

No SQL orienté document: Mongo DB

2020/2021

Tasnim.abar@supcom.tn

Plan

- Introduction
- Terminologie
- Structure de données
- Mongo DB: Pratique
- Architecture Mongo : Sharding
- Partitionnement des données
- Mongo DB & MapReduce

Introduction

- Ecrit en C++.
- Orienté documents à schéma flexible et distribuable.
- Interprète les requêtes Javascript (coté serveur).
- Distribué sous licence AGPL (Licence publique générale Affero).

Introduction

- Données représentées dans un schéma flexible.
- Données stockées sous forme de documents (paires clef/valeur) JSONlike.
- La structure du document n'est pas fixée à sa création.
- Les documents sont organisés sous forme de collections.

Terminologie

Relationnel	Orienté document
schéma	database
table	Collection: peut contenir des documents de structures différentes et les documents peuvent contenir eux-mêmes d'autres documents imbriqués.
colonne	Champ
ligne	Document (JSON, XML)
Index	Index
Clé primaire	Une seule, représentée par le <i>champ_id</i>
Jointure	Références et Données Imbriquées

Structure de données

- Les documents sont contenus dans des collections, qui correspondent plus ou moins aux tables des SGBDR.
- Les documents d'une même collection ont une structure similaire.
- Les collections peuvent être regroupées dans des espaces de noms, et sont stockées dans des BDs (databases).
- Un espace de noms est simplement un préfixe ajouté aux collections (et séparé de celles-ci par un point), qui permet de les regrouper, un peu comme le concept de schémas de la norme SQL.
- Une BD peut être considérée comme une collection de collections.

Structure de données

Un document:

- Stocké sur le disque sous forme de document BSON :
- ✓ Documents BSON (Binary JSON) : représentation binaire sérialisées d'une document JSON.
- Chaque document crée contient le **champ id** :
 - ✓ Sa valeur peut être fournie ou générée automatiquement.
 - ✓ Type primary key.
 - ✓ Indexé. - Structure de données JSON-like, composée de paires clef/valeur.
 - ✓ Les champs indexés ont une taille limite (1Mo)
 - ✓ Les noms des champs ne peuvent pas commencer par un \$, contenir le caractère « . » ou le caractère « null »

```
{  
  _id: ObjectId("5099803df3f4948bd2f98391"),  
  name: { first: "Alan", last: "Turing" },  
}
```

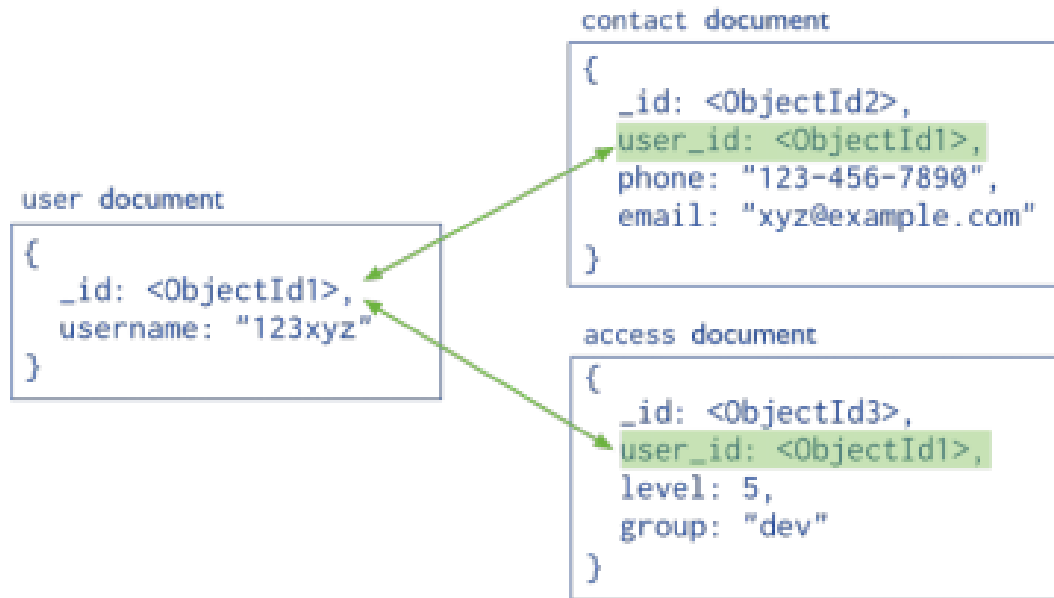
Structure de données

Limite d'un document:

