Dernière mise-à-jour : 2020/02/21 07:43

LRF151 - Administration du Serveur MongoDB sous RHEL 7

Dans cette formation vous allez apprendre :

- Ce qui est le NOSQL,
- Ce qui est MongoDB,
- Comment installer et configurer MongoDB,
- Comment utiliser le client mongo,
- Quels sont les clients graphiques disponibles pour MongoDB,
- Quelle est la structure des données de MongoDB,
- Quel est le langage des requêtes de MongoDB,
- Comment utiliser ce langage efficacement.

Présentation

Qu'est-ce le NOSQL ?

La naissance des outils NOSQL (*Not Only SQL*) a ses origines dans les limitations des **propriétés ACID**. En effet, c'est les grandes sociétés du web, qui, amenées à traiter des volumes de données très importants, ont été les premières confrontées aux limitations intrinsèques des SGBD relationnels traditionnels.

De ce fait, chaque société a adopté ou a dévélopé sa propre solution de gestion de données :

- Google - **BigTable (Fr)**,
- Facebook - **Cassandra (Fr)** puis **HBase (Fr)**,
- Ubuntu One - **CouchDB (Fr)**,
- Baidu - **Hypertable (Fr)**,
- Amazon - **Dynamo (En)**

- LinkedIn - **Voldemort (En)**,
- SourceForge.net - **MongoDB**.

C'est à travers de ce dernier que ce cours propose de vous faire découvrir la mouvance NOSQL.

Présentation de MongoDB

Dans le cas de MongoDB, les données prennent la forme de **documents** enregistrés dans des **collections**. Une collection peut contenir un nombre quelconque de documents. Cependant, les champs d'un enregistrement sont libres et peuvent être différents d'un enregistrement à un autre dans la même collection. Le seul champ commun et obligatoire est le champ de la clé principale : **_id**.

Fonctionnalités de MongoDB

- La réplication permet de dupliquer les serveurs de base de données pour répondre à une montée 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 soit pour répartir les données géographiquement,
- Système de fichiers **GridFS** qui 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.

Historique du Projet

MongoDB (de *Humongous* qui veut dire énorme ou immense) :

- est développé depuis 2007 par MongoDB,
- a été industriellement viable en 2010 avec la version 1.4,
- est écrit en langage C++.

Versions Majeures

Version de MongoDB	Améliorations par rapport à la version précédente
MongoDB 1.2.x	Création d'index plus rapide Fonctions JavaScript stockées La commande fsync configurable Plusieurs petites fonctionnalités et corrections
MongoDB 1.4	Amélioration de la mémoire d'indexation Une meilleure détection des expressions régulières
MongoDB 1.6	L'option w (et wtimeout) peut se propager vers plusieurs serveurs La commande findAndModify supporte les upserts Option $ showDiskLoc permet de voir l'emplacement du disque d'un document Prise en charge des sockets IPv6 et UNIX
MongoDB 1.8	Mode écriture avant Journaling pour faciliter la récupération du crash et la durabilité dans le moteur de stockage Correction d'un problème de concurrence avec de nombreuses connexions entrantes
MongoDB 2.0	Journaling est activé par défaut dans la version 2.0 pour les versions 64 bits La commande compact est maintenant disponible pour le compactage des index Réduction de la taille de la pile par défaut Améliorations des Indices de performance Les applications peuvent maintenant utiliser l' authentification avec les clusters fragmentées
MongoDB 2.2	Opérations d'agrégation Collections TTL permet supprimer les données périmées d'une collection Augmentation de la capacité du serveur pour les opérations simultanées Amélioration de la sensibilisation Data Center avec Tag Aware Sharding
MongoDB 2.4	Fonctionnalité Text Search Ajout d'un nouvel indice de 2dsphere Ajout d'un Index Hashed pour indexer des documents Améliorations de la sécurité
MongoDB 2.6	Agrégation Améliorées Text Search activé par défaut Nouveau protocole d'écriture Package MSI pour MongoDB Disponible pour Windows

Version de MongoDB	Améliorations par rapport à la version précédente
MongoDB 3.0	MongoDB 3.0 introduit le WiredTiger comme moteur de stockage Amélioration du moteur de stockage MMAPv Augmentation du nombre de Replica Set Members Amélioration des Clusters fragmentées Améliorations des requêtes
MongoDB 3.2	WiredTiger comme moteur par défaut de stockage Amélioration des Clusters Index partiels disponibles pour indexer des documents Nouveaux opérateurs de requête SpiderMonkey JavaScript Engine

Exécutables

Le tableau suivant indique les noms des exécutables selon la base de donnée utilisée :

	MongoDB	MySQL	Oracle	Informix	DB2
Serveur	mongod	mysqld	oracle	IDS	DB2 Server
Client	mongo	mysql	sqlplus	DB-Access	DB2 Client

Avantages et Inconvénients

Avantages

MongoDB :

- ne nécessite pas de schéma prédéfini des données d'où sa grande flexibilité,
- est une base de données orientée documents qui s’adapte parfaitement à de nombreuses applications, telles les applications de gestion de dossiers, factures, commandes ou produits,
- supporte une évolutivité à la charge (*scaling*) par l'ajout de machines,
- supporte des indexes secondaires,
- propose un langage complet de requêtes et une stricte cohérence et stabilité,
- propose du calcul en mémoire, d'où la lecture et écriture rapides,

- propose la réplication en mode maître/esclave et le basculement automatique,
- propose le Sharding (répartition automatique de données sur plusieurs machines).

Inconvénients

MongoDB :

- n'a pas de possibilité de réaliser de jointures, les données sont généralement embarquées dans le même document,
- ne peut pas gérer de transactions complexes,
- impose que la charge du contrôle des données soit reportée du côté de l'application puisque il n'y a pas de schéma,
- propose un langage d'interrogation qui lui est propre (donc, non standardisé), pratique mais qui s'avère limité.

Installation de MongoDB

Créez le fichier **/etc/yum.repos.d/mongodb-org-3.2.repo** :

```
[root@centos7 ~]# vi /etc/yum.repos.d/mongodb-org-3.2.repo
[root@centos7 ~]# cat /etc/yum.repos.d/mongodb-org-3.2.repo
[mongodb-org-3.2]
name=MongoDB Repository
baseurl=https://repo.mongodb.org/yum/redhat/$releasever/mongodb-org/3.2/x86_64/
gpgcheck=1
enabled=1
gpgkey=https://www.mongodb.org/static/pgp/server-3.2.asc
```

Installez mongo :

```
[root@centos7 ~]# yum install mongodb-org
Loaded plugins: fastestmirror, langpacks
adobe-linux-x86_64
| 2.9 kB  00:00:00
```

```
base
| 3.6 kB  00:00:00
extras
| 3.4 kB  00:00:00
mongodb-org-3.2
| 2.5 kB  00:00:00
updates
| 3.4 kB  00:00:00
(1/6): adobe-linux-x86_64/primary_db
| 2.7 kB  00:00:01
(2/6): extras/7/x86_64/primary_db
| 101 kB  00:00:00
(3/6): base/7/x86_64/group_gz
| 156 kB  00:00:01
(4/6): mongodb-org-3.2/7/primary_db
|  72 kB  00:00:02
(5/6): base/7/x86_64/primary_db
| 5.7 MB  00:00:03
(6/6): updates/7/x86_64/primary_db
| 2.8 MB  00:00:02
Determining fastest mirrors
 * base: ftp.ciril.fr
 * extras: ftp.ciril.fr
 * updates: centos.crazyfrogs.org
Resolving Dependencies
--> Running transaction check
---> Package mongodb-org.x86_64 0:3.2.16-1.el7 will be installed
--> Processing Dependency: mongodb-org-tools = 3.2.16 for package: mongodb-org-3.2.16-1.el7.x86_64
--> Processing Dependency: mongodb-org-shell = 3.2.16 for package: mongodb-org-3.2.16-1.el7.x86_64
--> Processing Dependency: mongodb-org-server = 3.2.16 for package: mongodb-org-3.2.16-1.el7.x86_64
--> Processing Dependency: mongodb-org-mongos = 3.2.16 for package: mongodb-org-3.2.16-1.el7.x86_64
--> Running transaction check
---> Package mongodb-org-mongos.x86_64 0:3.2.16-1.el7 will be installed
---> Package mongodb-org-server.x86_64 0:3.2.16-1.el7 will be installed
```

```
---> Package mongodb-org-shell.x86_64 0:3.2.16-1.el7 will be installed
---> Package mongodb-org-tools.x86_64 0:3.2.16-1.el7 will be installed
--> Finished Dependency Resolution

Dependencies Resolved

================================================================================================================================
================================================================================
 Package                                          Arch                                    Version
Repository                                      Size
================================================================================================================================
================================================================================
Installing:
 mongodb-org                                      x86_64                                  3.2.16-1.el7
mongodb-org-3.2                                5.8 k
Installing for dependencies:
 mongodb-org-mongos                               x86_64                                  3.2.16-1.el7
mongodb-org-3.2                                5.7 M
 mongodb-org-server                               x86_64                                  3.2.16-1.el7
mongodb-org-3.2                                 13 M
 mongodb-org-shell                                x86_64                                  3.2.16-1.el7
mongodb-org-3.2                                6.8 M
 mongodb-org-tools                                x86_64                                  3.2.16-1.el7
mongodb-org-3.2                                 41 M

Transaction Summary
================================================================================================================================
================================================================================
Install  1 Package (+4 Dependent packages)

Total download size: 66 M
Installed size: 202 M
Is this ok [y/d/N]: y
```

Activez et démarrez le service :

```
[root@centos7 ~]# systemctl status mongod
● mongod.service - SYSV: Mongo is a scalable, document-oriented database.
   Loaded: loaded (/etc/rc.d/init.d/mongod; bad; vendor preset: disabled)
   Active: inactive (dead)
     Docs: man:systemd-sysv-generator(8)
[root@centos7 ~]# systemctl enable mongod
mongod.service is not a native service, redirecting to /sbin/chkconfig.
Executing /sbin/chkconfig mongod on
[root@centos7 ~]# systemctl start mongod
[root@centos7 ~]# systemctl status mongod
● mongod.service - SYSV: Mongo is a scalable, document-oriented database.
   Loaded: loaded (/etc/rc.d/init.d/mongod; bad; vendor preset: disabled)
   Active: active (running) since Mon 2017-09-18 13:46:09 CEST; 4s ago
     Docs: man:systemd-sysv-generator(8)
  Process: 14244 ExecStart=/etc/rc.d/init.d/mongod start (code=exited, status=0/SUCCESS)
   CGroup: /system.slice/mongod.service
           └─14255 /usr/bin/mongod -f /etc/mongod.conf

Sep 18 13:46:08 centos7.fenestros.loc systemd[1]: Starting SYSV: Mongo is a scalable, document-oriented
database....
Sep 18 13:46:08 centos7.fenestros.loc runuser[14251]: pam_unix(runuser:session): session opened for user mongod
by (uid=0)
Sep 18 13:46:09 centos7.fenestros.loc mongod[14244]: Starting mongod: [  OK  ]
Sep 18 13:46:09 centos7.fenestros.loc systemd[1]: Started SYSV: Mongo is a scalable, document-oriented database..
```

Le client MongoDB est invoqué avec la commande mongo :

```
[root@centos7 ~]# mongo
MongoDB shell version: 3.2.16
connecting to: test
Welcome to the MongoDB shell.
For interactive help, type "help".
```

```
For more comprehensive documentation, see
    http://docs.mongodb.org/
Questions? Try the support group
    http://groups.google.com/group/mongodb-user
Server has startup warnings:
2017-09-18T13:46:09.416+0200 I CONTROL  [initandlisten]
2017-09-18T13:46:09.416+0200 I CONTROL  [initandlisten] ** WARNING: soft rlimits too low. rlimits set to 4096
processes, 64000 files. Number of processes should be at least 32000 : 0.5 times number of files.
> exit
bye
[root@centos7 ~]#
```

Configuration

Le fichier de configuration de mongodb est **/etc/mongod.conf** :

```
[root@centos7 ~]# cat /etc/mongod.conf
systemLog:
  destination: file
  logAppend: true
  path: /var/log/mongodb/mongod.log

storage:
  dbPath: /var/lib/mongo
  journal:
    enabled: true

processManagement:
  fork: true  # fork and run in background
  pidFilePath: /var/run/mongodb/mongod.pid  # location of pidfile

net:
  port: 27017
```

```
  bindIp: 127.0.0.1
```

Dans ce fichier, sont notamment définit le lieu de stockage des données et des logs respectivement dans **/var/lib/mongo** et **/var/log/mongodb/mongod.log** par défaut.

