui fonctionnent ensemble. Le conteneur externe est connu sous le nom de cluster. Pour la manipulation de l’échec, chaque nœud contient une réplique, et en cas d'échec, la réplique prend en charge. Cassandra arrange les nœuds d'un cluster, dans un format de sonnerie, et attribue des données à leur disposition.

## **Le Modèle de données de Cassandra :**

Le modèle de données de Cassandra s'appuie sur un schéma dynamique, avec un modèle de données orienté colonne.

Cela signifie que, contrairement à une base de données relationnelle, il n'est pas nécessaire de modéliser toutes les colonnes puisqu'une ligne n'a, potentiellement, pas le même ensemble de colonnes.

Les colonnes et leurs métadonnées peuvent être ajoutées par l'application lorsque cela s'avère nécessaire.

Si on réfléchit bien, le modèle de données de Cassandra a été pensé pour répondre à des problématiques de données distribuées et diffère complètement d'un modèle classique de base de données relationnelle où les données sont stockées dans des tables qui sont, dans la plupart des cas, en relation entre elles. En outre, dans un modèle relationnel, les données sont généralement normalisées afin d'éviter la redondance et des jointures sont faites entre les tables sur des clés communes afin de satisfaire les requêtes.

**Squelette de Cassandra :**

On commence par le **Keyspace** qui est le conteneur des données de l'application (un peu comme une database ou un schéma pour une base de données relationnelle). Dans ces keyspaces se trouvent une ou plusieurs familles de colonnes (qui correspondent aux tables en base de données relationnelle).

Puis **les familles de colonnes** qui contiennent des **colonnes** ainsi qu'un ensemble de colonnes connexes qui sont identifiées par une clé de ligne. En outre, chaque ligne d'une famille de colonnes ne dispose pas nécessairement des mêmes colonnes qu'une autre ligne.

Enfin, Cassandra n'impose pas de relations entre les familles de colonnes au sens base de données relationnelles : il n'y a pas de clés étrangères et les jointures entre familles de colonnes ne sont pas supportées.

### **Colonnes :**

Une colonne est la plus petite unité du modèle de données de Cassandra. C'est un triplet contenant un nom, une valeur et un timestamp.Un timestamp sert à déterminer la mise à jour la plus récente

* La taille du nom peut être contenue jusqu'à 64 KO.
* La valeur quant à elle peut contenir 2 GO de données.
* La valeur n'est pas obligatoire
* La case valeur peut contenir plusieurs valeurs. C'est le cas par exemple d'une collection de chaînes de caractères

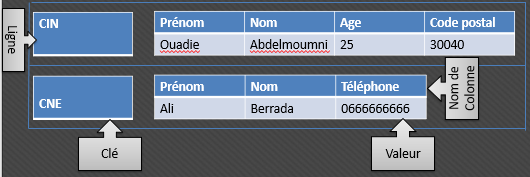


### **Ligne :**

Une ligne c’est un ensemble de colonnes ,elle est identifiée par un clé unique et peut contenir jusqu’à **Deux milliards de colonnes ,**et il est aussi possible d’utiliser des colonnes comme des clés .

Nous montrons sur la figure ci-dessous un exemple de deux lignes. Notez que :

1. Le nombre de colonnes n'est pas identique
2. Les colonnes ne sont pas forcément les mêmes.



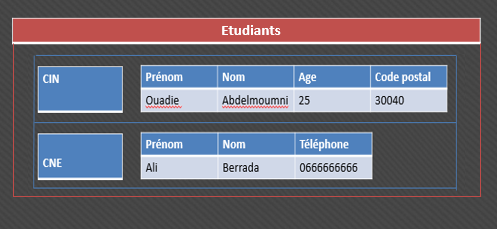
### **Une famille de colonnes :**

Une famille de colonnes ou *column family* en anglais est un regroupement logique de lignes

Lors de la définition d'une famille de colonnes, on peut y ajouter des informations concernant les métadonnées des colonnes. Toutefois c'est au moment de l'ajout d'une ligne que vous choisirez quelles sont les colonnes à exploiter.

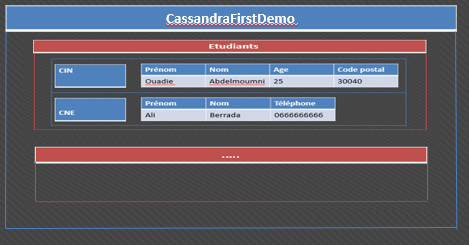
C'est ainsi que Cassandra distingue deux types de famille de colonnes :

* + Statique : les colonnes sont définies lors de la création ou modification de la famille de colonnes ;
  + Dynamique : les colonnes sont définies lors de la création ou modification d'une ligne.
* Ci-dessous un exemple de famille Cassandra



### **Le KeySpace :**

Finalement on a le Keyspace qui est un ensemble de famille de colonnes. Il s'agit d'une sorte de schéma si on compare au monde des bases de données relationnelles



## **Requêtage :**

### **Le KeySpace :**

Comme on a dit précédemment un keyspace dans Cassandra est un espace de noms qui définit la réplication des données sur les nœuds. Un cluster contient un keyspace par nœud. Étant donné ci - dessous est la syntaxe pour la création d' un keyspace utilisant l'instruction **CREATE keyspace.**

**Syntaxe  de création d’un Kesypace:**

CREATE KEYSPACE <identifier> WITH <properties>

Les propriétés peuvent être comme suite :

CREATE KEYSPACE “KeySpace Name”

WITH replication = {'class': ‘Strategy name’, 'replication\_factor' : ‘No.Of replicas’};

CREATE KEYSPACE “KeySpace Name”

WITH replication = {'class': ‘Strategy name’, 'replication\_factor' : ‘No.Of replicas’}

AND durable\_writes = ‘Boolean value’;

#### L'instruction CREATE keyspace possède deux propriétés:Replication et durable\_writes

#### **Réplication :**

L'option de réplication est de spécifier la **stratégie de réplication de placement et le nombre de répliques voulu.** Le tableau suivant nous cite toutes les stratégies de placement de la réplique.

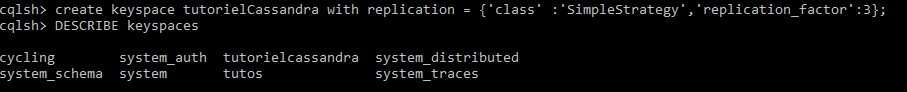
|  |  |
| --- | --- |
| Nom de la stratégie | La description |
| **Stratégie Simple '** | Indique un facteur de réplication simple pour le cluster. |
| **Stratégie Topologie de réseau** | En utilisant cette option, vous pouvez définir le facteur de réplication pour chaque centre de données indépendamment. |
| **Old stratégie Topologie réseau** | Ceci est une stratégie de réplication existante. |

**Exemples :**

On crée deux Keyspaces tutos et tutorielCassandra :



Puis à l’aide de la commande Describe keyspaces on récupère tous les keyspaces existants :

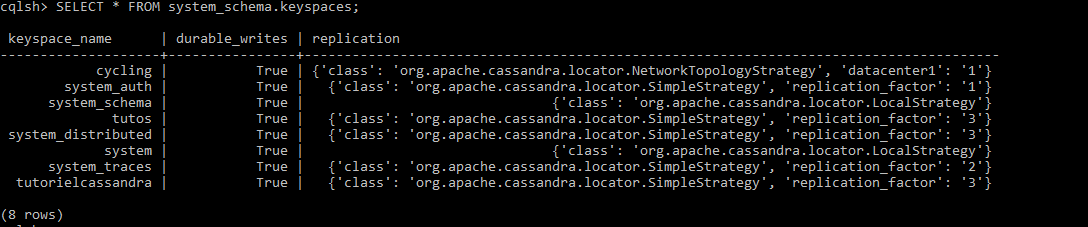


#### **Durable writes :**

La durabilité est la propriété qui écrit, une fois terminée, elle permet au données de persister de manière permanente, même si le serveur est tué ou se bloque .

Par défaut, les durable\_writes propriétés d'une table est définie sur **true, mais il peut être défini sur false.**

🡺On peut en lançant une requête select sur la table système\_Schema.keyspace récupérer les différents durable\_writes de chaque keySpace et les données sur sa réplication



On peut utiliser un Keyspace avec la commande use suivit du nom du keyspace

Syntax:USE <Nom\_Du\_KeySpace>

On voit qu’immédiatement après l’exécution de l’instruction on est pointé sur le Keyspace demandé



#### **Implémentation avec l’api Java :**

Il nous faut Le driver adéquat à cassandra

**Création du Cluster** :

Tout d’abord, créer une instance de la classe, puis ajouter un point de contact (adresse IP du nœud) en utilisant la méthode **addContactPoint () de l’objet Cluster.** Cette méthode retourne **Cluster.**

Cluster cluster = Cluster.builder().addContactPoint("127.0.0.1").build();



**Création de la session.**

Créer une instance de **session objet en utilisant la méthode connect () de la classe de cluster comme indiqué ci - dessous.** Cette méthode crée une nouvelle session et l'initialise. Si vous avez déjà un keyspace, vous pouvez le configurer à celui existant en passant le nom de keyspace au format de chaîne à cette méthode comme indiqué ci-dessous.

