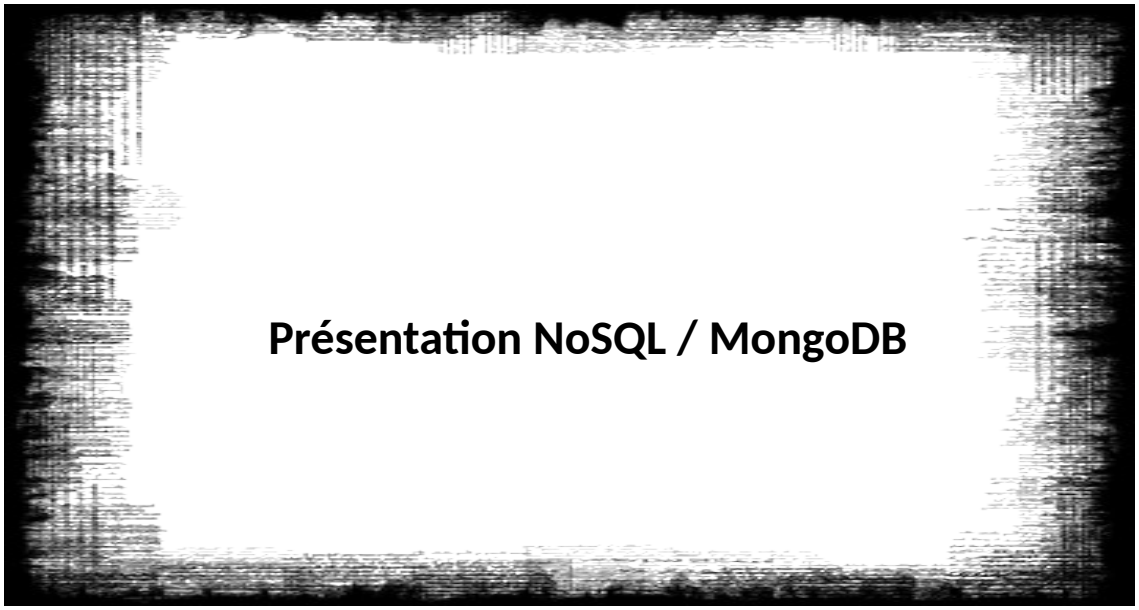
Administration

Pierrick Bouléry

Table des matières



Objectifs

Présentation du mouvement NoSQL
Motivations, historique
Les principaux produits
Présentation de MongoDB

Le mouvement NoSQL : Généralités

Les serveurs NoSQL visent à simplifier le modèle de données.
L'objectif est l'efficacité du stockage et de la restitution des données.
Par opposition aux serveurs SQL :

Pas de dictionnaire,
Pas de contraintes d'intégrité.
L'intégrité des données relève des applicatifs.
Les serveurs NoSQL ne gèrent généralement que le stockage,

Autre objectif : extensibilité horizontale, par ajout de serveurs (réplication, sharding).

Historique

NoSQL signifie 'Not Only SQL'.
L'expression apparaît en 1998 avec le moteur Strozzi NoSQL.
Elle est reprise en 2009 par Eric Evans lors d'une conférence sur les serveurs de bases de données distribuées.
La tendance, sous l'impulsion d'acteurs majeurs du Web, était à la gestion de données non-relationnelles.
Travaux menés dans les années 90 par Google avec BigTable ou FaceBook avec Cassandra.
Les grosses volumétries imposent de s'affranchir des contraintes classiques inhérentes aux SGBDR.
Parallèlement, émergence des méthodes agiles et des techniques RAD (Rapid Application Development) : nécessité d'un modèle de données adaptable aux besoins du projet et de ses évolutions.

Autres facteurs :

- Puissance croissante du matériel + diminution des coûts = multiplication des serveurs.
- Augmentation de la bande passante des réseaux.
- Développement des logiciels libres.

Technologies associées

- Google File System https://fr.wikipedia.org/wiki/Google_File_System
- Google Map Reduce https://fr.wikipedia.org/wiki/MapReduce
- Google Big Table https://fr.wikipedia.org/wiki/BigTable
- Amazon DynamoDB https://en.wikipedia.org/wiki/Amazon_DynamoDB
- Apache Cassandra https://fr.wikipedia.org/wiki/Cassandra_(base_de_donn%C3%A9es)

Théorème CAP

Ce 'théorème' est présenté en 2000 par Eric Brewer dans le cadre d'une conférence sur les systèmes distribués et retient 3 objectifs :

CAP = Consistency, Availability, Partition tolerance. (En français CDP = Consistance, Disponibilité, tolérence au Partitionnement)

- Consistency : Les nœuds du système possèdent des données identiques.

- Availability : Le système garantit une réponse aux requêtes même en cas de défaillance d'un nœud.

- Paritition Tolerance : Aucune panne, sauf panne complète du réseau, ne doit empêcher le système de répondre : en cas de partitionnement, chaque nœud doit fonctionner de manière autonome.

Le théorème : Au temps T, il est impossible de garantir les 3 en même temps. Généralement, on renonce à la consistance.

https://fr.wikipedia.org/wiki/Th%C3%A9or%C3%A8me_CAP

SQL/NoSql

SQL / SGBD relationnels : ACID : Atomicity, Consistency, Isolation, Durability

NoSQL : BASE : Basic Availability, Soft state, Eventual consistency

ACID

Repose sur le concept de transaction et de journalisation des transactions.

```
Begin work
        update ....
        if error
                rollback work
                return
        update ....
        if error
                rollback work
                return
        ../..
commit work
```

Atomicité : L'ensemble des opérations regroupées dans une transaction est réalisé complètement ou pas du tout.

Consistance : La base passe d'un état cohérent à un autre état cohérent.

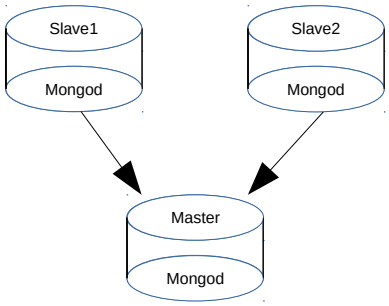
Isolation : Seule la session initiant la transaction voit les modifications apportées. Les sessions concurrentes voient les 'images avant'.

Durabilité : La journalisation permet de tracer les transactions réalisées. En cas de crash, les journaux peuvent être 'rejoués' après la restauration d'une sauvegarde.

BASE

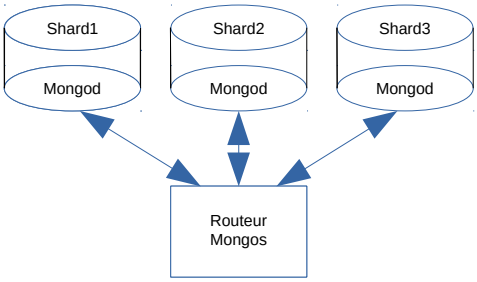
Les systèmes sont conçus pour être extensibles horizontalement. Avec MongoDB, les mécanismes sont la réplication et le sharding.

La réplication consiste à reproduire les mêmes données sur N serveur. Ici, un master et 2 slaves.



Les slaves se synchronisent sur le master. Indirectement, un slave peut se synchroniser sur un autre slave. L'ensemble s'appelle un 'replica set'.

Le sharding consiste à répartir les données selon une clé :



Environ un tiers des données sur le shard 1, un 2ème tiers sur le shard 2 et le reste sur le shard 3. Le système tend à s'auto équilibrer, éventuellement en basculant des données d'un shard à un autre.

On peut mixer le sharding et la réplication : N shards de replica sets à M nœuds.
Dans l'exemple, on aurait un système avec 3 shards * (1 master + 2 slaves) + le routeur mongos.
La réplication offre la redondance des données et le sharding amène l'équilibrage de charge.

'Base' est un acronyme qui s'applique aux architectures réparties, telles que déployées dans le monde NoSQL :

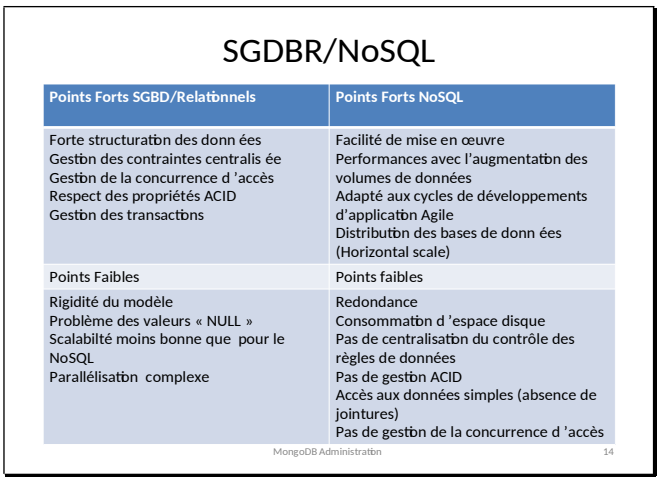
Base = Basically available, Soft state, Eventual consistency

Basically available : Le système répond aux requêtes même si les bases ne sont pas consistantes,

Soft state : l'état du système peut varier dans le temps même sans modification de données,

Eventual consistency : l' état du système tend vers la consistance. Le système devient consistant s'il n'y a plus d'entrée.

En résumé



SGDBR/NoSQL

Points Forts SGBD/Relationnels	Points Forts NoSQL
Forte structuration des données Gestion des contraintes centralisée Gestion de la concurrence d'accès Respect des propriétés ACID Gestion des transactions	Facilité de mise en œuvre Performances avec l'augmentation des volumes de données Adapté aux cycles de développements d'application Agile Distribution des bases de données (Horizontal scale)
Points Faibles	Points faibles
Rigidité du modèle Problème des valeurs « NULL » Scalabilté moins bonne que pour le NoSQL Parallélisation complexe	Redondance Consommation d'espace disque Pas de centralisation du contrôle des règles de données Pas de gestion ACID Accès aux données simples (absence de jointures) Pas de gestion de la concurrence d'accès

MongoDB Administration 14

Modèles de données et produits

Modèle clé / valeur

La clé, unique, est un identifiant pour chaque valeur de la base.
Ce modèle s'apparente aux tableaux associatifs tels qu'implémenté, par exemple en PERL ou PHP.
La valeur peut être un type simple ou un objet sérialisable.

Redis, Voldemort, Riak

Bases orientées documents

Dans ce modèle on stocke des documents dans des collections.
Modèle plus ou moins structuré : Les documents peuvent être des objets structurés (format JSON) ou ne stocker que de simples types et sont accédés par un identifiant.

MongoDB, CouchDB, Riak

Bases orientées colonnes

Les données sont stockées sous forme de colonnes.
Évolutivité du modèle par ajout dynamique de colonnes, sans coût de réorganisation.

Cassandra, BigTable, Hbase

Bases orientées graphes

Les données sont modélisées sous forme de nœuds reliés entre eux par des arcs nommés.
Modèle utilisé par exemple pour les réseaux sociaux.

Neo4j, FlockDB (Twitter)

Présentation de MongoDB

1996 - 2005 : la société Doubleclick (Kevin Ryan) développe une plate forme distribuée destinée au cloud : montée en charge facilitée par l'ajout de serveurs.

Écrit en JavaScript.

2010 – 2013 : Première version viable, la 1.4.

Le code est mis en open source sous licence AGPL. Le développement du moteur de base de données devient communautaire.

Création de la société 10gen qui deviendra MongoDB Inc (le nom MongoDB vient de l'anglais *humongous* qui signifie *énorme)*.

Cette société coordonne les développements et propose des services de support.

Le moteur MongoDB est réécrit en C++.

Aujourd'hui, très diffusé, MongoDB est utilisé entre autres par eBay, SouceForge ou le New York Times.

JSON / BSON

JSON est un format de données textuelles dérivé de Javascript :

https://fr.wikipedia.org/wiki/JavaScript_Object_Notation

Ce format est utilisé de façon externe : formalisation des requêtes utilisateur, imports / exports de données

Les types de données sont :

- Chaînes de caractères
- Nombres
- Dates
- Booléens
- Tableaux
- Objets
- NULL

Le format BSON est adopté pour le stockage des données. Il s'agit du format interne de MongoDB, développé par la société MongoDB inc.

BSON signifie 'binary JSON' : les applications voient les documents et leurs données comme des objets JSON.

JSON est léger, l'encodage/decodage JSON/BSON se fait à la volée par le moteur.

Avec ses métadonnées, BSON est plus rapide et plus efficace que JSON.

MongoDB identifie les objets par un _id.

Cet _id est généré par le moteur lors de l'insertion. Systématiquement indexe, c'est la méthode privilégiée d'accès aux données.

Lors de l'export d'une table quelconque (table SQL, fichier Excel..) si on dispose d'une clé primaire, on peut forcer l'_id.

Exemple de document JSON identifiant une personne :

```
{
_id: ObjectId("5146bb52d8524270060001f3"),
age: 25,
city: "Los Angeles",
email: "mark@abc.com",
user_name: "Mark Hanks"
}
```

Les documents sont stockés dans une collection, elle même rattachée à une base.
Le document est structuré en couples champ : valeur

Ex : { age : 25, city : « Los Angeles » }

Les collections sont 'schéma less', c'est à dire que le schéma de données existe au niveau du document mais pas au niveau collection, encore moins au niveau base.

Les documents suivants cohabitent parfaitement dans la même collection :

```
{
_id: ObjectId("5146bb52d8524270060001f3"),
age: 25,
city: "Los Angeles",
email: "mark@abc.com",
user_name: "Mark Hanks"
}
{
_id: ObjectId("5146bb52d8524270060001f5"),
user_name: "Robert Jones",
phone: "123456789",
}
```

Cette souplesse qu'apporte le format JSON permet de s'affranchir de structures de collections et de dictionnaire de données.

Elle élimine le problème des valeurs NULL, inhérent au modèle relationnel.

Le format JSON offre la possibilité d'imbriquer les documents :

```
{
_id: ObjectId("5146bb52d8524270060001f3"),
    identity : { age: 25, user_name: "Mark Hanks" },
    contact : { city: "Los Angeles", email: "mark@abc.com" }
}
{
_id: ObjectId("5146bb52d8524270060001f5"),
    identity : { user_name: "Robert Jones" },
    contact : { city: "Dallas", phone: "123456789" }
}
```

Cette représentation sous forme de documents / sous documents est pratique et permet de retrouver la totalité de l'information par l'accès à son '_id'.

La contrepartie est une forme de redondance des données :

```
{
_id: ObjectId("5146bb52d8524270060001f3"),
    identity : { age: 25, user_name: "Mark Hanks" },
    job : { role : "Manager", agency : "Los Angeles" },
    contact : { email: "mark@abc.com" }
}
{
_id: ObjectId("5146bb52d8524270060001f5"),
    identity : { user_name: "Robert Jones" },
    job : { role : "Sales", agency : "Dallas" },
    contact : { phone: "123456789" }
}
```

Dans le monde SQL, on aurait 3 tables et 2 jointures :

Comparatif SQL/MongoDB

SQL	Mongodb
database	database
table	collection
row/rangée	document/JSON/BSON
colonne	champ

Autres caractéristiques

- nombreuses fonctions de traitement des données,
- agrégats,
- MapReduce,
- interpréteur JavaScript intégré,
- procédures stockées,
- gestion des utilisateurs et des droits d'accès,
- indexation,
- import/export de données dans différents formats,
- sauvegardes, point in time recovery

../..

Interfaces de programmation

- C
- C++
- Java
- JavaScript
- .NET
- C#, F#
- PowerShell
- Perl
- PHP
- Python
- Ruby

Objectifs

Installation
Serveur mongod, Client mongo
Création de la première base/collection
Éléments d'architecture

Installation par package

La version actuelle est la 4.0.6. Les installations se réalisent sous 'root'. L'installation créé un utilisateur mongod dans /etc/passwd.

Les packages à installer sont :

Mongodb-org-4.0.6-1.el7.x86_64	Le meta package qui installera automatiquement les 4 suivants
Mongodb-org-server-4.0.6-1.el7.x86_64	Le démon mongod et les fichiers de configuration
Mongodb-org-shell-4.0.6-1.el7.x86_64	Le shell mongo
Mongodb-org-tools-4.0.6-1.el7.x86_64	Les outils d'administration : mongoimport, mongoexport,bsondump, mongodump, mongorestore, mongofiles, mongostat, et mongotop.
Mongodb-org-mongos-4.0.6-1.el7.x86_64	Le démon mongos (sharding)

Centos

Créer un fichier mongodb-org-4.0.repo sous /etc/yum.repos.d :

```
[MongoDB]
name=MongoDB Repository
baseurl=http://repo.mongodb.org/yum/redhat/$releasever/mongodb-org/4.0/x86_64/
gpgcheck=1
enabled=1
gpgkey=https://www.mongodb.org/static/pgp/server-4.0.asc
```

Puis

```
yum install mongodb-org
```

Debian

Obtenir la clé publique

```
sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv
9DA31620334BD75D9DCB49F368818C72E52529D4
```

Créer un fichier 'mongodb-org-4.0.list' sous /etc/apt/sources.list.d/

```
deb http://repo.mongodb.org/apt/debian stretch/mongodb-org/4.0 main
```

Puis

```
apt-get update
apt-get install -y mongodb-org
```

Exécutables installés

Le serveur s'appelle mongod. L'interpréteur de commande, mongo. Ce shell mongo permet de soumettre les requêtes, mais supporte aussi nativement le JavaScript.

Les exécutables sont installés sous /usr/bin ; on n'aura donc pas de problème de PATH.

Les autres exécutables installés sont :

- Les outils de sauvegarde/restauration :
  mongodump, mongorestore, bsondump
- Les outils d'export/import (txt, csv....)
  mongoexport,mongoimport
- Le gestionnaire de shrading
  mongos
- Les outils de surveillance
  mongotop, mongostat
- Le stockage des BLOBs
  mongofiles - gridfs

Mongod

Par défaut :

- Le serveur mongod s'attend à placer les données sous /data/db
  (C:\data\db sous Windows).
- tourne en avant plan
- affiche sa trace sur la sortie standard
- écoute le port 27017 sur la seule adresse de loopback.