Le client Mongo

Une manipulation intéressante à faire consiste à modifier le prompt du client mongo pour afficher la base de donnée courante. Par défaut le prompt est minimaliste :

```
>
```

Pour connaître le base de donnée sur laquelle on travaille il convient de de taper “db”

```
>db
test
>
```

Créez donc le fichier $HOME/.mongorc.js :

```
[root@centos7 ~]# vi $HOME/.mongorc.js
[root@centos7 ~]# cat $HOME/.mongorc.js
prompt = function(){return db+">";}
```

Constatez que la base de données courante se trouve dans le prompt :

```
[root@centos7 ~]# mongo
MongoDB shell version: 3.2.16
connecting to: test
Server has startup warnings:
2017-09-18T13:46:09.416+0200 I CONTROL  [initandlisten]
2017-09-18T13:46:09.416+0200 I CONTROL  [initandlisten] ** WARNING: soft rlimits too low. rlimits set to 4096
```

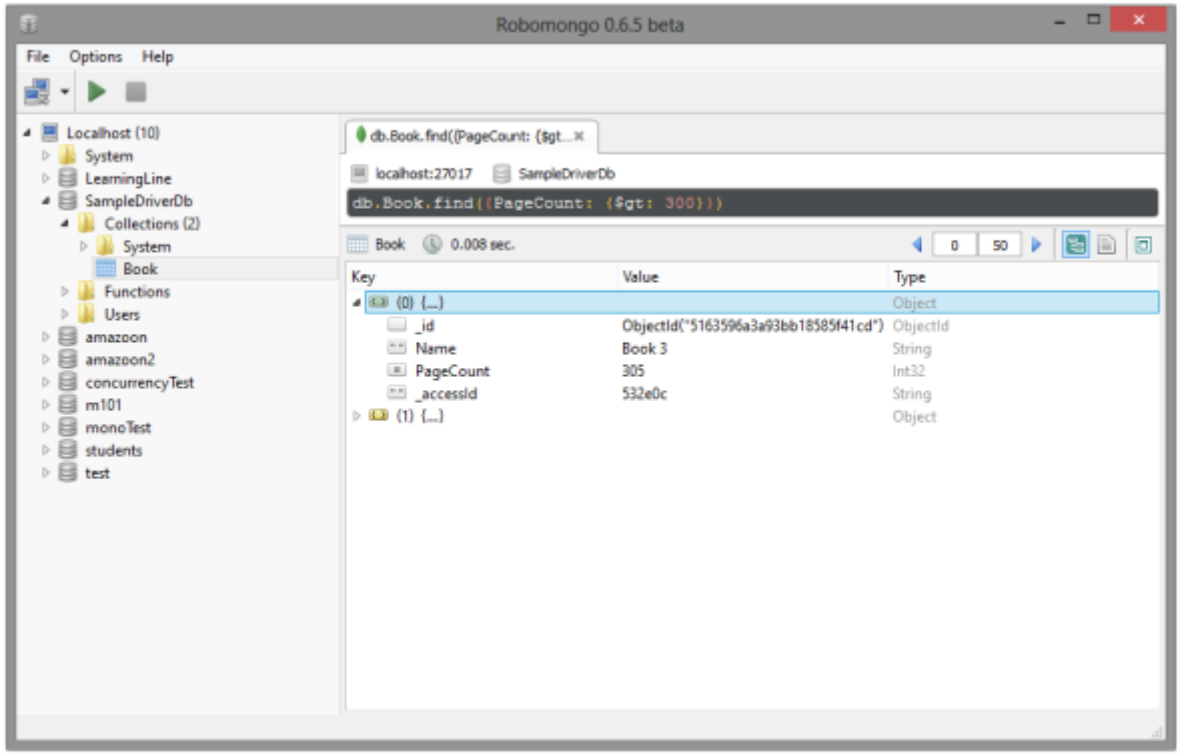
```
processes, 64000 files. Number of processes should be at least 32000 : 0.5 times number of files.
test>
```

Autres Clients Mongodb

Interfaces graphiques

RoboMongo

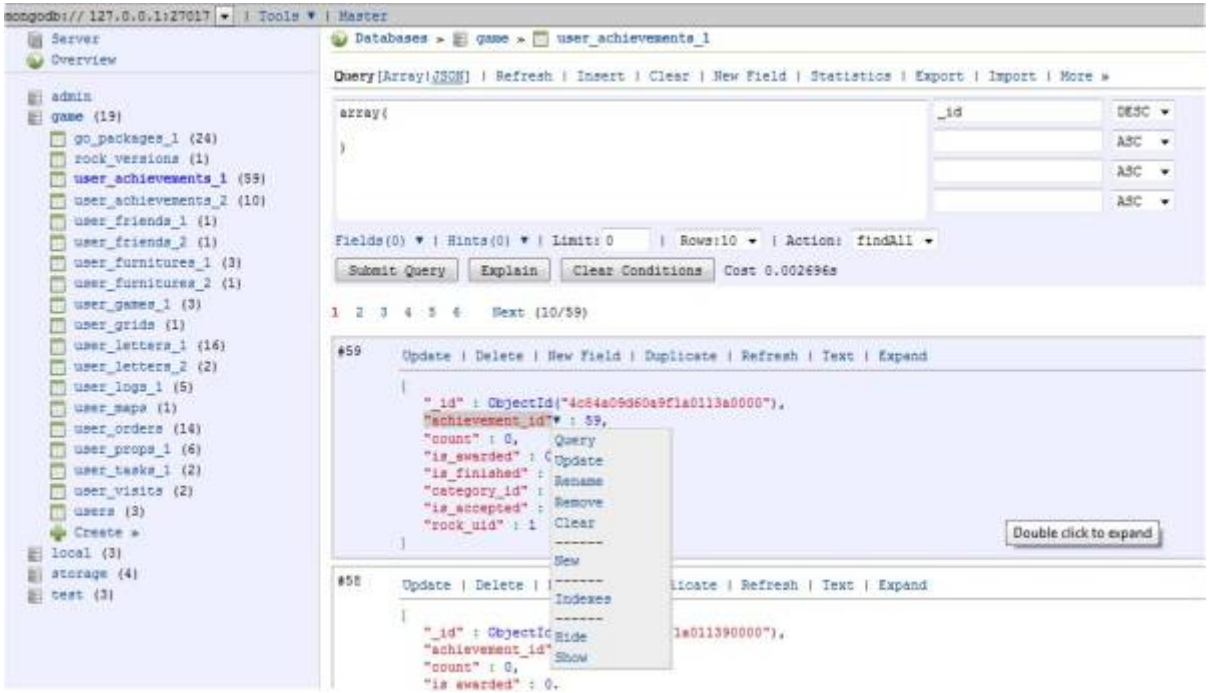
RoboMongo est un client graphique disponible pour toutes les plate-formes sur le site **https://robomongo.org/**. L'installation est très simple :



A Faire - Installez RoboMongo et connectez-vous à MongoDB.

RockMongo

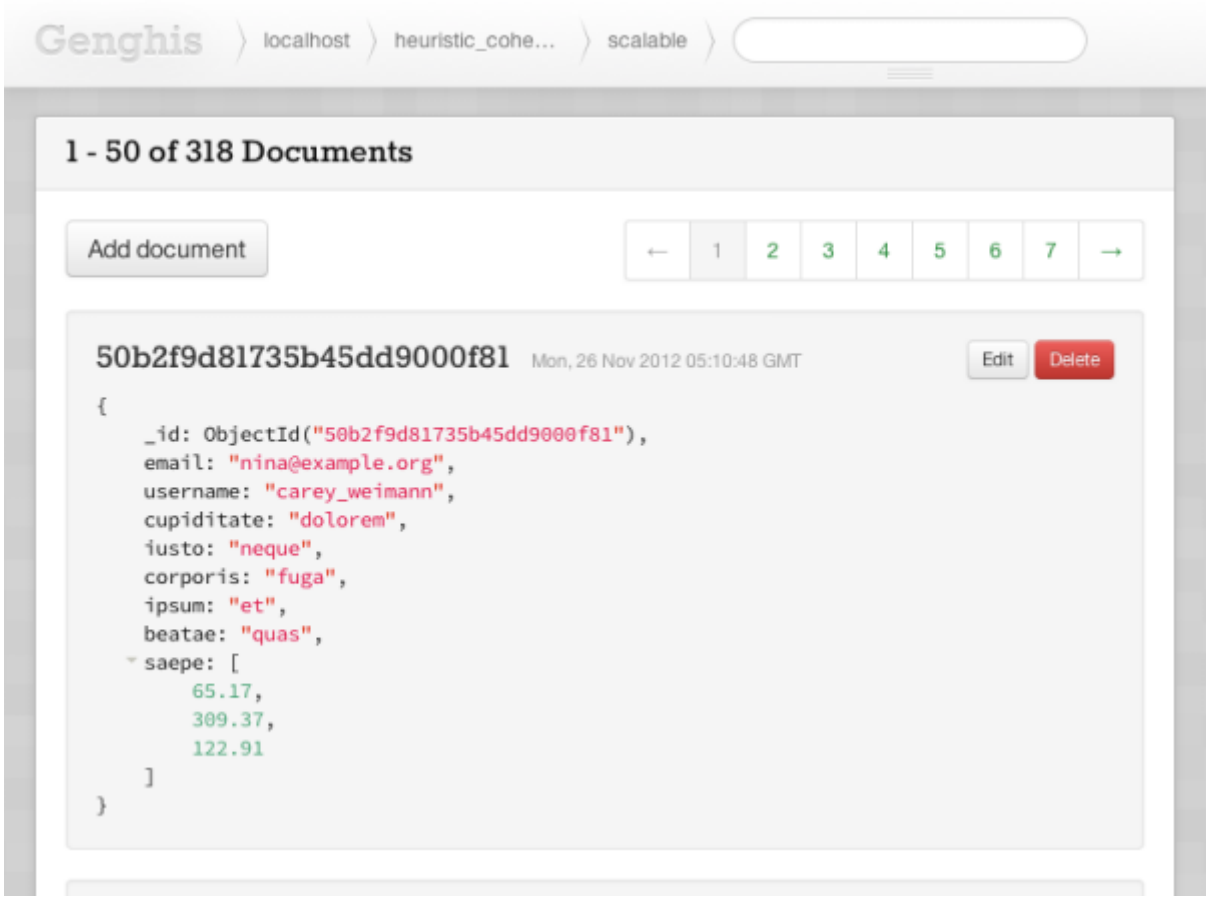
RockMongo est une application PHP disponible sur le site **http://www.rockmongo.com** et qui nécessite un serveur Apache/PHP. De plus, il est nécessaire d'installer l'extension PHP de connexion à MongoDB.



Genghis

Genghis est une interface d'administration de bases de données NoSQL sous MongoDB permettant une lecture adaptée à chaque périphérique. Il

s'agit d'une interface semblable a l'outil phpMyAdmin :



Les API pour MongoDB (Application Programming Interface)

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

Pour le langage C++

mongo-cxx-driver permet de se connecter à une instance mongod et de créer une connexion client à une instance en cours d'exécution mongod.

- **La librairie bsoncxx** permet de gérer les documents.

Pour gérer les applications, on doit inclure bson/bson.h dans le fichier code de l'application afin de pouvoir gérer les documents.

Structure des Données

Notions de Documents

On associe les documents aux bases de données orientées documents qui sont destinées aux applications qui gèrent des documents. Ce type de bases de données peut être ou non une sur-couche d'une base de données relationnelle.

Dans un système de base de données relationnelles, les informations sont stockées par ligne dans des tables. Ces tables sont mises en relation en utilisants des clés primaires et étrangères. Or dans MongoDb, l'information est modélisée sur un document au format BSON basé sur JSON (Javascript Object Notation).

MongoDB ne nécessite donc pas de 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.

De plus, contrairement aux bases de données SQL, il n'est pas nécessaire de faire des requêtes avec des jointures pour connaitre des informations. Dans MongoDb, il suffit de lire le document qui nous intéresse, d'où l'avantage de modéliser les données sur un document.

Notions de Collections

Le terme collection est issu du monde NoSQL et correspond par analogie à la notion de table dans les bases de données relationnelles (MySQL, PosgreSQL, Microsoft Access, Oracle, etc.). Une collection permet de stocker des documents, notion analogue à celle de l'enregistrement.

En comparant le schéma de structure d'une base de MongoDB à celui d'une base de donnée relationelle comme SQL, on peut faire le parallèle suivant:

Base de données relationnelle (SQL)	Base de données NoSQL (MongoDB)
database	database
table	collection
row	document où document BSON
column	field
index	index
primary key	primary key

Pour avoir un apercu, on peut définir par exemple 2 documents comme ceci:

```
{_id: "Her", acteurs : [{nom:"Johansson", prenom:"Scarlett"}, {nom:"Phoenix", prenom:"Joaquim"}]}
{_id: "Avengers", acteurs : [{nom:"Johansson", prenom:"Scarlett"}]}
```

Dans l'exemple ci-dessus, on peut voir que l'actrice **[{nom:“Johansson”, prenom:“Scarlett”}]** est dupliquée. Dans le monde NoSQL, on n'hésite pas à dénormaliser le schéma de la base de données pour favoriser les performances à la lecture. L'important est de savoir quelles requêtes seront faites pour décider du format des documents.