Session session = cluster.connect( );

Session session = cluster.connect(“ Notre KeySpace ” );



Maintenant on est connecté à notre nœud Cassandra,et on peut même lui spécifier de ce connecter à un Keyspace dès le début

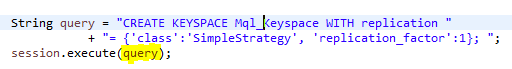


On peut exécuter des requêtes **CQL en utilisant la méthode execute () de la classe de session.** Passez la requête soit sous forme de chaîne ou comme un objet de classe **Statement à la méthode execute ().** Tout ce qui est donné à cette méthode sous forme de chaîne sera exécutée sur le **cqlsh.**

String query = "CREATE KEYSPACE Nom\_KeySpace WITH replication "

+ "= {'class':'SimpleStrategy', 'replication\_factor':1}; ";

session.execute(query);



Puis on peut utiliser ce Keyspace pour mettre en place nos traitement avec du code java de la manière suivante

execute(“ USE Nom\_Du\_Keyspace ” );



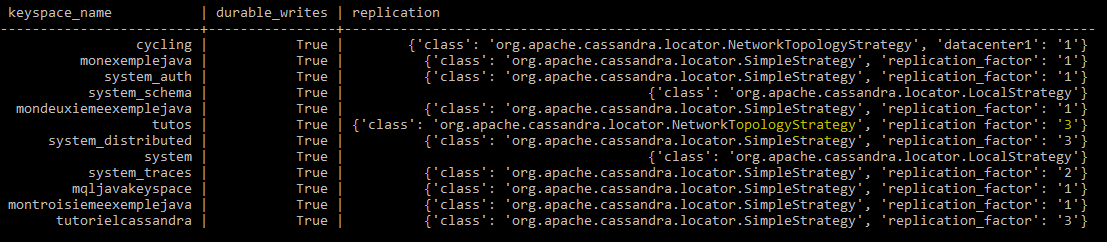
🡺 Toutes les requêtes sur un Keyspace Sont exécutées de la même manière avec le code Java (Alter,Drop…).

**Syntaxe de modification d’un Keyspace:**

ALTER KEYSPACE <Nom\_Du\_Keyspace> WITH <properties>

ALTER KEYSPACE “Mql\_Keyspace”

WITH replication = {'class': ‘Strategy Name’, 'replication\_factor' : ‘No.Of replicas’};

🡺On voit que la modification a bien été faite et que la stratégie et qe le facteur de réplication sont bels bien différents

**Syntaxe  de suppression d’un Keyspace:**

La suppression un keyspace en utilisant la **keyspace commande Drop.** Étant donné ci-dessous est la syntaxe pour laisser tomber un keyspace.

DROP KEYSPACE “KeySpace name”

cqlsh> DROP KEYSPACE mql\_Keyspace;

Pensez à vérifier que la suppression a été bien faite en utilisant la commande **Describe** sur les keyspaces comme ça on est sûr que le Keyspace a bel et bien été supprimer.

### **La Table :**

#### **Création d’une table :**

On peut créer une table en utilisant la commande **CREATE TABLE.** Étant donné ci-dessous est la syntaxe pour la création d'une table.

CREATE TABLE Nom\_De\_La\_Table(

column1 name datatype PRIMARYKEY,

column2 name data type,

column3 name data type.

)

La clé primaire est une colonne qui est utilisé pour identifier de manière unique une ligne. Par conséquent, la définition d'une clé primaire est obligatoire lors de la création d'une table. Une clé primaire est constituée d'une ou plusieurs colonnes d'une table. On peut définir une clé primaire d'une table, comme illustré ci-dessous.

CREATE TABLE Nom\_De\_La\_Table(

column1 name datatype PRIMARYKEY,

column2 name data type,

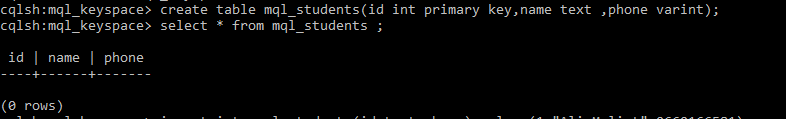
column3 name data type,

**PRIMARY KEY (column1)**

)

Étant donné ci-dessous est un exemple pour créer une table dans Cassandra utilisant cqlsh. Ici, nous:

* Utilisons mql\_Keyspace comme keyspace
* Créerons une table nommée **mql\_students**



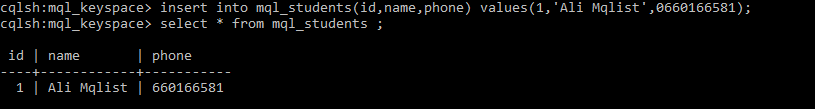
#### **Insertion des données dans une table :**

Après on insère des données dans les colonnes d'une ligne dans une table en utilisant la commande **INSERT.**Étant donné ci-dessous est la syntaxe pour créer des données dans une table.

INSERT INTO <tablename>

(<column1 name>, <column2 name>....)

VALUES (<value1>, <value2>....)



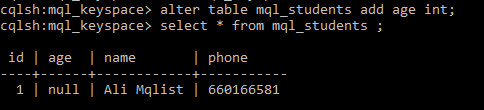
#### **Modification d’une table :**

1. **Ajout de colonne :**

Pour l’ajout on utilise la commande ALTER puis **add** le nom de la colonne, avec cette commande on peut ajouter une colonne à une table. Tout en ajoutant des colonnes, on doit veiller à ce que le nom de colonne n’est pas en conflit avec les noms de colonnes existantes. Étant donné ci-dessous est la syntaxe pour ajouter une colonne à une table.

ALTER TABLE table name

ADD new column datatype;



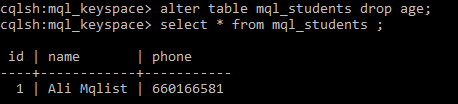
On voit bien que la colonne a été ajoutée et vu qu’avant on n’avait pas âge ça fait qu’elle est insérée nulle par défaut.

1. **Suppression de colonne :**

Pour la suppression on utilise de la commande ALTER puis **drop** le nom de la colonne. Avant de supprimer une colonne d'une table. Étant donné ci-dessous est la syntaxe pour supprimer une colonne d'une table en utilisant la commande ALTER.

ALTER table name

DROP column name;



🡺On voit que la colonne a bel et bien été supprimée

1. **Modification des données :**

UPDATE est la commande utilisée pour mettre à jour les données dans un tableau. Les mots-clés suivants sont utilisés tout en mettant à jour les données dans un tableau :

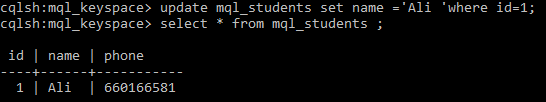
* **where** : Cette clause est utilisée pour sélectionner la ligne mise à jour.
* **Set** : Réglez la valeur à l’aide de ce mot clé.

UPDATE <tablename>

SET <column name> = <new value>

<column name> = <value>....

WHERE <condition>

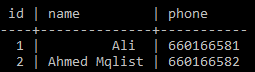
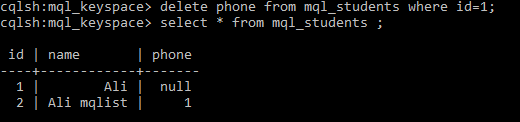


🡺Après on peut observer les mises à jour faites à partir de nos traitements.

1. **Suppression des données :**

On peut supprimer des données d’une table à l’aide de la commande **DELETE ,**sa syntaxe est la suivante :

DELETE <nom\_de\_colonne>FROM <nom\_de\_la\_table> WHERE <condition>;

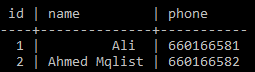
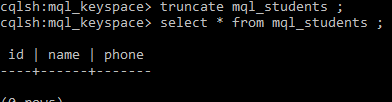
 

🡺Si on ne sépécifie pas le nom de la colonne toute la ligne sera supprimée

#### **Trancate (suppression de toutes les lignes) d’une table :**

On peut tronquer une table en utilisant la commande TRUNCATE. Lorsque vous tronquer une table, toutes les lignes de la table sont définitivement supprimés. Étant donné ci-dessous est la syntaxe de cette commande.

TRUNCATE nom\_table;

🡺C’est vrai que toutes les lignes de toute la table ont été supprimées

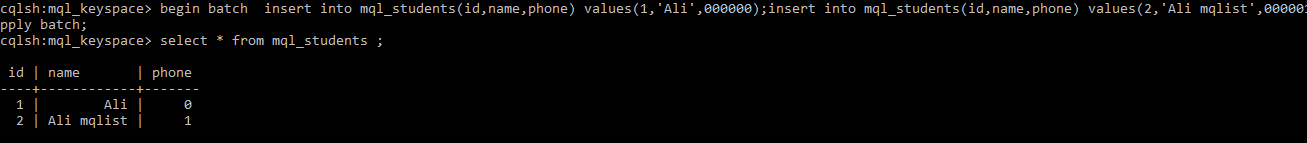
#### **Batch (lot d’instructions) :**

Pour l’utilisation de BATCH, on exécuter plusieurs instructions de modification (insert, update, delete) simultanément en utilisant une **Begin batch** pour dire qu’on va commencer notre lot d’instruction et en finissant par **Apply batch** . Sa syntaxe est la suivante :

BEGIN BATCH

<Query1> ; <query2> ; <query3>

APPLY BATCH



🡺On voit que l’ensemble des deux insertions a bel et bien été fait.

### **Les Collections :**

#### **Les listes :**

La liste est utilisée dans les cas où

* L’ordre des éléments doit être maintenue.
* Une valeur doit être stockée plusieurs fois.

On peut obtenir les valeurs d'un type de données de liste à l'aide de l'indice des éléments dans la liste

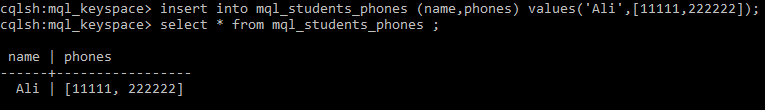
CREATE TABLE nom\_table(name text PRIMARY KEY,

notre\_liste list<type\_de\_données>);

INSERT INTO Nom\_table(name, notre\_liste) VALUES (‘Ali’,

[value1,value2])

Ci-dessous l’exemple concret de l’utilisation d’une liste



**Les Sets :**

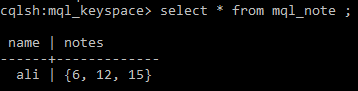
Le Set est un type de données qui est utilisé pour stocker un groupe d'éléments. Les éléments d'un ensemble seront retournés dans un ordre de tri.