Exemple où le répertoire /data/db n'existe pas :

```
[root@grouik VMs]# mongod
2017-09-18T15:44:20.675+0200 I CONTROL  [initandlisten] MongoDB starting : pid=23441 port=27017 dbpath=/data/
db 64-bit host=grouik.localdomain
2017-09-18T15:44:20.675+0200 I CONTROL  [initandlisten] db version v3.4.3
2017-09-18T15:44:20.675+0200 I CONTROL  [initandlisten] git version:
f07437fb5a6cca07c10bafa78365456eb1d6d5e1
2017-09-18T15:44:20.675+0200 I CONTROL  [initandlisten] OpenSSL version: OpenSSL 1.0.2k-fips  26 Jan 2017
2017-09-18T15:44:20.675+0200 I CONTROL  [initandlisten] allocator: tcmalloc
2017-09-18T15:44:20.675+0200 I CONTROL  [initandlisten] modules: none
2017-09-18T15:44:20.675+0200 I CONTROL  [initandlisten] build environment:
2017-09-18T15:44:20.675+0200 I CONTROL  [initandlisten]     distarch: x86_64
2017-09-18T15:44:20.675+0200 I CONTROL  [initandlisten]     target_arch: x86_64
2017-09-18T15:44:20.675+0200 I CONTROL  [initandlisten] options: {}
2017-09-18T15:44:20.717+0200 I STORAGE  [initandlisten] exception in initAndListen: 29 Data directory /data/db not
found., terminating
2017-09-18T15:44:20.717+0200 I NETWORK  [initandlisten] shutdown: going to close listening sockets...
2017-09-18T15:44:20.717+0200 I NETWORK  [initandlisten] shutdown: going to flush diaglog...
2017-09-18T15:44:20.717+0200 I CONTROL  [initandlisten] now exiting
2017-09-18T15:44:20.717+0200 I CONTROL  [initandlisten] shutting down with code:100
[root@grouik VMs]#
```

On peut créer ce répertoire /data/db où en choisir un autre avec l'option --dbpath de la ligne de commandes.

Le répertoire de données doit impérativement appartenir à l'utilisateur mongod. Lors de la première utilisation, le répertoire sera initialisé.

L'option --fork permet de lancer le processus en tâche de fond et, dans ce cas, on prévoit de rediriger la trace d'exécution dans un fichier quelconque (--logpath) :

```
-bash-4.3$ mongod --dbpath /opt/mongod --fork --logpath /tmp/mongod
about to fork child process, waiting until server is ready for connections.
forked process: 25207
child process started successfully, parent exiting
-bash-4.3$
```

La trace d'exécution peut être suivie :

```
-bash-4.3$ tail -f /tmp/mongod
2017-09-18T18:01:28.323+0200 I CONTROL  [initandlisten]
2017-09-18T18:01:28.323+0200 I CONTROL  [initandlisten] ** WARNING: Access control is not enabled for the database.
2017-09-18T18:01:28.323+0200 I CONTROL  [initandlisten] **          Read and write access to data and configuration is unrestricted.
2017-09-18T18:01:28.323+0200 I CONTROL  [initandlisten]
2017-09-18T18:01:28.663+0200 I FTDC     [initandlisten] Initializing full-time diagnostic data capture with directory '/opt/mongod/diagnostic.data'
2017-09-18T18:01:29.121+0200 I INDEX    [initandlisten] build index on: admin.system.version properties: { v: 2, key: { version: 1 }, name: "incompatible_with_version_32", ns: "admin.system.version" }
2017-09-18T18:01:29.121+0200 I INDEX    [initandlisten]     building index using bulk method; build may temporarily use up to 500 megabytes of RAM
2017-09-18T18:01:29.123+0200 I INDEX    [initandlisten] build index done.  scanned 0 total records. 0 secs
2017-09-18T18:01:29.123+0200 I COMMAND  [initandlisten] setting featureCompatibilityVersion to 3.4
2017-09-18T18:01:29.124+0200 I NETWORK  [thread1] waiting for connections on port 27017
```

L'installation par package crée un fichier /etc/mongod.conf qui peut être utilisé pour préciser les options de démarrage (ci dessous un extrait) :

```
systemLog:

.../...

  # Log file to send write to instead of stdout - has to be a file, not directory
  path: /var/log/mongodb/mongod.log

.../...

processManagement:
  # Fork server process (false by default)
  fork: true

.../...

# net Options - Network interfaces settings
net:
  # Specify port number (27017 by default)
  port: 27017

  # Comma separated list of ip addresses to listen on (all local ips by default)
  bindIp: 127.0.0.1,::1

.../...

storage:
  # Directory for datafiles (defaults to /data/db/)
  dbPath: /opt/mongod
```

Ce fichier est au format YAML (YAML Aint' Markup Langage).

Attention, ce format ne reconnait pas les tabulations. Utiliser des espaces pour les indentations.

Dans cet extrait, figurent les sections systemLog, processManagement, net et storage. Sous chaque section, les paramêtres, indentés.

Ex: dans la section net, spécification du port à 27017 et restriction de l'écoute réseau sur la seule adresse de loopback (Ipv4, Ipv6).

Démarrage de mongod avec le fichier de configuration (option -f ou --config) :

```
-bash-4.3$ mongod -f /etc/mongod.conf
about to fork child process, waiting until server is ready for connections.
forked process: 26202
child process started successfully, parent exiting
-bash-4.3$
```

Autres paramètres de mongod

Paramètre mongod.conf	Paramètre ligne de commandes	Commentaire
systemLog.quiet	--quiet	trace un peu moins bavarde
processManagement.fork	--fork	exécute mongod en arrière plan rediriger la trace dans un fichier avec un logpath
systemLog.logAppend	--logappend	ajoute la trace mongod au fichier log plutôt que de l'écraser
net.port	--port	par défaut 27017. chaque instance mongod sur un même serveur doit écouter un port différent
net.bindIp	--bind_ip	adresse(s) à écouter. Par défaut 127.0.0.1. si on commente cette ligne, mongod écoute toute ses interfaces, de toute provenance. prévoir des règles firewall.
net.http	--httpinterface	active l'interface http http://localhost:28017 pour chaque instance, le port mongod+1000 l'interface http, obsolète, est désactivée depuis la version 3.2
storage.mmapv1.preallocDatafiles	--noprealloc	si true, les fichiers de datas seront initialisés lors des premiers insert. préjudiciable à la performance. mmapv1 uniquement.
storage.mmapv1.nsSize	--nssize	taille totale pour les 'namespaces'. mmapv1 uniquement. (paragraphe dédié plus loin)
storage.journal	--nojournal	si true, désactive la journalisation. Evidemment déconseillé.

Paramètre mongod.conf	Paramètre ligne de commandes	Commentaire
storage.directoryPerDB	--directoryperdb	crée un répertoire par base pour y stocker les fichiers. Utile notamment si les répertoires sont attachés à leur propre filesystem.
security.authorization	--auth	si true, active l'authentification

Documentation mongod.conf :
https://docs.mongodb.com/manual/reference/configuration-options/
Pour les options en ligne de commandes : mongod –help

Lors du premier démarrage le répertoire et ses fichiers sont initialisés (ce qui peut prendre un certain temps).

Le fichier mongod.lock empêche un second démarrage intempestif.

```
-bash-4.3$ ls -l /opt/mongod
total 208
-rw-r--r--.  1 mongod mongod 16384 18 sept. 19:09 collection-0-1983312679691188357.wt
-rw-r--r--.  1 mongod mongod 16384 18 sept. 19:09 collection-2-1983312679691188357.wt
drwxr-xr-x. 2 mongod mongod  4096 18 sept. 19:09 diagnostic.data
-rw-r--r--.  1 mongod mongod 16384 18 sept. 19:09 index-1-1983312679691188357.wt
-rw-r--r--.  1 mongod mongod 16384 18 sept. 19:09 index-3-1983312679691188357.wt
-rw-r--r--.  1 mongod mongod 16384 18 sept. 19:09 index-4-1983312679691188357.wt
drwxr-xr-x. 2 mongod mongod  4096 18 sept. 18:01 journal
-rw-r--r--.  1 mongod mongod 16384 18 sept. 19:09 _mdb_catalog.wt
-rw-r--r--.  1 mongod mongod     0 18 sept. 19:09 mongod.lock
-rw-r--r--.  1 mongod mongod 32768 18 sept. 19:09 sizeStorer.wt
-rw-r--r--.  1 mongod mongod    95 18 sept. 18:01 storage.bson
-rw-r--r--.  1 mongod mongod    49 18 sept. 18:01 WiredTiger
-rw-r--r--.  1 mongod mongod  4096 18 sept. 19:09 WiredTigerLAS.wt
-rw-r--r--.  1 mongod mongod    21 18 sept. 18:01 WiredTiger.lock
-rw-r--r--.  1 mongod mongod   994 18 sept. 19:09 WiredTiger.turtle
-rw-r--r--.  1 mongod mongod 53248 18 sept. 19:09 WiredTiger.wt
```

Le produit est livré avec un script de démarrage automatique :
/usr/lib/systemd/system/mongod.service.

(sous RH, /etc/selinux/config → permissive)

Pour activer le service	'systemctl enable mongod.service'.
Pour le démarrer	'systemctl start mongod.service'
Pour l'arrêter	'systemctl stop mongod.service'
Pour connaître l'état du service	'systemctl status mongod.service'

Mongo

Mongo est le client privilégié pour MongoDB. Il fonctionne en ligne de commandes, à l'instar de psql pour Postgres ou sqlplus pour Oracle.

Quelques options :

-u [--username]	user
-p [--password]	password
--host	hostname
--port	port_number
--verbose / --quiet	± bavard
-h [--help]	aide sur les options
--version	affiche quelques éléments de version

Syntaxe : mondo ***db***
où ***db*** peut être:

mabase	base 'mabase' sur la machine locale
192.168.0.X/mabase	base 'mabase' à l'adresse 192.168.0.X
192.168.0.X:27018/mabase	base 'mabase' à l'adresse 192.168.0.X sur le port 27018

Connexion sur la machine locale sans référence à une base

```
-bash-4.3$ mongo
MongoDB shell version v3.4.3
connecting to: mongodb://127.0.0.1:27017
MongoDB server version: 3.4.3
Server has startup warnings:
2017-09-19T14:34:05.603+0200 I STORAGE  [initandlisten]
2017-09-19T14:34:05.603+0200 I STORAGE  [initandlisten] ** WARNING: Using the XFS filesystem is strongly recommended with the WiredTiger storage engine
2017-09-19T14:34:05.603+0200 I STORAGE  [initandlisten] **          See http://dochub.mongodb.org/core/prodnotes-filesystem
2017-09-19T14:34:06.688+0200 I CONTROL  [initandlisten]
2017-09-19T14:34:06.688+0200 I CONTROL  [initandlisten] ** WARNING: Access control is not enabled for the database.
2017-09-19T14:34:06.688+0200 I CONTROL  [initandlisten] **          Read and write access to data and configuration is unrestricted.
2017-09-19T14:34:06.688+0200 I CONTROL  [initandlisten]
> db
test
> // la base par défaut est la base test
```

Après l'installation, la connexion est possible sans authentification. MongoDB s'appuie sur l'authentification OS puisque les seules connexions possibles le sont depuis localhost.

Premiers pas

```
> show dbs
admin  0.000GB
local  0.000GB
> show databases
admin  0.000GB
local  0.000GB
```

```
> use mydb
switched to db mydb
>
> db.xx.insert( {prenom: "Pierrick"} )
WriteResult({ "nInserted" : 1 })
>
> show databases
admin  0.000GB
local  0.000GB
mydb   0.000GB
>
> show collections
xx
>
> db.xx.find()
{ "_id" : ObjectId("59c12adfa8182647f7b03e7b"), "prenom" : "Pierrick" }
>
> db
mydb
>
> quit()
```

Les commandes show dbs et show databases sont équivalentes.

Pour sélectionner une base, on utilise la syntaxe ‘use <base>’.

On crée une collection à la volée avec le premier insert :

La syntaxe est db.xx.insert(../..) où db désigne la base courante et xx, la collection.

Le document est au format JSON : { prenom:"Pierrick" }.

MongoDB retourne un document WriteResult indiquant le succès de l’insertion.

L’insertion crée la collection et la base.

Pour visualiser le contenu on utilise la commande find(). On constate la génération automatique de l’_id.

Il est possible de choisir la base à l’invocation de mongo : mongo mydb.

Il est aussi possible de se connecter sans référence à une base quelconque :

```
mongo -nodb
```

La connexion peut alors s’établir depuis le shell mongo :

```
[root@grouik VMs]# mongo --nodb
MongoDB shell version v3.4.3
> conn = new Mongo("127.0.0.1:27017")
connection to 127.0.0.1:27017
> db = conn.getDB("test")
test
```

Sans argument, mongo se connecte en local à la base ‘test’. Il s’agit dans un premier temps d’une simple connexion logique.

```
> db
test
> show dbs
admin  0.000GB
local  0.000GB
mydb   0.000GB
>
> // on est connecté à la base test qui n'existe pas
>
> db.xx.insert({xx:"xx"})
WriteResult({ "nInserted" : 1 })
>
> show dbs
admin  0.000GB
local  0.000GB
mydb   0.000GB
test   0.000GB
>
> // la base test est créée en même temps que la première collection xx
>
```

La sortie de mongo s'effectue avec la commande exit ou quit().

Customiser le prompt

Le prompt mongo peut être redéfini dans le fichier $HOME/.mongorc.js avec la fonction JavaScript suivante :

```
function prompt() {
   var username = "anon";
   var user = db.runCommand({connectionStatus : 1}).authInfo.authenticatedUsers[0];
   var host = db.getMongo().toString().split(" ")[2];
   var current_db = db.getName();

   if (!!user) {
      username = user.user;
   }

   return username + "@" + host + ":" + current_db + "> ";
}
```

Le prompt customisé permet d'identifier la base et l'utilisateur courant :

```
pierrick@grouik MongoDB]$ mongo                          // connexion anonyme – base test
MongoDB shell version v3.4.9
connecting to: mongodb://127.0.0.1:27017
MongoDB server version: 3.4.9
anon@127.0.0.1:27017:test> use admin                     // changement de base
switched to db admin
anon@127.0.0.1:27017:admin> db.auth("root","root")       // changement d'identité
1
root@127.0.0.1:27017:admin>
```

Les clients graphiques

Compass

Compass est proposé sur le site officiel.

https://www.mongodb.com/download-center/compass

Mongobooster

https://nosqlbooster.com/

La version gratuite est bridée.

Mongochef

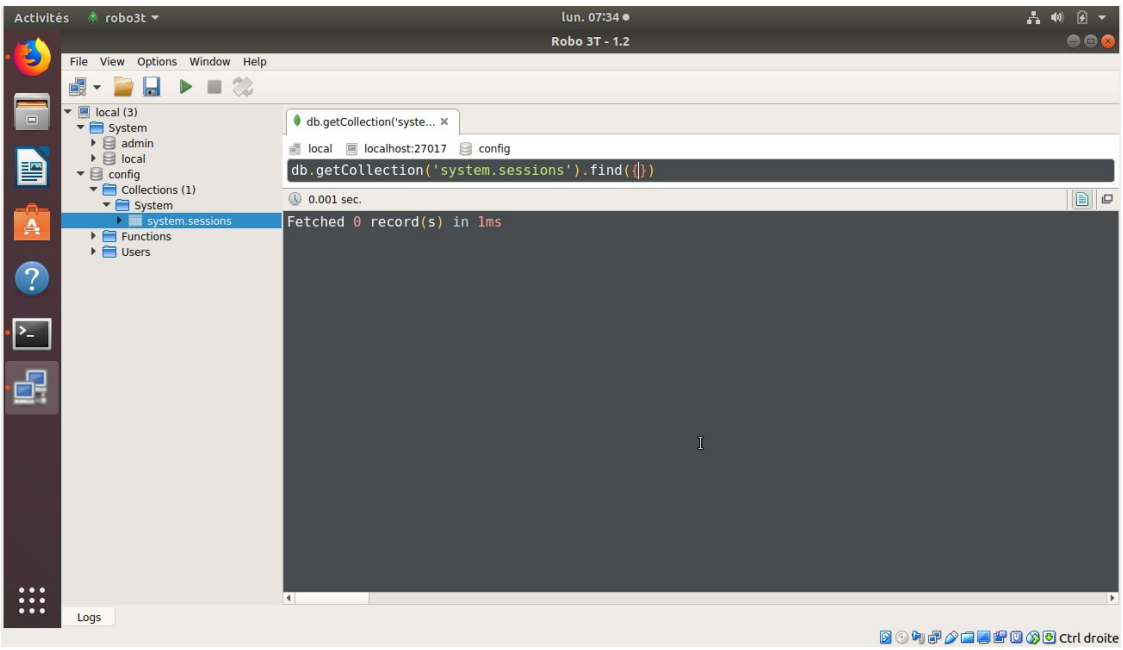
https://studio3t.com/

Version gratuite valide 30 jours

Robomongo

Rebaptisé Robo 3T, un produit libre et gratuit.

https://robomongo.org/download



Arrêt de MongoDB

En ligne de commandes

1) Utilisation de l'option shutdown de la commande mongod

```
$ mongod --dbpath <chemin_vers_datas> --shutdown
```

2) Lancé en avant plan, mongod peut être arrêté avec un simple CTRL-C, le signal est intercepté et l'arrêt est sécurisé.

3) Lancé en arrière plan le processus peut être tué par un 'kill <PID>' ou par un killall mongod.

Le numéro de PID peut être déterminé par la commande 'ps' ou peut être indentifié grâce au contenu du fichier /var/run/mongod/mongod.pid (ou du fichier <DBPATH>/mongod.lock).

Attention :

- Le kill envoie par défaut le signal TERM (kill -15 <PID>. Ne jamais utiliser un kill -9 !
- Le killall tue tous les processus, or il peut y avoir plusieurs mongod sur le même serveur

4) Par systemctl
Comme vu précédemment, 'systemctl stop mongod.service'

Par l'interface Mongo

Il faut des droits administrateur si l'authentification est activée.