Le Format 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**.

Ce format se base donc sur 2 types d'éléments:

- la paire clé/valeur, par exemple “nom”: “Phoenix”
- le tableau, par exemple “couleurs-primaires” : [“cyan”, “jaune”, “magenta”]

Le Format BSON

Le format BSON ou **Binary JSON** correspond au format des données manipulés par MongoDB. C'est un format d'échange de données informatiques utilisé principalement comme stockage de données et format de transfert de données par le réseau dans la 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.

Comparaison BSON/JSON

Voici un tableau comparatif de BSON ET JSON :

Comparaison BSON/JSON		
Type de Champs	**JSON**	**BSON**
Number	X	X
String	X	X
Boolean	X	X
Array	X	X

Comparaison BSON/JSON		
Type de Champs	**JSON**	**BSON**
Object	X	X
Null	X	X
Float	-	X
Date	-	X
Regular Expression	-	X
JavaScript Code	-	X

On constate que le format BSON supporte plus de types que le format JSON. Le codage BSON complète la représentation JSON en y ajoutant des types de données supplémentaires, tels que les formats virgule flottante ou date entre autres.

Format d'un Document BSON

Dernièrement, voici un document au format BSON stockant des informations concernant un film :

```
 {
  "_id": "movie:100",
  "title": "The Social network",
  "summary": "On a fall night in 2003, Harvard undergrad and
     programming genius Mark Zuckerberg sits down at his
     computer and heatedly begins working on a new idea. (...)",
  "year": 2010,
  "director": {"last_name": "Fincher",
               "first_name": "David"},
  "actors": [
    {"first_name": "Jesse", "last_name": "Eisenberg"},
    {"first_name": "Rooney", "last_name": "Mara"}
   ]
}
```

Le Langage des Requêtes

Requêtes de base

Se placer dans une base de données/Créer une base de données

Pour se placer dans une base de données ou créer une base inexistente, il convient d'utiliser la requête **use** :

```
use <database>
```

Créer une collection

Pour créer une collection, il convient d'utiliser la requête **db.createCollection(<collection>)** :

```
db.createCollection("movies")
```

Important - Une base est constituée d'un ensemble de collections, l'équivalent d'une table en relationnel.

Visualiser la liste des collections

Pour visualiser la liste des collections, il convient d'utiliser la requête **show** :

```
show collections
```

Inserer un document BSON dans une collection

Pour inserer un document BSON dans une collection, il convient d'utiliser la requête **db.<collection>.insert (<document>)** :

```
db.movies.insert ({"title": "Batman", "year": 1989})
db.movies.insert ({"produit": "Grulband", prix: 230, enStock: true})
```

Important - Notez que la structure du deuxième document n'a rien à voir avec le premier. Il n'y a pas de schéma (et donc pas de contrainte) dans MongoDB. On est libre de tout faire, ce qui revient à reporter les problèmes (contrôles, contraintes, tests sur la structure) vers l'application. Notez aussi que lorsque l'on veut insérer des documents BSON dans la base MongoDB, au lieu de les insérer un par un, on peut utiliser l'utilitaire d'import de MongoDB, qui prend en entrée un tableau BSON contenant la liste des objets à insérer.

Affecter un identifiant explicitement

Pour affecter un identifiant explicitement, il convient d'utiliser la requête **db.<collection>.insert(<document>)** en spécifiant la valeur de **_id** :

```
db.movies.insert ({_id: "1", "produit": "Kramölk", prix: 10, enStock: true})
```

Remplacer un document

Pour remplacer un document, il convient d'utiliser la requête **db.<collection>.update(<document>)** :

```
db.movies.update( { "title": "Mr Smith" }, { "title": "Mr Cool", "year": 2000 } )
```

Modifier des champs d'un document

Pour modifier des champs d'un document, il convient d'utiliser la requête **db.<collection>.update(<document>)** et **$set** :

```
db.movies.update( { "title": "Avengers" }, { $set: { "summary": "c'est un film divertissant" } } )
```

Compter le nombre de documents dans la collection

Pour compter le nombre de documents dans la collection, il convient d'utiliser la requête **db.<collection>.count()** :

```
db.movies.count()
```

Supprimer un document

Pour supprimer un ou plusieurs documents il convient d'utiliser la requête **db.<collection>.remove(<document>)** à laquelle on passe en paramètre le query permettant d'identifier les documents visés :

```
db.movies.remove({"produit": "Grulband"})
```

Supprimer un champ dans un document

Pour supprimer un champ dans un document, il convient d'utiliser la requête **db.<collection>.update(<document>)** et **$unset** :

```
db.movies.update({"title": "Batman"}, {$unset : {year : 1989}})
```

Supprimer une collection

Pour supprimer une collection, il convient d'utiliser la requête **db.<collection>.drop()** :

```
db.movies.drop()
```

Trier des documents

Pour trier les documents, il convient d'utiliser la requête **db.movies.find ().sort().skip().limit()** :

```
db.movies.find ().sort({"title": 1}).skip(9).limit(12)
```

Important - Notez qu'implicitement, cela suppose qu'il existe un ordre sur le parcours des documents. Par défaut, cet ordre est dicté par le stockage physique: MongoDB fournit les documents dans l'ordre où il les trouve (dans les fichiers). On peut trier explicitement, ce qui rend le résultat plus déterministe.

Rechercher des documents dans une collection

Pour afficher les contenu d'une collection, il convient d'utiliser la requête **db.<collection>.find(<document>)** :

```
db.movies.find()
```

On obtient des objets (javascript, encodés en BSON), par exemple :

```
{ "_id" : ObjectId("5422d9095ae45806a0e66474"), "nom" : "nfe024" }
```

Important - Notez que MongoDB associe un identifiant unique à chaque document, de nom conventionnel **_id**, et lui attribue une valeur si elle n'est pas indiquée explicitement.

Il est aussi possible de rechercher en connaissant l'identifiant :

```
db.movies.find ({"_id": "movie:2"})
```

ou en utilisant n'importe quel attribut :

```
db.movies.find ({"title": "Alien"})
```

Projections

Les requêtes précédentes ramènent l'intégralité des objets satisfaisant les critères de recherche. Il est aussi possible de faire des projections, en passant un second argument à la fonction find() :

```
db.movies.find ({"actors.last_name": "Tarantino"}, {"title": true, "actors": 'j'} )
```

Le second argument est un objet BSON dont les attributs sont ceux à conserver dans le résultat. La valeur des attributs dans cet objet-projection ne prend que deux interprétations. Toute valeur autre que **0** ou **null** indique que l'attribut doit être conservé. Si on choisit au contraire d'indiquer les attributs à exclure, on leur donne la valeur **0** ou **null**. Par exemple, la requête suivante retourne les films sans les acteurs et sans le résumé :

```
db.movies.find ({"actors": null, "summary": 0})
```

Opérateurs ensemblistes

Les opérateurs du langage SQL **in**, **not in**, **any** et **all** se retrouvent dans le langage d'interrogation. La différence, notable, est que SQL applique ces opérateurs à des relations (elles-mêmes obtenues par des requêtes) alors que dans le cas de MongoDB, ce sont des tableaux JSON. MongoDB ne permet pas d'imbriquer des requêtes.

Par exemple, on cherche les films dans lesquels joue au moins un des artistes dans une liste dont on connaît l'identifiant :

```
db.artists.find({"actors._id": {$in: ["artist:34","artist:98","artist:1"]}})
```

Le **not in** correspond à l'opérateur $nin.

```
db.artists.find({"_id": {$nin: ["artist:34","artist:98","artist:1"]}})
```

Comme dernier example, voici comment trouver les films qui n'ont pas d'attribut summary :

```
db.movies.find({"summary": {$exists: false}}, {"title": 1})
```

Opérateurs booléens

Par défaut, quand on exprime plusieurs critères, c'est une conjonction (and) qui est appliquée. On peut l'indiquer explicitement. Voici la syntaxe (les films tournés avec Leonardo DiCaprio en 1997) :

```
db.movies.find({$and : [{"year": "1997"}, {actors.last_name: "DiCaprio"]} )
```

L'opérateur **and** s'applique à un tableau de conditions. Bien entendu il existe un opérateur **or** avec la même syntaxe. Les films parus en 1997 ou avec Leonardo DiCaprio :

```
db.movies.find({$or : [{"year": "1997"}, {actors.last_name: "DiCaprio"]} )
```

Comparaison de la structure des requêtes entre SQL/MongoDB

Tables/Collections

CREATE TABLE

SQL	MongoDB
`CREATE TABLE people (id MEDIUMINT NOT NULL AUTO_INCREMENT, user_id Varchar(30),\ age Number, status char(1), PRIMARY KEY (id))`	`db.people.insertOne( {user_id: "abc123", age: 55, status: "A"} )` ou simplement `db.createCollection("people")`

DROP TABLE

SQL	MongoDB
`DROP TABLE people`	`db.people.drop()`

ALTER TABLE

SQL	MongoDB
`ALTER TABLE people ADD join_date DATETIME`	`db.people.updateMany( { },{ $set: { join_date: new Date() } } )`
`ALTER TABLE people DROP COLUMN join_date`	`db.people.updateMany( { },{ $unset: { "join_date": "" } } )`

CREATE INDEX

SQL	MongoDB
`CREATE INDEX idx_user_id_asc ON people(user_id)`	`db.people.createIndex( { user_id: 1 } )`
`CREATE INDEX idx_user_id_asc_age_desc ON people(user_id, age DESC)`	`db.people.createIndex( { user_id: 1, age: -1 } )`

Rows/documents

INSERT

SQL	MongoDB
`INSERT INTO people(user_id, age, status) VALUES ("bcd001", 45, "A")`	`db.people.insertOne({ user_id: "bcd001", age: 45, status: "A" })`

SELECT

SQL	MongoDB
`SELECT * FROM people`	`db.people.find()`
`SELECT id, user_id, status FROM people`	`db.people.find( { },{ user_id: 1, status: 1 } )`
`SELECT user_id, status FROM people`	`db.people.find( { },{ user_id: 1, status: 1, _id: 0 } )`
`SELECT * FROM people WHERE status = "A"`	`db.people.find( { status: "A" } )`
`SELECT user_id, status FROM people WHERE status = "A"`	`db.people.find( { status: "A" },{ user_id: 1, status:1, _id: 0 } )`
`SELECT * FROM people WHERE status != "A"`	`db.people.find( { status: { $ne: "A" } } )`
`SELECT * FROM people WHERE status = "A" AND age = 50`	`db.people.find( { status: "A", age: 50 } )`
`SELECT * FROM people WHERE status = "A" OR age = 50`	`db.people.find( { $or: [ { status: "A" } , { age: 50 } ] } )`
`SELECT * FROM people WHERE age > 25`	`db.people.find( { age: { $gt: 25 } } )`
`SELECT * FROM people WHERE age < 25`	`db.people.find( { age: { $lt: 25 } } )`
`SELECT * FROM people WHERE age > 25 AND age <= 50`	`db.people.find( { age: { $gt: 25, $lte: 50 } } )`
`SELECT * FROM people WHERE status = "A" ORDER BY user_id ASC`	`db.people.find( { status: "A" } ).sort( { user_id: 1 } )`
`SELECT * FROM people WHERE status = "A" ORDER BY user_id DESC`	`db.people.find( { status: "A" } ).sort( { user_id: -1 } )`

`SELECT * FROM people WHERE user_id like "bc%"`	`db.people.find( { user_id: /^bc/ } )`
	ou
	`db.people.find( { user_id: { $regex: ^bc/ } } )`

`SELECT * FROM people WHERE user_id like "%bc%"`	`db.people.find( { user_id: /bc/ } )`
	ou
	`db.people.find( { user_id: { $regex: /bc/ } } )`

`SELECT COUNT(*) FROM people`	`db.people.count()`
	ou
	`db.people.find().count()`

`SELECT COUNT(user_id) FROM people`	`db.people.count( { user_id: { $exists: true } } )`
	ou
	`db.people.find( { user_id: { $exists: true } } ).count()`

SQL	MongoDB

`SELECT COUNT(*) FROM people WHERE age > 30`	`db.people.count( { age: { $gt: 30 } } )`
	ou
	`db.people.find( { age: { $gt: 30 } } ).count()`