L'exemple suivant crée un exemple de table avec deux colonnes, nom et téléphone. Pour stocker plusieurs numéros de téléphone, nous utilisons ensemble.

CREATE TABLE nom\_table (name text PRIMARY KEY, notre\_Set set<type de données>);

Lors de l'insertion des données dans les éléments d'un ensemble, entrez toutes les valeurs séparées par des virgules entre accolades {}, comme indiqué ci-dessous.





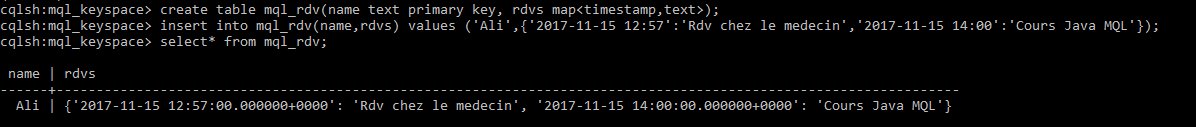
**Les MAPS :**

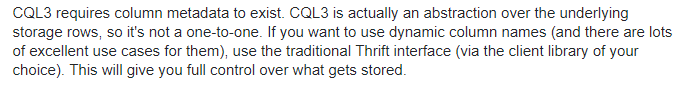
La Map est un type de données qui est utilisée pour stocker une paire d'éléments de valeur-clé.

CREATE TABLE nom de la table (name text PRIMARY KEY, notre Map

map<type, type>);

Lors de l’insertion des données dans les éléments d'une carte, saisir toutes les paires **clé: valeur séparées par des virgules entre accolades {} , comme indiqué ci - dessous.**





# MongoDB



**MongoDB :**

MongoDB est un système de base de données NoSQL ,il est plutôt orienté documents(Un document est la représentation d'une donnée en BSON, Binary JSON. Extension du JSON) , qui est écrit en C++,pouvant être répartit sur une multitude de machines et ne nécessitant pas de schéma prédéfini de données ce qui le rend d’une grande flexibilité(contrairement) aux bases de données relationnelles .Son objectif est la gestion d’énormes quantités de données ,mais la question c’est comment ? Son moteur de base de données facilite l’extension (ce qu’on appelle scaling ) chose mise en évidence par le support de l’accroissement énorme des données par l’ajout de machines .

## Avantages et inconvénients de MongoDB :

### Avantages :

Vue que MongoDB ne nous impose pas de schéma on peut lui trouver un bon nombre d’avantages :

* Un grand nombre de fonctions qui le rendent polyvalente : les index géographiques, le framework d’agrégation, le map/reduce intégré et les nombreux outils de chargement de données en masse ou de sauvegarde et de restauration.
* Le concept de schéma n’existe pas ,du coup chaque document est libre de suivre sa propre structure ce qui nous permet une grande flexibilité de données.
* MongoDB assure la scalabilité: la réplication et le Sharding (distribution de données sur plusieurs machines): son modèle de données orientées document lui permet de répartir automatiquement les données sur plusieurs serveurs (Cela permet aux développeurs de se concentrer sur la programmation de l'application)

### Inconvénients :

Tandis que cette flexibilité dont on vante les mérites peut poser des problèmes lorsqu’on utilise MongoDB :

* Impossible de réaliser des jointures « les données sont généralement embraquées dans le même document ».
* La flexibilité, la performance et la scalabilité se font au dépend de la capacité de gérer des transactions complexes
* On a pas de schéma de imposé certes mais cela peut vite tourner en notre défaveur puisque le contrôle de données sera reporté côté application .
* MongoDB offre un langage d’interrogation propre à lui (non standard ) ce qui est d’un côté pratique mais nous limite beaucoup .

## **Les fonctionnalités de MongoDB :**

MongoDB est donc une base de données NoSQL de type document qui dispose de plusieurs fonctionnalités intéressantes. Voici les fonctionnalités générales de MongoDB :

* La réplication permet de dupliquer les serveurs de base de données pour répondre à une monté en charge ou une tolérance de panne.
* Le Sharding (distribution de données sur plusieurs machines pour assurer la scalabilité) permet de répartir les données sur plusieurs serveurs soit pour simplement augmenter les performances ou pour répartir les données géographiquement.
* Système de fichiers GridFS permet de stocker simplement des fichiers en base de données.
* Le SIG ou Système d’information géographique permet de manipuler simplement des positions sur un plan ou sur le globe terrestre.
* La fonction de recherche : MongoDB intègre un système de recherche optimisé en fonction de la langue utilisée.
* MongoDB offre des nombreuses fonctionnalités que l’on trouve dans le monde relationnel (count, groupBy, etc.) mais aussi le support de la recherche full-text, la recherche géo-spatiale ou MapReduce (manipulation et distribution de données dans un cluster) - supporte l’indexation pour l’optimisation des recherches

## **Les versions :**

MongoDB a été développé en 2007 par MongoDB,une entreprise qui s’adonnait aux systèmes de Cloud computing .Sa première version de MongoDB considérée comme industriellement viable a été la 1.4 réalisé en 2010 , la dernière version « par laquelle on travaille actuellement » est la 3.4

## **Organisation :**

* Un serveur MongoDB est composé de plusieurs bases de données
* Une base de données contient des collections.
* Les collections possèdent les documents
* Chaque document possède un identifiant unique généré par MongoDB, le champ \_id

## **Structures de données :**

### Notions de documents

Dans SGBDR « Système de base de données relationnelles les données sont stockées sous formes de lignes dans des tables interrogeables par des requêtes SQL. Ces tables sont liées entre elles à travers clés primaires et étrangères.

Or que dans le cas de MongoDB ces données sont modélisées dans un format BSon basé sur JSon.

Du coup MongoDB n’a pas besoin d’un schéma prédéfini comme en SQL, il n'est pas nécessaire par exemple de définir des colonnes avec un nom et un type et on peut insérer n'importe quel document BSON. Lors de l'insertion d'un document, MongoDB ajoute automatiquement un index nommé par

défaut \_id. La méthode insert retourne l'identifiant du document inséré.

Il n'est pas obligatoire de définir des colonnes avec un nom et un type de données bien précis et on peut insérer n'importe quel document BSON. Lors de l'insertion d'un document, MongoDB ajoute automatiquement un index nommé par défaut \_id(ou bien on peut le faire de manière explicite ). La méthode insert retourne l'identifiant du document inséré.

**Exemple d’un document :**

Document={nom :’Mqlist’, prénom :’Ali’ ,emails[‘email1’,’email2]}

### Notions collections

La notion de collection correspond dans le monde de SQL des bases de données relationnelles (SQL SERVER ,Oracle,MySQL,PosgreSql…)

Onretrouve ci-dessous un tableau comparatif entre les deux :

|  |  |
| --- | --- |
| Bases de données relationnelles  (SQL) | Bases de données NoSQL  (NoSQL) |
| Table | Collection |
| Ligne | Document |
| Index | Index |
| Jointures | Données embarquées dans le document |

## **Le Format BSON(Binary JSON),comparaison avec JSON :**

### **Le Format BSON**

Le format BSON est celui utilisé par MongoDB.En gros c’est un format d’échange de données utilisé essentiellement comme stockage de données et format de transfert de données par le réseau dans une base de données MongoDB.

C'est un format binaire permettant de représenter des structures de données simples et des tableaux associatifs (appelées objets ou des documents dans MongoDB). Le nom BSON est basé sur le terme JSON et signifie Binary JSON

### **Comparaison avec JSON**

JSON ou JavaScript Object Notation, est un format de données textuelles dérivé de la notation des objets du langage JavaScript. Il permet de représenter de l’information structurée Un document JSON a pour fonction de représenter de l'information accompagnée d'étiquettes permettant d'en interpréter les divers éléments, sans aucune restriction sur le nombre de celles-ci. Un document JSON ne comprend que deux types d'éléments structurels : des ensembles de paires nom / valeur et des listes ordonnées de valeurs.

**Voici un tableau comparatif entre JSON et BSON**



🡺On déduit que le format BSON supporte plus de type que JSON,et qu’il est complémentaire à JSON vu qu’il y ajoute es types de données supplémentaires, tels que les formats virgule flottante ou date entre autres.

## **Démarrage avec MongoDB  :**

Le serveur se lance à l’aide de la commande mongod.exe et L’interpréteur de commandes Javascript correspond à la commande mongo.exe

Maintenant et vu que notre interpréteur de données est démarré nous allons procéder à nos premières manipulations :

### **1-Utilisation d’une base de données :**

Il nous faut tout d’abord une base de données de test sur laquelle on va effectuer tous nos tests, on l’appellera **personnes** .

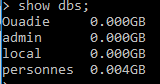
Pour créer cette base de données il suffit de spécifier à MongoDB qu’on veut l’utiliser « et ceci on utilisant la commande use [Le Nom de la BD] » et là deux cas se présentent : ou bien la base de données existe déjà donc il va passer sous cette base de données même ou bien elle existe pas et il va la créer et switcher vers cette même base de données

Use DATABASE\_NAME



Et si on veut voir toutes les bases de données, vous pouvez utiliser la commande **show dbs ;**

show dbs



🡺 Cela nous donne même la quantité de données stockée en GB dans chaque base de données

Pour supprimer une base de données il suffit de se positionner sur cette dernière et exécuter la commande :

db.dropDatabase()

### **2-Utilisation d’une collection :**

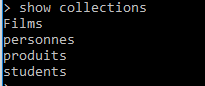
Dans le cas d’une base de données relationnelle on crée des tables pour y stocker nos données. Dans le cas de mongodb, notre base de données contient des **collections**dans lesquelles on ajoute nos documents qui stockent nos données. Une collection est donc un ensemble de documents de même nature.