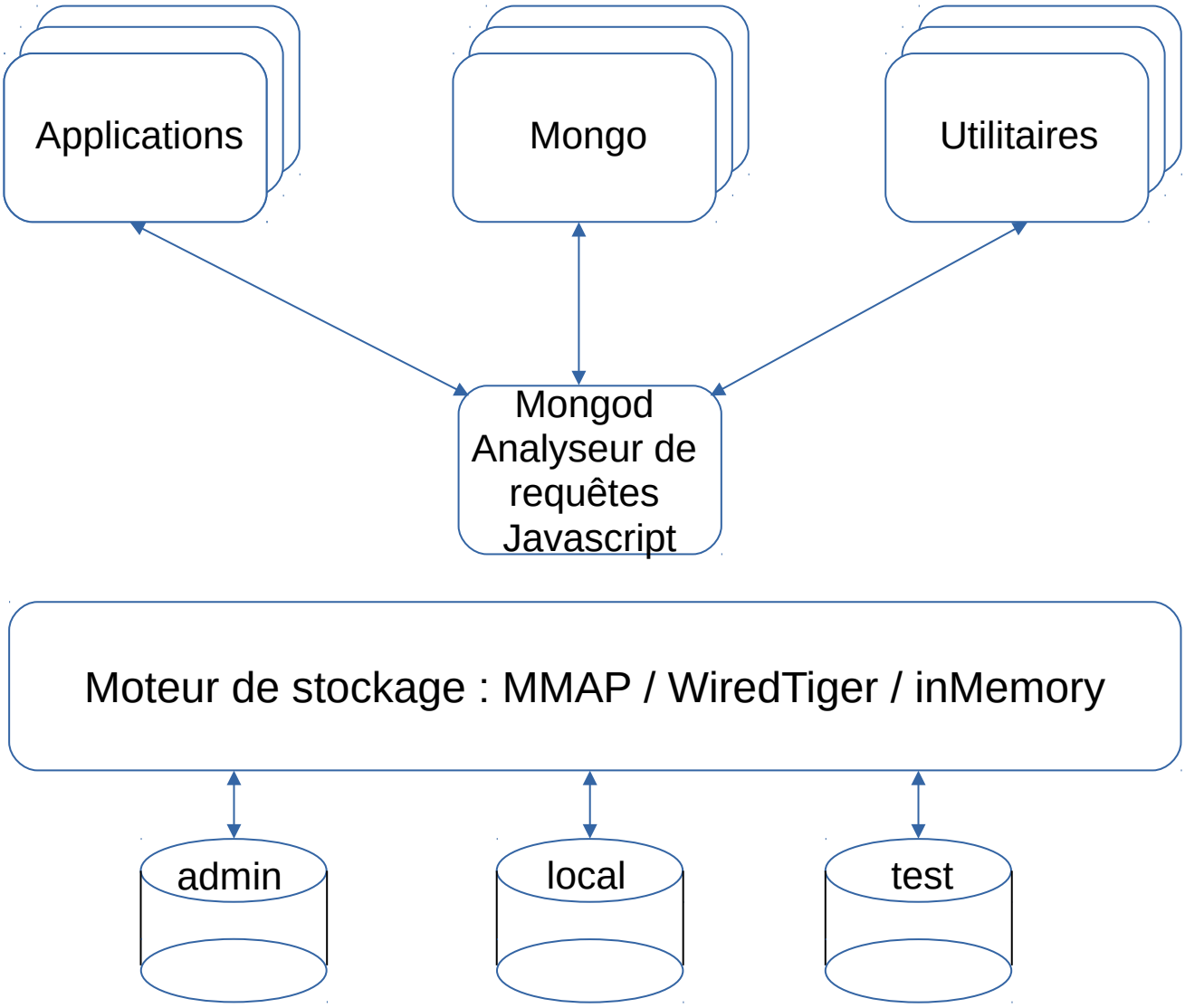
```
> use admin
switched to db admin
> db.shutdownServer()
server should be down...
2017-09-20T14:57:08.841+0200 I NETWORK  [thread1] trying reconnect to 127.0.0.1:27017 (127.0.0.1) failed
2017-09-20T14:57:08.841+0200 W NETWORK  [thread1] Failed to connect to 127.0.0.1:27017, in(checking socket for error after poll), reason: Connection refused
2017-09-20T14:57:08.841+0200 I NETWORK  [thread1] reconnect 127.0.0.1:27017 (127.0.0.1) failed failed
>
> // En cas d'urgence on peut forcer l'arrêt
>
> db.adminCommand({shutdown:1, force:true})
```

Éléments d'architecture

MongoDB peut gérer plusieurs bases : à l'initialisation, le moteur crée les bases admin et local. La base test est la base par défaut, vide.

Historiquement, il existait un moteur MMAP. Avec la version 3.0, est apparu le moteur WiredTiger, devenu le moteur par défaut en version 3.2 (Déc. 2015).
Déconseillé en version 4, le moteur MMAP ne sera plus supporté en 4.2. Le moteur inMemory est utile pour gérer les données en mémoire centrale.

A la différence de MySQL, le moteur utilisé est propre à une instance.



L’instance MongoDB

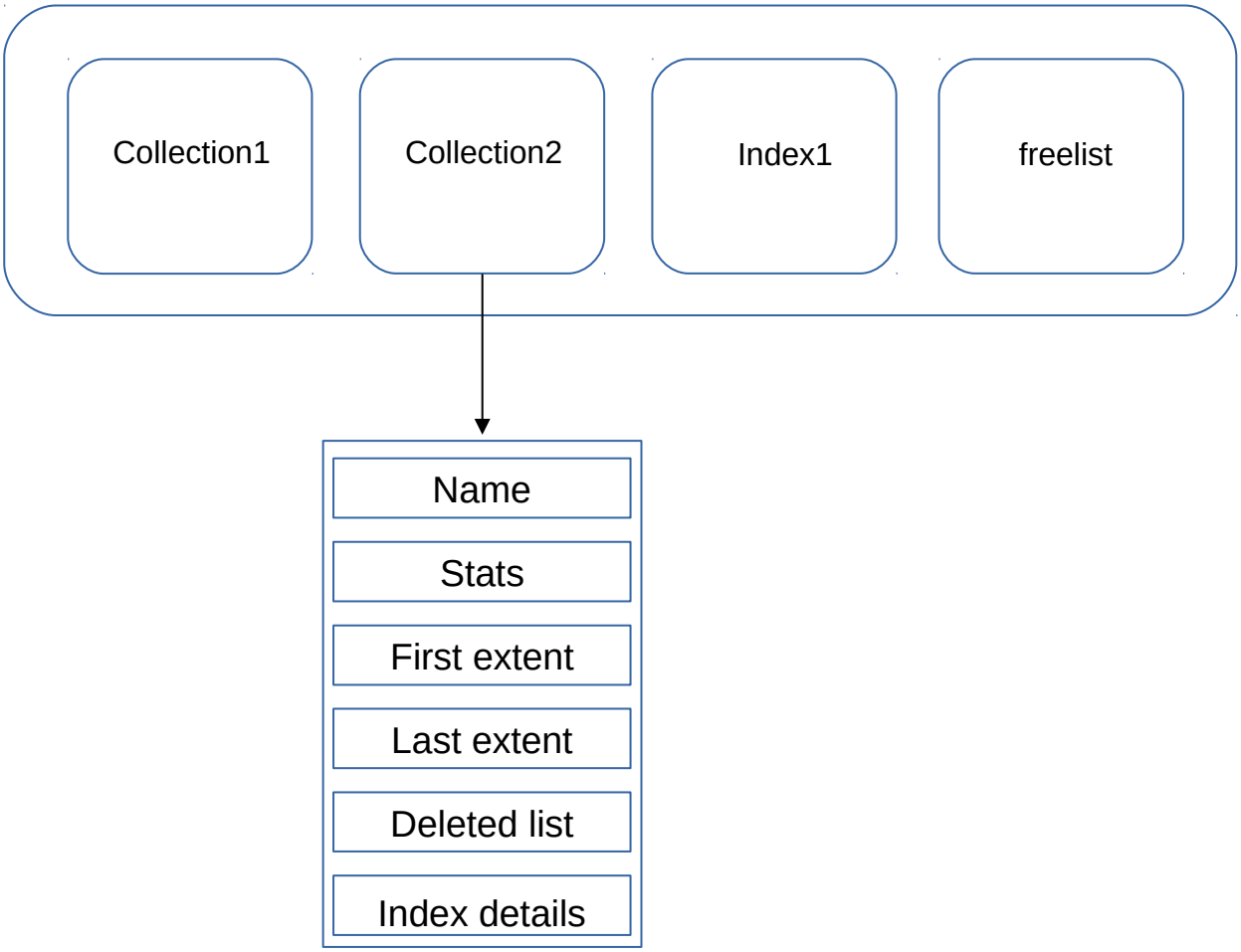
- Le processus serveur mongod,
- Le moteur de stockage choisi pour l’instance : MMAP/WiredTiger/InMemory
  (option --storageEngine en ligne de commandes,
  option engine: de la section storage dans le fichier /etc/mongod.conf)
- Le cache mémoire dédié sous wiredTiger,
- Le port d’écoute (27017 par défaut),
- Le répertoire de stockage des données (/data/db par défaut / option --dbpath),
- Le journal pour le recovery après un plantage du serveur,
- Un fichier de configuration (/etc/mongod.conf).

Par défaut connecté à la base 'test', on passe d’une base à l’autre avec la commande ‘use’.

Chaque objet de chaque base (collections, index) constitue un NameSpace. Les NameSpace sont alloués par extents.

L’objet est désigné par son namespace : 'base.collection'. Précédemment, on a créé le namespace 'mydb.xx'. Un namespace occupe en moyenne 700 octets.

Structure d’un namespace



Le moteur MMAP

Moteur original qui a connu plusieurs améliorations successives : performances, gestion de la journalisation, introduction de mécanismes de gestion de la concurrence d'accès au niveau collection…. Ce n'est pourtant plus le moteur par défaut.

Le moteur MMAP connait un certain nombre de limites relatives aux namespaces :
un fichier .ns par base de données (ci dessous, le répertoire par défaut d'un mongod/MMAP)

```
[root@grouik VMs]# ls -l /opt/mongo-mmap/
total 163860
-rw-------. 1    root root  67108864 21 sept. 15:25 admin.0
-rw-------. 1    root root  16777216 21 sept. 15:25 admin.ns
drwxr-xr-x. 2 root root        4096 21 sept. 15:32 diagnostic.data
drwxr-xr-x. 2 root root        4096 21 sept. 15:25 journal
-rw-------. 1    root root  67108864 21 sept. 15:25 local.0
-rw-------. 1    root root  16777216 21 sept. 15:25 local.ns
-rw-r--r--. 1    root root           5 21 sept. 15:25 mongod.lock
-rw-r--r--. 1    root root          69 21 sept. 15:25 storage.bson
drwxr-xr-x. 2 root root        4096 21 sept. 15:25 _tmp
```

On constate 2 premiers fichiers de datas par base, admin.0 et local.0 de 64M chacun + 2 fichiers de méta datas, admin.ns et local.ns de 16M chacun. On créé une collection dans la base test :

```
[root@grouik VMs]# mongo --quiet
> show dbs
admin  0.078GB
local  0.078GB
>
> db
test
>
> db.xx.insert({xx:"xx"})
WriteResult({ "nInserted" : 1 })
>
> show dbs
admin  0.078GB
local  0.078GB
test   0.078GB
> quit()
```

```
[root@grouik VMs]# ls -l /opt/mongo-mmap/
total 245780
-rw-------.  1 root root 67108864 21 sept. 15:25 admin.0
-rw-------.  1 root root 16777216 21 sept. 15:25 admin.ns
drwxr-xr-x. 2 root root        4096 21 sept. 15:33 diagnostic.data
drwxr-xr-x. 2 root root        4096 21 sept. 15:25 journal
-rw-------.  1 root root 67108864 21 sept. 15:25 local.0
-rw-------.  1 root root 16777216 21 sept. 15:25 local.ns
-rw-r--r--.  1 root root           5 21 sept. 15:25 mongod.lock
-rw-r--r--.  1 root root          69 21 sept. 15:25 storage.bson
-rw-------.  1 root root 67108864 21 sept. 15:33 test.0
-rw-------.  1 root root 16777216 21 sept. 15:33 test.ns
drwxr-xr-x. 2 root root        4096 21 sept. 15:33 _tmp
[root@grouik VMs]#
```

On constate la création des fichiers test.0 et test.ns. Les fichiers .ns sont par défaut limités à 16M (ce qui fait à peu près 24000 collections/index par base). Ces fichiers contiennent des méta données relatives aux collections. Les données sont contenues dans les fichiers test.0, puis test.1, test.2, test.3 etc….

L'utilisation d'une boucle JavaScript va créer 50000 collections avec 50000 inserts d'une chaîne de caractères.

On redémarre le serveur avec l'option --nssize afin d'augmenter la taille du fichier .ns (2Go maximum). La base peut être redimensionnée (ici * 4) en redémarrant mongod avec l'option --nssize 64 et en exécutant db.repairDatabase().

```
mongod --dbpath /opt/mongo-mmap –shutdown                                    // arrêt de l'instance
mongod --dbpath /opt/mongo-mmap --logpath /tmp/mongo --fork --storageEngine mmapv1 --port 27018 --nssize 64
```

```
[pierrick@grouik VMs]$ mongo --port 27018 --quiet
> show dbs
admin  0.078GB
local  0.078GB
test   1.953GB
> db.repairDatabase()
{ "ok" : 1 }
>
```

A l'issue de l'opération, le répertoire /opt/mongo-mmap contient

```
-rw-------.  1 root root   67108864 21 sept. 15:25 admin.0
-rw-------.  1 root root   16777216 21 sept. 15:25 admin.ns
drwxr-xr-x. 2 root root       4096 21 sept. 17:01 diagnostic.data
drwxr-xr-x. 2 root root       4096 21 sept. 17:00 journal
-rw-------.  1 root root   67108864 21 sept. 16:41 local.0
-rw-------.  1 root root   16777216 21 sept. 16:41 local.ns
-rw-r--r--.  1 root root          5 21 sept. 16:41 mongod.lock
-rw-r--r--.  1 root root         69 21 sept. 15:25 storage.bson
drwxr-xr-x. 2 root root       4096 21 sept. 16:18 test
-rw-------.  1 root root   67108864 21 sept. 16:59 test.0
-rw-------.  1 root root  134217728 21 sept. 16:59 test.1
-rw-------.  1 root root  268435456 21 sept. 16:59 test.2
-rw-------.  1 root root  536870912 21 sept. 16:59 test.3
-rw-------.  1 root root 1073741824 21 sept. 16:59 test.4
-rw-------.  1 root root   67108864 21 sept. 16:59 test.ns
```

On constate que le fichier test.ns est passé à 64M. Les fichiers de données test.0 à test.4 ont été créés avec des tailles de 64M à 1Go : La règle est le doublement de taille à chaque nouveau fichier créé jusqu'à 2Go. Au delà des 2Go, chaque nouveau fichier de données sera initialisé à 2Go.

Le moteur wiredTiger

Le moteur WiredTiger est un produit indépendant, open source.

Créé en 2010 par deux anciens ingénieurs de BerkeleyDB, il est intégrable à MySQL, par exemple. Apprécié pour ses performances sur les gros volumes.

Intégré à MongoDB en 2013 suite au rachat de la société par MongoDB Inc.

WiredTiger gère la concurrence d'accès au niveau document ainsi que le MVCC (Multi Version Concurency Control)

La compression des données est activée par défaut. Elle s'effectue à la volée selon les technologies snappy (collections) ou prefix (index).

La journalisation est activée et les fichiers de journalisation sont aussi compressés.

Il dispose de fonctions de cryptage en version entreprise.

Répertoire WiredTiger :

```
bash-4.3$ ls -l /opt/mongod
Total 64
-rw-rw-r--.   1 mongod mongod 4096 22 sept. 18:30 collection-0--3440689741430222131.wt
-rw-rw-r--.   1 mongod mongod 4096 22 sept. 18:30 collection-2--3440689741430222131.wt
drwxrwxr-x. 2 mongod mongod 4096 22 sept. 18:30 diagnostic.data
-rw-rw-r--.   1 mongod mongod 4096 22 sept. 18:30 index-1--3440689741430222131.wt
-rw-rw-r--.   1 mongod mongod 4096 22 sept. 18:30 index-3--3440689741430222131.wt
-rw-rw-r--.   1 mongod mongod 4096 22 sept. 18:30 index-4--3440689741430222131.wt
drwxrwxr-x. 2 mongod mongod 4096 22 sept. 18:30 journal
-rw-rw-r--.   1 mongod mongod 4096 22 sept. 18:30 _mdb_catalog.wt
-rw-r--r--.    1 mongod mongod    5 22 sept. 18:30 mongod.lock
-rw-rw-r--   . 1 mongod mongod 4096 22 sept. 18:30 sizeStorer.wt
-rw-rw-r--   . 1 mongod mongod   95 22 sept. 18:30 storage.bson
-rw-rw-r--   . 1 mongod mongod   49 22 sept. 18:30 WiredTiger
-rw-rw-r--   . 1 mongod mongod 4096 22 sept. 18:30 WiredTigerLAS.wt
-rw-rw-r--   . 1 mongod mongod   21 22 sept. 18:30 WiredTiger.lock
-rw-rw-r--   . 1 mongod mongod  842 22 sept. 18:30 WiredTiger.turtle
-rw-rw-r--   . 1 mongod mongod 4096 22 sept. 18:30 WiredTiger.wt
bash-4.3$
```

Fichiers et répertoires du dbpath :

diagnostic.data	existe sous MMAP comme sous WiredTiger statistiques du serveur destinées à un service support métriques horodatées 'metrics.YYYY.MM.DDTHH-MM-SSZ-XXXXX'
journal	Fichiers journaux WiredTigerLog.XXXXXXXXXX WiredTigerPrelog.XXXXXXXXXX
_mdb_catalog.wt	Liaison entre les méta datas (namespaces) et les fichiers de données, collections et index
collection-X—XXXXXXXXXXXXXXXXXXX.wt **index-X—XXXXXXXXXXXXXXXXXXX.wt**	Une collection Un index
mongod.lock	Verrou. Contient le PID de mongod
storage.bson	Meta-données du serveur
wiredTiger	Fichier de version
wiredTiger.lock	Fichier verrou
wiredTiger.turtle	Fichier de configuration
Autres : sizeStorer.wt, wiredTiger.wt, wiredTigerLAS.wt	Fichiers propres à WiredTiger

Les caches mémoire

MongoDB utilise le cache mémoire de l'OS.

La commande linux 'free', ci dessous retourne qu'1,5Go de données lues sur le disque sont présentes dans le cache OS:

```
[pierrick@grouik VMs]$ free
              total        used        free      shared  buff/cache   available
Mem:       16371776     3114228    11684780      108336     1572768    13359360
Swap:      16383996           0    16383996
[pierrick@grouik VMs]$
```

Potentiellement, toute la mémoire disponible (available) peut être utilisée par le cache OS.
A l'instar d'autres SGBD (Postgres…), MongoDB utilise cette technique du Memory Map Files.
A intervalles réguliers, l'OS provoque un sync du cache.