`SELECT DISTINCT(status) FROM people`	`db.people.distinct( "status" )`

`SELECT * FROM people LIMIT 1`	`db.people.findOne()`
	ou
	`db.people.find().limit(1)`

`SELECT * FROM people LIMIT 5 SKIP 10`	`db.people.find().limit(5).skip(10)`

EXPLAIN SELECT

SQL	MongoDB
`EXPLAIN SELECT * FROM people WHERE status = "A"`	`db.people.find( { status: "A" } ).explain()`

UPDATE

SQL	MongoDB
`UPDATE people SET status = "C" WHERE age > 25`	`db.people.updateMany( { age: { $gt: 25 } } , { $set: { status: "C" } } )`
`UPDATE people SET age = age + 3 WHERE status = "A"`	`db.people.updateMany( { status: "A" } , { $inc: { age: 3 } } )`

DELETE FROM

SQL	MongoDB
`DELETE FROM people WHERE status = "D"`	`db.people.deleteMany( { status: "D" } )`
`DELETE FROM people`	`db.people.deleteMany({})`

LAB #1 - Utilisation de requêtes de base

Création d'une collection

Commencez par créer une collection nommé Movies qui enregistre des informations sur des films :

```
test>db.createCollection("movies")
{ "ok" : 1 }
test>
```

Création de documents

Créez maintenant les documents de la collection :

```
test>db.movies.insert({ "_id" : "movie:1", "titre" : "Batman", "année" : 1989, "acteurs" : [ { "prénom" :
"Michael", "nom" : "Keaton" } ] })
WriteResult({ "nInserted" : 1 })
test>db.movies.insert({ "_id" : "movie:2", "titre" : "Beetlejuice", "année" : 1988, "acteurs" : [ { "prénom" :
"Michael", "nom" : "Keaton" }, { "prénom" : "Geena", "nom" : "Davis" } ] })
WriteResult({ "nInserted" : 1 })
test>db.movies.insert({ "_id" : "movie:3", "titre" : "Jurassik Park", "réalisateur" : { "prénom" : "Steven",
"nom" : "Spielberg" }, "année" : 1990, "acteurs" : [ { "prénom" : "Sam", "nom" : "Neil" }, { "prénom" : "Laura",
"nom" : "Dern" } ] })
WriteResult({ "nInserted" : 1 })
test>db.movies.insert({ "_id" : "movie:4", "titre" : "Avengers", "année" : 2012 })
WriteResult({ "nInserted" : 1 })
test>db.movies.insert({ "_id" : "movie:5", "titre" : "Hobbit", "année" : 2012 })
WriteResult({ "nInserted" : 1 })
test>
```

Important - Notez que vous avez rentré 2 fois le nom et prénom du champ acteurs pour les 2 premiers documents crées.

Recherche de documents

Utilisez maintenant la commande find pour vérifier que les 5 documents ont bien été crées dans la collection movies :

```
test>db.movies.find()
{ "_id" : "movie:1", "titre" : "Batman", "année" : 1989, "acteurs" : [ { "prénom" : "Michael", "nom" : "Keaton" }
] }
{ "_id" : "movie:2", "titre" : "Beetlejuice", "année" : 1988, "acteurs" : [ { "prénom" : "Michael", "nom" :
"Keaton" }, { "prénom" : "Geena", "nom" : "Davis" } ] }
{ "_id" : "movie:3", "titre" : "Jurassik Park", "réalisateur" : { "prénom" : "Steven", "nom" : "Spielberg" },
"année" : 1990, "acteurs" : [ { "prénom" : "Sam", "nom" : "Neil" }, { "prénom" : "Laura", "nom" : "Dern" } ] }
{ "_id" : "movie:4", "titre" : "Avengers", "année" : 2012 }
{ "_id" : "movie:5", "titre" : "Hobbit", "année" : 2012 }
test>
```

Modifiez ensuite le document ayant pour _id : movie 3) en utilisant la commande update :

```
test>db.movies.update({"_id": "movie:3"}, {$set: {titre: "le parc des dinosaures"}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
test>
```

Important - Notez que le retour indique la réussite de l'oprération de mise à jour - **“nModified” : 1**.

Vérifiez que le champ a bien été modifié dans le document en consultant la collection movies :

```
test>db.movies.find()
{ "_id" : "movie:1", "titre" : "Batman", "année" : 1989, "acteurs" : [ { "prénom" : "Michael", "nom" : "Keaton" }
] }
{ "_id" : "movie:2", "titre" : "Beetlejuice", "année" : 1988, "acteurs" : [ { "prénom" : "Michael", "nom" :
"Keaton" }, { "prénom" : "Geena", "nom" : "Davis" } ] }
```

```
{ "_id" : "movie:3", "titre" : "le parc des dinosaures", "réalisateur" : { "prénom" : "Steven", "nom" :
"Spielberg" }, "année" : 1990, "acteurs" : [ { "prénom" : "Sam", "nom" : "Neil" }, { "prénom" : "Laura", "nom" :
"Dern" } ] }
{ "_id" : "movie:4", "titre" : "Avengers", "année" : 2012 }
{ "_id" : "movie:5", "titre" : "Hobbit", "année" : 2012 }
test>
```

Important - Notez que le titre du film ayant pour **“_id” : “movie:3”** a bien été modifié.

Affichez maintenant les films n'ayant pas de réalisateur :

```
test>db.movies.find({"réalisateur": null})
{ "_id" : "movie:1", "titre" : "Batman", "année" : 1989, "acteurs" : [ { "prénom" : "Michael", "nom" : "Keaton" }
] }
{ "_id" : "movie:2", "titre" : "Beetlejuice", "année" : 1988, "acteurs" : [ { "prénom" : "Michael", "nom" :
"Keaton" }, { "prénom" : "Geena", "nom" : "Davis" } ] }
{ "_id" : "movie:4", "titre" : "Avengers", "année" : 2012 }
{ "_id" : "movie:5", "titre" : "Hobbit", "année" : 2012 }
test>
```

Suppression d'un document

Supprimez le film dont le titre est **“Batman”** :

```
test>db.movies.remove({"titre": "Batman"})
WriteResult({ "nRemoved" : 1 })
test>
```

Constatez le résultat de la requête :

```
test>db.movies.find()
{ "_id" : "movie:2", "titre" : "Beetlejuice", "année" : 1988, "acteurs" : [ { "prénom" : "Michael", "nom" :
"Keaton" }, { "prénom" : "Geena", "nom" : "Davis" } ] }
{ "_id" : "movie:3", "titre" : "le parc des dinosaures", "réalisateur" : { "prénom" : "Steven", "nom" :
"Spielberg" }, "année" : 1990, "acteurs" : [ { "prénom" : "Sam", "nom" : "Neil" }, { "prénom" : "Laura", "nom" :
"Dern" } ] }
{ "_id" : "movie:4", "titre" : "Avengers", "année" : 2012 }
{ "_id" : "movie:5", "titre" : "Hobbit", "année" : 2012 }
test>
```

Supprimez maintenant les films ayant **“année”: 2012** :

```
test>db.movies.remove({"année": 2012})
WriteResult({ "nRemoved" : 2 })
test>db.movies.find()
{ "_id" : "movie:2", "titre" : "Beetlejuice", "année" : 1988, "acteurs" : [ { "prénom" : "Michael", "nom" :
"Keaton" }, { "prénom" : "Geena", "nom" : "Davis" } ] }
{ "_id" : "movie:3", "titre" : "le parc des dinosaures", "réalisateur" : { "prénom" : "Steven", "nom" :
"Spielberg" }, "année" : 1990, "acteurs" : [ { "prénom" : "Sam", "nom" : "Neil" }, { "prénom" : "Laura", "nom" :
"Dern" } ] }
```

Important - Notez la suppression de deux documents : **“_id” : “movie:4”** et **“_id” : “movie:5”**.

LAB #2 - Rechercher, filtrer et trier

La Requête find()

Connectez-vous à MongoDB en utilisant le client **mongo** :

```
[trainee@centos7 ~]$ su -
Mot de passe : fenestros
Dernière connexion : lundi 18 septembre 2017 à 13:39:03 CEST sur pts/0
[root@centos7 ~]# mongo
MongoDB shell version: 3.2.16
connecting to: test
Server has startup warnings:
2017-09-18T13:46:09.416+0200 I CONTROL  [initandlisten]
2017-09-18T13:46:09.416+0200 I CONTROL  [initandlisten] ** WARNING: soft rlimits too low. rlimits set to 4096
processes, 64000 files. Number of processes should be at least 32000 : 0.5 times number of files.
test>
```

Commencez par créer une base de données appelée **find** :

```
test>use find
switched to db find
find>
```

Critères de recherche

En sachant que le client mongo interprète du JavaScript, créez 100 000 documents dans la base de données **find** dans une collection appelée **products** :

```
find>for(i=1; i<=100000; i++){var tenthousand = i%10000; var thousand = i%1000; var hundred = i%100;
db.products.insert({counter:i, tenthousand:tenthousand, thousand:thousand, hundred:hundred })}
WriteResult({ "nInserted" : 1 })
```

Saisissez maintenant la commande **db.products.find()** :

```
find>db.products.find()
{ "_id" : ObjectId("59c0fb1a3b6221e3d36169a6"), "counter" : 1, "tenthousand" : 1, "thousand" : 1, "hundred" : 1 }
{ "_id" : ObjectId("59c0fb1a3b6221e3d36169a7"), "counter" : 2, "tenthousand" : 2, "thousand" : 2, "hundred" : 2 }
```

```
{ "_id" : ObjectId("59c0fb1a3b6221e3d36169a8"), "counter" : 3, "tenthousand" : 3, "thousand" : 3, "hundred" : 3 }
{ "_id" : ObjectId("59c0fb1a3b6221e3d36169a9"), "counter" : 4, "tenthousand" : 4, "thousand" : 4, "hundred" : 4 }
{ "_id" : ObjectId("59c0fb1a3b6221e3d36169aa"), "counter" : 5, "tenthousand" : 5, "thousand" : 5, "hundred" : 5 }
{ "_id" : ObjectId("59c0fb1a3b6221e3d36169ab"), "counter" : 6, "tenthousand" : 6, "thousand" : 6, "hundred" : 6 }
{ "_id" : ObjectId("59c0fb1a3b6221e3d36169ac"), "counter" : 7, "tenthousand" : 7, "thousand" : 7, "hundred" : 7 }
{ "_id" : ObjectId("59c0fb1a3b6221e3d36169ad"), "counter" : 8, "tenthousand" : 8, "thousand" : 8, "hundred" : 8 }
{ "_id" : ObjectId("59c0fb1a3b6221e3d36169ae"), "counter" : 9, "tenthousand" : 9, "thousand" : 9, "hundred" : 9 }
{ "_id" : ObjectId("59c0fb1a3b6221e3d36169af"), "counter" : 10, "tenthousand" : 10, "thousand" : 10, "hundred" :
10 }
{ "_id" : ObjectId("59c0fb1a3b6221e3d36169b0"), "counter" : 11, "tenthousand" : 11, "thousand" : 11, "hundred" :
11 }
{ "_id" : ObjectId("59c0fb1a3b6221e3d36169b1"), "counter" : 12, "tenthousand" : 12, "thousand" : 12, "hundred" :
12 }
{ "_id" : ObjectId("59c0fb1a3b6221e3d36169b2"), "counter" : 13, "tenthousand" : 13, "thousand" : 13, "hundred" :
13 }
{ "_id" : ObjectId("59c0fb1a3b6221e3d36169b3"), "counter" : 14, "tenthousand" : 14, "thousand" : 14, "hundred" :
14 }
{ "_id" : ObjectId("59c0fb1a3b6221e3d36169b4"), "counter" : 15, "tenthousand" : 15, "thousand" : 15, "hundred" :
15 }
{ "_id" : ObjectId("59c0fb1a3b6221e3d36169b5"), "counter" : 16, "tenthousand" : 16, "thousand" : 16, "hundred" :
16 }
{ "_id" : ObjectId("59c0fb1a3b6221e3d36169b6"), "counter" : 17, "tenthousand" : 17, "thousand" : 17, "hundred" :
17 }
{ "_id" : ObjectId("59c0fb1a3b6221e3d36169b7"), "counter" : 18, "tenthousand" : 18, "thousand" : 18, "hundred" :
18 }
{ "_id" : ObjectId("59c0fb1a3b6221e3d36169b8"), "counter" : 19, "tenthousand" : 19, "thousand" : 19, "hundred" :
19 }
{ "_id" : ObjectId("59c0fb1a3b6221e3d36169b9"), "counter" : 20, "tenthousand" : 20, "thousand" : 20, "hundred" :
20 }
Type "it" for more
find>
```