Nous allons créer une collection d'**étudiants**. La création de la collection se fait implicitement, elle dès qu’on insert le premier document, ou bien explicitement à travers la fonction **db.createCollection( « NomCollection »).**

db.COLLECTION\_NAME.insert(document)



Pour voir les collections existantes on utilise la méthode **db.showCollections()**



Puis on va créer un objet :



Puis on insère cet objet à notre collection comme on l’a mentionné au paravant, il nous précise même le nombre de ligne insérées



La commande se décompose comme ceci :

db.<nom de la collection>.insert( <document> ).Nous avons donné pour nom de la collection : *students*,le document que nous avons inséré est le suivant



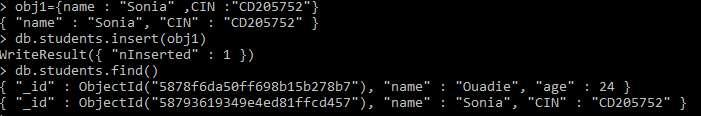
Dès lors on peut vérifier si la ligne a été bel et bien insérée

db.<nom de la collection>.find( )



On notera que L’id est attribué automatiquement et que si on veut spécifier notre id on le mentionne au début avec \_id

Et comme on l’a dit au début MongoDB est schemaless ,ce qui veut dire que les documents ne doivent pas respecté obligatoirement le même format



### **3-Recherche :**

**Critères de recherches** :

Après notre insertion, nous avons pu récupérer notre document en utilisant la méthode **find()**.

La méthode *find* prend pour premier paramètre un objet JSON qu'elle va prendre comme critère de recherche.

Si par exemple on fait





On va avoir tous les documents qui correspondent à *age* = 24 et name =Ouadie.

En SQL, cela revient à faire une requête avec une clause where.

On peut aussi ajouter de la précision à notre recherche on ajouter plusieurs autres critères pour la recherche :



Et pour connaitre le nombre de résultats retournés il suffit d’invoquer la méthode count()



**Opérateurs conditionnels :**

On peut aussi ajouter quelques critères de recherches pour plus affiner notre recherche avec les opérandes suivantes :

* $gt : plus grand que
* $lt : plus petit que
* $gte : plus grand ou égal à
* $lte: plus petit ou égal à

Et les quatre opérandes précédents fonctionnent pour les nombres et les chaînes de caractères, pour les chaines de caractère l’ordre appliqué est l’ordre alphabétique

* $or : pour récupérer les documents qui correspondent à 2 critères différents
* $and : pour cumuler des critères
* $in, $all, $exist, $type et $regex ...



### **Filtrer les résultats**

 Pour le moment, lorsque nous faisons une requête, nous récupérerons les documents dans leur ensemble. Si les documents sont volumineux, il peut être intéressant de ne récupérer que les valeurs qui nous intéressent.

La méthode find() prend un deuxième paramètre qui va nous servir à récupérer les propriétés qui nous intéressent.

Si on fait db.students.find({name: ‘Sonia’} , {\_id : 0 ,CIN:1})



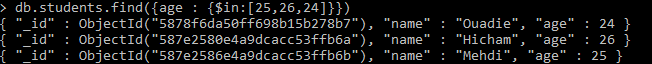
Cela nous retourne que le CIN de la personne dont le nom est Sonia et pas l’id parce qu’on spécifié le 0 comme paramètre qui veut dire on veut pas retourner l’id sion il nous retourne l’id par défaut



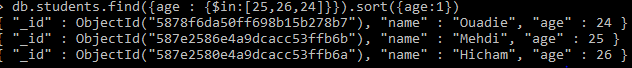
### **Trier les résultats**

 Pour trier les résultats, nous pouvons utiliser la méthode **sort()**.

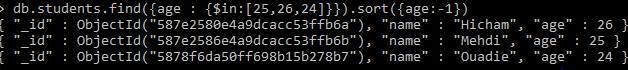
On va d’abord retourner les âges des étudiants stockées



Puis à l’aide de la méthode sort () on spécifie qu’on veut trier par ordre croissant (le paramètre 1 ) sinon on utilise le paramètre (-1)



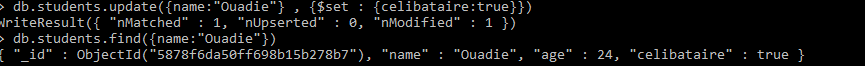
*Exemple tri par Ordre croissant*



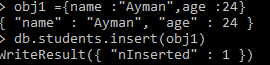
*Exemple tri par Ordre décroissant*

### **Ajout /Modification d’une propriété**

Pour qu’on puisse faire un ajout ou bien une modification dans un de nos documents on doit utiliser la méthode **update ()**



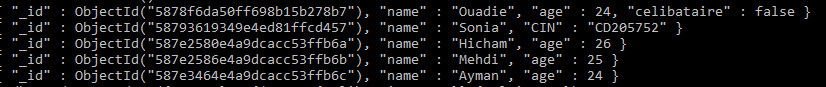
Maintenant on va insérer un autre objet qui aura dans l’âge 24 comme valeur ,du coup on a deux ligne avec la même valeur de âge .



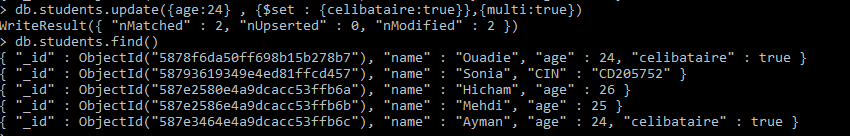
Après on tape la commande update pour modifier ceux qui ont l’âge égal à 24 on leur ajoute le paramètre célibataire dont la valeur = false.

Sauf que cela ne marche que pour la première ligne trouvée





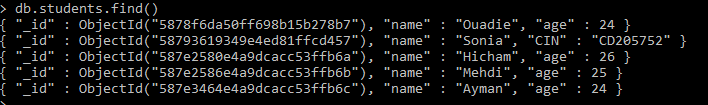
Pour modifier toutes les lignes avec comme critère âge = 24 on doit ajouter l’attribut multi : true



### **Suppression d’un attribut :**

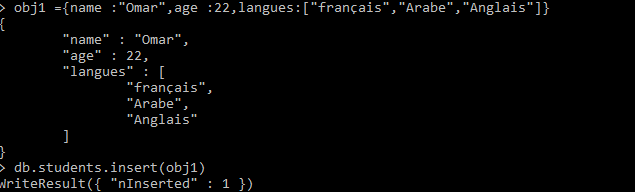
Nous allons supprimer la propriété que l'on vient d'ajouter. Pour cela, nous allons utiliser **$unset**.

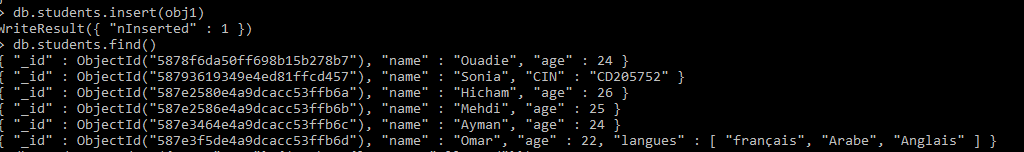




### **Mise à jour d’un tableau :**

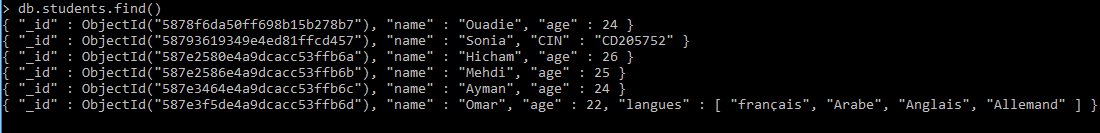
On va ajouter un nouvel objet avec un tableau langues.





Maintenant pour ajouter un élément, on utilise l'opérande **$push**





On voit qu’on ajouter la valeur Allemand au tableau des langues

Maintenant et grâce à l’opérande $pushAll on va ajouter plusieurs valeurs au même tableau





Avec l'opérande **$pop** on va supprimer ou bien le premier élément ou bien le dernier élément.

1er cas : avec la valeur -1

On va supprimer le premier élément, on notera que la langue française n’y est plus





2ème cas : avec la valeur -1

On va supprimer le dernier élément, on notera que la langue Espagnole n’y est plus





Avec l'opérande **$addToSet**, on ajoute sans doublon :

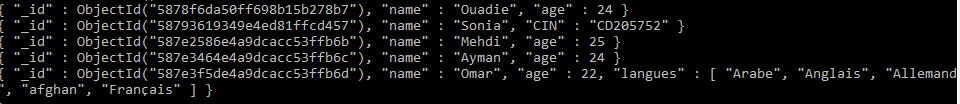


On voit bien que le nombre de lignes modifiées n’a pas changé par contre si on ajoute une langue inexistante elle va être insérée



### **Suppression d’un document :**

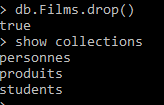




On remarque très bien que le document contenant comme nom :Hicham a été supprimé

### **Suppression d’une Collection :**

A l’aide de la fonction drop() qui nous retourne true si on l’a supprimé et false dans le cas contraire.