Le moteur WiredTiger possède son propre cache : par défaut, en version 3.4, WiredTiger initialise son cache à min((50 % de la RAM – 1Go) : 256Mb)

Paramètre mongod.conf	Paramètre ligne de commandes	Commentaire
wiredTiger.engineConfig.cacheSizeGB	--wiredTigerCacheSizeGB	taille du cache. type float. de 256Mo à 10To

Objectifs

Les bases
Les collections
Les documents
Lister, insérer, modifier et supprimer des documents
GridFs

Les bases de données

Une base de données MongoDB ne peut être gérée que par une instance MongoDB.
A l'initialisation, les deux bases admin et local sont créées avec leurs propres collections :

```
[pierrick@grouik VMs]$ mongo --quiet
>
> show databases
admin  0.000GB
local  0.000GB
>
> use admin
switched to db admin
>
> show collections
system.version
>
> db.system.version.find()
{ "_id" : "featureCompatibilityVersion", "version" : "3.4" }
>
> use local
switched to db local
>
> show collections
startup_log
>
> // la collection startup_log contient les paramètres de démarrage dans un format ± lisible.....
```

TP : Importer les fichiers 'communes.csv' et 'insee.csv' de la base communes :

```
-bash-4.3$  mongoimport -d communes -c communes --file communes.csv --type csv --headerline
2017-09-25T08:40:56.311+0200    connected to: localhost
2017-09-25T08:40:57.049+0200    imported 36595 documents
-bash-4.3$
-bash-4.3$  mongoimport -d communes -c insee --file insee.csv --type csv --headerline
2017-09-25T08:41:43.014+0200    connected to: localhost
2017-09-25T08:41:43.533+0200    imported 36742 documents
-bash-4.3$
```

Les 'mongoimport' ont pour effet de créer la base communes et les 2 collections :

```
> use communes
switched to db communes
>
> show collections
communes
insee
>
> db.communes.find().limit(2)
{ "_id" : ObjectId("59c8a4f84ade6b6c7aae4d0f"), "ID_GEOFLA" : "COMMUNE00000000000000002", "CODE_COM" : 33, "INSEE_COM" : 47033, "NOM_COM" : "BOUDY-DE-BEAUREGARD", "STATUT" : "Commune simple", "X_CHF_LIEU" : 516424, "Y_CHF_LIEU" : 6384852, "X_CENTROID" : 515575, "Y_CENTROID" : 6385938, "Z_MOYEN" : 112, "SUPERFICIE" : 1019, "POPULATION" : 406, "CODE_CANT" : 6, "CODE_ARR" : 3, "CODE_DEPT" : 47, "NOM_DEPT" : "LOT-ET-GARONNE", "CODE_REG" : 72, "NOM_REG" : "AQUITAINE" }
{ "_id" : ObjectId("59c8a4f84ade6b6c7aae4d10"), "ID_GEOFLA" : "COMMUNE00000000000000003", "CODE_COM" : 9, "INSEE_COM" : 32009, "NOM_COM" : "ARMOUS-ET-CAU", "STATUT" : "Commune simple", "X_CHF_LIEU" : 472979, "Y_CHF_LIEU" : 6278963, "X_CENTROID" : 473004, "Y_CENTROID" : 6278937, "Z_MOYEN" : 217, "SUPERFICIE" : 932, "POPULATION" : 95, "CODE_CANT" : 20, "CODE_ARR" : 3, "CODE_DEPT" : 32, "NOM_DEPT" : "GERS", "CODE_REG" : 73, "NOM_REG" : "MIDI-PYRENEES" }
>
> db.insee.find().limit(2)
{ "_id" : ObjectId("59c8a5274ade6b6c7aaedc2b"), "CodeINSEE" : 35259, "CodePostal" : 35120, "Commune" : "SAINT-BROLADRE" }
{ "_id" : ObjectId("59c8a5274ade6b6c7aaedc2c"), "CodeINSEE" : 66175, "CodePostal" : 66740, "Commune" : "SAINT-GENIS-DES-FONTAINES" }
>
```

Pour le stockage, on a utilisé les options 'directoryPerDB' et 'directoryForIndexes' (option propre au moteur wiredTiger) du fichier de config :

```
# storage Options - How and Where to store data
storage:
  # Directory for datafiles (defaults to /data/db/)
  dbPath: /opt/mongod
  directoryPerDB: true
../..
  wiredTiger:
    engineConfig:
      # The maximum size of the cache that WiredTiger will use for all data
      # (max(60% of RAM - 1GB, 1GB) by default)
      cacheSizeGB: 6
      directoryForIndexes: true
```

On a donc un répertoire 'communes' contenant les 2 sous-répertoires 'collection' et 'index'. Les fichiers .wt correspondent aux 2 collections et aux 2 indexes qui ont été automatiquement créés sur les champs '_id' des 2 collections :

```
[root@grouik pierrick]# ls -l /opt/mongod
total 220
drwxr-xr-x. 4 mongod mongod   4096 23 sept. 12:40 admin
drwxr-xr-x. 4 mongod mongod   4096 25 sept. 08:39 communes
drwxr-xr-x. 2 mongod mongod   4096 25 sept. 09:04 diagnostic.data
drwxr-xr-x. 2 mongod mongod   4096 25 sept. 06:19 journal
drwxr-xr-x. 4 mongod mongod   4096 23 sept. 12:40 local
-rw-r--r--.   1 mongod mongod 36864 25 sept. 08:42 _mdb_catalog.wt
-rw-r--r--.   1 mongod mongod     5 25 sept. 06:18 mongod.lock
-rw-r--r--.   1 mongod mongod 36864 25 sept. 08:42 sizeStorer.wt
-rw-r--r--.   1 mongod mongod    95 23 sept. 12:40 storage.bson
drwxr-xr-x. 4 mongod mongod   4096 24 sept. 13:36 test
-rw-r--r--.   1 mongod mongod    49 23 sept. 12:40 WiredTiger
-rw-r--r--.   1 mongod mongod   4096 25 sept. 06:18 WiredTigerLAS.wt
-rw-r--r--.   1 mongod mongod    21 23 sept. 12:40 WiredTiger.lock
-rw-r--r--.   1 mongod mongod  1003 25 sept. 08:43 WiredTiger.turtle
-rw-r--r--.   1 mongod mongod 94208 25 sept. 08:43 WiredTiger.wt
drwxr-xr-x. 4 mongod mongod   4096 23 sept. 16:02 yougo
[root@grouik pierrick]#
[root@grouik pierrick]# ls -l /opt/mongod/communes
total 8
drwxr-xr-x. 2 mongod mongod 4096 23 sept. 16:56 collection
drwxr-xr-x. 2 mongod mongod 4096 23 sept. 16:56 index
[root@grouik pierrick]#
[root@grouik pierrick]# ls -l /opt/mongod/communes/collection
total 36
-rw-r--r--. 1 mongod mongod 16384 23 sept. 16:56 13--4487126631861265603.wt
-rw-r--r--. 1 mongod mongod 20480 23 sept. 16:56 15--4487126631861265603.wt
[root@grouik pierrick]#
[root@grouik pierrick]# ls -l /opt/mongod/communes/index
total 32
-rw-r--r--. 1 mongod mongod 16384 23 sept. 16:56 14--4487126631861265603.wt
-rw-r--r--. 1 mongod mongod 16384 23 sept. 16:56 16--4487126631861265603.wt
[root@grouik pierrick]#
```

La création des bases et collections se fait donc à la volée. Pour supprimer la base mydb :

```
> db
mydb
>
> // on doit être positionné dans la base pour pouvoir la détruire
>
> db.dropDatabase()
{ "dropped" : "mydb", "ok" : 1 }
>
> // alternativement, on peut utiliser la syntaxe :
> db.runCommand( {dropDatabase : 1} )
```

La commande dropDatabase() pose un verrou exclusif global le temps de l'opération.
Les fichiers de données sont supprimés du dbpath.

La connexion à la base est maintenue, bien que la base soit supprimée. (simple connexion logique)

```
> db
mydb
>
> show dbs
admin  0.000GB
local  0.000GB
>
```

Attention, la commande dropDatabase() ne supprime pas les utilisateurs.
Pour cela il faut utiliser la commande dropAllUsersFromDatabase, réservée aux utilisateurs possédant des droits 'admin'

```
> use mydb
> db.runCommand( { dropAllUsersFromDatabase: 1 } )
```

Les collections

Opérations basiques :

```
> show collections
communes
insee
>
> // création d'une collection vide et sans structure
> db.createCollection("xx")
{ "ok" : 1 }
>
> // insertion d'un premier document
> db.xx.insert( {xx:"xx"} )
WriteResult({ "nInserted" : 1 })
>
> // affichage du contenu
> db.xx.find()
{ "_id" : ObjectId("59c6986415fa7643d190f9bd"), "xx" : "xx" }
>
> // le champ _id est automatiquement généré et indexé par MongoDB
>
> // la collection communes contient + de 35000 documents. si on fait un db.communes.find()  mongo affiche les
> // documents 20 par 20
> // on passe aux suivants en tapant it (iterate). on peut chaîner le find() avec un limit(x)
>
> // la collection xx peut être détruite
> db.xx.drop()
true
>
```

L'intérêt du createCollection est de pouvoir préciser des options :

capped	<boolean>	collection 'encadrée' true pour activer une collection cappée (*). Nécessite un champ size ou max.
size	<number>	taille maximum en octet pour les collections cappées.
max	<number>	nombre maximum de documents dans la collection cappée. la taille est prioritaire.
autoIndexId	<boolean>	si false, désactive la création automatique d'un index sur le champ _id. plus supporté en 4.0.
Validator	<document>	Document précisant les regles de validation.
validationLevel	"off/strict/moderate"	Validation pour les insert/update Si 'moderate' pas de vérification pour les documents exitants
validationAction	"error/warn"	Si error, l'insertion échoue Si warn, signalisation dans les logs.

(*) Cappé s'emploie par exemple en finance. Un prêt cappé : le taux d'intérêt oscille entre x et y.

Collection cappée

```
> db.createCollection("xx", {capped: true, max: 3, size: 1000})
{ "ok" : 1 }
> db.xx.insert( {doc: "doc1"})
WriteResult({ "nInserted" : 1 })
>
> db.xx.insert( {doc: "doc2"})
WriteResult({ "nInserted" : 1 })
>
> db.xx.insert( {doc: "doc3"})
WriteResult({ "nInserted" : 1 })
>
> db.xx.find()
{ "_id" : ObjectId("59c6c16e15fa7643d190f9be"), "doc" : "doc1" }
{ "_id" : ObjectId("59c6c17215fa7643d190f9bf"), "doc" : "doc2" }
{ "_id" : ObjectId("59c6c17515fa7643d190f9c0"), "doc" : "doc3" }
>
> db.xx.insert( {doc: "doc4"})
WriteResult({ "nInserted" : 1 })
>
> db.xx.find()
{ "_id" : ObjectId("59c6c17215fa7643d190f9bf"), "doc" : "doc2" }
{ "_id" : ObjectId("59c6c17515fa7643d190f9c0"), "doc" : "doc3" }
{ "_id" : ObjectId("59c6c18115fa7643d190f9c1"), "doc" : "doc4" }
>
```

Cette fonctionnalité permet de maîtriser la volumétrie. On verra, au chapitre sur les index, qu'il existe aussi des index 'Time to Live'. Ces index éliminent les données au fur et à mesure de leur vieillissement.

Autoindex

Jusqu'en 4.0 : True par défaut. Passer à false est utile si on dispose par ailleurs d'une clé primaire.

Note : l'_id contenant le timestamp de création des documents, l'index sur l'_id permet de les trouver par date de création,

```
> db.createCollection("xx", {autoIndexId:false} )
{
        "note" : "the autoIndexId option is deprecated and will be removed in a future release",
        "ok" : 1
}
> db.xx.insert({_id:"",ss:"1750544123123",nom:"dupont"})
WriteResult({ "nInserted" : 1 })
>
> db.xx.insert({_id:"",ss:"2750544456456",nom:"durand"})
WriteResult({ "nInserted" : 1 })
>
> db.xx.find()
{ "_id" : "", "ss" : "1750544123123", "nom" : "dupont" }
{ "_id" : "", "ss" : "2750544456456", "nom" : "durand" }
>
```

Dans cette hypothèse, on créé un index unique sur le champ "ss", et on force l'id à "".

Fonctionnalité déconseillée en version 3 et supprimée en 4.

Validator

Un des avantages des bases de données relationnelles est la prédictibilité des données qui 'collent' aux structures des tables. Dans Mongo, on perd cette garantie. Les documents de validation permettent de contrôler la présence de champs, leur type, leur forme (avec une expression régulière)...

La validation peut être 'strict' (validation à chaque insert/update + existant) ou 'moderate' et générer une erreur (et un rejet) ou un simple avertissement. (voir exemple plus loin).

Id d'un document

Valeur unique d'identification d'un document, sur 12 octets, de type ObjectId.

```
> db.xx.insert( { x: ObjectId() })
WriteResult({ "nInserted" : 1 })
>
> db.xx.find()
{ "_id" : ObjectId("59c7ae83bc30888a2ba6e25c"), "x" : ObjectId("59c7ae83bc30888a2ba6e25b") }
>
```

MongoDb 'calcule' l'_id :

- les 4 premiers octets se composent du timestamp courant
  (nbre de secondes depuis l'unix epoch, 01/01/1970).
- 3 octets dérivés de l'adresse MAC de la machine
- 2 octets identifiant le numéro de processus
- les 3 derniers octets constituent un compteur.

L'id est donc unique au sein d'une instance MongoDb, tous objets confondus. (aucune référence aux notions de bases/collections)

On dispose d'un index sur la date de création du document.

Un document peut stocker l'_id d'un autre document. Charge à l'application de réaliser la jointure.

Un document peut s'assurer de l'existence d'un autre document dans sa clause 'validator' (embryon d'intégrité référentielle)

Les documents

Un document JSON est une liste de couples champ:"valeur" :

{ champ1 : "valeur1", champ2 : "valeur2", ..., champN : "valeurN" }

Les nom de champs ne peuvent commencer par le caractère "$", ni contenir le caractère ".", ni avoir la valeur "null". Les valeurs sont typées.

Types de données simples

type	commentaire	exemple
null	représente une valeur nulle	{ x : null }
boolean	true / false	{ x : true }
numeric	entiers / flottants. Flottant (double) par défaut. Les entiers doivent être explicitement précisées.	{ x : NumberInt(3), pi : 3.14 }
string		{ x : "Hello world !" }
date	dates stockées sur des entiers à 64 bits. dates unix : nombre de millisecondes depuis le 01/01/1970. soit : ± 292 millions d'années avant ou après. le fuseau horaire n'est pas stocké.	{ x : new Date() }

Autres types de données

type	commentaire	exemple
expressions régulières	Les expressions régulières sont possibles, par exemple pour filtrer la méthode find()	{ x : /test/i }
tableaux	Liste de valeurs entre crochets, de types élémentaires ou complexes, éventuellement hétérogènes	{ x : ["a", "b", "c"] } { y : [123, "abcd", true, null] } { z : [123, ["abc", "def"], 456]}
documents imbriqués	Un document peut contenir d'autres documents.	{ x : { y : "z"} } { ventes : { qte : 5, art : { code : 123, art : "un arcticle", px : 100 } } }

Le format JSON

JSON (JavaScript Object Notation) est définit par la RFC 4627.
Documenté sur le site http://www.json.org/json-fr.html
L'information, textuelle, est structurée et est beaucoup plus souple qu'XML.

Manipulable comme un objet par tout langage de programmation, ce format permet une écriture souple et une lecture simple des données traitées et autorise plusieurs présentations de la même information.

Exemple :

```
{
prenom : "Jean",
nom :      "Dubois",
    adresse : {
        numero :        "5",
        voie :          "Place des Fêtes",
        codepostal :    "75019",
        ville :         "Paris"
        },
age : 55
}
```

Alternativement :

```
{
   identite : { prenom : "Jean", nom : "Dubois", age : 55 },
   adresse : { numero :  "5",  voie : "Place des Fêtes", codepostal :    "75019", ville : "Paris" }
}
```

Le format BSON

BSON (Binary JSON) est la représentation interne de MongoDB enrichie par exemple des expressions régulières où de blocs de code JavaScript.

- Efficacité :
  représentation des données minimisant l'espace.
- Parcours (traversability) :
  BSON sacrifie de l'espace afin de faciliter le traitement du format de données.
  Ex : les chaînes de caractères sont préfixées par leur taille.
- Performance :
  BSON est conçu pour être codé et décodé à la volée. Il utilise les représentations C pour les types, compatibles avec la majorité des langages de programmation modernes.

La méthode find()

db.<NomCollection>.find()	retourne tous les documents d'une collection.
db.<NomCollection>.find().pretty()	améliore la lisibilité.
db.<NomCollection>.find().linit(x)	limite le nombre de documents retournés (Il existe aussi la méthode findOne() qui retourne le premier document satisfaisant.
db.<NomCollection>.find().skip(y)	saute les y premiers documents

db.<NomCollection>.findOne()	trouve le premier document satisfaisant.
db.<NomCollection>.count()	compte les enregistrements
db.<NomCollection>.sort(<Critère>)	Trie les enregistrements selon le critère précisé.

Exemple :

```
> db.xx.insert(
                {prenom : "Jean", nom : "Dubois",
                adresse : {
                            numero : "5",  voie : "Place des Fêtes",
                            codepostal : "75019",  ville : "Paris"
                          },
                age : 55 }
)
WriteResult({ "nInserted" : 1 })
>
```

```
> db.xx.find()
{ "_id" : ObjectId("5c7a5edbf469bbe452b5c1e8"), "prenom" : "Jean", "nom" : "Dubois", "adresse" : { "numero" : "5",
"voie" : "Place des Fêtes", "codepostal" : "75019", "ville" : "Paris" }, "age" : 55 }
>
> db.xx.find().pretty()
{
        "_id" : ObjectId("5c7a5edbf469bbe452b5c1e8"),
        "prenom" : "Jean",
        "nom" : "Dubois",
        "adresse" : {
                "numero" : "5",
                "voie" : "Place des Fêtes",
                "codepostal" : "75019",
                "ville" : "Paris"
        },
        "age" : 55
}
```

La methode find() est à priori mono-collection. Toutefois, depuis la version 3.2, il existe l'opérateur $lookup qui permet de faire des jointures. (Voir framework aggregate).

find() accepte une partie projection : selection des champs interessant l'application. Par ailleurs, possibilité de selectionner les documents (query) et enfin possiblité de trier les resultats.

La syntaxe complète est **db.collection.find(query, projection)**

La clause 'projection' définit la liste des champs qu'on retient sous la forme

{ champ1: booléenne , champ2: booléenne }

La booléenne peut prendre les valeurs suivantes:

1 ou true pour inclure le champ dans le résultat.
0 ou false pour exclure le champ.

La méthode inclura toujours le champ '_id'. Pour l'exclure, il faut l'indiquer explicitement :

```
> db.communes.find( { NOM_COM : "CHELLES" }, { NOM_COM : 1, NOM_DEPT : 1, _id : 0 }  )
{ "NOM_COM" : "CHELLES", "NOM_DEPT" : "OISE" }
{ "NOM_COM" : "CHELLES", "NOM_DEPT" : "SEINE-ET-MARNE" }
>
```

Ici, la partie 'query' filtre la recherche sur les seules villes s'appelant 'CHELLES' et la projection demande à ne retenir que les noms de la commune et du département.

Le document 'query' { NOM_COM : "CHELLES" } pose implicitement une condition d'égalité.

D'autre opérateurs de comparaison existent : $eq, $gt, $gte, $lt, $lte, $ne, $in, $nin
Ces opérateurs peuvent être composés avec les opérateurs logiques $and, $or, $nor, $not...