- Taille max d'un document : 4Mo
- ✓ Utilisation de l'API GridFS pour stocker des documents plus larges que la taille autorisée (GridFS permet de diviser les documents en chunks de même taille, et les stocke sous forme de documents séparés)
- MongoDB préserve l'ordre des champs du document, comme défini à leur création, sauf que le champs `_id` doit toujours figurer en premier.
- Les opérations de update qui incluent le renommage d'un champs peut entraîner le changement de l'ordre des champs dans le document.

Structure de données

- Deux manières des relations entre les données: Références et Données Imbriquées
- Références :
 - ✓ Inclusion de liens ou références d'un document à un autre associées : On dit qu'on utilise des Modèles de Données Normalisés



Structure de données

- Données Imbriquées (Embedded Data) : Sauvegarde des données associées dans la même structure de documents.
- Il est possible d'inclure des documents dans un champ ou un tableau
- ✓ Permet aux applications d'extraire et manipuler plusieurs niveaux de hiérarchie en une seule instruction
- ✓ Ce sont les Modèles de Données Dénormalisés



Structure de données

- Données Imbriquées (Embedded Data) : Sauvegarde des données associées dans la même structure de documents.
- Il est possible d'inclure des documents dans un champ ou un tableau
- ✓ Permet aux applications d'extraire et manipuler plusieurs niveaux de hiérarchie en une seule instruction
- ✓ Ce sont les Modèles de Données Dénormalisés



Structure de données

- Comment choisir entre Références et Données Imbriquées?
- On choisit les références quand:
 - ✓ L'imbrication va produire des données dupliquées sans grands avantages en terme de performance de lecture.
 - ✓ On veut modéliser de larges ensembles de données hiérarchiques.
- On choisit les données imbriquées quand:
 - ✓ On a des relations contains entre les éléments (modèles one-to-one)
Exemple: personne et nom.
 - ✓ On a des relations one-to-many, où les documents fils (many) apparaissent toujours dans le contexte des documents parents (one)
Exemple : personne et plusieurs téléphones.

Mongo DB: Pratique

- Caractéristiques d'une instance MongoDB :
 - ✓ Un port d'écoute (par défaut 27017).
 - ✓ Un processus serveur.
 - ✓ Un répertoire racine de stockage.
 - ✓ Un fichier de log.
 - ✓ Un fichier de configuration : mongod.conf.
 - ✓ Les outils et commandes MongoDB :
 - ❖ mongod : le moteur de base.
 - ❖ mongo : le shell Javascript.
 - ❖ mongos : le contrôleur de Sharding (partitionnement).
 - ✓ Les outils d'import/export : mongoimport, mongoexport

Mongo DB: Pratique

- La distribution de MongoDB inclue un outil en ligne de commande (MongoDB interactive shell).
- Cet utilitaire permet d'envoyer des commandes au serveur à partir de la ligne de la commande.
- Il permet aussi d'exécuter des fichiers JavaScript pour exécuter des scripts d'administration.
- Ce shell permet de :
 - ✓ Consulter le contenu de la base.
 - ✓ Tester des requêtes de consultation.
 - ✓ Créer des indexes.
 - ✓ Exécuter des scripts de maintenance.
 - ✓ Administrer la base.