Notez l'instruction **Type “it” for more** :

```
find>it
{ "_id" : ObjectId("59c0fb1a3b6221e3d36169ba"), "counter" : 21, "tenthousand" : 21, "thousand" : 21, "hundred" :
21 }
{ "_id" : ObjectId("59c0fb1a3b6221e3d36169bb"), "counter" : 22, "tenthousand" : 22, "thousand" : 22, "hundred" :
22 }
{ "_id" : ObjectId("59c0fb1a3b6221e3d36169bc"), "counter" : 23, "tenthousand" : 23, "thousand" : 23, "hundred" :
23 }
{ "_id" : ObjectId("59c0fb1a3b6221e3d36169bd"), "counter" : 24, "tenthousand" : 24, "thousand" : 24, "hundred" :
24 }
{ "_id" : ObjectId("59c0fb1a3b6221e3d36169be"), "counter" : 25, "tenthousand" : 25, "thousand" : 25, "hundred" :
25 }
{ "_id" : ObjectId("59c0fb1a3b6221e3d36169bf"), "counter" : 26, "tenthousand" : 26, "thousand" : 26, "hundred" :
26 }
{ "_id" : ObjectId("59c0fb1a3b6221e3d36169c0"), "counter" : 27, "tenthousand" : 27, "thousand" : 27, "hundred" :
27 }
{ "_id" : ObjectId("59c0fb1a3b6221e3d36169c1"), "counter" : 28, "tenthousand" : 28, "thousand" : 28, "hundred" :
28 }
{ "_id" : ObjectId("59c0fb1a3b6221e3d36169c2"), "counter" : 29, "tenthousand" : 29, "thousand" : 29, "hundred" :
29 }
{ "_id" : ObjectId("59c0fb1a3b6221e3d36169c3"), "counter" : 30, "tenthousand" : 30, "thousand" : 30, "hundred" :
30 }
{ "_id" : ObjectId("59c0fb1a3b6221e3d36169c4"), "counter" : 31, "tenthousand" : 31, "thousand" : 31, "hundred" :
31 }
{ "_id" : ObjectId("59c0fb1a3b6221e3d36169c5"), "counter" : 32, "tenthousand" : 32, "thousand" : 32, "hundred" :
32 }
{ "_id" : ObjectId("59c0fb1a3b6221e3d36169c6"), "counter" : 33, "tenthousand" : 33, "thousand" : 33, "hundred" :
33 }
{ "_id" : ObjectId("59c0fb1a3b6221e3d36169c7"), "counter" : 34, "tenthousand" : 34, "thousand" : 34, "hundred" :
34 }
{ "_id" : ObjectId("59c0fb1a3b6221e3d36169c8"), "counter" : 35, "tenthousand" : 35, "thousand" : 35, "hundred" :
35 }
{ "_id" : ObjectId("59c0fb1a3b6221e3d36169c9"), "counter" : 36, "tenthousand" : 36, "thousand" : 36, "hundred" :
36 }
{ "_id" : ObjectId("59c0fb1a3b6221e3d36169ca"), "counter" : 37, "tenthousand" : 37, "thousand" : 37, "hundred" :
```

```
37 }
{ "_id" : ObjectId("59c0fb1a3b6221e3d36169cb"), "counter" : 38, "tenthousand" : 38, "thousand" : 38, "hundred" :
38 }
{ "_id" : ObjectId("59c0fb1a3b6221e3d36169cc"), "counter" : 39, "tenthousand" : 39, "thousand" : 39, "hundred" :
39 }
{ "_id" : ObjectId("59c0fb1a3b6221e3d36169cd"), "counter" : 40, "tenthousand" : 40, "thousand" : 40, "hundred" :
40 }
Type "it" for more
find>
```

Important - Bien que pratique, il faudrait taper la commande **it** un grand nombre de fois pour atteindre le document 90 000 !

Tapez maintenat la commande **db.products.find({tenthousand:9999})** :

```
find>db.products.find({tenthousand:9999})
{ "_id" : ObjectId("59c0fb2d3b6221e3d36190b4"), "counter" : 9999, "tenthousand" : 9999, "thousand" : 999,
"hundred" : 99 }
{ "_id" : ObjectId("59c0fb403b6221e3d361b7c4"), "counter" : 19999, "tenthousand" : 9999, "thousand" : 999,
"hundred" : 99 }
{ "_id" : ObjectId("59c0fb533b6221e3d361ded4"), "counter" : 29999, "tenthousand" : 9999, "thousand" : 999,
"hundred" : 99 }
{ "_id" : ObjectId("59c0fb663b6221e3d36205e4"), "counter" : 39999, "tenthousand" : 9999, "thousand" : 999,
"hundred" : 99 }
{ "_id" : ObjectId("59c0fb793b6221e3d3622cf4"), "counter" : 49999, "tenthousand" : 9999, "thousand" : 999,
"hundred" : 99 }
{ "_id" : ObjectId("59c0fb8e3b6221e3d3625404"), "counter" : 59999, "tenthousand" : 9999, "thousand" : 999,
"hundred" : 99 }
{ "_id" : ObjectId("59c0fba13b6221e3d3627b14"), "counter" : 69999, "tenthousand" : 9999, "thousand" : 999,
"hundred" : 99 }
{ "_id" : ObjectId("59c0fbb43b6221e3d362a224"), "counter" : 79999, "tenthousand" : 9999, "thousand" : 999,
"hundred" : 99 }
```

```
{ "_id" : ObjectId("59c0fbc93b6221e3d362c934"), "counter" : 89999, "tenthousand" : 9999, "thousand" : 999,
"hundred" : 99 }
{ "_id" : ObjectId("59c0fbdc3b6221e3d362f044"), "counter" : 99999, "tenthousand" : 9999, "thousand" : 999,
"hundred" : 99 }
find>
```

Important - Vous obtenez les documents qui correspondent à **tenthousand = 9999**.

Précisez maintenant un deuxième critère de recherche :

```
find>db.products.find({counter:19999, tenthousand:9999})
{ "_id" : ObjectId("59c0fb403b6221e3d361b7c4"), "counter" : 19999, "tenthousand" : 9999, "thousand" : 999,
"hundred" : 99 }
find>
```

Important - Vous obtenez un seul document où counter=19999 et tenthousand=9999.

Pour connaître le nombre de documents retournés lors d'une recherche spécifique, utilisez la commande **count()** :

```
find>db.products.find({thousand:289, hundred:89})
{ "_id" : ObjectId("59c0fb1a3b6221e3d3616ac6"), "counter" : 289, "tenthousand" : 289, "thousand" : 289, "hundred"
: 89 }
{ "_id" : ObjectId("59c0fb1d3b6221e3d3616eae"), "counter" : 1289, "tenthousand" : 1289, "thousand" : 289,
"hundred" : 89 }
{ "_id" : ObjectId("59c0fb1e3b6221e3d3617296"), "counter" : 2289, "tenthousand" : 2289, "thousand" : 289,
"hundred" : 89 }
{ "_id" : ObjectId("59c0fb203b6221e3d361767e"), "counter" : 3289, "tenthousand" : 3289, "thousand" : 289,
"hundred" : 89 }
```

```
{ "_id" : ObjectId("59c0fb223b6221e3d3617a66"), "counter" : 4289, "tenthousand" : 4289, "thousand" : 289,
"hundred" : 89 }
{ "_id" : ObjectId("59c0fb243b6221e3d3617e4e"), "counter" : 5289, "tenthousand" : 5289, "thousand" : 289,
"hundred" : 89 }
{ "_id" : ObjectId("59c0fb263b6221e3d3618236"), "counter" : 6289, "tenthousand" : 6289, "thousand" : 289,
"hundred" : 89 }
{ "_id" : ObjectId("59c0fb283b6221e3d361861e"), "counter" : 7289, "tenthousand" : 7289, "thousand" : 289,
"hundred" : 89 }
{ "_id" : ObjectId("59c0fb2a3b6221e3d3618a06"), "counter" : 8289, "tenthousand" : 8289, "thousand" : 289,
"hundred" : 89 }
{ "_id" : ObjectId("59c0fb2c3b6221e3d3618dee"), "counter" : 9289, "tenthousand" : 9289, "thousand" : 289,
"hundred" : 89 }
{ "_id" : ObjectId("59c0fb2e3b6221e3d36191d6"), "counter" : 10289, "tenthousand" : 289, "thousand" : 289,
"hundred" : 89 }
{ "_id" : ObjectId("59c0fb2f3b6221e3d36195be"), "counter" : 11289, "tenthousand" : 1289, "thousand" : 289,
"hundred" : 89 }
{ "_id" : ObjectId("59c0fb313b6221e3d36199a6"), "counter" : 12289, "tenthousand" : 2289, "thousand" : 289,
"hundred" : 89 }
{ "_id" : ObjectId("59c0fb333b6221e3d3619d8e"), "counter" : 13289, "tenthousand" : 3289, "thousand" : 289,
"hundred" : 89 }
{ "_id" : ObjectId("59c0fb353b6221e3d361a176"), "counter" : 14289, "tenthousand" : 4289, "thousand" : 289,
"hundred" : 89 }
{ "_id" : ObjectId("59c0fb373b6221e3d361a55e"), "counter" : 15289, "tenthousand" : 5289, "thousand" : 289,
"hundred" : 89 }
{ "_id" : ObjectId("59c0fb393b6221e3d361a946"), "counter" : 16289, "tenthousand" : 6289, "thousand" : 289,
"hundred" : 89 }
{ "_id" : ObjectId("59c0fb3b3b6221e3d361ad2e"), "counter" : 17289, "tenthousand" : 7289, "thousand" : 289,
"hundred" : 89 }
{ "_id" : ObjectId("59c0fb3d3b6221e3d361b116"), "counter" : 18289, "tenthousand" : 8289, "thousand" : 289,
"hundred" : 89 }
{ "_id" : ObjectId("59c0fb3f3b6221e3d361b4fe"), "counter" : 19289, "tenthousand" : 9289, "thousand" : 289,
"hundred" : 89 }
Type "it" for more
find>db.products.find({thousand:289, hundred:89}).count()
```

```
100
find>
```

Important - Notez qu'il existe **100** documents correspondant à notre critère **({thousand:289, hundred:89})**.

Utiliser des Opérandes

Il est possilible d'utiliser des opérandes avec la requête find() :

Opérande	Description
$gt	Supérieur à
$gte	Supérieur ou égal à
$lt	Inférieur à
$lte	Inférieur ou égal à

Important - Les 4 opérandes ci-dessus fonctionnent pour des nombres et des chaines de caractères. Dans le cas des chaines de caractères, l'ordre appliqué est l'ordre alphabétique.

Par exemple le nombre de documents qui ont pour la valeur **thousand** supérieure ou égale à 525 et la valeur **hundred** inférieure à 90 :

```
find>db.products.find({thousand:{$gte:525}, hundred:{$lt:90}}).count()
42500
find>
```

Filtrer les champs

Saisissez la requête **db.products.find({tenthousand: 525})** :

```
find>db.products.find({tenthousand: 525})
{ "_id" : ObjectId("59c0fb1b3b6221e3d3616bb2"), "counter" : 525, "tenthousand" : 525, "thousand" : 525, "hundred"
: 25 }
{ "_id" : ObjectId("59c0fb2e3b6221e3d36192c2"), "counter" : 10525, "tenthousand" : 525, "thousand" : 525,
"hundred" : 25 }
{ "_id" : ObjectId("59c0fb413b6221e3d361b9d2"), "counter" : 20525, "tenthousand" : 525, "thousand" : 525,
"hundred" : 25 }
{ "_id" : ObjectId("59c0fb543b6221e3d361e0e2"), "counter" : 30525, "tenthousand" : 525, "thousand" : 525,
"hundred" : 25 }
{ "_id" : ObjectId("59c0fb673b6221e3d36207f2"), "counter" : 40525, "tenthousand" : 525, "thousand" : 525,
"hundred" : 25 }
{ "_id" : ObjectId("59c0fb7a3b6221e3d3622f02"), "counter" : 50525, "tenthousand" : 525, "thousand" : 525,
"hundred" : 25 }
{ "_id" : ObjectId("59c0fb8f3b6221e3d3625612"), "counter" : 60525, "tenthousand" : 525, "thousand" : 525,
"hundred" : 25 }
{ "_id" : ObjectId("59c0fba23b6221e3d3627d22"), "counter" : 70525, "tenthousand" : 525, "thousand" : 525,
"hundred" : 25 }
{ "_id" : ObjectId("59c0fbb53b6221e3d362a432"), "counter" : 80525, "tenthousand" : 525, "thousand" : 525,
"hundred" : 25 }
{ "_id" : ObjectId("59c0fbca3b6221e3d362cb42"), "counter" : 90525, "tenthousand" : 525, "thousand" : 525,
"hundred" : 25 }
find>
```

Ajoutez maintenant un deuxième paramètre destiné à filtrer les champs retournés par la requête :

```
find>db.products.find({tenthousand: 525} , {counter:1})
{ "_id" : ObjectId("59c0fb1b3b6221e3d3616bb2"), "counter" : 525 }
{ "_id" : ObjectId("59c0fb2e3b6221e3d36192c2"), "counter" : 10525 }
{ "_id" : ObjectId("59c0fb413b6221e3d361b9d2"), "counter" : 20525 }
{ "_id" : ObjectId("59c0fb543b6221e3d361e0e2"), "counter" : 30525 }
{ "_id" : ObjectId("59c0fb673b6221e3d36207f2"), "counter" : 40525 }
{ "_id" : ObjectId("59c0fb7a3b6221e3d3622f02"), "counter" : 50525 }
```

```
{ "_id" : ObjectId("59c0fb8f3b6221e3d3625612"), "counter" : 60525 }
{ "_id" : ObjectId("59c0fba23b6221e3d3627d22"), "counter" : 70525 }
{ "_id" : ObjectId("59c0fbb53b6221e3d362a432"), "counter" : 80525 }
{ "_id" : ObjectId("59c0fbca3b6221e3d362cb42"), "counter" : 90525 }
find>
```

Important - Notez que vous n'avez plus que les champs **_id** et **counter** dans les résultats. Le champs **_id** est systématiquement renvoyé sauf quand il est explicitement indiqué le contraire dans la requête. **counter** est renvoyé parce que dans le second paramètre se trouve la valeur **1**.