Une originalité de MongoDB est que les opérateurs logiques s'utilisent en notation préfixée :

```
> db.communes.find( { $and : [ { POPULATION : { $lt : 1000 } },{ NOM_COM : { $regex : /^C/ } } ] } ,
{ "NOM_COM" : 1, "POPULATION" : 1} ).sort( {"NOM_COM" : -1}).limit(5);
{ "_id" : ObjectId("59c8a4f94ade6b6c7aaeced5"), "NOM_COM" : "CYS-LA-COMMUNE", "POPULATION" : 154 }
{ "_id" : ObjectId("59c8a4f84ade6b6c7aae99ea"), "NOM_COM" : "CUZY", "POPULATION" : 144 }
{ "_id" : ObjectId("59c8a4f84ade6b6c7aae7f9d"), "NOM_COM" : "CUZORN", "POPULATION" : 873 }
{ "_id" : ObjectId("59c8a4f84ade6b6c7aae96ca"), "NOM_COM" : "CUZION", "POPULATION" : 443 }
{ "_id" : ObjectId("59c8a4f84ade6b6c7aaeadfd"), "NOM_COM" : "CUZIEU", "POPULATION" : 410 }
>
```

Lecture :

1) { POPULATION : { $lt : 1000 } } // La commune a moins de 1000 habitants
{ NOM_COM : { $regex : /^C/ } } // Le nom de la commune commence par 'C'

2) { $and : [{ POPULATION : { $lt : 1000 } }, { NOM_COM : { $regex : /^C/ } }] }
Le $and préfixé opère sur un tableau [] des 2 conditions qui doivent être réunies.

Les opérateurs de comparaison

$eq	opérateur testant l'égalité, l'équivalence est { champ : valeur }
$gt	supérieur à
$gte	supérieur ou égal à une valeur spécifique
$lt	inférieur à une valeur spécifique
$lte	inférieur ou égal à une valeur spécifique
$ne	différent
$in	appartient à la liste de valeurs en argument
$nin	n'appartient pas à la liste de valeurs en argument

Les opérateurs logiques

$or	combinaison d'expressions avec la clause OR. Retourne 'true' (vrai) si une des conditions est validée
$and	combinaison d'expressions avec la clause AND Retourne 'true' (vrai) si toutes les conditions sont validées
$not	inverse les effets d'une condition et renvoie les documents qui ne correspondent pas à la condition
$nor	combinaison d'expressions avec la clause NOR (not or) Renvoie les documents si toutes les conditions ont échouées

Autres opérateurs

$regex	sélectionne les documents qui correspondent à une expression régulière spécifique
$exists	Sélectionne les documents en fonction de l'existence d'une information

Les expression régulières

Syntaxe

{ <champ>: { $regex: /pattern/, $options: '<options>' } }
{ <champ>: { $regex: 'pattern', $options: '<options>' } }
{ <champ>: { $regex: /pattern/<options> } } // cette syntaxe est la plus fréquente

Exemples

/XXX/ : contient l'expression XXX
/^XXX/ : commence par l'expression XXX
/XX*Y/ : contient XX , puis Y à une position quelconque

Options :

m : prend en compte les sauts de lignes pour les débuts et fin de ligne dans un document (\n)
i : compare en minuscule ou majuscule les caractères

L'opérateur $exists

Syntaxe
{ champ: { $exists: <boolean> } }

```
> db.Tests.find({})
{ "_id" : ObjectId("55e56fecb3bac13cb7a6d567"), x  : "a" }
{ "_id" : ObjectId("55e56fecb3bac13cb7a6d568"), w : "c" }
{ "_id" : ObjectId("55e56fecb3bac13cb7a6d569"), z  : "g" }
> db.Tests.find({"w" : {$exists:true} })
{ "_id" : ObjectId("55e56fecb3bac13cb7a6d568"), w : "c" }
> db.Tests.find({"w" : {$exists:false} })
{ "_id" : ObjectId("55e56fecb3bac13cb7a6d567"), x : "a" }
{ "_id" : ObjectId("55e56fecb3bac13cb7a6d569"), z : "g" }
```

La méthode count()

Retourne le nombre de documents extraits. Plus court : db.communes.count({../..})

```
> db.communes.find( {NOM_COM: 'CHELLES'},{} ).count()
2
>
```

La méthode sort()

Trie les documents selon le critère passé en paramètre. Tri ascendant ou descendant

```
> // tri ascendant
>
> db.communes.find( {NOM_COM: 'CHELLES'},{NOM_COM: 1, POPULATION: 1, _id: 0} ).sort({POPULATION:1})
{ "NOM_COM" : "CHELLES", "POPULATION" : 469}
{ "NOM_COM" : "CHELLES", "POPULATION" : 52817}
>
> // tri descendant
>
> db.communes.find( {NOM_COM: 'CHELLES'},{NOM_COM: 1,  POPULATION: 1, _id: 0} ).sort({POPULATION:-1})
{ "NOM_COM" : "CHELLES", "POPULATION" : 52817}
{ "NOM_COM" : "CHELLES", "POPULATION" : 469}
```

Les curseurs

La méthode find() retourne un curseur sur l'ensemble des documents satisfaisant les critères de recherche.

Le shell mongo les affiche 20 par 20. On passe à la page suivante en tapant 'it' (iterate)

Le nombre de documents par page se paramètre en modifiant la variable DBQuery.shellBatchSize

```
> DBQuery.shellBatchSize=2
2
> db.communes.find()
{ "_id" : ObjectId("59c8a4f84ade6b6c7aae4d0f"), "ID_GEOFLA" : "COMMUNE00000000000000002", "CODE_COM" : 33,
"INSEE_COM" : 47033, "NOM_COM" : "BOUDY-DE-BEAUREGARD", "STATUT" : "Commune simple", "X_CHF_LIEU" :
516424, "Y_CHF_LIEU" : 6384852, "X_CENTROID" : 515575, "Y_CENTROID" : 6385938, "Z_MOYEN" : 112, "SUPERFICIE" :
1019, "POPULATION" : 406, "CODE_CANT" : 6, "CODE_ARR" : 3, "CODE_DEPT" : 47, "NOM_DEPT" : "LOT-ET-GARONNE",
"CODE_REG" : 72, "NOM_REG" : "AQUITAINE" }
{ "_id" : ObjectId("59c8a4f84ade6b6c7aae4d10"), "ID_GEOFLA" : "COMMUNE00000000000000003", "CODE_COM" : 9,
"INSEE_COM" : 32009, "NOM_COM" : "ARMOUS-ET-CAU", "STATUT" : "Commune simple", "X_CHF_LIEU" : 472979,
"Y_CHF_LIEU" : 6278963, "X_CENTROID" : 473004, "Y_CENTROID" : 6278937, "Z_MOYEN" : 217, "SUPERFICIE" : 932,
"POPULATION" : 95, "CODE_CANT" : 20, "CODE_ARR" : 3, "CODE_DEPT" : 32, "NOM_DEPT" : "GERS", "CODE_REG" : 73,
"NOM_REG" : "MIDI-PYRENEES" }
Type "it" for more
>
```

Le shell mongo est programmable :

```
> var j=db.communes.find( {NOM_COM: 'CHELLES'},{NOM_COM: 1, NOM_DEPT: 1, POPULATION: 1, _id: 0} )
> while (j.hasNext()) { print (tojson(j.next())); }
{ "NOM_COM" : "CHELLES", "POPULATION" : 469, "NOM_DEPT" : "OISE" }
{
        "NOM_COM" : "CHELLES",
        "POPULATION" : 52817,
        "NOM_DEPT" : "SEINE-ET-MARNE"
}
>
```

Le while() peut se transformer en forEach et le print (tojson()) en printjson. Comme en Perl, il existe une variable implicite qui désigne l'objet courant.

```
var j =  db.communes.find( {NOM_COM: 'CHELLES'},{NOM_COM: 1, NOM_DEPT: 1, POPULATION: 1, _id: 0} )
j.forEach(printjson);

{ "NOM_COM" : "CHELLES", "POPULATION" : 469, "NOM_DEPT" : "OISE" }
{
        "NOM_COM" : "CHELLES",
        "POPULATION" : 52817,
        "NOM_DEPT" : "SEINE-ET-MARNE"
}
>
```

Les opérations CRUD

Create, Read, Update, Delete sont les 4 opérations de base qu'offre un SGBD.
MongoDB retourne un document WriteResult() ou BulkWriteResult().

Les informations retournées sont :

nInserted	le nombre de documents insérés, excluant les documents "upsert"
nUpserted	le nombre de documents insérés par un "upsert"
nMatched	le nombre de documents sélectionnés pour un update
nModified	le nombre de documents effectivement modifiés
nRemoved	le nombre de documents supprimés
writeErrors	informations en cas erreur

Insert

La méthode 'insert()' perdure, pour des raisons de compatibilité, mais est obsolète. On préférera, selon le cas, utiliser 'insertOne()' et 'insertMany()'.

```
> db.xx.insert({ prenom : "Jean", nom : "Dubois",
...
...             adresse : {
...                         numero :     "5",
...                         voie :       "Place des Fêtes",
...                         codepostal : "75019",
...                         ville :      "Paris"
...                         },
... age : 55
... }
... )
WriteResult({ "nInserted" : 1 })
>
```

L'insert retourne un document WriteResult qui confirme l'insertion. L'insertOne retourne un 'acknowledge' et l'_id du document inséré.

```
> db.xx.insertOne({ prenom : "Jean", nom : "Dubois",
...
...           adresse : {
...                     numero :     "5",
...                     voie :       "Place des Fêtes",
...                     codepostal : "75019",
...                     ville :      "Paris"
...                     },
... age : 55
... }
... )
{
        "acknowledged" : true,
        "insertedId" : ObjectId("5c7a7b3b31412dac5bb424a2")
}
>
```

Il n'est par rare d'utiliser des variables pour stocker des documents JSON et d'utiliser ces variables pour des opérations d'insert ou d'update.

```
x ={ prenom : "Jean", nom : "Dubois",

          adresse : {
                    numero :     "5",
                    voie :       "Place des Fêtes",
                    codepostal : "75019",
                    ville :      "Paris"
                    },
age : 55
}
> db.xx.insert(x)
WriteResult({ "nInserted" : 1 })
>
```

InsertMany - Bulk insert

Il est possible d'insérer des documents en masse, groupés en tableau :

```
> var x= [ {prenom: 'pierre'}, {prenom: 'paul'}, {prenom: 'jacques'} ]
> db.xx.insert(x)
BulkWriteResult({
          "writeErrors" : [ ],
          "writeConcernErrors" : [ ],
          "nInserted" : 3,
          "nUpserted" : 0,
          "nMatched" : 0,
          "nModified" : 0,
          "nRemoved" : 0,
          "upserted" : [ ]
})
>
```

```
db.xx.find().pretty()
{ "_id" : ObjectId("59c7e3cee138a50567db060e"), "prenom" : "pierre" }
{ "_id" : ObjectId("59c7e3cee138a50567db060f"), "prenom" : "paul" }
{ "_id" : ObjectId("59c7e3cee138a50567db0610"), "prenom" : "jacques" }
>
```

L'insertion retourne un document 'BulkWriteResult' synthétisant les insertions et éventuelles erreurs. MongoDB préconise la méthode insertMany pour ce genre de situation :

```
> db.xx.insertMany([{prenom:"pierre"},{prenom:"paul"}])
{
        "acknowledged" : true,
        "insertedIds" : [
                ObjectId("5c7ab0df31412dac5bb424a6"),
                ObjectId("5c7ab0df31412dac5bb424a7")
        ]
}
>
```

Update

La méthode update() permet de mettre à jour des données existantes.

Syntaxe
db.<NomCollection>.update(<query>, <update>,<options>)

Exemple

```
> db.communes.update({},{$set : {annee : "2017" } } );
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.communes.find({annee:"2017"},{NOM_COM:1});
{ "_id" : ObjectId("553f74ce4ce2b576def797bf"), "NOM_COM" : "ARMOUS-ET-CAU" }
>
```

Par défaut, seul le premier document est mis à jour. Pour plusieurs documents, prévoir l'option {multi:true} ou utiliser le raccourci updateMany().

```
> db.communes.update({},{$set : {annee : "2017" } }, {multi: true} );
WriteResult({ "nMatched" : 36595, "nUpserted" : 0, "nModified" : 36594 })
>
> // l'option multi: true provoque la mise à jour de toute la collection. le WriteResult retourne 36595 documents
> // concernés et 36594 effectivement mis à jour. Le premier document avait déjà été mis à jour.
>
> db.communes.find().limit(3)
{ "_id" : ObjectId("59c8a4f84ade6b6c7aae4d0f"), "ID_GEOFLA" : "COMMUNE0000000000000002", "CODE_COM" : 33, "INSEE_COM" : 47033, "NOM_COM" : "BOUDY-DE-BEAUREGARD", "STATUT" : "Commune simple", "X_CHF_LIEU" : 516424, "Y_CHF_LIEU" : 6384852, "X_CENTROID" : 515575, "Y_CENTROID" : 6385938, "Z_MOYEN" : 112, "SUPERFICIE" : 1019, "POPULATION" : 406, "CODE_CANT" : 6, "CODE_ARR" : 3, "CODE_DEPT" : 47, "NOM_DEPT" : "LOT-ET-GARONNE", "CODE_REG" : 72, "NOM_REG" : "AQUITAINE", "annee" : "2017" }
../..
```

```
../..
{ "_id" : ObjectId("59c8a4f84ade6b6c7aae4d10"), "ID_GEOFLA" : "COMMUNE0000000000000003", "CODE_COM" : 9, "INSEE_COM" : 32009, "NOM_COM" : "ARMOUS-ET-CAU", "STATUT" : "Commune simple", "X_CHF_LIEU" : 472979, "Y_CHF_LIEU" : 6278963, "X_CENTROID" : 473004, "Y_CENTROID" : 6278937, "Z_MOYEN" : 217, "SUPERFICIE" : 932, "POPULATION" : 95, "CODE_CANT" : 20, "CODE_ARR" : 3, "CODE_DEPT" : 32, "NOM_DEPT" : "GERS", "CODE_REG" : 73, "NOM_REG" : "MIDI-PYRENEES", "annee" : "2017" }
{ "_id" : ObjectId("59c8a4f84ade6b6c7aae4d11"), "ID_GEOFLA" : "COMMUNE0000000000000004", "CODE_COM" : 225, "INSEE_COM" : 38225, "NOM_COM" : "MEAUDRE", "STATUT" : "Commune simple", "X_CHF_LIEU" : 898640, "Y_CHF_LIEU" : 6450689, "X_CENTROID" : 898625, "Y_CENTROID" : 6451597, "Z_MOYEN" : 1184, "SUPERFICIE" : 3371, "POPULATION" : 1378, "CODE_CANT" : 41, "CODE_ARR" : 1, "CODE_DEPT" : 38, "NOM_DEPT" : "ISERE", "CODE_REG" : 82, "NOM_REG" : "RHONE-ALPES", "annee" : "2017" }
>
```

Query	définit le ou les critères de sélection	
Update	introduit les modifications à effectuer. Utilise les opérateur $set et $unset { $set: { champ: "valeur" } } ou { $set: { chp1: "val1", chp2:"val2 } }	
Options	multi: <bool>	si la valeur est true, mise à jour de tous les documents. si la valeur est false, seul le premier document est mis à jour. la valeur par défaut est false
	upsert: <bool>	upsert = update / insert si la valeur est true, insert du nouveau document si l'update échoue. la valeur par défaut est false.
	writeConcern: <document>	document qui exprime le writeConcern. brièvement, le writeConcern permet de s'assurer de la bonne écriture dans les fichiers de données, sur tous les replicasets d'un cluster, de l'écriture dans le journal....

Note : les raccourcis updateOne() et updateMany() équivalent à update avec l'option multi à true ou false.

On peut ajouter un ou plusieurs champ(s) à un document existant :

```
> db.communes.update({},{$set: {xx: "xx",yy:"yy"}},{multi:1})
WriteResult({ "nMatched" : 36595, "nUpserted" : 0, "nModified" : 36595 })
>
```

Dans un update(), l'utilisation des variables peut s'avérer utile.

Dans l'exemple suivant, la variable x contient le document relatif à la première ville de CHELLES. La variable est mise à jour. (par défaut multi : false)

```
> var x=db.communes.findOne({NOM_COM: 'CHELLES'})
> x
{
        "_id" : ObjectId("59c8a4f84ade6b6c7aae5e9f"),
        "ID_GEOFLA" : "COMMUNE00000000000004493",
        "CODE_COM" : 145,
        "INSEE_COM" : 60145,
        "NOM_COM" : "CHELLES",
        "STATUT" : "Commune simple",
        "X_CHF_LIEU" : 702507,
        "Y_CHF_LIEU" : 6917025,
        "X_CENTROID" : 703292,
        "Y_CENTROID" : 6916489,
        "Z_MOYEN" : 109,
        "SUPERFICIE" : 903,
        "POPULATION" : 469,
        "CODE_CANT" : 1,
        "CODE_ARR" : 3,
        "CODE_DEPT" : 60,
        "NOM_DEPT" : "OISE",
        "CODE_REG" : 22,
        "NOM_REG" : "PICARDIE",
        "annee" : "2017",
        "xx" : "xx",
        "yy" : "yy"
}
>
> x.xx='xxxx'
xxxx
>
> db.communes.update({NOM_COM: 'CHELLES'},x)
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
> var x=db.communes.findOne({NOM_COM: 'CHELLES'})
> x
{
        "_id" : ObjectId("59c8a4f84ade6b6c7aae5e9f"),
        "ID_GEOFLA" : "COMMUNE00000000000004493",
        "CODE_COM" : 145,
        "INSEE_COM" : 60145,
        "NOM_COM" : "CHELLES",
        "STATUT" : "Commune simple",
        "X_CHF_LIEU" : 702507,
        "Y_CHF_LIEU" : 6917025,
        "X_CENTROID" : 703292,
        "Y_CENTROID" : 6916489,
        "Z_MOYEN" : 109,
        "SUPERFICIE" : 903,
        "POPULATION" : 469,
        "CODE_CANT" : 1,
        "CODE_ARR" : 3,
        "CODE_DEPT" : 60,
        "NOM_DEPT" : "OISE",
        "CODE_REG" : 22,
        "NOM_REG" : "PICARDIE",
        "annee" : "2017",
        "xx" : "xxxx",
        "yy" : "yy"
}
> // La mise à jour s'est bien effectuée.
```

Pour supprimer un champ dans un document, on utilise l'opérateur $unset

```
> db.communes.update( { }, { $unset: { xx: "", yy:"" } } , {multi: true} )
WriteResult({ "nMatched" : 36595, "nUpserted" : 0, "nModified" : 36595 })
>
```

Remove

Pour supprimer des données, on utilise la méthode remove().

Syntaxe
db.<NomCollection>.remove(<query>, <justOne>)

query	définit le ou les critères de sélection
justOne	si true, supprime uniquement le premier élément correspondant aux critères. si false, supprime tous les éléments false par défaut Remarque : la logique est inversée vis à vis de l'update (!)

Exemple

```
> use test
switched to db test
> show collections
xx
>
> db.xx.find({ prenom : {$exists: true} })
{ "_id" : ObjectId("59c7bfbdbc30888a2ba6e25d"), "prenom" : "Jean", "nom" : "Dubois", "adresse" : { "numero" : "5",
"voie" : "Place des Fêtes", "codepostal" : "75019", "ville" : "Paris" }, "age" : 55 }
{ "_id" : ObjectId("59c7cf26bc30888a2ba6e25f"), "prenom" : "Jean", "nom" : "Dubois", "adresse" : { "numero" : "5",
"voie" : "Place des Fêtes", "codepostal" : "75019", "ville" : "Paris" }, "age" : 55 }
{ "_id" : ObjectId("59c7e3cee138a50567db060e"), "prenom" : "pierre" }
{ "_id" : ObjectId("59c7e3cee138a50567db060f"), "prenom" : "paul" }
{ "_id" : ObjectId("59c7e3cee138a50567db0610"), "prenom" : "jacques" }
>
> db.xx.remove({ prenom : {$exists: true} })
WriteResult({ "nRemoved" : 5 })
>
> db.xx.find({ prenom : {$exists: true} })
>
```

La méthode remove() permet aussi de supprimer tous les documents d'une collection ; Il suffit d'appliquer un critère vide et de laisser l'option justOne à false.