Mongo DB: Pratique

- Pour afficher la base en cours :
 > db
- Pour afficher la liste des bases :
 > show dbs
- Mongo créé par défaut deux bases :
 - ✓ local : une base qui a la particularité de n'être jamais dupliqué et qui peut servir à stocker des documents qui n'ont d'utilité que sur une machine.
 - ✓ test : base vide sans particularité.
- Il peut aussi contenir une base admin permettant à ses utilisateurs d'utiliser des commandes d'administration (comme l'arrêt du serveur) qui ne sont pas disponibles avec les autres bases et config qui est utilisée en mode partitionnement pour stocker des informations sur les nœuds

Mongo DB: Pratique

- Création / Suppression - Démarrage de l'outil en ligne de commande :

`$ mongo`

- Création de bases de données :

`> use <db name>`

- Suppression de bases de données :

`> use <db name> ;`

`> db.runCommand({dropDatabase : 1}) ;` ou

`> db.dropDatabase() ;`

- Collection: équivalent à une table en relationnel:
- ✓ Deux modes de création:
 - Automatiquement lors d'une insertion d'un document
 - En utilisant la commande `db.createCollection`

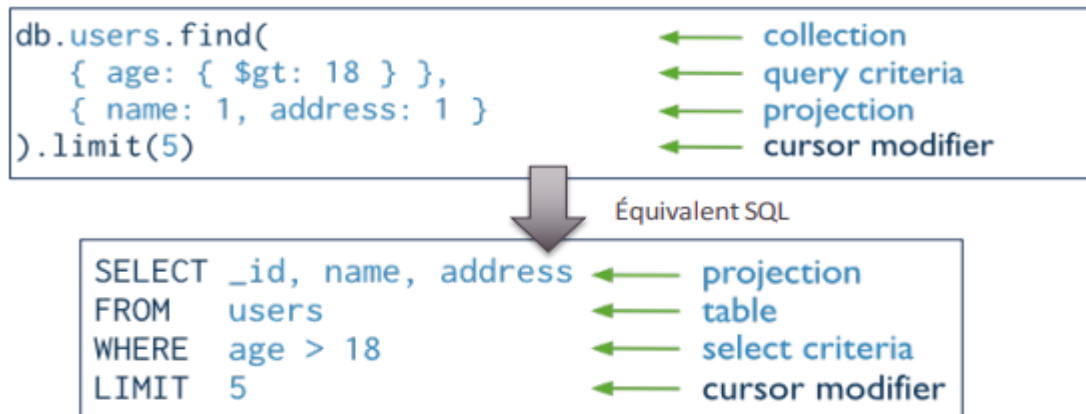
Mongo DB: Pratique

- Les opérations sur les collections:
 - Créer : - > `db.createCollection('TEKUP')`.
 - Supprimer : - > `db.collection.drop()`.
 - Lister : - > `show collections`.
 - > `db.getCollections()`.
 - Insérer un document :
 - > `show collections`.
 - > `db.<collection name>.insert({var1 : "valeur", var2 : "valeur", var3 : "valeur",})` ;
 - Supprimer : - > `db.<nom collection>.drop()` ;
 - Lister les bases créées: > `show dbs`
 - Création de la base > `use bibliotheque`
 - > `show collections`
 - Renommer une Collection :> `db.Editeurs.renameCollection("editeurs");`

Mongo DB: Pratique

Lecture des documents:

- Cible une unique collection spécifique de documents
- Cible un critère ou des conditions spécifiques, pour identifier le document à retourner
- Peut inclure une projection sur les champs du document à retourner
- Peut définir des Modificateurs pour imposer des limites, un ordre, un filtre...
- Opérations de lecture:
- **db.collection.find()** : pour l'extraction de données.
- **db.collection.findOne()** : retourne un seul document.



Mongo DB: Pratique

Ecriture des documents:

- Création, mise à jour ou suppression de données
- Ces opérations modifient les données d'une seule collection
- Définition de critères pour sélectionner les documents à modifier ou supprimer
- Opérations d'écriture:
- **db.collection.insert()** : pour l'insertion d'un nouveau document.

Si vous ajoutez un document sans champ `_id`, le système l'ajoute lui-même en générant un champ de type `ObjectId`

```
Collection
  ↓
db.users.insert(
  {
    name: "sue",
    age: 26,
    status: "A",
    groups: [ "news", "sports" ]
  }
)
```

Mongo DB: Pratique

Ecriture des documents:


- **db.collection.update()** : mise à jour d'un document.