## API pour le langage JAVA :

Java-MongoDB-Driver est le pilote Java pris en charge pour MongoDB

com.mongodb: il s'agit du paquet de base permettant de créer une connexion client à une instance En cours d' exécution mongod

com.mongodb.client: ce paquet permet l'accès à une base de données MongoDB

On se connecte à une instance MongoDB en cours d’exécution sur le localhost (port par défaut

27017)

Pour gérer les documents dans l'application, on utilise les paquets suivants :

\* org.bson.Document

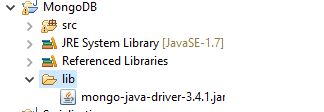
\* com.mongodb.MongoClient

\* com.mongodb.client.MongoCollection

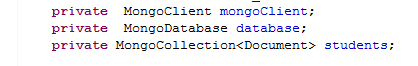
\* com.mongodb.client.MongoDatabase

**Exemple :**

On ajoute notre Jar à notre projet(vaut mieux télécharger la dernière version )



On aura besoin des variables suivantes pour établir la connexion à la base de données et accéder au document et à la collection voulue



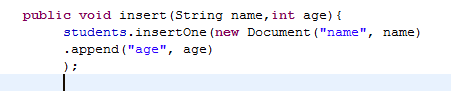
On instancie une instance de type MongoClient pour établir la connexion avec le serveur Mongodb et qui contient la méthode getDatabase() qui nous retournera la base de données voulue



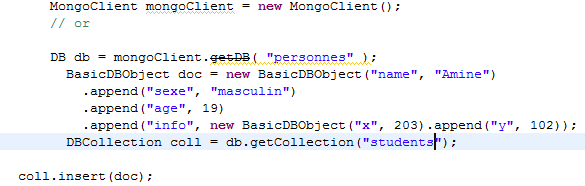
On récupère le nom de la base de données dans un objet de type MongoDatabase qui à son tour nous permet de récupérer la collection voulue à l’aide de la méthode getCollections()



Après on pourra insérer un nouveau document dans notre base de données à l’aide de la méthode insertOne() qui prend comme paramètre un objet de type Document et qui l’insèrera dans la collection voulue de notre base de données



On peut aussi insérer un document on instanciant un objet de type **BasicDBObject**



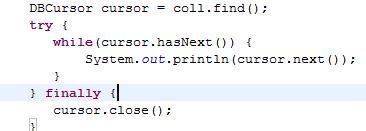
On instancie un Objet de type MongoClient pour établir la connexion avec le serveur ,si on spécifie rien dans les paramètre Il va choisir le premier server enregistré sinon on peut lui passer comme paramètres («l.ocalhost »,et le numéro du port )

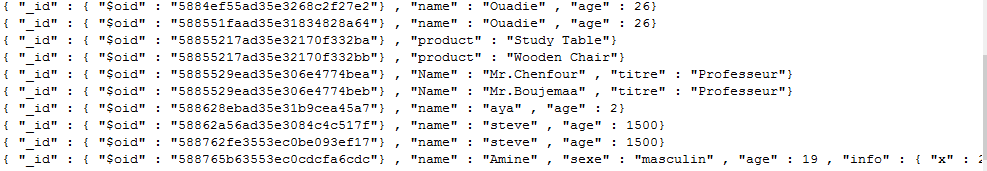
-Puis on récupère notre base de données à l’aide de la méthode dépricated getDB() qui nous retourne un objet de type DB qui à son tour contient la méthode getCollection qui nous retourne la collection voulue .

Il nous reste plus qu’à créer notre objet le remplir avec nos paramètres et l’insérer à l’aide de la méthode insert ()

-On pourra afficher le premier document de notre collection à l’aide de la méthode **findOne**() qui nous retourne le premier élément

-Pour récupérer tous les documents dans une collection on va utiliser **un curseur** sur la méthode **find*()*** retournée par la collection choisie,puis on exécutera une boucle pour retourner tous les documents tant que le curseur n’est pas arrivé à la fin



Résultat : 

Redis



**Redis :**

Redis (de l'anglais REmote DIctionary Server qui peut être traduit par « serveur de dictionnaire distant » et jeu de mot avec Redistribute) est

* Un système de gestion de base de données clef-valeur scalable, à très hautes performances
* Ecrit avec le langage de programmation C. Il fait partie de la mouvance NoSQL et vise à fournir les performances les plus élevées possibles.
* Très rapide, il est fait pour manipuler des données en mémoire, il peut être utilisé comme serveur de cache ou comme base de données (les données peuvent être persisté sur le disque). C'est un outil pratique pour faire de la performance

**Avantages :**

* Haute performance - Redis peut lire la vitesse est de 110 000 fois / s, la vitesse d'écriture est 81000 fois / s.
* Types de données riches - soutien Redis binaires Strings de cas, listes, Hashes, Ensembles et Ordonné type de l'opération de données de jeux.
* Atom - Redis Toutes les opérations sont atomiques, atomiques tandis que Redis soutient également la mise en œuvre de l'ensemble de l'opération et après quelques-uns.
* Le riche ensemble de fonctionnalités - Redis soutient également publication / abonnement, notifications, expiration de la clé et donc sur les caractéristiques.

**Prise en main Redis :**

**Requêtage :**

### **Démarrer le client Redis :**

Pour exécuter une commande nécessite un client sur le service de Redis de redis. Redis client avant Redis télécharger le package d'installation.

Dans l'exemple ci - dessous nous nous connectons au service de Redis local et exécuter la commande **PING, qui est utilisé pour détecter le service Redis est démarré.**Il nous retourne Pong pour nous dire qu’il a bel et bien reçu notre ping.

redis 127.0.0.1:6379> PING

PONG



### **Les traitements sur un couple (clé,valeur) dans Redis :**

**Ecriture :**

On commence à découvrir les commandes Redis. Ce que nous allons voir vous servira directement lorsque vous coderez en Java, Les drivers Redis reprennent la syntaxe de l’interpréteur.

nous allons créer des entités type clé-valeurs très simples. La syntaxe est SET <key> <value>

redis 127.0.0.1:6379> SET nom\_clé valeur

OK



**Lecture :**

Pour récupérer notre value on y procède par récupération de la clé avec le : GET <key> <value> .

redis 127.0.0.1:6379> GET nom\_clé

« Valeur »



**Supprimer :**

La syntaxe pour supprimer une clé est DEL <key>,une fois la clé supprimé on a un retour entier de la valeur de ‘1’ sinon un retour de la valeur de ‘0’

redis 127.0.0.1:6379> DEL nom\_clé

« 1 si oui elle a été supprimée », « 0 si non »

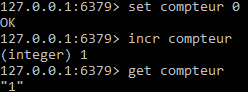
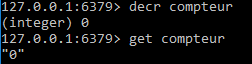


**Incrémentation /Décrémentation de la valeur d’une clé :**

Pour incrémenter ou décrémenter la valeur d’une clé il suffit de la syntaxe suivante

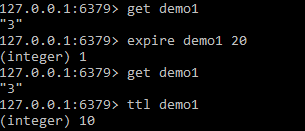
redis 127.0.0.1:6379> INCR/DECR nom\_clé

« 1 si oui elle a été supprimé », « 0 si non »

**Clé temporaire :**

SETEX et EXPIRE permettent de stocker une clé temporaire qui expire au bout du temps demandé (Expire <key> <Time> | SETEX <key> <Value> <Time>) . La commande TTL vous donne le nombre de secondes avant que Redis n’efface cette clé



🡺A chaque fois qu’on modifie une valeur d’une clé sa date d’expiration est réinitialisée.

### **Structures de données dans Redis :**

Redis est Très intéressant du point de vu où il propose des structures de données comme les Lists, les Sets ou les Hashs :

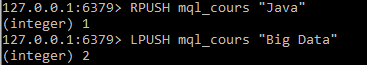
**Les listes :**

Les listes sur Redis sont simplement : des listes de String. On retrouve la complexité qu’on trouve dans une liste classique et aussi l’ordre d’insertion qui est conservé. On peut ajouter un élément au début ou à la fin avec un appel de complexité O(1). Une liste peut contenir jusqu’à 232 éléments (plus de 4 milliards).

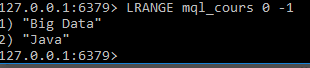
Les commandes principales à connaitre :

RPUSH : ajout à droite de la liste.

LPUSH : Ajout à gauche de la liste



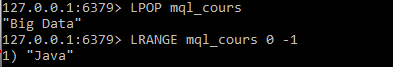
LRANGE : Elle permet de retourner une liste depuis l’indice donné en départ et l’indice donné en arrivé (exemple LRANGE liste 0-1 récupère depuis l’indice 0 jusqu’à la dernière case de la liste)



LLEN :Pour retourner la longueur de la liste



LPOP/RPOP : supprimer un élément au début ou bien à la fin de la liste



**Les SETs :**

Les SETs sur Redis sont des collections non ordonnées de String.Ce qui fait que les SET sont des indexes particulièrement efficaces c’est que chaque clé est unique, et si vous insérez une nouvelle valeur avec la même clé, vous allez simplement écraser l’ancienne valeur.

Les commandes principales à connaitre :

SADD : pour créer une Set

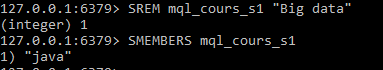


SMEMBERS : Pour récupérer les valeurs stockées dans notre SET

🡺On voit bien que si on essaye d’insérer un doublon la SET ne l’accepte pas et nous retourne 0 (qui veut dire que l’insertion n’a pas été faite)

SREM :Pour supprimer une valeur de notre SET

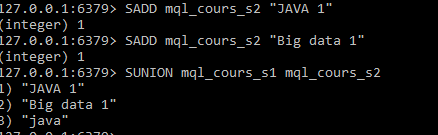


SISMEMBER :Pour savoir si une valeur existe dans notre SET



Vu qu’on a supprimé le BIG DATA il ne retourne qu’il existe pas dans la SET ,ce qui est le contraire pour le Module JAVA.

SUNION :Si on veut unir deux SETS et avoir le résultat en un seul SET ,c’est efficace pour croiser des SETS



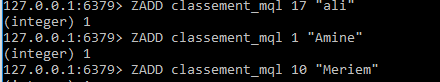
**Les Sorted Set (ou ZSET):**

Les ZSet. Il s’agit de SET, avec la possibilité d’attacher un score(classement) à chaque valeur. Le score doit forcément être une valeur numérique.