La collection est vidée mais existe toujours. Équivalent du truncate SQL.

```
> db.xx.remove({})                    //
WriteResult({ "nRemoved" : 8 })
> db.xx.find()
> show collections
xx
>
```

Note : les raccourcis deleteOne() et deleteMany() sont préférables et jouent sur l'option 'justOne'.

Validator

Réaliser le TP 3.

https://docs.mongodb.com/manual/core/schema-validation/

drop

La méthode drop() supprime la collection et ses index

```
> db.xx.drop()
true
>
```

GridFS

GridFS est une technique utilisable pour stocker des BLOBs (binary long objects). Les BLOBs sont des fichiers images, vidéo, audio etc..... Il s'agit de contourner la limite de 16Mo pour un document.

Le fichier est découpé en chunks de 255k. Chaque chunk est stocké dans un document séparé. GridFS utilise deux collections fs.files et fs.chunks pour stocker respectivement les méta données et les chunks.

Chaque chunk est identifié par son _id. La collection fs.files est la collection 'parent'.

```
{
  filename: "test.txt",                                  // Le document
  chunkSize: NumberInt(261120),                          // taille du chunk 255 Ko
  uploadDate: ISODate("2017-09-02T11:32:33.557Z"),
  md5: "7b762939321e146569b07f72c62cca4f",               // Somme de contrôle
  length: NumberInt(646)                                 // Nombre de chunks
}
```

Le champ 'files_id' de la collection fs.chunks fait la jointure avec l'_id du document parent.

```
{
  files_id: ObjectId("534a75d19f54bfec8a2fe44b"),
  n: NumberInt(0),
  data: "Mongo Binary Data"
}
```

Ajout d'un document à GridFS

On utilise la commande 'put' du binaire 'mongofiles' :

```
pierrick@grouik ~]$ mongofiles -d test -u root -p root --authenticationDatabase=admin put xx.iso
2017-10-09T15:50:23.569+0200        connected to: localhost
added file: xx.iso
[pierrick@grouik ~]$
```

Il en résulte 268 chunks (69 953 536 / 255 / 1024 = 267,89)

```
root@127.0.0.1:27017:test> db.fs.files.find({},{_id:1, length:1})
{ "_id" : ObjectId("59db7e9f52dabf17e09363be"), "length" : 69953536 }
root@127.0.0.1:27017:test>
root@127.0.0.1:27017:test> db.fs.chunks.find({},{_id:0, files_id:1, n:1})
../..
{ "files_id" : ObjectId("59db7e9f52dabf17e09363be"), "n" : 120 }
{ "files_id" : ObjectId("59db7e9f52dabf17e09363be"), "n" : 121 }
../..
Type "it" for more
```

Objectifs

Javascript
Python
Php, perl
Java

Les scripts Javascript.

On a vu en introduction le script HOME/.mongorc.js qui définit la fonction JavaScript :

```
function prompt() {
   var username = "anon";
   var user = db.runCommand({connectionStatus : 1}).authInfo.authenticatedUsers[0];
   var host = db.getMongo().toString().split(" ")[2];
   var current_db = db.getName();

   if (!!user) {
      username = user.user;
   }

   return username + "@" + host + ":" + current_db + "> ";
}
```

La fonction s'appelle prompt. Les parenthèses sont un élément de syntaxe, que la fonction accepte des paramètres ou pas.

Le corps de la fonction est enserré entre accolades. Cette notation est héritée du C et on la retrouve en Java ou d'autres langages.

Une variable est déclarée et initialisée par le mot-clé 'var'.

Ici, on appelle la commande 'connectionStatus"=' qui retourne un document JSON :

```
> db.runCommand({connectionStatus:1})
{
        "authInfo" : {
                "authenticatedUsers" : [ ],
                "authenticatedUserRoles" : [ ]
        },
        "ok" : 1
}
```

Si le serveur est en mode authentifié (voir chapitre Utilisateurs et roles), la commande retourne le nom d'utilisateur.

La commande 'db.getMongo()' retourne l'IP et le numéro de port de la connexion et la commande 'db.getName()' retourne la base courante.

```
> db.getMongo()
connection to 127.0.0.1:27017
>
```

Les méthodes toString et split sont des fonctions Javascript explicites.

La fonction retourne la chaîne de connexion complète par 'return' (l'opérateur de concaténation est le '+') .

Il est possible de démarrer mongo avec un fichier .js :

```
pierrick@localhost:~/job$ mongo xx.js
```

Sous le shell mongo, il est possible le lancer un script externe :

```
> load('/home/pierrick/mongodb/script.js')
```

Au sein du script js, on peut initier une connexion avec le constructeur Mongo()

- conn=new Mongo() ;
- conn=new Mongo(<host>:<port>) ;

Une fois la connexion établie, sélectionner la base avec getDB()

- db=conn.getDB(<base>) ;

La connexion peut s'établir directement avec

- db=connect("<host>:<port>/<base>") ;

Par défaut :

- db=connect("localhost:27017/test") ;

Les raccourcis du shell mongo en bref :

show dbs	db.adminCommand('listDatabases') ;
use <db>	db.getSibingDB('<db>) ;
show collections	db.getCollectionsNames() ;
show logs	db.adminCommand({'getLog' :'*'}) ;
show log <log>	db.adminCommand({'getLog' :'<log>'}) ;

Pour visualiser une collection, on utilise un curseur :

```
.../...
cur=db.produits.find()

while(cur.hasNext()) {
        doc=cur.next()
        print(doc.designation,doc.prix,doc.stock);
        }
.../.....
```

Plus d'info : http://zetcode.com/db/mongodbjavascript/

L'interface PyMongo

Pour accéder à Mongodb depuis python, on utilise le module ‘pymongo’.

```
[root@localhost VMs]# pip3 install pymongo
WARNING: Running pip install with root privileges is generally not a good idea. Try `pip3 install --user` instead.
Collecting pymongo
  Downloading https://files.pythonhosted.org/packages/11/88/dd1f8c4281a60791b043f55e338d0521049208f21e3de19ddc9c160dbbef/pymongo-3.7.1-cp36-cp36m-manylinux1_x86_64.whl (405kB)
    100% |████████████████████████████████| 409kB 1.4MB/s
Installing collected packages: pymongo
Successfully installed pymongo-3.7.1
[root@localhost Vms]#
```

Établir la connexion avec MongoClient

La première étape consiste à établir la connexion par la création d’un objet MongoClient par appel du constructeur :

```
>>> from pymongo import MongoClient
>>> client = MongoClient()
```

Équivalant aux 2 notations suivantes :

```
>>> client = MongoClient('localhost', 27017)
>>>
>>> client = MongoClient('mongodb://localhost:27017/')
```

Se connecter à une base

On utilise un objet de classe MongoClient (ici client) pour choisir la base.

```
>>> db = client.test
```

On peut utiliser la notation suivante, notamment si le nom de la base contient des caractères spéciaux :

```
>>> db = client['ma-base']
```

Accéder à une collection

```
>>> collection = db.ma-collection
```

ou (notation 'dictionnaire')

```
>>> collection = db['ma-collection']
```

Documents

```
>>> import datetime
>>> post = {"author": "Mike",
...         "text": "My first blog post!",
...         "tags": ["mongodb", "python", "pymongo"],
...         "date": datetime.datetime.utcnow()}
```

Dans cet exemple on utilise le type datetime qui sera automatiquement converti en son équivalent BSON.

Insérer un document

Utilisation de la méthode `insert_one()` :

```
>>> posts = db.posts
>>> post_id = posts.insert_one(post).inserted_id
>>> post_id
ObjectId('...')
```

`Insert_one()` retourne une instance d' `InsertOneResult`.

Pour informations complémentaires sur le champ `"_id"`, voir documentation on _id.

On peut obtenir la liste des collections :

```
>>> db.collection_names()
['posts']
```

Recherche de document avec la méthode `find_one()`

La méthode find_one() retourne le premier document satisfaisant les critères de recherche. (ou None en cas de recherche infructueuse)

Exemple :

```
>>> import pprint
>>> pprint.pprint(posts.find_one())
{'_id': ObjectId('...'),
 author': Mike',
 'date': datetime.datetime(...),
 'tags': ['mongodb', 'python', 'pymongo'],
 'text': 'My first blog post!'}
```

Le résultat est un dictionnaire.

`Find_one()` peut accepter une valeur en paramètre :

```
>>> pprint.pprint(posts.find_one({"author": "Mike"}))
{'_id': ObjectId('...'),
 'author': 'Mike',
 'date': datetime.datetime(...),
 'tags': ['mongodb', 'python', 'pymongo'],
 'text': 'My first blog post!'}
```

Une recherche sur le prénom “Eliot” ne produit rien :

```
>>> posts.find_one({"author": "Eliot"})
>>>
```

Recherche selon l’objectId :

```
>>> post_id
ObjectId(...)
>>> pprint.pprint(posts.find_one({"_id": post_id}))
{'_id': ObjectId('...'),
 'author': 'Mike',
 'date': datetime.datetime(...),
 'tags': ['mongodb', 'python', 'pymongo'],
 'text': 'My first blog post!'}
```

Attention : un ObjectId n’est pas sa représentation sous forme de chaîne de caractères :

```
>>> post_id_as_str = str(post_id)
>>> posts.find_one({"_id": post_id_as_str}) # Pas de résultat
>>>
```

Dans les applications Web, il est fréquent d'obtenir l'ObjectId depuis l'URL. Dans ce cas, il convient de trans typer depuis un string avant d'invoquer la méthode find_one() :

```
from bson.objectid import ObjectId

# The web framework gets post_id from the URL and passes it as a string

def get(post_id):
    # Convert from string to ObjectId:
    document = client.db.collection.find_one({'_id': ObjectId(post_id)})
```

A voir : [When I query for a document by ObjectId in my web application I get no result]

Bulk Inserts

Les insertions en masse se font en passant une liste de documents à la méthode insert_many(). Il n'y a qu'une seule requête pour n insertions.

```
>>> new_posts = [{"author": "Mike",
...               "text": "Another post!",
...               "tags": ["bulk", "insert"],
...               "date": datetime.datetime(2009, 11, 12, 11, 14)},
...              {"author": "Eliot",
...               "title": "MongoDB is fun",
...               "text": "and pretty easy too!",
...               "date": datetime.datetime(2009, 11, 10, 10, 45)}]
>>> result = posts.insert_many(new_posts)
>>> result.inserted_ids
[ObjectId('...'), ObjectId('...')]
```

L' `insert_many()` retourne deux `ObjectId` , un pour chaque document inséré..

Recherches sur plus d'un document

Pour une recherche élargie, on utilise la méthode `find()`. Cette méthode retourne un curseur (Voir le lien `Cursor`) .

On utilise une boucle pour itérer l'ensemble des document retournés.

```
>>> for post in posts.find():
...   pprint.pprint(post)
...
```

```
{'_id': ObjectId('...'),
 'author': 'Mike',
 'date': datetime.datetime(...),
 'tags': ['mongodb', 'python', 'pymongo'],
 'text': u'My first blog post!'}
{'_id': ObjectId('...'),
 'author': 'Mike',
 'date': datetime.datetime(...),
 'tags': ['bulk', 'insert'],
 'text': 'Another post!'}
{'_id': ObjectId('...'),
 'author': 'Eliot',
 'date': datetime.datetime(...),
 'text': 'and pretty easy too!',
 'title': 'MongoDB is fun'}
```

Comme avec la méthode `find_one()`, on peut restreindre la recherche en passant un critère à la méthode `find()`. Le critère s'exprime dans un document JSON :

```
>>> for post in posts.find({"author": "Mike"}):
...   pprint.pprint(post)
...
{'_id': ObjectId('...'),
 'author': 'Mike',
 'date': datetime.datetime(...),
 'tags': ['mongodb', 'python', 'pymongo'],
 'text': 'My first blog post!'}
{'_id': ObjectId('...'),
 'author': 'Mike',
 'date': datetime.datetime(...),
 'tags': ['bulk', 'insert'],
 'text': 'Another post!'}
```

La méthode count()

Cette méthode retourne simplement le nombre de documents répondant au critère de recherche :

```
>>> posts.count()
3
>>> posts.find({"author": "Mike"}).count()
2
```

Recherches par intervalles – Tris

MongoDB accepte des méthodes de recherche plus complexes (advanced queries).

La requête suivante sélectionne les documents antérieurs à une date donnée (opérateur $lt = less than), et les trie selon l'auteur :

```
>>> d = datetime.datetime(2009, 11, 12, 12)
>>> for post in posts.find({"date": {"$lt": d}}).sort("author"):
...   pprint.pprint(post)
...
{'_id': ObjectId('...'),
 'author': u'Eliot',
 'date': datetime.datetime(...),
 'text': u'and pretty easy too!',
 'title': u'MongoDB is fun'}
{'_id': ObjectId('...'),
 'author': u'Mike',
 'date': datetime.datetime(.pprint.pprint(post)..),
 'tags': [u'bulk', u'insert'],
 'text': u'Another post!'}
```

Les interfaces PHP et Perl

Exemple de code Php

```
#!/usr/bin/php

<?php
   // connect to mongodb
   $m = new MongoClient();
   echo "Connection to database successfully\n";

   // select a database
   $db = $m->mydb;
   echo "Database mydb selected\n";

  // create collection
   $collection = $db->createCollection("mycol");
   echo "Collection created succsessfully\n";

   $document = array(
      "title" => "MongoDB",
      "description" => "database",
      "likes" => 100,
      "url" => "http://www.tutorialspoint.com/mongodb/",
      "by" => "tutorials point"
   );

   $collection->insert($document);
   echo "Document inserted successfully"

   $cursor = $collection->find();
   // iterate cursor to display title of documents

   foreach ($cursor as $document) {
      echo $document["title"] . "\n";
   }

?>
```

Exemple de code Perl

```
#!/usr/bin/perl

use strict;
use warnings;
use 5.010;

use MongoDB;
use Data::Dumper qw(Dumper);

my $client = MongoDB::MongoClient->new();

#my $client = MongoDB::MongoClient->new(host => 'localhost', port => 27017);

my $db   = $client->get_database( 'test_perl' );

my $people_coll = $db->get_collection('people');

$people_coll->insert_one( {
   name => 'First',
});

$people_coll->insert_one( {
   name => 'Second',
});

my $people = $people_coll->find;
while (my $p = $people->next) {
   print Dumper $p;
}

$db->drop;
```

L'interface Java

Différents pilotes sont proposés en fonction de la version de Mongodb et du JDK.

https://mongodb.github.io/mongo-java-driver/

Les packages à importer sont 'com.mongodb' et 'com.mongodb.client'. L'encodage et décodage du format bson suppose l'import de la classe 'org.bson'.

Les exemples à suivre sont validés avec java 1.8 et un driver en version 3.6 intitulé 'mongo-java-driver-3.6.1.jar'.

Pour les projets 'Maven' intégrer la séquence de dépendance suivante :

```
  <dependency>
     <groupId>org.mongodb</groupId>
     <artifactId>mongo-java-driver</artifactId>
     <version>3.6.1</version>
  </dependency>
```

Connexion

La connexion au serveur se fait avec le constructeur MongoClient

```
MongoClient mongoClient = new MongoClient("host", port);
```

Par défaut, localhost et 27017.

Autre syntaxe (utilisation d'une URI) :

```
MongoClient mongoClient = new MongoClient(new MongoClientURI("mongodb://localhost:27017/");
```

Cette seconde syntaxe permet de préciser plus d'options :

Se connecter à un serveur avec authentification :

```
MongoClient mongoClient = new MongoClient(new
MongoClientURI("mongodb://user1:pwd1@host1/?authSource=db1"););
```

L'utilisateur user1 est déclaré (avec son MdP) auprès de la base db1 sur le host1.
(Voir chapitre 'Utilisateurs et rôles')

Se connecter à un 'replica set' :

```
MongoClient mongoClient = new MongoClient( new
MongoClientURI("mongodb://host1:27017,host2:27017,host3:27017"));
```

en précisant au moins deux membres du 'replica set'

```
MongoClient mongoClient = new MongoClient( new MongoClientURI(
"mongodb://host1:27017,host2:27017,host3:27017/?replicaSet=myReplicaSet"));
```

en précisant le nom 'replica set' et au moins un membre.

Le driver reconnaît le rôle du membre du 'replica set', primary ou secondary....
(Voir chapitre Replication)

Pour se connecter à un cluster sharded, se connecter à l'instance 'mongos' comme à un seveur ordinaire. (Voir chapitre Sharding)

Bases - Collections

Pour se connecter à une base on utilise le constructeur MongoDatabase

```
MongoDatabase database = mongoClient.getDatabase("test");
```

Pour sélectionner une collection :

```
MongoCollection<Document> collection = database.getCollection("col_x");
```

Pour créer une collection (si option particulière, les collections se créent, par défaut, 'à la volée')

```
database.createCollection("col_y", new
        CreateCollectionOptions().capped(true).sizeInBytes(0x50000)) ;
```

Documents

Récupérer les documents :

```
FindIterable<Document> results = collection.find();
```

Boucler :

```
for (Document document : results) {
                System.out.println(document.getString("nom"));
```

Poser un filtre sur la sélection :

```
FindIterable<Document> results = collection.find(
                and(gte(prix, 2.0), (lt, 3.0))
                ) ;
```

Ces filtres sont disponibles après avoir importé le module com.mongodb.client.Filters.*

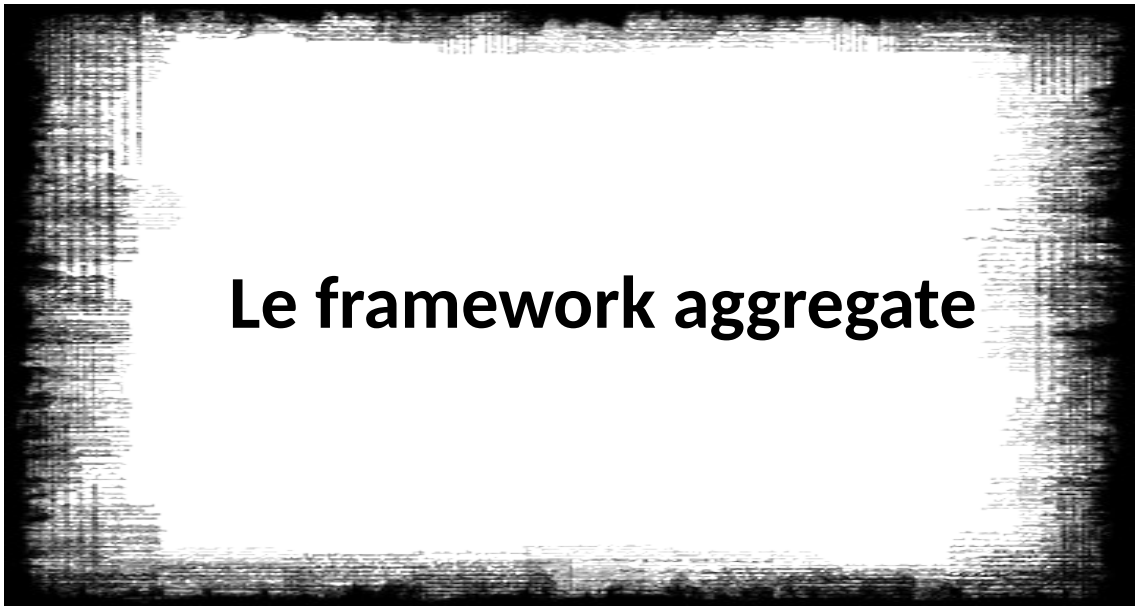
```
import static com.mongodb.client.model.Filters.*;
```

Pour accéder aux champs d'un document :

- getString()
- getInt()
- getDouble()
- getDate()
- getBoolean()

Insérer un document :

```
Document prod1 = new Document("designation", "Gaufrettes vanille")
                        .append("prix", "1.90").append("stock","500");
collection.insertOne(prod1) ;
```



Objectifs

Les opérateurs project, match, group, sort,
Unwind,
Lookup
Les tableaux

La fonction aggregate – Généralités

Les traitements vus dans le chapitre précédent sont des traitements assez élémentaires. Une base de données est utile pour analyser les données, éventuellement les restructurer. (calculs, agrégats….)