Une opération update modifie un seul document par défaut (plusieurs docs si **multi:true**)

Existence de l'option **upsert: true**, : si le document à modifier n'existe pas dans la collection, il est automatiquement inséré.

- **db.collection.remove()** : supprimer un document.

```
db.users.update(  
  { age: { $gt: 18 } },  
  { $set: { status: "A" } },  
  { multi: true }  
)
```



```
db.users.remove(  
  { status: "D" }  
)
```



Mongo DB: Pratique

Quelques opérateurs utilisés avec Mongo

opérateur	exemple	Rôle
<code>.aggregate()</code>	<code>db.myColl.aggregate([{ \$match: { status: "A" } }, { \$group: { _id: "\$category", count: { \$sum: 1 } } }]);</code>	Calcule les valeurs agrégées des données d'une collection.
<code>.sort{field:1}</code>	<code>db.bios.find().sort({ name: 1 })</code>	renvoie les documents de la collection de bios triés par ordre croissant selon le champ de nom
<code>.limit()</code>	<code>db.collection.find().limit(N)</code>	Selectionner les N premiers documents
<code>.skip()</code>	<code>db.collection.find().skip(M)</code>	Selectionner les documents à partir de Mième document
<code>createIndex({field:1})</code>	<code>db.inventory.createIndex({ quantity: 1 })</code>	Créer un index sur le champ quantity

Mongo DB: Pratique

opérateur	exemple	Rôle
\$gt	<code>db.collection.find({ qty: { \$gt: 4 } })</code>	retourner les documents dont la valeur de qté est supérieure à 4.
\$ne	<code>db.inventory.find({ price: { \$ne: 200 } })</code>	sélectionner tous les documents de la collection inventory où la valeur du champ prix n'est pas égale à 200
\$lte	<code>db.inventory.find({ { quantity: { \$lte: 20 } } })</code>	sélectionner tous les documents de la collection d'inventaire où la valeur du champ de quantité est inférieure ou égale à 20.
\$or	<code>db.inventory.find({ \$or: [{ quantity: { \$lt: 20 } }, { price: 10 }] })</code>	sélectionner tous les documents de la collection d'inventaire où la valeur du champ de quantité est inférieure à 20 ou la valeur du champ de prix est égale à 10.
\$and	<code>db.inventory.find({ \$and: [{ price: { \$ne: 1.99 } }, { price: { \$exists: true } }] })</code> <code>db.inventory.find({ price: { \$ne: 1.99, \$exists: true } })</code>	sélectionner tous les documents de la collection où: la valeur du champ de prix n'est pas égale à 1,99 et le champ de prix existe.

Mongo DB: Pratique

opérateur	exemple	Rôle
\$exists	<code>db.records.find({ a: { \$exists: true } })</code>	sélectionner tous les documents de la collection où le champ a existe.
\$match	<code>{ \$match: { \$expr: { <aggregation expression> } } }</code> <code>db.articles.aggregate([{ \$match : { author : "dave" } }]);</code>	permet la sélection (find). sélectionner les documents où le champ auteur est égal à dave
\$project	<code>db.books.aggregate([{ \$project : { _id: 0, title : 1 , author : 1 } }])</code>	exclut le champ _id et inclut le titre et les champs auteur dans les documents de sortie:
\$group	<code>db.sales.aggregate([{ \$group : { _id : "\$item" } }])</code>	récupérer les valeurs distinctes des « item » de la collection sales.