Les ZSet sont utilisés comme des indexes, et sont très pratiques lorsque par exemple vous voulez montrer les 10 deals les moins chers ou les mieux notés sur un site ou bien pour des opérations récurrentes de classement

Les commandes principales à connaitre :

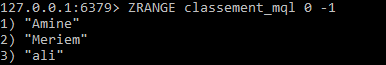
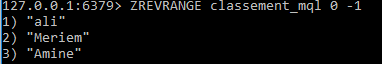
ZADD : pour créer une ZSET



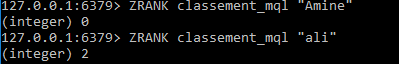
On voit que chaque étudiant a son score avant sa valeur

ZRANGE : pour récupérer l’ensemble des valeurs classées de façon croissante

ZREVRANGE : pour récupérer l’ensemble des valeurs classées de façon décroissante

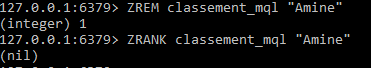
 

ZRANK :Pour savoir la position d’une valeur dans la ZSET



Cette commande retourne l’indice de la valeur donnée en paramètre

ZREM : pour la suppression d’une valeur donnée dans la ZSET



On supprime Amine de la liste ,puis on lance une recherche sur sa position ,ça retourne null,car il existe plus

**Les Hashs:**

Les Hashs sur Redis sont de purs Map de String. Naturellement, c’est la structure qui semble la plus adaptée pour représenter des objets JAVA par exemple

Les commandes principales à connaitre :

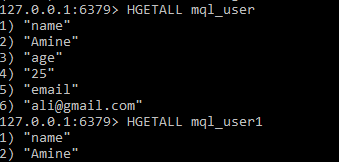
HSET/HMSET :Pour insérer respectivement une ou plusieurs valeurs dans une Hash





Dans le premier cas on a ajouté une seule clé valeur dans l’ensemble de la SET ,pour le second on a inséré un ensemble de couples clé valeur.

HGETALL : Permet de retourner toutes les informations dans une HASH donnée



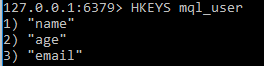
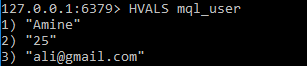
NB : chaque clé est suivie de sa valeur !

HGET : Permet de retourner l’information sur un couple de clé valeur bien précis dans une HASH donnée.



HKEYS : Permet de retourner toutes les clés d’une HASH

HVALS : Permet de retourner toutes les valeurs d’une HASH

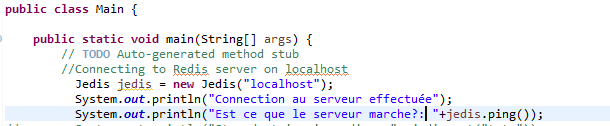
 

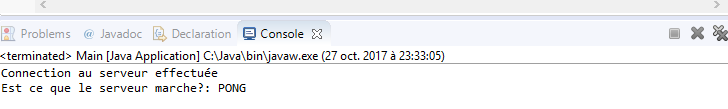
### **API JAVA pour Redis :**

On devra tout d’abord télécharger l’api Jedis pour Redis afin de pouvoir avoir accès à la base de données Redis à travers son port et son adresse.

**Description du Code :**D’abord on instancie un objet de la classe Jedis avec comme paramètre dans le constructeur localhost ,c’est lui automatiquement qui gérera l’écoute sur le port et l’adresse de la base de données .

Afin de nous assurer que cela marche parfaitement on utilise la fonction ping() qui permet de lancer un ping sur le serveur .Vous vous rappelez que le Serveur répond par PONG ?



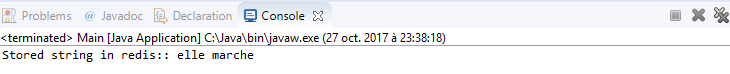


🡺Une fois la connexion effectuée on peut commencer nos traitements à nos aises.

Maintenant on va essayer d’insérer un couple de clé valeur dans notre base de données Redis

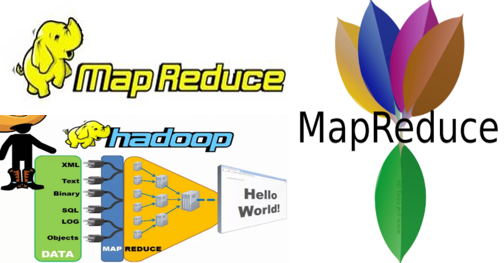


On utilise la fonction set paramétrée respectivement par la clé et la valeur



🡺On voit très bien que la clé valeur a bel et bien été insérer

Map /Reduce



**Map/Reduce :**

MapReduce est un paradigme de calcul qui traite de grandes masses de données (plusieurs téra-octets pour le moins) en parallèle sur de grands clusters (plusieurs milliers de noeuds), de façon à être tolérant aux pannes.

MapReduce est un des patrons de l’architecture dans le monde des calculs parallèles et distribués de données très volumineuses .

MapReduce permet de manipuler de grandes quantités de données en les distribuant dans un cluster de machines pour être traitées. Ce modèle connaît un vif succès auprès de sociétés possédant d'importants centres de traitement de données telles que Facebook ,Yahoo ou encore Google.

**Inspiration fonctionnelle :**

En programmation fonctionnelle :

* + Map : consiste à appliquer une même fonction à tous les éléments de la liste

map(f)[x0, ..., xn] = [f(x0, ...,f(xn)]

map(\*4)[2, 3, 6] = [8, 12, 24]

* + Reduce : Applique une fonction récursivement à une liste et retourne un seul résultat

reduce(f)[x0, ..., xn] = f(x0, f(x1, f(x2, ...)))

reduce(+)[2, 3, 6] = (2 + (3 + 6)) = 11

**Map /Reduce sur les (clé,valeur) :**

1. L'ensemble des données à traiter est découpé en plusieurs lots ou sous-ensembles.
2. Dans une première étape, l'étape MAP, Cette opération transforme la paire(clé, valeur)représentant le lot en une liste de nouvelles paires(clé, valeur)constituant ainsi des résultats intermédiaires du traitement à effectuer sur les données complètes.
3. Avant d'être envoyés à l'étape REDUCE, les résultats intermédiaires sont regroupés et triés par clé. C'est l'étape deSHUFFLE and SORT.
4. Enfin, l'étape REDUCE consiste à appliquer l'opérationreduce, spécifiée pour notre problème, à chaque clé. Elle agrège tous les résultats intermédiaires associés à une même clé et renvoie donc pour chaque clé une valeur unique.

**Exemple WORDCOUNT ,le helloWord de map/Reduce :**

A titre d’exemple on va illustrer l’exemple typique de MapReduce avec le ‘WordCount’

Prenons en entrée un document textuel : l'objectif de Wordcount est de calculer le nombre d'occurrences de chaque mot dans le document.

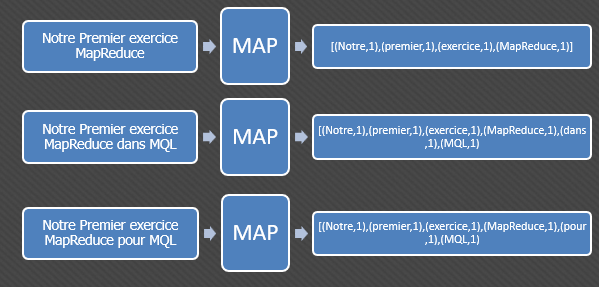
Document : Notre Premier exercice MapReduce, Notre Premier exercice MapReduce dans MQL , Notre Premier exercice MapReduce pour MQL)

On va diviser ce document en 3 sous-documents (pour la parallélisation des traitements)

* Document 1 :’ ‘Notre Premier exercice MapReduce’
* Document 2 : ‘Notre Premier exercice MapReduce dans MQL ’
* Document 3 : ‘Notre Premier exercice MapReduce pour MQL

### **L’étape Map :**

Il nous faut maintenant déterminer la clé de notre opération ,cette clé sera les mots de notre corpus.

****

**🡺A la fin de notre étape MAP on a plusieurs liste de paires (clés,valeurs).**

### **L’étape Suffle and Sort :**

Cette étape entièrement gérée par le FrameWork Hadoop plus exactement (Yarn/MapReduce) nous permet de regrouper et de trier par clé commune les résultats fournis par l’étape Map.

|  |  |
| --- | --- |
| **(Notre,[1,1,1])** | **(Dans,[1])** |
| (premier,[1,1,1]) | (Mql ,[1,1]) |
| (exercice,[1,1,1]) | (MapReduce,[1,1,1]) |

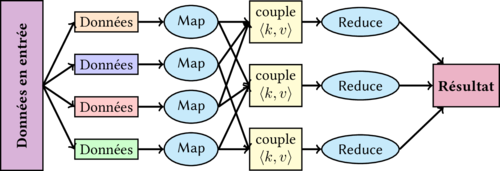
🡺Nous avons donc à notre disposition un ensemble de pairs qui attendent la dernière étape (Reduce) pour que chaque clé soit unique, non redondante ayant sont nombre d’occurrence comme valeur.

### **L’étape Reduce :**

Pour notre exemple WordCount l’opération Reduce consiste à sommer toutes les valeurs de toute les clés données par les opérations suivantes

|  |  |
| --- | --- |
| **(Notre,3)** | **(Dans,1)** |
| (premier,3) | (Mql,2) |
| (exercice,3) | (MapReduce,3) |

**Récapitulatif de l’opération** :



Hadoop



**Hadoop :**

Hadoop est un framework open source, écrit en Java et géré par Apache.Conçu pour répondre aux besoins du Big Data, tant au plan technique qu’économique. Hadoop est capable de stocker et traiter de manière efficace un grand nombre de donnés, en reliant plusieurs serveurs entre eux pour travailler en parallèle.

Hadoop offre une grande flexibilité. Ses performances évoluent de manière quasi linéaire en fonction du nombre de machines constituant le cluster. Plus le nombre de nœuds est élevé moins le temps d’exécution des jobs est court .