Saisissez donc la requête suivante qui indique que celle-ci ne doit **pas** retournée le champs **_id** :

```
find>db.products.find({tenthousand: 525} , {_id:0, counter:1})
{ "counter" : 525 }
{ "counter" : 10525 }
{ "counter" : 20525 }
{ "counter" : 30525 }
{ "counter" : 40525 }
{ "counter" : 50525 }
{ "counter" : 60525 }
{ "counter" : 70525 }
{ "counter" : 80525 }
{ "counter" : 90525 }
find>
```

Trier

Saisissez la requête suivante :

```
find>db.products.find({thousand : {$in: [500, 600, 700]}})
```

```
{ "_id" : ObjectId("59c0fb1b3b6221e3d3616b99"), "counter" : 500, "tenthousand" : 500, "thousand" : 500, "hundred"
: 0 }
{ "_id" : ObjectId("59c0fb1b3b6221e3d3616bfd"), "counter" : 600, "tenthousand" : 600, "thousand" : 600, "hundred"
: 0 }
{ "_id" : ObjectId("59c0fb1b3b6221e3d3616c61"), "counter" : 700, "tenthousand" : 700, "thousand" : 700, "hundred"
: 0 }
{ "_id" : ObjectId("59c0fb1d3b6221e3d3616f81"), "counter" : 1500, "tenthousand" : 1500, "thousand" : 500,
"hundred" : 0 }
{ "_id" : ObjectId("59c0fb1d3b6221e3d3616fe5"), "counter" : 1600, "tenthousand" : 1600, "thousand" : 600,
"hundred" : 0 }
{ "_id" : ObjectId("59c0fb1d3b6221e3d3617049"), "counter" : 1700, "tenthousand" : 1700, "thousand" : 700,
"hundred" : 0 }
{ "_id" : ObjectId("59c0fb1f3b6221e3d3617369"), "counter" : 2500, "tenthousand" : 2500, "thousand" : 500,
"hundred" : 0 }
{ "_id" : ObjectId("59c0fb1f3b6221e3d36173cd"), "counter" : 2600, "tenthousand" : 2600, "thousand" : 600,
"hundred" : 0 }
{ "_id" : ObjectId("59c0fb1f3b6221e3d3617431"), "counter" : 2700, "tenthousand" : 2700, "thousand" : 700,
"hundred" : 0 }
{ "_id" : ObjectId("59c0fb213b6221e3d3617751"), "counter" : 3500, "tenthousand" : 3500, "thousand" : 500,
"hundred" : 0 }
{ "_id" : ObjectId("59c0fb213b6221e3d36177b5"), "counter" : 3600, "tenthousand" : 3600, "thousand" : 600,
"hundred" : 0 }
{ "_id" : ObjectId("59c0fb213b6221e3d3617819"), "counter" : 3700, "tenthousand" : 3700, "thousand" : 700,
"hundred" : 0 }
{ "_id" : ObjectId("59c0fb233b6221e3d3617b39"), "counter" : 4500, "tenthousand" : 4500, "thousand" : 500,
"hundred" : 0 }
{ "_id" : ObjectId("59c0fb233b6221e3d3617b9d"), "counter" : 4600, "tenthousand" : 4600, "thousand" : 600,
"hundred" : 0 }
{ "_id" : ObjectId("59c0fb233b6221e3d3617c01"), "counter" : 4700, "tenthousand" : 4700, "thousand" : 700,
"hundred" : 0 }
{ "_id" : ObjectId("59c0fb253b6221e3d3617f21"), "counter" : 5500, "tenthousand" : 5500, "thousand" : 500,
"hundred" : 0 }
{ "_id" : ObjectId("59c0fb253b6221e3d3617f85"), "counter" : 5600, "tenthousand" : 5600, "thousand" : 600,
"hundred" : 0 }
```

```
{ "_id" : ObjectId("59c0fb253b6221e3d3617fe9"), "counter" : 5700, "tenthousand" : 5700, "thousand" : 700,
"hundred" : 0 }
{ "_id" : ObjectId("59c0fb263b6221e3d3618309"), "counter" : 6500, "tenthousand" : 6500, "thousand" : 500,
"hundred" : 0 }
{ "_id" : ObjectId("59c0fb273b6221e3d361836d"), "counter" : 6600, "tenthousand" : 6600, "thousand" : 600,
"hundred" : 0 }
Type "it" for more
find>
```

Important - Notez que le résultat démontre une alternance des valeurs de **thousand**, alternativement 500, 600 et 700.

Triez maintenant les résultats sur la valeur de **thousand** dans le sens croissant :

```
find>db.products.find({thousand : {$in: [500, 600, 700]}}).sort({thousand:1})
{ "_id" : ObjectId("59c0fb1b3b6221e3d3616b99"), "counter" : 500, "tenthousand" : 500, "thousand" : 500, "hundred"
: 0 }
{ "_id" : ObjectId("59c0fb1d3b6221e3d3616f81"), "counter" : 1500, "tenthousand" : 1500, "thousand" : 500,
"hundred" : 0 }
{ "_id" : ObjectId("59c0fb1f3b6221e3d3617369"), "counter" : 2500, "tenthousand" : 2500, "thousand" : 500,
"hundred" : 0 }
{ "_id" : ObjectId("59c0fb213b6221e3d3617751"), "counter" : 3500, "tenthousand" : 3500, "thousand" : 500,
"hundred" : 0 }
{ "_id" : ObjectId("59c0fb233b6221e3d3617b39"), "counter" : 4500, "tenthousand" : 4500, "thousand" : 500,
"hundred" : 0 }
{ "_id" : ObjectId("59c0fb253b6221e3d3617f21"), "counter" : 5500, "tenthousand" : 5500, "thousand" : 500,
"hundred" : 0 }
{ "_id" : ObjectId("59c0fb263b6221e3d3618309"), "counter" : 6500, "tenthousand" : 6500, "thousand" : 500,
"hundred" : 0 }
{ "_id" : ObjectId("59c0fb283b6221e3d36186f1"), "counter" : 7500, "tenthousand" : 7500, "thousand" : 500,
"hundred" : 0 }
{ "_id" : ObjectId("59c0fb2a3b6221e3d3618ad9"), "counter" : 8500, "tenthousand" : 8500, "thousand" : 500,
```

```
"hundred" : 0 }
{ "_id" : ObjectId("59c0fb2c3b6221e3d3618ec1"), "counter" : 9500, "tenthousand" : 9500, "thousand" : 500,
"hundred" : 0 }
{ "_id" : ObjectId("59c0fb2e3b6221e3d36192a9"), "counter" : 10500, "tenthousand" : 500, "thousand" : 500,
"hundred" : 0 }
{ "_id" : ObjectId("59c0fb303b6221e3d3619691"), "counter" : 11500, "tenthousand" : 1500, "thousand" : 500,
"hundred" : 0 }
{ "_id" : ObjectId("59c0fb323b6221e3d3619a79"), "counter" : 12500, "tenthousand" : 2500, "thousand" : 500,
"hundred" : 0 }
{ "_id" : ObjectId("59c0fb343b6221e3d3619e61"), "counter" : 13500, "tenthousand" : 3500, "thousand" : 500,
"hundred" : 0 }
{ "_id" : ObjectId("59c0fb363b6221e3d361a249"), "counter" : 14500, "tenthousand" : 4500, "thousand" : 500,
"hundred" : 0 }
{ "_id" : ObjectId("59c0fb373b6221e3d361a631"), "counter" : 15500, "tenthousand" : 5500, "thousand" : 500,
"hundred" : 0 }
{ "_id" : ObjectId("59c0fb393b6221e3d361aa19"), "counter" : 16500, "tenthousand" : 6500, "thousand" : 500,
"hundred" : 0 }
{ "_id" : ObjectId("59c0fb3b3b6221e3d361ae01"), "counter" : 17500, "tenthousand" : 7500, "thousand" : 500,
"hundred" : 0 }
{ "_id" : ObjectId("59c0fb3d3b6221e3d361b1e9"), "counter" : 18500, "tenthousand" : 8500, "thousand" : 500,
"hundred" : 0 }
{ "_id" : ObjectId("59c0fb3f3b6221e3d361b5d1"), "counter" : 19500, "tenthousand" : 9500, "thousand" : 500,
"hundred" : 0 }
Type "it" for more
find>
```

Dernièrement, triez dans le sens décroissant :

```
find>db.products.find({thousand : {$in: [500, 600, 700]}}).sort({thousand:-1})
{ "_id" : ObjectId("59c0fb1b3b6221e3d3616c61"), "counter" : 700, "tenthousand" : 700, "thousand" : 700, "hundred"
: 0 }
{ "_id" : ObjectId("59c0fb1d3b6221e3d3617049"), "counter" : 1700, "tenthousand" : 1700, "thousand" : 700,
"hundred" : 0 }
{ "_id" : ObjectId("59c0fb1f3b6221e3d3617431"), "counter" : 2700, "tenthousand" : 2700, "thousand" : 700,
```

```
"hundred" : 0 }
{ "_id" : ObjectId("59c0fb213b6221e3d3617819"), "counter" : 3700, "tenthousand" : 3700, "thousand" : 700,
"hundred" : 0 }
{ "_id" : ObjectId("59c0fb233b6221e3d3617c01"), "counter" : 4700, "tenthousand" : 4700, "thousand" : 700,
"hundred" : 0 }
{ "_id" : ObjectId("59c0fb253b6221e3d3617fe9"), "counter" : 5700, "tenthousand" : 5700, "thousand" : 700,
"hundred" : 0 }
{ "_id" : ObjectId("59c0fb273b6221e3d36183d1"), "counter" : 6700, "tenthousand" : 6700, "thousand" : 700,
"hundred" : 0 }
{ "_id" : ObjectId("59c0fb293b6221e3d36187b9"), "counter" : 7700, "tenthousand" : 7700, "thousand" : 700,
"hundred" : 0 }
{ "_id" : ObjectId("59c0fb2b3b6221e3d3618ba1"), "counter" : 8700, "tenthousand" : 8700, "thousand" : 700,
"hundred" : 0 }
{ "_id" : ObjectId("59c0fb2d3b6221e3d3618f89"), "counter" : 9700, "tenthousand" : 9700, "thousand" : 700,
"hundred" : 0 }
{ "_id" : ObjectId("59c0fb2e3b6221e3d3619371"), "counter" : 10700, "tenthousand" : 700, "thousand" : 700,
"hundred" : 0 }
{ "_id" : ObjectId("59c0fb303b6221e3d3619759"), "counter" : 11700, "tenthousand" : 1700, "thousand" : 700,
"hundred" : 0 }
{ "_id" : ObjectId("59c0fb323b6221e3d3619b41"), "counter" : 12700, "tenthousand" : 2700, "thousand" : 700,
"hundred" : 0 }
{ "_id" : ObjectId("59c0fb343b6221e3d3619f29"), "counter" : 13700, "tenthousand" : 3700, "thousand" : 700,
"hundred" : 0 }
{ "_id" : ObjectId("59c0fb363b6221e3d361a311"), "counter" : 14700, "tenthousand" : 4700, "thousand" : 700,
"hundred" : 0 }
{ "_id" : ObjectId("59c0fb383b6221e3d361a6f9"), "counter" : 15700, "tenthousand" : 5700, "thousand" : 700,
"hundred" : 0 }
{ "_id" : ObjectId("59c0fb3a3b6221e3d361aae1"), "counter" : 16700, "tenthousand" : 6700, "thousand" : 700,
"hundred" : 0 }
{ "_id" : ObjectId("59c0fb3c3b6221e3d361aec9"), "counter" : 17700, "tenthousand" : 7700, "thousand" : 700,
"hundred" : 0 }
{ "_id" : ObjectId("59c0fb3e3b6221e3d361b2b1"), "counter" : 18700, "tenthousand" : 8700, "thousand" : 700,
"hundred" : 0 }
{ "_id" : ObjectId("59c0fb3f3b6221e3d361b699"), "counter" : 19700, "tenthousand" : 9700, "thousand" : 700,
```

```
"hundred" : 0 }
Type "it" for more
find>
```

LAB #3 - Requêtes sur la base movies

Préparation

Sortez du client mongo, puis télécharger à partir de la section Fichiers de ce cours le fichier au format JSON dont nous aurons besoin pour ce LAB :

- movies.json
  - ce fichier contient la liste de films complets comprenant tous les noms et prénoms des artistes, répétés à chaque occurrence.