Mongo DB: Pratique

Quelques opérateurs utilisés avec Mongo: Les tableaux

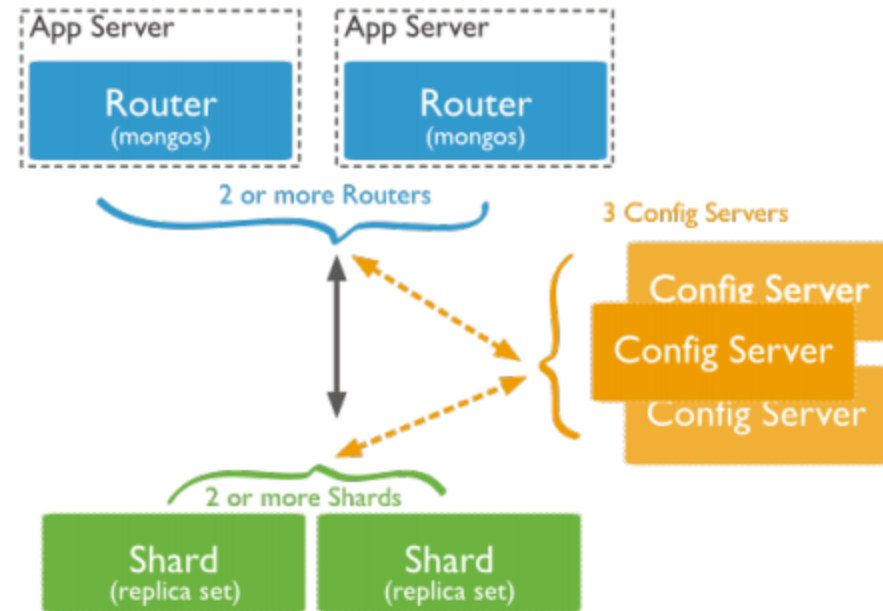
opérateur	exemple	Rôle
\$in	<code>db.tekup.find({ enseignants: { \$in: [« Ahmed", "Mourad"]} })</code>	retourne des documents dans la collection tekup où le champ tableau enseignants contient l'élément « Ahmed" ou « Mourad"
\$all	<code>db.tekup.find({ enseignants: { \$all: [" Ahmed", "Mourad"]} })</code>	retourne des documents dans la collection tekup où le champ tableau enseignants contient l'élément " Ahmed" et " Mourad"
\$size	<code>db.AAA.find({ BBB: { \$size: 4 } })</code>	retourner des documents dans la collection AAA où la taille du tableau BBB est 4:
\$unwind	<code>db.sales.aggregate(\$unwind: "\$items")</code>	génère un nouveau document pour chaque élément du tableau « item »

Sharding

- MongoDB utilise la notion de Sharding (ou horizontal scaling) pour stocker ses données dans le cluster.
- Division des données et leur distribution entre plusieurs serveurs ou shards.
- Réduction de la quantité de données que le serveur a besoin de stocker.
- Réduction du nombre d'opérations que chaque machine gère.

Sharding

- **Shards**: Stocker les données (les données sont distribuées et répliquées sur les shards)
 - **Query Routers**: Instances mongos, Interfaçage avec les applications clientes: permet de Rediriger les opérations vers le shard approprié et retourne le résultat au client.
 - **Config Servers**: Stocker les méta - données du cluster, Définir le mapping entre les data et les shards.
- ✓ 3 Config Servers doivent être définis.

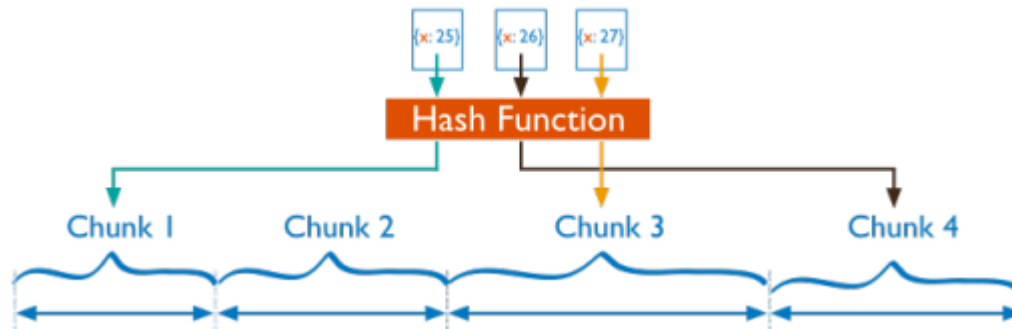


Partitionnement des données

- **Shard key:**
 - ✓ permet le Partitionnement des données au niveau des collections.
 - ✓ Champ simple ou composé indexé qui existe dans chaque document de la collection.
 - ✓ MongoDB divise les valeurs de la clef en morceaux (chunks) et les distribuent de manière équitable entre les shards.
 - ✓ On utilise l'une de ces 3 méthodes pour la répartition des clefs:
 - ❖ Basée sur le Hash
 - ❖ Basée sur le rang
 - ❖ Basée sur les tags