Le 'framework aggregate' offre quantité de méthodes pour traiter l'information. Le donnés passent comme un flux au travers d'un pipeline où chaque étape réalise un traitement particulier :

- filtrage,
- projection,
- regroupement,
- tri…

Syntaxe

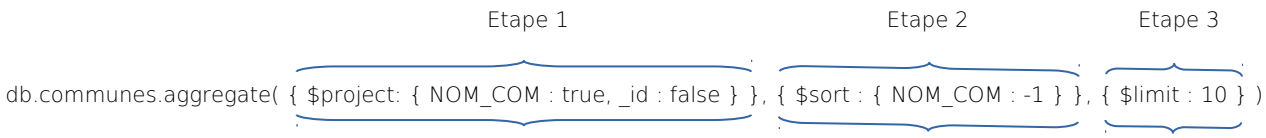
db.<collection>.aggregate(<oper>, <options>)

Les étapes (<oper>,<options>) sont chaînées dans un pipe. A l'étape n, le flux en entée est traité et transmis en sortie à l'étape n+1.

$project	restructure chaque document du flux, en ajoutant ou supprimant des champs. Ex : prix * qte => montant
$match	filtre les documents regroupés suivant le ou les critères de sélection.
$group	regroupe les documents par une expression spécifique. Ex : sum, avg…
$limit	limite le nombre de documents à retourner.
$skip	'saute' un certain nombre de documents en entrée.
$sort	trie les documents. (ordre ascendant ou descendant).
$unwind	transforme chaque élément d'un tableau en un document séparé.
$lookup	Réalise des jointures de collections

Exemple

```
> db.communes.aggregate( { $project: { NOM_COM : true, _id : false } }, { $sort : { NOM_COM : -1 } }, { $limit : 10 } )
{ "NOM_COM" : "ZUYTPEENE" }
{ "NOM_COM" : "ZUYDCOOTE" }
{ "NOM_COM" : "ZUTKERQUE" }
{ "NOM_COM" : "ZUDAUSQUES" }
{ "NOM_COM" : "ZUANI" }
{ "NOM_COM" : "ZOZA" }
{ "NOM_COM" : "ZOUFFTGEN" }
{ "NOM_COM" : "ZOUAFQUES" }
{ "NOM_COM" : "ZOTEUX" }
{ "NOM_COM" : "ZONZA" }
>
```



Les virgules séparatrices de la fonction aggregate construisent le pipe. Les étapes successives ne sont que des opérations déjà vues, réécrites autrement.

On peut gagner en lisibilité :

```
db.communes.aggregate(
                        { $project: { NOM_COM : true, _id : false } },
                        { $sort : { NOM_COM : -1 } },
                        { $limit : 10 }
                        )
```

Project

L'opérateur $project permet de restructurer chaque document du flux :

- Sélectionner les champs à retenir avec true
  (par défaut tous les champs sont à false à part _id)
- Supprimer le champ _id avec false
- Permutter les champs : { <champ4> : true, <champ2 : true, <champ3> : true }
- Ajouter un champ calculé

Exemple :

```
> db.commandes.find({},{_id : 0})
{ "article" : "xx1", "qte" : 3, "px" : 2.5 }
{ "article" : "xx2", "qte" : 4, "px" : 6 }
{ "article" : "xx3", "qte" : 5, "px" : 4.2 }
>
> db.commandes.aggregate({ $project: {_id: 0, article: 1, qte: 1, px: 1, montant: {$multiply : [ "$qte", "$px" ]} }})
{ "article" : "xx1", "qte" : 3, "px" : 2.5, "montant" : 7.5 }
{ "article" : "xx2", "qte" : 4, "px" : 6, "montant" : 24 }
{ "article" : "xx3", "qte" : 5, "px" : 4.2, "montant" : 21 }
>
```

Le nouveau champ calculé est généré à la volée en le baptisant (ici, montant) auquel on ajoute une expression calculée.

Expressions mathématiques

Opérateurs $add, $substract, $multiply, $mod, $divide suivis d'une liste d'arguments entre [ ].

- $add et $multiply acceptent un nombre quelconque d'arguments.
- $substract, $mod et $divide n'acceptent que deux arguments.

Ex :
total : { $add : ["$x", "$y", "$z"] }
impair : { $mod : ["$nbr", 2] }

Expressions sur les dates

$year, $month, $week, $dayOfMonth, $dayOfWeek, $dayOfYear, $hour, $minute, $second

Expressions sur les chaînes de caractères

$substr, $concat, $toLower, $toUpper, $strcasecmp

Comparaisons

$cmp, $eq, $ne, $gt, $gte, $lt, $lte

Combinaisons logiques

$and, $or, $not, $cond, $ifNull

Match

Filtre les documents suivant le ou les critères de sélection.

```
> use yougo
switched to db yougo
>
> db.user_type.find()
{ "_id" : ObjectId("4e65fd75e8bf1bb31c2492d9"), "code" : "CONSULTANT", "description" : "Consultant" }
{ "_id" : ObjectId("4e65fd75e8bf1bb31c2492db"), "code" : "DIRECTION", "description" : "Direction" }
{ "_id" : ObjectId("4e65fd75e8bf1bb31c2492dc"), "code" : "COMMERCIAL", "description" : "Commercial" }
{ "_id" : ObjectId("4e65fd75e8bf1bb31c2492da"), "code" : "RH", "description" : "Ressources Humaines" }
>
> db.user_type.aggregate( { $match: { $or: [ {code: "DIRECTION"}, {code: "RH"} ] } } )
{ "_id" : ObjectId("4e65fd75e8bf1bb31c2492db"), "code" : "DIRECTION", "description" : "Direction" }
{ "_id" : ObjectId("4e65fd75e8bf1bb31c2492da"), "code" : "RH", "description" : "Ressources Humaines" }
>
```

Comme en SQL, évaluer les expressions $match le plus tôt possible, afin d'éliminer les documents non-voulus rapidement.

Group

L'opérateur $group regroupe les documents selon une clé et effectue une opération de type arithmétique sur les documents du groupe.

Total de la population de Loire-Atlantique

```
> db.communes.aggregate(
                                        { $match: { NOM_DEPT: "LOIRE-ATLANTIQUE" } },
                                        { $project: { NOM_DEPT: 1, POPULATION: 1, _id: 0 } },
                                        { $group: { _id: "Total L.A", total: { $sum: "$POPULATION"} } }  )
{ "_id" : "Total L.A", "total" : 1296364 }
>
```

Total de la population par département

```
> db.communes.aggregate(
                                        { $group: { _id: "$NOM_DEPT", total_pop: { $sum: "$POPULATION"} } }
                                )
{ "_id" : "LORRAINE", "total_pop" : 104 }
{ "_id" : "SEINE-ET-MARNE", "total_pop" : 1338126 }
{ "_id" : "AUDE", "total_pop" : 359967 }
{ "_id" : "HAUTE-LOIRE", "total_pop" : 224907 }
{ "_id" : "MEURTHE-ET-MOSELLE", "total_pop" : 733124 }
{ "_id" : "SEINE-MARITIME", "total_pop" : 1259635 }
{ "_id" : "AVEYRON", "total_pop" : 275813 }
{ "_id" : "VOSGES", "total_pop" : 378830 }
{ "_id" : "HAUTE-SAONE", "total_pop" : 239695 }
{ "_id" : "PUY-DE-DOME", "total_pop" : 635469 }
{ "_id" : "EURE", "total_pop" : 588111 }
{ "_id" : "GERS", "total_pop" : 188893 }
{ "_id" : "CORREZE", "total_pop" : 242454 }
{ "_id" : "DORDOGNE", "total_pop" : 415168 }
{ "_id" : "MOSELLE", "total_pop" : 1045146 }
{ "_id" : "ARIEGE", "total_pop" : 152286 }
{ "_id" : "MAINE-ET-LOIRE", "total_pop" : 790343 }
{ "_id" : "LOT", "total_pop" : 174754 }
{ "_id" : "TARN-ET-GARONNE", "total_pop" : 244545 }
{ "_id" : "LOIRE", "total_pop" : 749053 }
Type "it" for more
>
```

Les mêmes, triés

```
db.communes.aggregate( [
                                { $group: { _id: "$NOM_DEPT", total_pop: { $sum: "$POPULATION"} }  },
                                { $sort: { _id: 1 } }
                        ] )
{ "_id" : "AIN", "total_pop" : 603827 }
{ "_id" : "AISNE", "total_pop" : 541302 }
{ "_id" : "ALLIER", "total_pop" : 342729 }
{ "_id" : "ALPES-DE-HAUTE-PROVENCE", "total_pop" : 160959 }
{ "_id" : "ALPES-MARITIMES", "total_pop" : 1081244 }
{ "_id" : "ARDECHE", "total_pop" : 317277 }
{ "_id" : "ARDENNES", "total_pop" : 283110 }
{ "_id" : "ARIEGE", "total_pop" : 152286 }
{ "_id" : "AUBE", "total_pop" : 303997 }
{ "_id" : "AUDE", "total_pop" : 359967 }
{ "_id" : "AVEYRON", "total_pop" : 275813 }
{ "_id" : "BAS-RHIN", "total_pop" : 1097518 }
{ "_id" : "BOUCHES-DU-RHONE", "total_pop" : 1975896 }
{ "_id" : "CALVADOS", "total_pop" : 685262 }
{ "_id" : "CANTAL", "total_pop" : 147577 }
{ "_id" : "CHARENTE", "total_pop" : 352705 }
{ "_id" : "CHARENTE-MARITIME", "total_pop" : 625682 }
Type "it" for more
```

Le champ _id est généré, il est obligatoire et il définit la clé de groupage.

Un ou plusieurs champs résultat sont définis avec une expression de calcul à effectuer selon la clé de groupage.

$group ne trie pas les documents.
Le champ _id peut être mis à une valeur nulle.

Les agrégats

$sum	la somme
$avg	la moyenne
$first	le premier document
$last	le dernier document
$max	la plus grande valeur
$min	la plus petite valeur

Sort

Trie le flux de donnés selon une ou plusieurs clé(s)

Syntaxe

```
{ $sort: { <ch1>: <sens>, <ch2>: <sens> ... } }
```
où le sens vaut 1 ou -1.

```
> use yougo
switched to db yougo
>
> db.user_type.find()
{ "_id" : ObjectId("4e65fd75e8bf1bb31c2492d9"), "code" : "CONSULTANT", "description" : "Consultant" }
{ "_id" : ObjectId("4e65fd75e8bf1bb31c2492db"), "code" : "DIRECTION", "description" : "Direction" }
{ "_id" : ObjectId("4e65fd75e8bf1bb31c2492dc"), "code" : "COMMERCIAL", "description" : "Commercial" }
{ "_id" : ObjectId("4e65fd75e8bf1bb31c2492da"), "code" : "RH", "description" : "Ressources Humaines" }
>
> // méthode déjà vue
>
> db.user_type.find().sort( { code: 1 } )
{ "_id" : ObjectId("4e65fd75e8bf1bb31c2492dc"), "code" : "COMMERCIAL", "description" : "Commercial" }
{ "_id" : ObjectId("4e65fd75e8bf1bb31c2492d9"), "code" : "CONSULTANT", "description" : "Consultant" }
{ "_id" : ObjectId("4e65fd75e8bf1bb31c2492db"), "code" : "DIRECTION", "description" : "Direction" }
{ "_id" : ObjectId("4e65fd75e8bf1bb31c2492da"), "code" : "RH", "description" : "Ressources Humaines" }
>
> db.user_type.aggregate( { $sort: { code: 1 } } )
{ "_id" : ObjectId("4e65fd75e8bf1bb31c2492dc"), "code" : "COMMERCIAL", "description" : "Commercial" }
{ "_id" : ObjectId("4e65fd75e8bf1bb31c2492d9"), "code" : "CONSULTANT", "description" : "Consultant" }
{ "_id" : ObjectId("4e65fd75e8bf1bb31c2492db"), "code" : "DIRECTION", "description" : "Direction" }
{ "_id" : ObjectId("4e65fd75e8bf1bb31c2492da"), "code" : "RH", "description" : "Ressources Humaines" }
>
```

Dans ses formes simples, l'aggregate est ± redondant vis à vis des méthodes complétant le find().

Unwind

L'opérateur $unwind génère un document pour chaque élément d'un tableau.

```
> db.commandes.find()
{ "_id" : ObjectId("59c96b83b1a80c8004b35ee1"), "article" : "xx1", "qte" : 3, "px" : 2.5, "ventes" : [ 10, 20, 30 ] }
{ "_id" : ObjectId("59c96b93b1a80c8004b35ee2"), "article" : "xx2", "qte" : 4, "px" : 6, "ventes" : [ 40, 50, 60 ] }
{ "_id" : ObjectId("59c96ba1b1a80c8004b35ee3"), "article" : "xx3", "qte" : 5, "px" : 4.2, "ventes" : [ 4, 5, 6 ] }
>
> db.commandes.aggregate( { $unwind: "$ventes" } )
{ "_id" : ObjectId("59c96b83b1a80c8004b35ee1"), "article" : "xx1", "qte" : 3, "px" : 2.5, "ventes" : 10 }
{ "_id" : ObjectId("59c96b83b1a80c8004b35ee1"), "article" : "xx1", "qte" : 3, "px" : 2.5, "ventes" : 20 }
{ "_id" : ObjectId("59c96b83b1a80c8004b35ee1"), "article" : "xx1", "qte" : 3, "px" : 2.5, "ventes" : 30 }
{ "_id" : ObjectId("59c96b93b1a80c8004b35ee2"), "article" : "xx2", "qte" : 4, "px" : 6, "ventes" : 40 }
{ "_id" : ObjectId("59c96b93b1a80c8004b35ee2"), "article" : "xx2", "qte" : 4, "px" : 6, "ventes" : 50 }
{ "_id" : ObjectId("59c96b93b1a80c8004b35ee2"), "article" : "xx2", "qte" : 4, "px" : 6, "ventes" : 60 }
{ "_id" : ObjectId("59c96ba1b1a80c8004b35ee3"), "article" : "xx3", "qte" : 5, "px" : 4.2, "ventes" : 4 }
{ "_id" : ObjectId("59c96ba1b1a80c8004b35ee3"), "article" : "xx3", "qte" : 5, "px" : 4.2, "ventes" : 5 }
{ "_id" : ObjectId("59c96ba1b1a80c8004b35ee3"), "article" : "xx3", "qte" : 5, "px" : 4.2, "ventes" : 6 }
>
```

Si le document contient un tableau vide, le document sera ignoré.

```
> db.commandes.find()
{ "_id" : ObjectId("59c96b83b1a80c8004b35ee1"), "article" : "xx1", "qte" : 3, "px" : 2.5, "ventes" : [ 10, 20, 30 ] }
{ "_id" : ObjectId("59c96b93b1a80c8004b35ee2"), "article" : "xx2", "qte" : 4, "px" : 6, "ventes" : [ 40, 50, 60 ] }
{ "_id" : ObjectId("59c96ba1b1a80c8004b35ee3"), "article" : "xx3", "qte" : 5, "px" : 4.2, "ventes" : [ 4, 5, 6 ] }
{ "_id" : ObjectId("59ca14761c2e338354c69870"), "article" : "xx4", "qte" : 10, "px" : 7.5, "ventes" : [ ] }
>
> db.commandes.aggregate( { $unwind: "$ventes" } )
{ "_id" : ObjectId("59c96b83b1a80c8004b35ee1"), "article" : "xx1", "qte" : 3, "px" : 2.5, "ventes" : 10 }
{ "_id" : ObjectId("59c96b83b1a80c8004b35ee1"), "article" : "xx1", "qte" : 3, "px" : 2.5, "ventes" : 20 }
{ "_id" : ObjectId("59c96b83b1a80c8004b35ee1"), "article" : "xx1", "qte" : 3, "px" : 2.5, "ventes" : 30 }
{ "_id" : ObjectId("59c96b93b1a80c8004b35ee2"), "article" : "xx2", "qte" : 4, "px" : 6, "ventes" : 40 }
{ "_id" : ObjectId("59c96b93b1a80c8004b35ee2"), "article" : "xx2", "qte" : 4, "px" : 6, "ventes" : 50 }
{ "_id" : ObjectId("59c96b93b1a80c8004b35ee2"), "article" : "xx2", "qte" : 4, "px" : 6, "ventes" : 60 }
{ "_id" : ObjectId("59c96ba1b1a80c8004b35ee3"), "article" : "xx3", "qte" : 5, "px" : 4.2, "ventes" : 4 }
{ "_id" : ObjectId("59c96ba1b1a80c8004b35ee3"), "article" : "xx3", "qte" : 5, "px" : 4.2, "ventes" : 5 }
{ "_id" : ObjectId("59c96ba1b1a80c8004b35ee3"), "article" : "xx3", "qte" : 5, "px" : 4.2, "ventes" : 6 }
>
```

Lookup

Réalise un 'outer join' entre deux collections.

Les collections doivent se trouver dans la même base et les collections ne peuvent pas être 'shardées'.