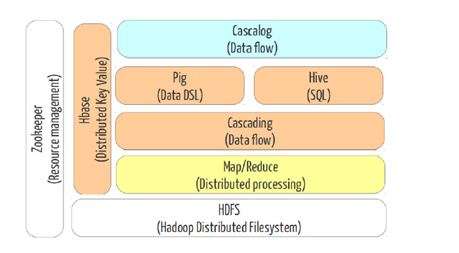
Hadoop fonctionne sur le principe des grilles de calcul consistant à répartir l’exécution d’un traitement intensif de données sur plusieurs noeuds ou grappes de serveurs.

Java est le langage de préférence pour écrire des programme Hadoop natifs. Néanmoins, il est possible d’utiliser python, bash, ruby, perl …

**Composantes de Hadoop :**

Le framework Hadoop comprend un grand nombre de composants Open Source, tous connectés à un ensemble de modules cœur destinés à capturer, traiter, gérer et analyser d’importants volumes de données. Ces technologies sont entre autres :

* HDFS (Hadoop Distributed File System) est le système de stockage primaire utilisé par les applications Hadoop. HDFS permet de gérer la réplication de multiples blocs de données et leur distribution sur les nœuds de calcul à travers un cluster pour permettre des calculs fiables et extrêmement rapides.
* MapReduce est une plate-forme de programmation conçue pour écrire des applications permettant le traitement rapide et parallélisé de vastes quantités de données réparties sur plusieurs clusters de nœuds de calcul.
* HBase est un système de gestion de base de données non relationnelle, distribuée et orientée colonnes, prenant pour modèle Big Table de Google.
* HCatalog est une couche de métalangage permettant d'attaquer les données HDFS via des schémas de type tables de données en lecture/écriture.
* Hive est un système d'entrepôt de données facilitant l'agrégation des données, les requêtes ad hoc, et l'analyse de grands ensembles de données stockées dans les systèmes de fichiers compatibles Hadoop. Hive dispose d'un langage de type SQL appelé HiveQL.
* Pig est une plate-forme d'analyse de vastes ensembles de données. Pig comprend un langage de haut niveau gérant la parallélisation des traitements d'analyse.
* Oozie est un outil de workflow dont l'objectif est de simplifier la coordination et la séquence de différents traitements. Le système permet aux utilisateurs de définir des actions et les dépendances entre ces actions.
* ZooKeeper est un service centralisé pour gérer les informations de configuration, de nommage, et assurer la synchronisation des différents serveurs via un cluster.

****

**Hadoop File System (HDFS) :**

HDFS est un système de fichiers distribué. C’est-à-dire :

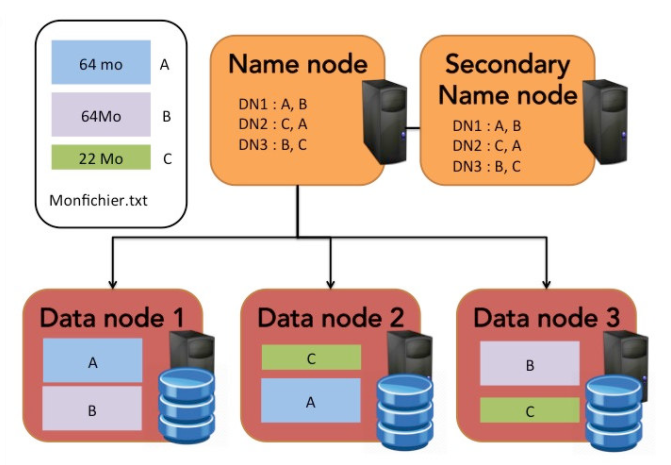
* Un dispositif de stockage et d’accès à des fichiers
* Ces fichiers sont stockés sur un grand nombre de machines de manière à rendre invisible la position exacte d’un fichier. L’accès est transparent, quelle que soient les machines qui contiennent les fichiers.

Il faut aussi savoir que dans HDFS :

* Les fichiers sont physiquement **découpés en blocs d'octets de grande taille** (par défaut 64 Mo) pour optimiser les temps de transfert et d'accès ;
* Ces blocs sont ensuite **répartis** sur plusieurs machines, permettant ainsi de traiter un même fichier en parallèle. Cela permet aussi de ne pas être limité par la capacité de stockage d'une seule machine pour au contraire tirer parti de tout l'espace disponible du cluster de machines ;
* Enfin, pour garantir une tolérance aux pannes, les blocs de chaque fichier sont **répliqués**, de manière intelligente, sur plusieurs machines.

### **L’architecture de stockage :**

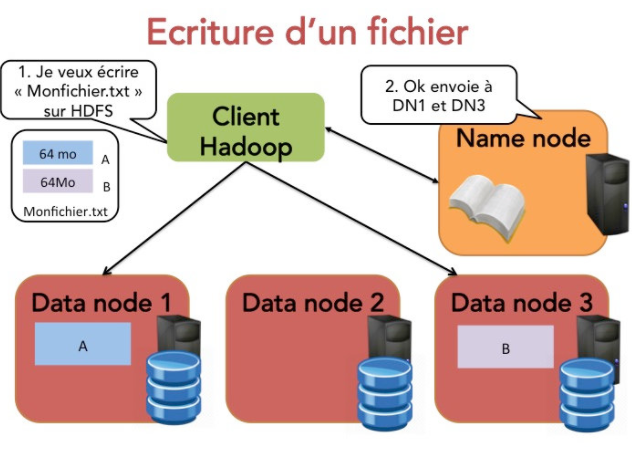
L’architecture dont on parle ici est une architecture maitre-esclave .

* Le nœud maitre appelé **name node** contient et stocke tous les noms et blocs des fichiers ainsi que leur localisation dans le cluster. On peut donc le voir comme un gros annuaire.
* Une autre machine, appelée **secondary name node** sert de namenode de secours en cas de défaillance du nœud maître et il a donc pour rôle de faire des sauvegardes régulières de l'annuaire.
* Les autres nœuds, les esclaves, sont les nœuds de stockage en tant que tels. Ce sont les **data nodes** qui ont pour rôle la gestion des opérations de stockage locales (création, suppression et réplication de blocs) sur instruction du name node.
* 

### **Ecriture d’un fichier HDFS :**

Si on souhaite écrire un fichier dans HDFS, on utilise un client Hadoop. Le principe est assez simple :

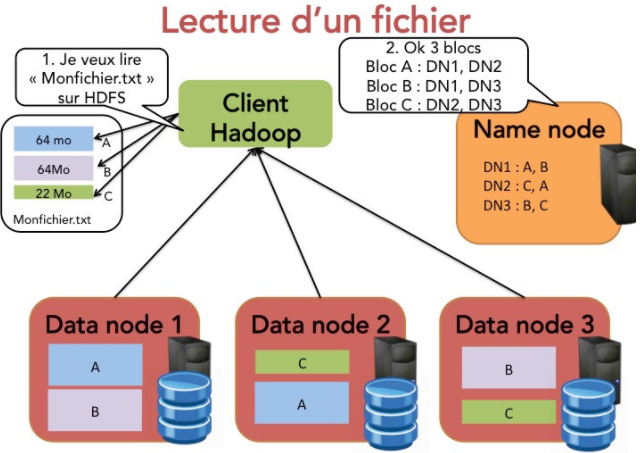
1. Le client indique au name node qu'il souhaite écrire un bloc.
2. Le name node indique le data node à contacter.
3. Le client envoie le bloc au data node.
4. Les data nodes répliquent les blocs entre eux.
5. Le cycle se répète pour le bloc suivant.

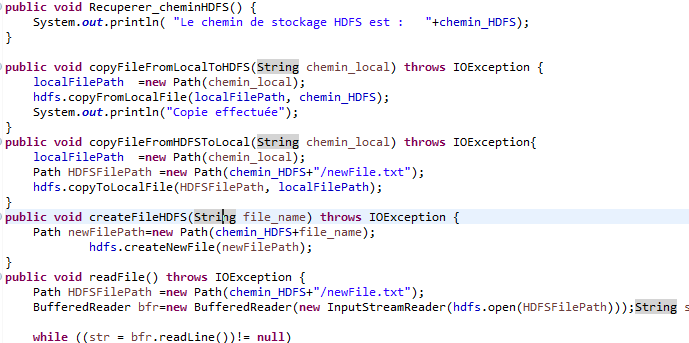


### **Lectured’un fichier HDFS :**

Il y a deux possibilités pour manipuler HDFS :

1. Soit via l'API Java.
2. Soit directement depuis un terminal via les commandes

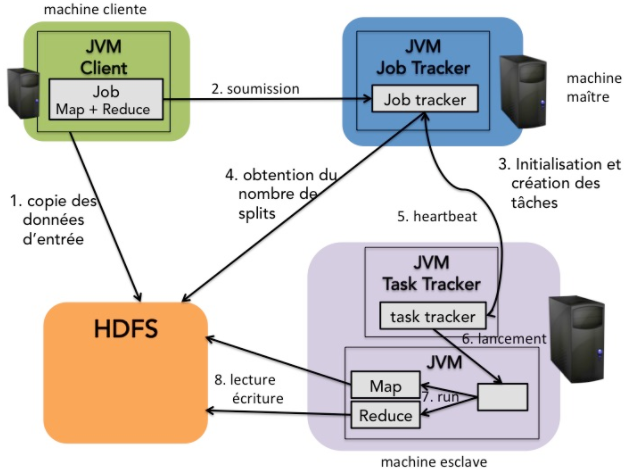




**Traitement distribué :**

Le traitement distribué sera géré par Hadoop en interne avec une nouvelle fois à la base une architecture de type maitre/esclave :

* Le ***job tracker*** est processus maître qui va se charger de l’ordonnancement des traitements et aussi de la gestion de l’ensemble des ressources du système .Il reçoit (du client ) les tâches Map/Reduce à exécuter (.Jar Java ) ainsi que les données d’entrées et le répertoire ou stockées les données de sortie.
* 🡺Il est pour cela en communication directe avec le **Name Node** d’HDFS
* Le job tracker est en charge de planifier l'exécution des tâches et de les distribuer sur des **task trackers**. Comme il sait où sont situés les blocs de données, il peut optimiser la colocalisation traitements/données.
* Un **task tracker** est une unité de calcul du cluster. Il assure, en lançant une nouvelle machine virtuelle java (JVM), l'exécution et le suivi des tâches MAP ou REDUCE s'exécutant sur son nœud et qu'il reçoit du **job tracker**.
* Il dispose d'un nombre limité de slots d'exécution et donc un nombre limité de tâches MAP, REDUCE ou SHUFFLE pouvant s'éxécuter simultanément sur le nœud. Il est aussi en communication constante avec le **job tracker** pour l'informer de l'état d'avancement des tâches (*heartbeat call*)
* S’il y a une panne ou en cas de défaillance, le **job tracker** en est informé et doit pouvoir ordonner la réexécution de la tâche.