Partitionnement des données

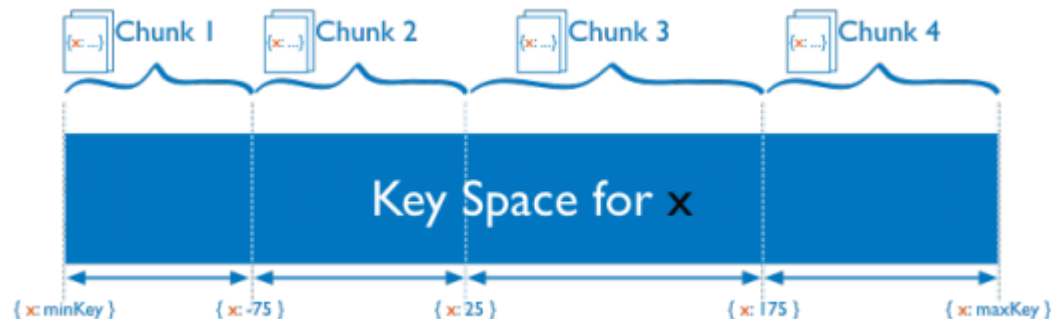
- **Basé sur le Hash:**
 - ✓ Calcul de la valeur du hash d'un champ, puis utilise ces hash pour créer des partitions .
 - ✓ Deux documents ayant des clefs proches ont peu de chance de se trouver dans le même shard.
 - ✓ Distribution aléatoire d'une collection dans le cluster, d'où une distribution plus équitable .
 - ✓ Moins efficace, car le temps de recherche de la donnée est plus grand.
 - ✓ Dans le cas d'une requête portant sur des données se trouvant dans un intervalle défini, le système doit parcourir plusieurs shards



Partitionnement des données

- **Basé sur le rang:**

- ✓ Définition d'intervalles qui ne se chevauchent pas, dans lesquelles les valeurs de la shard key peuvent se trouver.
- ✓ Permet aux documents avec des clefs proches de se trouver dans le même shard.
- ✓ Plus facile de retrouver le shard pour une donnée.
- ✓ Risque de distribution non équitable des données (par exemple si la clef est le temps, alors toutes les requêtes dans un même intervalle de temps sont sur le même serveur, d'où une grande différence selon les heures de grande ou de faible activité).



Partitionnement des données

- **Basé sur le tag**

- ✓ Les administrateurs peuvent définir des tags, qu'ils associent à des intervalles de clef .
- ✓ Ils associent ces tags aux différents shards en essayant de respecter la distribution équitable des données.
- ✓ Un balancer migre les données taggées vers les shards adéquats § Le meilleur moyen pour assurer une bonne répartition des données

Partitionnement des données

- Pour activer le Sharding pour une base de donnée:
`>db.runCommand({enablesharding : <nom de db>})`
- Définir une clé de partitionnement pour le sharding :
`>db.runCommand({shardcollection : "<namespace>", key : { "<nom champ>" : 1 } }) ;`
- Afficher les informations sur le sharding :
`>db.printShardingStatus() ;`
`>db.<collection>.stats() ;`