Création de base de données

Connectez-vous à MongoDB avec le client **mongo** et créez une base de données **movies** contenant une collection **movies**,

```
[root@centos7 ~]# mongo
MongoDB shell version: 3.2.16
connecting to: test
Server has startup warnings:
2017-09-19T11:51:43.992+0200 I CONTROL  [initandlisten]
2017-09-19T11:51:43.992+0200 I CONTROL  [initandlisten] ** WARNING: soft rlimits too low. rlimits set to 4096
processes, 64000 files. Number of processes should be at least 32000 : 0.5 times number of files.
test>use movies
switched to db movies
movies>db.createCollection("movies")
{ "ok" : 1 }
movies>exit
bye
```

Importer les données

Utilisez maintenant la commande **mongoimport** pour importer le fichier dans MongoDB :

```
[root@centos7 ~]# mongoimport -d movies -c movies --file movies.json --jsonArray
2017-09-20T11:19:44.012+0200    connected to: localhost
2017-09-20T11:19:44.122+0200    imported 88 documents
```

Important - Notez que l'argument jsonArray indique à mongoimport qu'il s'agit d'un tableau d'objets à créer individuellement, et pas d'un unique document JSON.

Re-connectez-vous à MongoDB en utilisant le client mongo :

```
[root@centos7 ~]# mongo
MongoDB shell version: 3.2.16
connecting to: test
Server has startup warnings:
2017-09-19T11:51:43.992+0200 I CONTROL  [initandlisten]
2017-09-19T11:51:43.992+0200 I CONTROL  [initandlisten] ** WARNING: soft rlimits too low. rlimits set to 4096
processes, 64000 files. Number of processes should be at least 32000 : 0.5 times number of files.
test>
```

En utilisant la base de données **movies**, vérifiez que vous pouvez trouver les 88 documents :

```
movies>db.movies.count ()
88
```

Exercices

Trouvez maintenant les informations suivantes :

- tous les titres,
- le résumé de Spider-Man,
- le metteur en scène de Gladiator,
- les titres des films avec Kirsten Dunst,
- les films ayant un résumé,
- les films qui ne sont ni des drames ni des comédies,
- les titres des films et les noms des acteurs,
- les films où Clint Eastwood est acteur mais pas réalisateur.

Corrigés

Tous les titres

```
movies>db.movies.find({}, {"title": 1})
{ "_id" : "movie:1", "title" : "Vertigo" }
{ "_id" : "movie:2", "title" : "Alien" }
{ "_id" : "movie:3", "title" : "Titanic" }
{ "_id" : "movie:4", "title" : "Sacrifice" }
{ "_id" : "movie:5", "title" : "Volte/Face" }
{ "_id" : "movie:6", "title" : "Sleepy Hollow" }
{ "_id" : "movie:7", "title" : "American Beauty" }
{ "_id" : "movie:8", "title" : "Impitoyable" }
{ "_id" : "movie:9", "title" : "Gladiator" }
{ "_id" : "movie:10", "title" : "Blade Runner" }
{ "_id" : "movie:11", "title" : "Piège de cristal" }
{ "_id" : "movie:12", "title" : "58 minutes pour vivre" }
{ "_id" : "movie:13", "title" : "Van Gogh" }
{ "_id" : "movie:14", "title" : "Seven" }
{ "_id" : "movie:15", "title" : "Twelve Monkeys" }
{ "_id" : "movie:16", "title" : "Le last_name de la rose" }
```

```
{ "_id" : "movie:17", "title" : "Pulp fiction" }
{ "_id" : "movie:18", "title" : "Mary à tout prix" }
{ "_id" : "movie:19", "title" : "Terminator" }
{ "_id" : "movie:20", "title" : "Les dents de la mer" }
Type "it" for more
movies>
```

Le résumé de Spider-Man

```
movies>db.movies.find({"title": "Spider-Man"}, {"summary": 1})
{ "_id" : "movie:47", "summary" : "Orphelin, Peter Parker est élevé par sa tante May et son oncle Ben dans le quartier Queens de New York. Tout en poursuivant ses études à l'université, il trouve un emploi de photographe au journal Daily Bugle. Il partage son appartement avec Harry Osborn, son meilleur ami, et rêve de séduire la belle Mary Jane.Cependant, après avoir été mordu par une araignée génétiquement modifiée, Peter voit son agilité et sa force s'accroître et se découvre des pouvoirs surnaturels. Devenu Spider-Man, il décide d'utiliser ses nouvelles capacités au service du bien.Au même moment, le père de Harry, le richissime industriel Norman Osborn, est victime d'un accident chimique qui a démesurément augmenté ses facultés intellectuelles et sa force, mais l'a rendu fou. Il est devenu le Bouffon Vert, une créature démoniaque qui menace la ville. Entre lui et Spider-Man, une lutte sans merci s'engage." }
```

Le metteur en scène de Gladiator

```
movies>db.movies.find({"title": "Gladiator"}, {"director": 1})
{ "_id" : "movie:9", "director" : { "_id" : "artist:4", "last_name" : "Scott", "first_name" : "Ridley", "birth_date" : "1937" } }
```

Les titres des films avec Kirsten Dunst

```
movies>db.movies.find({"actors.last_name": "Dunst"}, {"title": 1})
```

```
{ "_id" : "movie:67", "title" : "Marie Antoinette" }
```

Les films ayant un résumé

```
movies>db.movies.find({"summary": {$exists: true}}, {"title": 1})
{ "_id" : "movie:1", "title" : "Vertigo" }
{ "_id" : "movie:2", "title" : "Alien" }
{ "_id" : "movie:3", "title" : "Titanic" }
{ "_id" : "movie:4", "title" : "Sacrifice" }
{ "_id" : "movie:5", "title" : "Volte/Face" }
{ "_id" : "movie:6", "title" : "Sleepy Hollow" }
{ "_id" : "movie:7", "title" : "American Beauty" }
{ "_id" : "movie:8", "title" : "Impitoyable" }
{ "_id" : "movie:9", "title" : "Gladiator" }
{ "_id" : "movie:10", "title" : "Blade Runner" }
{ "_id" : "movie:11", "title" : "Piège de cristal" }
{ "_id" : "movie:12", "title" : "58 minutes pour vivre" }
{ "_id" : "movie:13", "title" : "Van Gogh" }
{ "_id" : "movie:14", "title" : "Seven" }
{ "_id" : "movie:15", "title" : "Twelve Monkeys" }
{ "_id" : "movie:16", "title" : "Le last_name de la rose" }
{ "_id" : "movie:17", "title" : "Pulp fiction" }
{ "_id" : "movie:18", "title" : "Mary à tout prix" }
{ "_id" : "movie:19", "title" : "Terminator" }
{ "_id" : "movie:20", "title" : "Les dents de la mer" }
Type "it" for more
```

Les films qui ne sont ni des drames ni des comédies

```
movies>db.movies.find({"genre": {$nin: ["Drame", "Comédie"]}}, {"title": 1, "genre": 1})
{ "_id" : "movie:1", "title" : "Vertigo", "genre" : "drama" }
```

```
{ "_id" : "movie:2", "title" : "Alien", "genre" : "Science-fiction" }
{ "_id" : "movie:3", "title" : "Titanic", "genre" : "drama" }
{ "_id" : "movie:4", "title" : "Sacrifice", "genre" : "drama" }
{ "_id" : "movie:5", "title" : "Volte/Face", "genre" : "Action" }
{ "_id" : "movie:6", "title" : "Sleepy Hollow", "genre" : "Fantastique" }
{ "_id" : "movie:8", "title" : "Impitoyable", "genre" : "Western" }
{ "_id" : "movie:9", "title" : "Gladiator", "genre" : "drama" }
{ "_id" : "movie:10", "title" : "Blade Runner", "genre" : "Action" }
{ "_id" : "movie:11", "title" : "Piège de cristal", "genre" : "Action" }
{ "_id" : "movie:12", "title" : "58 minutes pour vivre", "genre" : "Action" }
{ "_id" : "movie:13", "title" : "Van Gogh", "genre" : "drama" }
{ "_id" : "movie:14", "title" : "Seven", "genre" : "crime" }
{ "_id" : "movie:15", "title" : "Twelve Monkeys", "genre" : "Science-fiction" }
{ "_id" : "movie:16", "title" : "Le last_name de la rose", "genre" : "crime" }
{ "_id" : "movie:17", "title" : "Pulp fiction", "genre" : "Action" }
{ "_id" : "movie:19", "title" : "Terminator", "genre" : "Science-fiction" }
{ "_id" : "movie:20", "title" : "Les dents de la mer", "genre" : "Horreur" }
{ "_id" : "movie:21", "title" : "Le silence des agneaux", "genre" : "crime" }
{ "_id" : "movie:22", "title" : "Godzilla", "genre" : "Action" }
Type "it" for more
```

Les titres des films et les noms des acteurs

```
movies>db.movies.find({}, {"title": 1, "actors.first_name": 1, "actors.last_name": 1})
{ "_id" : "movie:1", "title" : "Vertigo", "actors" : [ { "first_name" : "James", "last_name" : "Stewart" }, {
"first_name" : "Kim", "last_name" : "Novak" }, { "first_name" : "Arthur", "last_name" : "Pierre" } ] }
{ "_id" : "movie:2", "title" : "Alien", "actors" : [ { "first_name" : "Sigourney", "last_name" : "Weaver" } ] }
{ "_id" : "movie:3", "title" : "Titanic", "actors" : [ { "first_name" : "Kate", "last_name" : "Winslet" }, {
"first_name" : "Leonardo", "last_name" : "DiCaprio" } ] }
{ "_id" : "movie:4", "title" : "Sacrifice", "actors" : [ ] }
{ "_id" : "movie:5", "title" : "Volte/Face", "actors" : [ { "first_name" : "John", "last_name" : "Travolta" }, {
"first_name" : "Nicolas", "last_name" : "Cage" } ] }
{ "_id" : "movie:6", "title" : "Sleepy Hollow", "actors" : [ { "first_name" : "Johnny", "last_name" : "Depp" }, {
```

```
"first_name" : "Christina", "last_name" : "Ricci" }, { "first_name" : "Christopher", "last_name" : "Walken" } ] }
{ "_id" : "movie:7", "title" : "American Beauty", "actors" : [ { "first_name" : "Kevin", "last_name" : "Spacey"
}, { "first_name" : "Anette", "last_name" : "Bening" } ] }
{ "_id" : "movie:8", "title" : "Impitoyable", "actors" : [ { "first_name" : "Clint", "last_name" : "Eastwood" },
{ "first_name" : "Gene", "last_name" : "Hackman" }, { "first_name" : "Morgan", "last_name" : "Freeman" } ] }
{ "_id" : "movie:9", "title" : "Gladiator", "actors" : [ { "first_name" : "Russell", "last_name" : "Crowe" }, {
"first_name" : "Adam", "last_name" : "Baldwin" }, { "first_name" : "Ryan", "last_name" : "ONeal" }, {
"first_name" : "Marisa", "last_name" : "Berenson" } ] }
{ "_id" : "movie:10", "title" : "Blade Runner", "actors" : [ { "first_name" : "Harrison", "last_name" : "Ford" },
{ "first_name" : "Rutger", "last_name" : "Hauer" } ] }
{ "_id" : "movie:11", "title" : "Piège de cristal", "actors" : [ { "first_name" : "Bruce", "last_name" : "Willis"
} ] }
{ "_id" : "movie:12", "title" : "58 minutes pour vivre", "actors" : [ { "first_name" : "Bruce", "last_name" :
"Willis" } ] }
{ "_id" : "movie:13", "title" : "Van Gogh", "actors" : [ { "first_name" : "Jacques", "last_name" : "Dutronc" } ]
}
{ "_id" : "movie:14", "title" : "Seven", "actors" : [ { "first_name" : "Kevin", "last_name" : "Spacey" }, {
"first_name" : "Morgan", "last_name" : "Freeman" }, { "first_name" : "Brad", "last_name" : "Pitt" } ] }
{ "_id" : "movie:15", "title" : "Twelve Monkeys", "actors" : [ { "first_name" : "Bruce", "last_name" : "Willis" }
] }
{ "_id" : "movie:16", "title" : "Le last_name de la rose", "actors" : [ { "first_name" : "Sean", "last_name" :
"Connery" }, { "first_name" : "Christian", "last_name" : "Slater" } ] }
{ "_id" : "movie:17", "title" : "Pulp fiction", "actors" : [ { "first_name" : "John", "last_name" : "Travolta" },
{ "first_name" : "Bruce", "last_name" : "Willis" }, { "first_name" : "Quentin", "last_name" : "Tarantino" }, {
"first_name" : "Samuel L.", "last_name" : "Jackson" }, { "first_name" : "Rosanna", "last_name" : "Arquette" }, {
"first_name" : "Uma", "last_name" : "Thurman" }, { "first_name" : "Christopher", "last_name" : "Walken" }, {
"first_name" : "Harvey", "last_name" : "Keitel" }, { "first_name" : "Tim", "last_name" : "Roth" } ] }
{ "_id" : "movie:18", "title" : "Mary à tout prix", "actors" : [ { "first_name" : "Cameron", "last_name" : "Diaz"
}, { "first_name" : "Mat", "last_name" : "Dillon" } ] }
{ "_id" : "movie:19", "title" : "Terminator", "actors" : [ { "first_name" : "Arnold", "last_name" :
"Schwartzenegger" } ] }
{ "_id" : "movie:20", "title" : "Les dents de la mer", "actors" : [ { "first_name" : "Roy", "last_name" :
"Scheider" }, { "first_name" : "Robert", "last_name" : "Shaw" }, { "first_name" : "Richard", "last_name" :
"Dreyfus" } ] }
```

```
Type "it" for more
```