*Schéma du traitement distribué*

**API Java pour Hadoop et exemple d’implémentation Map/Reduce :**

Hadoop est écrit en Java et donc fournit des classes et des interfaces Java pour l'écriture des programmes MapReduce. Nous verrons un exemple d’utilisation de Map/Reduce sous Java.

**Classe Map :**

Une classe Map <générique> , qui héritera de la classe **org.apache.hadoop.Mapper**.la classe Map sera paramétrée avec les valeurs suivantes :(in\_key, in\_value,out\_key, inter\_value)

Vue que la classe étends la classe Mapper de Hadoop,on génère automatiquement la fonction map :

* Prend en argument une entrée (key« étant de type général Object » ,value de type Text).
* Mais aussi un contexte d’exécution (de type Context) dans lequel la fonction map va écrire ses résultats

🡺La fonction map doit pouvoir lancer des exceptions, de type IOException ou InterruptedException

Nous venons de voir que cette classe est paramétrée par 4 types.

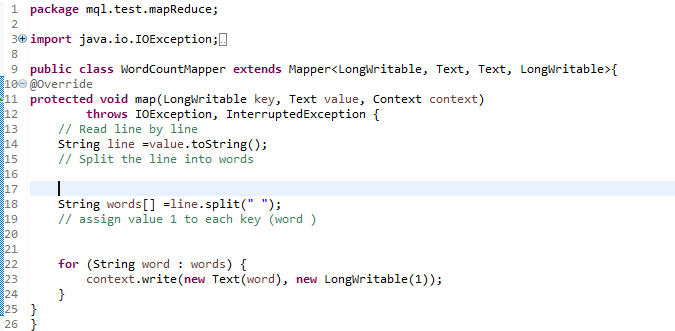
Nous ne pouvons les types standard de Java. Il faut utiliser des types spéciaux qui vont permettre la transmission efficace des données entre les différentes machines du cluster.

* Les valeurs doivent implémenter l'interface writable de l'API Hadoop qui permet la sérialisation et la désérialisation (carles machines doivent s'échanger des données).
* Les clés doivent implémenter l'interface WritableComparable<T>

|  |  |
| --- | --- |
| **Type** | **Description** |
| **Text** | **String (chaine UTF8)** |
| **BooleanWritable** | **Boolean** |
| **IntWritable** | **Int (Entier sur 32 bits )** |
| **LongWritable** | **Long (Entier sur 64 bits)** |
| **FloatWritable** | **Float (Réel sur 32 bit)** |
| **DoubleWritable** | **Double(Réel sur 64 bits)** |

**Les différents types de données Hadoop**

**Exemple de la classe Mapper**

**

**Classe Reduce :**

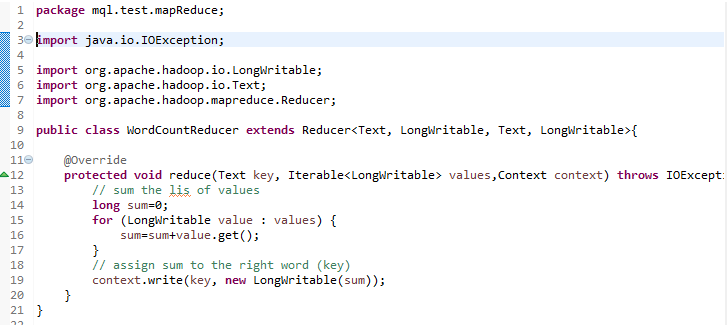
Une classe Reduce <générique> , qui héritera de la classe org.apache.hadoop.Reducer.la classe Reduce sera paramétrée avec les même paramètres que ceux de Map.

Vue que la classe étends la classe Reducer de Hadoop,on génère automatiquement la fonction Reduce :

* Prend en argument une entrée (key« étant de type général Object » ,liste des valeurs de type Iterable<TypeValue>).
* Mais aussi un contexte d’exécution (de type Context) dans lequel la fonction reduce va écrire ses résultats

🡺La fonction reduce doit pouvoir lancer des exceptions, de type IOException ou InterruptedException

**Exemple de la classe Reducer**

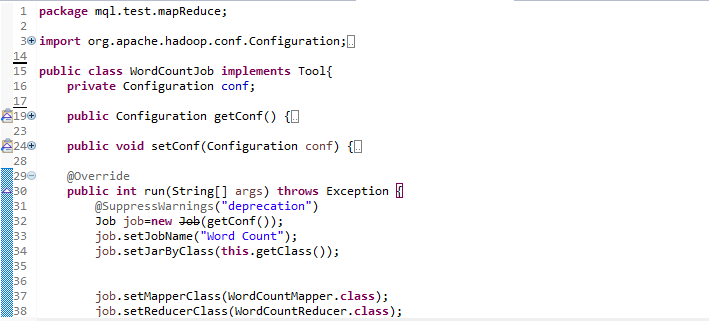


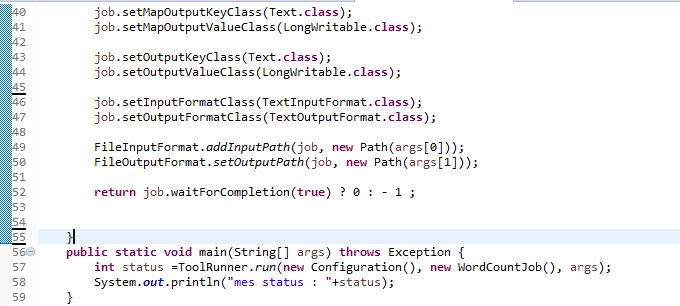
**Classe Job :**

Une classe Job qui implémentera l’interface Tool contiendra notre méthode main() qui permettra de :

* Récupérer la configuration générale du cluster.
* Créer un job.
* Préciser quelles sont les classes Map et Reduce du programme.
* Préciser les types de clés et de valeur correspondant à notre problème.
* Indiquer où sont les données d'entrée et de sortie dans HDFS.
* Lancer l'exécution de la tâche

**Exemple de la classe Job**

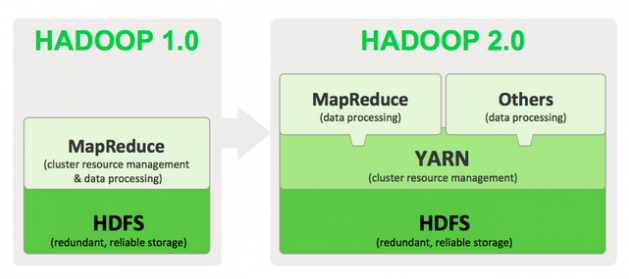




**Hadoop 2.X et Yarn**

L'architecture d'Hadoop a été modifiée pour introduire YARN : ***Yet Another Ressource Negociator***, un framework permettant d'exécuter n'importe quel type d'application distribuée sur un cluster Hadoop, pas uniquement les applications MapReduce.

YARN propose en effet de séparer la gestion des ressources du cluster et la gestion des jobs MapReduce, permettant ainsi de généraliser cette gestion des ressources à d'autres applications. L'idée principale est de considérer que les nœuds ont des ressources (mémoire et CPU) qui seront allouées aux applications quand elles le demandent.



En particulier, dans YARN, les fonctionnalités du job tracker sont réparties entre :

* Le **resource manager** qui est le chef d'orchestre des ressources du cluster. Il ordonnance les requêtes clients et pilote le cluster par l'intermédiaire de **node managers** qui s'exécutent sur chaque nœud de calcul. Il a donc pour rôle de contrôler toutes les ressources du cluster et l'état des machines qui le constituent. Il gère donc le cluster en maximisant l'utilisation de ressources.
* L'**application master (AM)** qui est un processus s'exécutant sur toutes les machines esclaves et qui gère, en discussion avec le **resource manager**, les ressources nécéssaires au travail soumis.

De même, les fonctionnalités du **task tracker** sont aussi réparties sur une même machine entre:

* Des **containers** qui sont des abstractions de ressources sur un nœud dédiées soit à l'exécution de tâches comme Map et Reduce, soit à l'exécution d'un **application master**.
* Un **node manager** qui héberge des **containers** et gére donc les ressources du nœud. Il est en communication via un *heartbeat* avec le ressource manager.

Le schéma de soumission et d'exécution d'un job dans cette nouvelle architecture est donc le suivant :

* Un client hadoop copie ses données sur HDFS.
* Le client soumet le travail à effectuer au **resource manager** sous la forme d'une archive .JAR et des noms des fichiers d'entrée et de sortie.
* Le **resource manager** alloue alors un container pour l'**application master** sur un **node manager**.
* L'**application master** demande au **resource manager** un ou plusieurs containers avec des préférences de localisation dépendant de la localité des données d'entrée du travail.
* Le **resource manager** alloue alors un ou plusieurs containers (child) à l'**application master**.
* L'**application master** choisit parmi la liste des tâches (par exemple Map et Reduce) et demarre une instance de la tâche choisie dans un des **containers** qui lui a été alloué. Il collabore alors avec le **node manager** pour utiliser les ressources acquises. Il communique aussi souvent avec le **resource manager** (message *heartbeat*) pour la tolérance aux pannes.