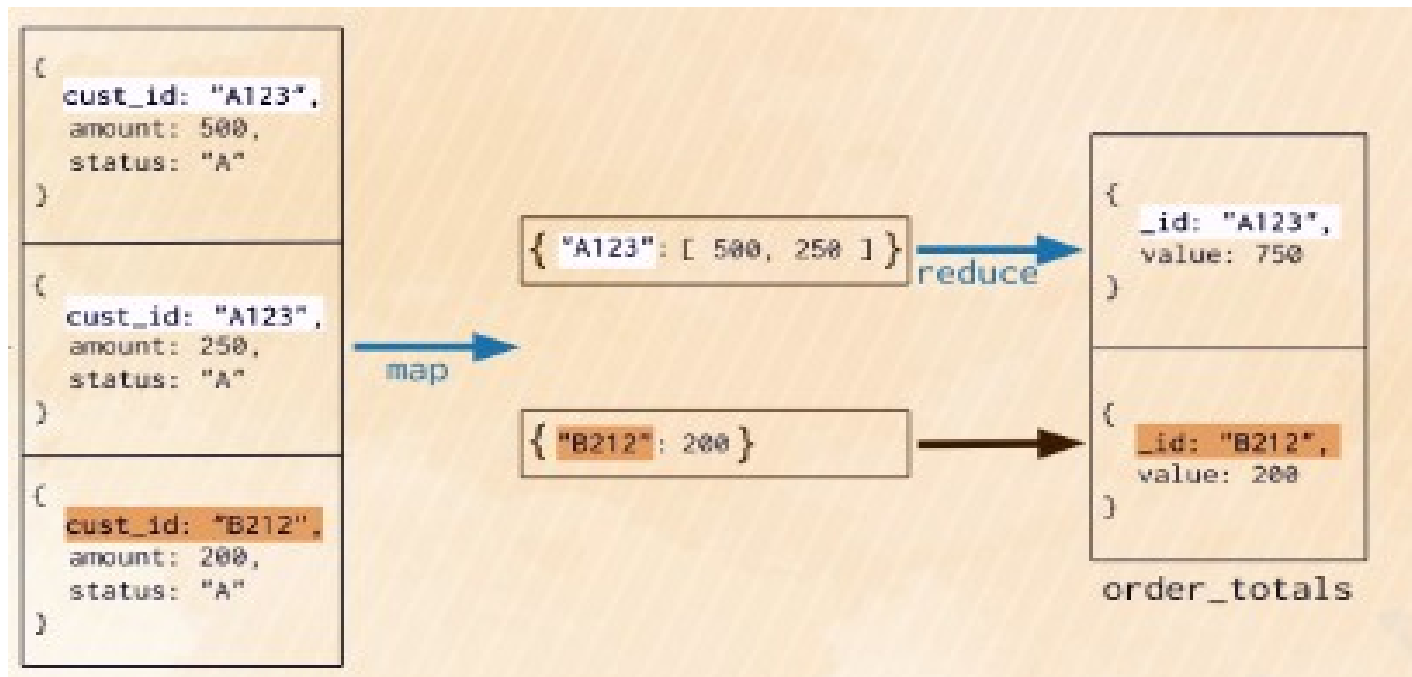
Mongo & MapReduce

- MongoDB possède également une fonction mapreduce() interne.
- Elle est relativement limitée (essentiellement par le type de traitement qu'on peut effectuer sur les données), mais peut avoir son utilité.

```
db.COLLECTION.mapreduce(FONCTION_MAP,  
                        FONCTION_REDUCE,  
                        {query: {CONDITIONS}},  
                        out: "NOM_SORTIE"))
```