Les films où Clint Eastwood est acteur mais pas réalisateur

```
movies>db.movies.find({"actors.last_name": "Eastwood", "director.last_name": {$ne: "Eastwood"}}, {"title": 1})
{ "_id" : "movie:32", "title" : "Le bon, la brute et le truand" }
```

LAB #4 - Jointures

Préparation

Sortez du client mongo, puis téléchargez à partir de la section Fichiers de ce cours les deux fichiers au format JSON dont nous aurons besoin pour ce LAB :

- movies_ref.json
  - ce fichier contient la liste des films avec références, les identifiants des artistes, et impose donc d'effectuer des jointures
- artists.json
  - ce fichier contient la liste des artistes.

Création des bases de données

Connectez-vous à MongoDB avec le client **mongo** et créez une base de données **moviesref** contenant deux collections **movies** et **artists**.

```
[root@centos7 ~]# mongo
MongoDB shell version: 3.2.16
connecting to: test
Server has startup warnings:
2017-09-19T11:51:43.992+0200 I CONTROL  [initandlisten]
2017-09-19T11:51:43.992+0200 I CONTROL  [initandlisten] ** WARNING: soft rlimits too low. rlimits set to 4096
```

```
processes, 64000 files. Number of processes should be at least 32000 : 0.5 times number of files.
movies>use moviesref
switched to db moviesref
moviesref>db.createCollection("movies")
{ "ok" : 1 }
moviesref>db.createCollection("artists")
{ "ok" : 1 }
moviesref>exit
bye
```

Importer les données

Utilisez maintenant la commande **mongoimport** pour importer les fichiers dans MongoDB :

```
[root@centos7 ~]# mongoimport -d moviesref -c movies --file movies-refs.json --jsonArray
2017-09-20T11:19:58.697+0200    connected to: localhost
2017-09-20T11:19:58.721+0200    imported 88 documents
[root@centos7 ~]# mongoimport -d moviesref -c artists --file artists.json --jsonArray
2017-09-20T11:20:14.384+0200    connected to: localhost
2017-09-20T11:20:14.405+0200    imported 206 documents
```

Important - Notez que l'argument jsonArray indique à mongoimport qu'il s'agit d'un tableau d'objets à créer individuellement, et pas d'un unique document JSON.

Re-connectez-vous à MongoDB en utilisant le client mongo :

```
[root@centos7 ~]# mongo
MongoDB shell version: 3.2.16
connecting to: test
Server has startup warnings:
```

```
2017-09-19T11:51:43.992+0200 I CONTROL  [initandlisten]
2017-09-19T11:51:43.992+0200 I CONTROL  [initandlisten] ** WARNING: soft rlimits too low. rlimits set to 4096
processes, 64000 files. Number of processes should be at least 32000 : 0.5 times number of files.
test>use moviesref
switched to db moviesref
```

Effectuer la jointure du côté client

Le serveur MongoDB ne sait pas effectuer de jointures. Pour cette raison celles-ci doivent être mises en place du côté client. Cela revient essentiellement à appliquer l'algorithme de jointures par boucle imbriquées en stockant des données temporaires dans des structures de données sur le client, et en effectuant des échanges réseaux entre le client et le serveur.

La première étape dans la jointure côté client consiste à chercher l'artiste Clint Eastwood et à le stocker dans l'espace mémoire du client :

```
moviesref>eastwood = db.artists.findOne({"first_name": "Clint", "last_name": "Eastwood"})
{
    "_id" : "artist:20",
    "last_name" : "Eastwood",
    "first_name" : "Clint",
    "birth_date" : "1930"
}
```

Ensuite il convient d'utiliser cette "variable" pour rechercher dans la collection **movies** :

```
moviesref>db.movies.find({"director._id": eastwood['_id']}, {"title": 1})
{ "_id" : "movie:8", "title" : "Impitoyable" }
{ "_id" : "movie:26", "title" : "Les pleins pouvoirs" }
{ "_id" : "movie:63", "title" : "Million Dollar Baby" }
```

MapReduce

Définition

Etant donné une collection de documents on applique un traitement en deux phases:

La première phase appelée map

Une fonction est appliquée à chaque document de la collection et produit une valeur placée dans un **accumulateur**.

La seconde phase appelée reduce

Les valeurs placées dans l'accumulateur sont traitées par une fonction d'agrégation **reduce** ,produisant une valeur finale.

Un exemple simple peut être vu en considérant la requête SQL suivante :

```
select count(*) from Collection
```

Dans ce cas :

- la phase **map** produit une valeur de 1 et la place dans l'**accumulateur** pour chaque document dans la collection,
- la phase **reduce** calcule la somme pour produire un résultat.

En allant un peu plus loins, considérez la requête SQL suivante :

```
select count(*) from Collection group by annee
```

Dans ce cas, les valeurs produites par le **map** sont **partionnées** en groupes où chaque groupe représente une année. Le **map** produit donc des paires **groupe, valeur** ou **année, valeur**.

Revenons maintenant à notre base de données de films. Notre but est de produire un document par réalisateur contenant la liste des films réalisés par ce réalisateur :

- la phase **map** : un groupe doit être créé par réalisateur contenant les films réalisés par ce dernier,
- la phase **reduce** : la création du document final.

Par exemple la phase **map** est la définition d'une variable **mapRealisateur** contenant une fonction. Saisissez donc cette commande dans l'interface du client mongo :

```
movies>var mapRealisateur = function() {
...               emit(this.director._id, this.title);
...       };
movies>
```

Important - La fonction contient l'instruction **emit** qui produit une paire **clef:valeur** constituée de l'identifiant du réalisateur et du titre du film. Le mot clef **this** indique le document actuel.

La phase **reduce** contient une fonction, **reduceRealisateur**, qui prend deux arguments **directorId**, l'identifiant du groupe auquel elle s'applique, et la liste des valeurs produites par le **map** sous forme d'un tableau javascript. Saisissez donc cette commande dans l'interface du client mongo :

```
movies>var reduceRealisateur = function(directorId, titres) {
...    var res = new Object();
...    res.director = directorId;
...    res.films = titres;
...    return res;
...  };
movies>
```

Important - La fonction construit la valeur de résultat comme un objet **res** auquel on affecte deux propriétés: **director** et **titres**.

Pour lancer le traitement, il convient d'exécuter la commande suivante qui appelle la fonction **mapReduce** sur la collection **movies** :

```
movies>db.movies.mapReduce(mapRealisateur,  reduceRealisateur, {out: {"inline": 1}} )
{
    "results" : [
        {
            "_id" : "artist:1",
            "value" : "Marie Antoinette"
        },
        {
            "_id" : "artist:10",
            "value" : "Volte/Face"
        },
        {
            "_id" : "artist:101",
            "value" : {
                "director" : "artist:101",
                "films" : [
                    "Eyes Wide Shut",
                    "Shining"
                ]
            }
        },
        {
            "_id" : "artist:111",
            "value" : {
                "director" : "artist:111",
                "films" : [
                    "Jeanne d'Arc",
                    "Le cinquième élément",
                    "Léon",
                    "Nikita",
                    "Le grand bleu"
                ]
```

```
            }
        },
        {
            "_id" : "artist:122",
            "value" : {
                "director" : "artist:122",
                "films" : [
                    "King of New York",
                    "Bad Lieutenant"
                ]
            }
        },
        {
            "_id" : "artist:13",
            "value" : "Sleepy Hollow"
        },
        {
            "_id" : "artist:135",
            "value" : "The Matrix Revolutions"
        },
        {
            "_id" : "artist:138",
            "value" : "De bruit et de fureur"
        },
        {
            "_id" : "artist:142",
            "value" : "Usual suspects"
        },
        {
            "_id" : "artist:168",
            "value" : "Une journée en enfer"
        },
        {
            "_id" : "artist:17",
```

```
            "value" : {
                "director" : "artist:17",
                "films" : [
                    "American Beauty",
                    "Skyfall"
                ]
            }
        },
...
```

Important - Le premier paramètre est la fonction de **map**, le second la fonction de **reduce**, et le troisième indique la sortie, ici l'écran.

MapReduce peut prendre plusieurs options dont une s'avère particuluièrement utile, à savoir le résultat d'une requête :

```
movies>db.movies.mapReduce(mapRealisateur,  reduceRealisateur,
...          {out: {"inline": 1}, query: {"country": "USA"}} )
{
    "results" : [
        {
            "_id" : "artist:1",
            "value" : "Marie Antoinette"
        },
        {
            "_id" : "artist:10",
            "value" : "Volte/Face"
        },
        {
            "_id" : "artist:101",
            "value" : "Eyes Wide Shut"
        },
        {
```

```
            "_id" : "artist:122",
            "value" : {
                "director" : "artist:122",
                "films" : [
                    "King of New York",
                    "Bad Lieutenant"
                ]
            }
        },
        {
            "_id" : "artist:13",
            "value" : "Sleepy Hollow"
        },
        {
            "_id" : "artist:135",
            "value" : "The Matrix Revolutions"
        },
        {
            "_id" : "artist:142",
            "value" : "Usual suspects"
        },
        {
            "_id" : "artist:168",
            "value" : "Une journée en enfer"
        },
        {
            "_id" : "artist:17",
            "value" : {
                "director" : "artist:17",
                "films" : [
                    "American Beauty",
                    "Skyfall"
                ]
            }
```

```
        },
```

Références

- https://www.mongodb.com/fr
- https://fr.wikipedia.org/wiki/MongoDB
- http://blog.ippon.fr/2013/11/19/mongodb-est-moins-rapide-et-alors/
- http://www.next-decision.fr/les-editeurs/stockage/mongo-db
- https://docs.mongodb.com/manual/release-notes/
- http://b3d.bdpedia.fr/mongodb.html
- https://openclassrooms.com/courses/guide-de-demarrage-pour-utiliser-mongodb
- http://blog.xebia.fr/2010/12/15/mongodb-en-pratique/
- http://www.slideshare.net/marksmalley1/json-the-argonauts-and-mark
- https://fr.wikipedia.org/wiki/BSON
- https://blog.michaelckennedy.net/2013/04/22/a-roundup-of-mongodb-management-tools-nosql-database/
- http://rockmongo.com/
- http://genghisapp.com/
- http://b3d.bdpedia.fr/mongodb.html
- http://blog.xebia.fr/2010/12/15/mongodb-en-pratique/
- https://api.mongodb.com/java/current/
- http://javarticles.com/2016/01/inserting-document-using-mongodb-java-driver.html
- https://www.mongodb.com/blog/post/introducing-new-c-driver
- http://mongoc.org/libmongoc/1.2.2/tutorial.html

<html>

</html>