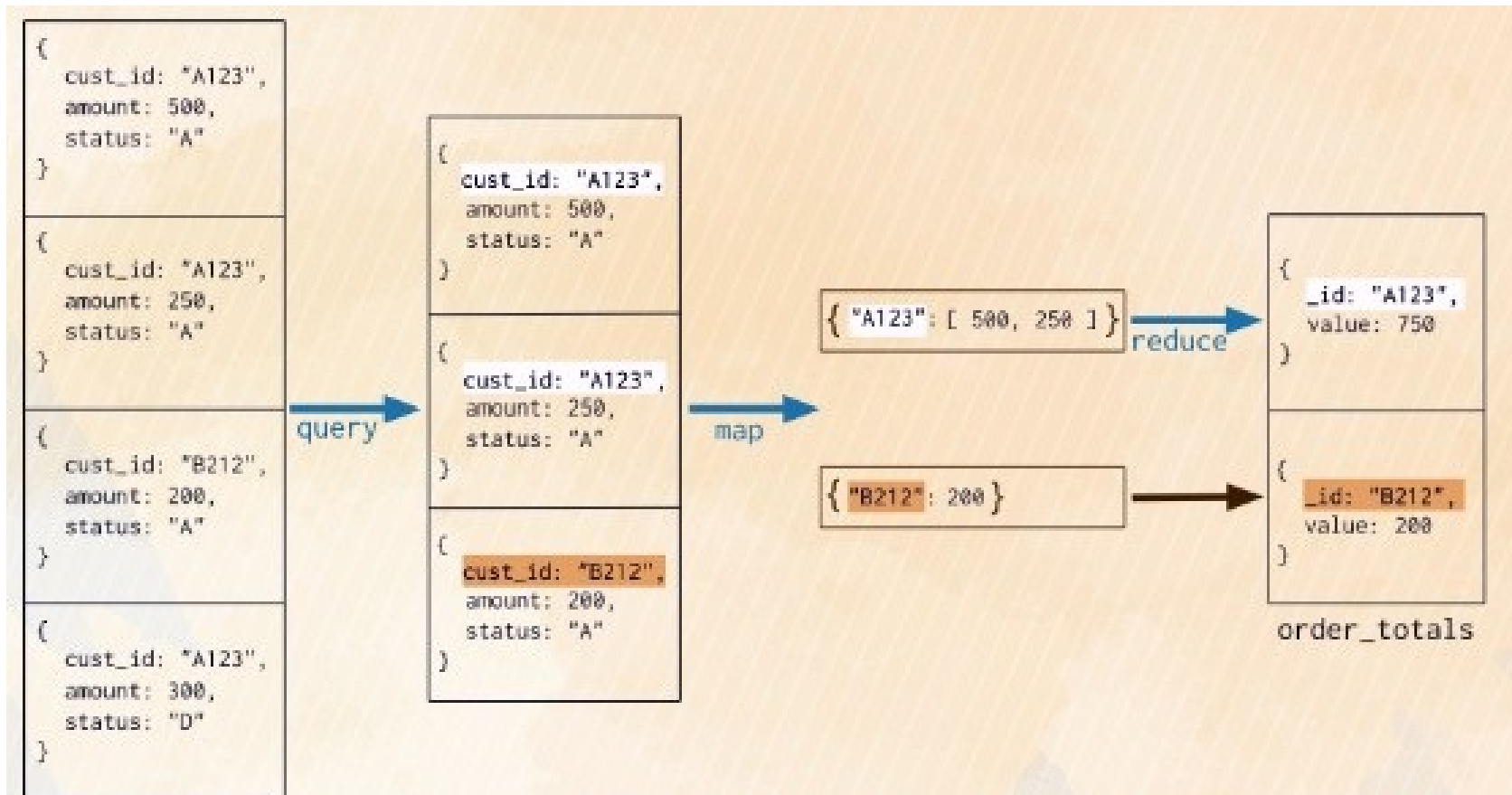

Mongo & MapReduce

- Collection : orders;
- Map -> **var** mapFunction = function () { emit(this.cust_id, this.amount); },
- Reduce -> function (key, value) { return Array .sum (values) };
- Output -> out : "order_totals "



Mongo & MapReduce

- Collection : orders;
- Map -> **var** mapFunction = function () { emit(this.cust_id, this.amount); };
- Reduce -> **var** reduceFunction = function (key, value) { return Array .sum (values) };
- Query -> query : {status : " A "},
- Output -> out : "order_totals "



Mongo & MapReduce

```
db.orders.mapReduce( mapFunction, reduceFunction, { out:  
  "order_totals" } );
```

- > Effectuez la la fonction map-reduce sur tous les documents de la collection ordres à l'aide de la fonction map mapFunction et de la fonction reduce reduceFunction1.
- >db.order_totals.find(); pour afficher le résultat final.