$lookup:

```
        {
        from: <collection to join>,
        localField: <field from the input documents>,
        foreignField: <field from the documents of the "from" collection>,
        as: <output array field>
        }
```

Exemple

```
> db
test
>
> db.commandes.find()
{ "_id" : ObjectId("59c96b83b1a80c8004b35ee1"), "article" : "xx1", "qte" : 3, "px" : 2.5, "ventes" : [ 10, 20, 30 ] }
{ "_id" : ObjectId("59c96b93b1a80c8004b35ee2"), "article" : "xx2", "qte" : 4, "px" : 6, "ventes" : [ 40, 50, 60 ] }
{ "_id" : ObjectId("59c96ba1b1a80c8004b35ee3"), "article" : "xx3", "qte" : 5, "px" : 4.2, "ventes" : [ 4, 5, 6 ] }
{ "_id" : ObjectId("59ca14761c2e338354c69870"), "article" : "xx4", "qte" : 10, "px" : 7.5, "ventes" : [ ] }
{ "_id" : ObjectId("59ca728076e78d4b1be69f30"), "article" : null, "qte" : null, "px" : null }
{ "_id" : ObjectId("59ca72cd76e78d4b1be69f31"), "article" : "xx1", "qte" : 6, "px" : 2.5, "ventes" : [ 10, 20, 30 ] }
>
> db.article.find()
{ "_id" : ObjectId("59ca731876e78d4b1be69f32"), "code" : "xx1", "designation" : "article xx1" }
{ "_id" : ObjectId("59ca732176e78d4b1be69f33"), "code" : "xx2", "designation" : "article xx2" }
{ "_id" : ObjectId("59ca732b76e78d4b1be69f34"), "code" : "xx3", "designation" : "article xx3" }
{ "_id" : ObjectId("59ca733476e78d4b1be69f35"), "code" : "xx4", "designation" : "article xx4" }
{ "_id" : ObjectId("59ca733e76e78d4b1be69f36"), "code" : "xx32", "designation" : "article xx32" }
>
> db.commandes.aggregate( [ { $lookup:
                                                {
                                                from: "article",
                                                localField: "article",
                                                foreignField: "code",
                                                as: "cmdes"
                                                }
                                        } ] )
{ "_id" : ObjectId("59c96b83b1a80c8004b35ee1"), "article" : "xx1", "qte" : 3, "px" : 2.5, "ventes" : [ 10, 20, 30 ],
"cmdes" : [ { "_id" : ObjectId("59ca731876e78d4b1be69f32"), "code" : "xx1", "designation" : "article xx1" } ] }
{ "_id" : ObjectId("59c96b93b1a80c8004b35ee2"), "article" : "xx2", "qte" : 4, "px" : 6, "ventes" : [ 40, 50, 60 ], "cmdes"
: [ { "_id" : ObjectId("59ca732176e78d4b1be69f33"), "code" : "xx2", "designation" : "article xx2" } ] }
{ "_id" : ObjectId("59c96ba1b1a80c8004b35ee3"), "article" : "xx3", "qte" : 5, "px" : 4.2, "ventes" : [ 4, 5, 6 ], "cmdes" :
[ { "_id" : ObjectId("59ca732b76e78d4b1be69f34"), "code" : "xx3", "designation" : "article xx3" } ] }
{ "_id" : ObjectId("59ca14761c2e338354c69870"), "article" : "xx4", "qte" : 10, "px" : 7.5, "ventes" : [ ], "cmdes" :
[ { "_id" : ObjectId("59ca733476e78d4b1be69f35"), "code" : "xx4", "designation" : "article xx4" } ] }
{ "_id" : ObjectId("59ca728076e78d4b1be69f30"), "article" : null, "qte" : null, "px" : null, "cmdes" : [ ] }
{ "_id" : ObjectId("59ca72cd76e78d4b1be69f31"), "article" : "xx1", "qte" : 6, "px" : 2.5, "ventes" : [ 10, 20, 30 ],
"cmdes" : [ { "_id" : ObjectId("59ca731876e78d4b1be69f32"), "code" : "xx1", "designation" : "article xx1" } ] }
```

La sortie est un peu bavarde....

```
> db.commandes.aggregate( [ { $lookup:
                                                {
                                                 from: "article",
                                                 localField: "article",
                                                 foreignField: "code",
                                                 as: "cmdes"
                                                }
                                    },
                                    { $unwind:  "$cmdes"  },
                                    { $project: { _id: 0, ventes: 0, "cmdes._id": 0, "cmdes.code": 0 } }
                                ] )
{ "article" : "xx1", "qte" : 3, "px" : 2.5, "cmdes" : { "designation" : "article xx1" } }
{ "article" : "xx2", "qte" : 4, "px" : 6, "cmdes" : { "designation" : "article xx2" } }
{ "article" : "xx3", "qte" : 5, "px" : 4.2, "cmdes" : { "designation" : "article xx3" } }
{ "article" : "xx4", "qte" : 10, "px" : 7.5, "cmdes" : { "designation" : "article xx4" } }
{ "article" : "xx1", "qte" : 6, "px" : 2.5, "cmdes" : { "designation" : "article xx1" } }
>
```

Les tableaux

Un tableau est un ensemble d'éléments :
{ t1: [elt1, elt2, ...] }

Pour insérer un tableau :
db.<NomCollection>.insert({<champ>: [elt1, elt2, etc] })

Exemple

```
> var g = { tab: ["a","b","c"] }
>
> db.Tests.insert(g)
WriteResult({ "nInserted" : 1 })
>
> db.Tests.find()
{ "_id" : ObjectId("55e40db93df985e9c7793942"), "tab" : [ "a", "b", "c" ] }
>
```

Push

L'opérateur $push, utilisé avec update() permet d' ajouter un élément dans un tableau existant.

db.<NomCollection>.update({},{ $push: { <champ1>: <valeur1>, ... } })

Exemple

```
> db.Tests.update({"_id" : ObjectId("55e40f933df985e9c7793944")}, { $push : {"tab" : "d"} } )
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
> db.Tests.find({})
{ "_id" : ObjectId("55e40f933df985e9c7793944"), "tab" : [ "a", "b", "c", "d" ] }
>
```

AddToSet

L'opérateur $addToSet, utilisé avec update() permet d' ajouter un élément à un tableau s'il n'est pas déjà présent.

db.<NomCollection>.update({},{ $addToSet: { <champ1>: <valeur1>, ... } })

Exemple

```
> db.Tests.find({})
{ "_id" : ObjectId("55e40f933df985e9c7793944"), "tab" : [ "a", "b", "c", "e", "f", "g" ] }
>
> db.Tests.update({"_id" : ObjectId("55e40f933df985e9c7793944")},{$addToSet : {"tab" : "c"} })
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 0 })
>
> db.Tests.find({})
{ "_id" : ObjectId("55e40f933df985e9c7793944"), "tab" : [ "a", "b", "c", "e", "f", "g" ] }
>
> db.Tests.update({"_id" : ObjectId("55e40f933df985e9c7793944")},{$addToSet : {"tab" : "d"} })
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
> db.Tests.find({})
{ "_id" : ObjectId("55e40f933df985e9c7793944"), "tab" : [ "a", "b", "c", "e", "f", "g", "d" ] }
>
```

Each

L'opérateur $each, utilisé avec $push ou $addToSet permet de traiter les éléments un par un.

```
db.<NomCollection>.update( {}, { $addToSet: { <champ>: { $each: [ <valeur1>,<valeur2> ... ] } } }  )
db.<NomCollection>.update( {}, { $push: { <champ>: { $each: [ <valeur1>, <valeur2> ... ] } } }  )
```

Exemple

```
> db.Tests.update({"_id" : ObjectId("55e40f933df985e9c7793944")},{$addToSet : {"tab" : ["h","i"]} })
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
> db.Tests.find({})
{ "_id" : ObjectId("55e40f933df985e9c7793944"), "tab" : [ "a", "b", "c", "e", "f", "g", "d", [ "h", "i" ] ] }
>
> db.Tests.update({"_id" : ObjectId("55e40f933df985e9c7793944")},{$addToSet : {"tab" : { $each : ["j","k"] } } })
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
> db.Tests.find({})
{ "_id" : ObjectId("55e40f933df985e9c7793944"), "tab" : [ "a", "b", "c", "e", "f", "g", "d", [ "h", "i" ], "j", "k" ] }
>
```

Position

L'opérateur $position permet d'insérer les données à partir de la position indiquée. Il s'utilise avec $push et $each

```
db.<NomCollection>.update( {},{
                              $push:  { <champ>: { $each: [ <valeur1>, <valeur2>, ... ],
                                      $position: <num> } } } )
```

Exemple

```
> db.Tests.find({})
{ "_id" : ObjectId("55e45315b3bac13cb7a6d563"), "tab" : [ "a", "b", "c", "f", "g" ] }
>
> db.Tests.update({"_id" : ObjectId("55e45315b3bac13cb7a6d563")},{$push: {"tab" :{$each:[ "e","f"],
                                                                              $position:3}} }  )
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
> db.Tests.find({})
{ "_id" : ObjectId("55e45315b3bac13cb7a6d563"), "tab" : [ "a", "b", "c", "e", "f", "f", "g" ] }
>
```

Pop

L'opérateur $pop permet de retirer le premier (arg -1) ou le dernier (arg 1) élément d'un tableau

```
db.<NomCollection>.update( {},{ $pop: { <champ>: <-1 | 1>, ... } }  )
```

```
> var g = { "tab" : ["a","b","c"] }
>
> db.Tests.insert(g)
WriteResult({ "nInserted" : 1 })
>
> db.Tests.find({})
{ "_id" : ObjectId("55e40f933df985e9c7793944"), "tab" : [ "a", "b", "c" ] }
>
> db.Tests.update({"_id" : ObjectId("55e40f933df985e9c7793944")}, { $pop : {"tab" : 1} } )
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
> db.Tests.find({})
{ "_id" : ObjectId("55e40f933df985e9c7793944"), "tab" : [ "a", "b" ] }
>
```

Pull

L'opérateur $pull permet d'enlever un élément d'un tableau suivant une valeur ou une condition spécifique

```
db.<NomCollection>.update( {},{
                                $pull:
                                        { <champ1>: <valeur|condition>,
                                        <champ2>: <valeur|condition>, ... } } )
```

Exemple

```
> db.Tests.find({})
{ "_id" : ObjectId("55e40f933df985e9c7793944"), "tab" : [ "a", "b", "c", "d", "e", "f", "g" ] }
>
> db.Tests.update({"_id" : ObjectId("55e40f933df985e9c7793944")} , {$pull : {"tab":"d"} } )
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
> db.Tests.find({})
{ "_id" : ObjectId("55e40f933df985e9c7793944"), "tab" : [ "a", "b", "c", "e", "f", "g" ] }
```

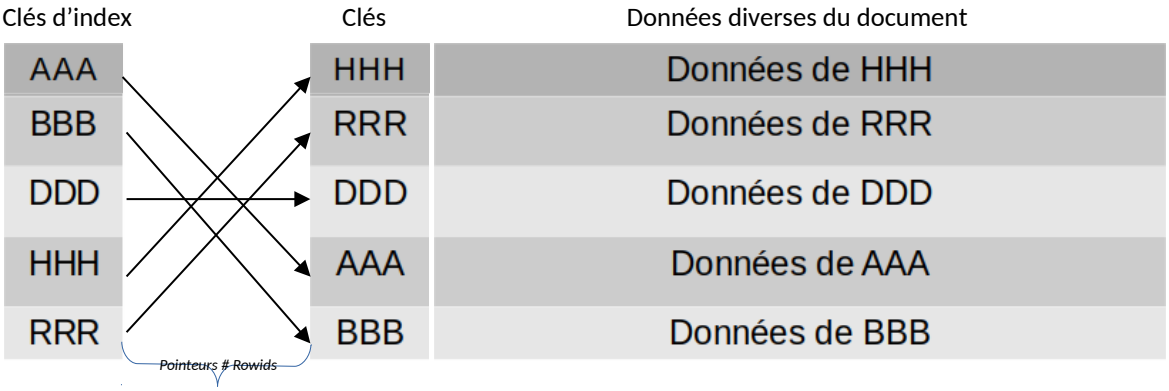
Objectifs

Type d’index
Création, réorganisation
Optimisation
Query planner
Hints

Les index – Généralités

Objectifs : Accélérer les accès en lecture des documents. Comme dans tout SGBD, les index optimisent les lectures et pénalisent les insertions / mises à jour. Sans index, les recherches de documents s'effectueraient de manière séquentielle.

Les index MongoDB sont structurés en arbre B+. Ce sont des structures adjacentes aux collections qui stockent la valeur du champ indexé et un pointeur vers la donnée. Les données sont stockées 'en tas', selon les insertions et mises à jour.



Animation : https://www.youtube.com/watch?v=coRJrcIYbF4

MongoDB génère automatiquement le champ _id, et l'indexe en tant que clé unique. Ce champ _id est donc le moyen d'accès privilégié aux données. Il existe différents types dindex :

Index simple	indexé sur un seul champ
Index composé	indexé sur plusieurs champs
Index unique	interdit les doublons (erreur en insertion)
Index multi-key	index sur les éléments d'un tableau
Index Time to live (TTL)	Index qui élimine les données obsolètes
Sparse Index	
Index Hash	
Index Text	

Index simple champ

Les index simples sont basés sur un seul champ des documents, unique ou non unique. On les oppose aux index composés qui reposent sur plusieurs champs. Exemple : dans une application d'assurance maladie, il y a un numéro de contrat (index simple, mono colonne et unique), un numéro de SS (index simple, mono colonne et unique) et les noms prénoms et date de naissance de l'assuré (index composé, potentiellement dupliqué).

Création d'index simple

```
db.<NomCollection>. createIndex({<champ>:1});
db.<NomCollection>. createIndex({<champ>:-1});
```

Options :
Trié dans l'ordre croissant (1 pour ASC)
Trié dans l'ordre descendant (-1 pour DESC)

```
> use test
switched to db test
>
> show collections
article
commandes
>
> db.article.createIndex( { code: 1 } )
{
        "createdCollectionAutomatically" : false,
        "numIndexesBefore" : 1,
        "numIndexesAfter" : 2,
        "ok" : 1
}
>
```

Index composé

Les index composés portent sur un sous-ensemble de champs de la collection.
L'ordre d'apparition des champs est important et doit correspondre à celui qui sera utilisé dans les recherches.
Un index composé peut contenir jusqu'à 31 champs maximum.
Un index portant sur le couple (nom, prenom) sera aussi un index sur le nom.

Création d'index composé

```
db.<NomCollection>.createIndex({<champ1>:1, <champ2>:1});
db.<NomCollection>.createIndex({<champ1>:-1, <champ2>:-1});
```

Options :
Tri dans l'ordre croissant (1 pour ASC)
Tri décroissant(-1 pour DESC)

```
> use communes
switched to db communes
>
> db.communes.createIndex( { NOM_COM: 1, NOM_DEPT: 1 } )
{
        "createdCollectionAutomatically" : false,
        "numIndexesBefore" : 1,
        "numIndexesAfter" : 2,
        "ok" : 1
}
>
```

Index unique

Peuvent être considérés comme des contraintes : Ils interdisent l'insertion de doublons.

```
db.<NomCollection>.createIndex({<champ>:1}, {unique: true});
db.<NomCollection>.createIndex({<champ>:-1}, {unique: true});
```

Options :
Tri dans l'ordre croissant (1 pour ASC)
Tri décroissant(-1 pour DESC)

```
> db.article.createIndex( { code: 1 }, { unique: true } )
{
        "createdCollectionAutomatically" : false,
        "numIndexesBefore" : 1,
        "numIndexesAfter" : 2,
        "ok" : 1
}
>
```

Les index multi-key

Ces index permettent d'indexer les éléments d'un tableau.

```
db.colllection.createIndex( { <champ_de_type_tableau>: < 1 or -1 > } )
```

Options :
Tri dans l'ordre croissant (1 pour ASC)
Tri décroissant(-1 pour DESC)

```
> db
test
>
> db.commandes.find()
{ "_id" : ObjectId("59c96b83b1a80c8004b35ee1"), "article" : "xx1", "qte" : 3, "px" : 2.5, "ventes" : [ { "janv" : 10 }, { "fev" : 20 }, { "mars" : 30 } ] }
{ "_id" : ObjectId("59c96b93b1a80c8004b35ee2"), "article" : "xx2", "qte" : 4, "px" : 6, "ventes" : [ { "janv" : 40 }, { "fev" : 45 }, { "mars" : 32 } ] }
{ "_id" : ObjectId("59c96ba1b1a80c8004b35ee3"), "article" : "xx3", "qte" : 5, "px" : 4.2, "ventes" : [ { "janv" : 22 }, { "fev" : 25 }, { "mars" : 23 } ] }
{ "_id" : ObjectId("59ca14761c2e338354c69870"), "article" : "xx4", "qte" : 10, "px" : 7.5, "ventes" : [ { "janv" : 32 }, { "fev" : 54 }, { "mars" : 43 } ] }
>
> // création de l'index sur le tableau des chiffes de ventes
>
> db.commandes.createIndex( { ventes: 1 } )
{
        "createdCollectionAutomatically" : false,
        "numIndexesBefore" : 1,
        "numIndexesAfter" : 2,
        "ok" : 1
}
>
> db.commandes.find( { "ventes.janv": { $gte: 20 } } )
{ "_id" : ObjectId("59c96b93b1a80c8004b35ee2"), "article" : "xx2", "qte" : 4, "px" : 6, "ventes" : [ { "janv" : 40 }, { "fev" : 45 }, { "mars" : 32 } ] }
{ "_id" : ObjectId("59c96ba1b1a80c8004b35ee3"), "article" : "xx3", "qte" : 5, "px" : 4.2, "ventes" : [ { "janv" : 22 }, { "fev" : 25 }, { "mars" : 23 } ] }
{ "_id" : ObjectId("59ca14761c2e338354c69870"), "article" : "xx4", "qte" : 10, "px" : 7.5, "ventes" : [ { "janv" : 32 }, { "fev" : 54 }, { "mars" : 43 } ] }
>
> db.commandes.find( { $and: [ { "ventes.fev": { $gte: 20 } }, { "ventes.fev": { $lte: 30 } } ] } )
{ "_id" : ObjectId("59c96b83b1a80c8004b35ee1"), "article" : "xx1", "qte" : 3, "px" : 2.5, "ventes" : [ { "janv" : 10 }, { "fev" : 20 }, { "mars" : 30 } ] }
{ "_id" : ObjectId("59c96ba1b1a80c8004b35ee3"), "article" : "xx3", "qte" : 5, "px" : 4.2, "ventes" : [ { "janv" : 22 }, { "fev" : 25 }, { "mars" : 23 } ] }
>
```

Les index Hash

Le principe consiste à appliquer une fonction de hachage à la clé d'index.

Ces index ne peuvent pas reposer sur plusieurs colonnes. Ils ne peuvent pas être unique (les fonctions de hachage ne garantissent qu'une probabilité d'unicité).

Les index hash sont très efficaces sur les conditions d'égalité, mais ne seront pas utilisés pour des requêtes d'inégalité ou d'intervalle. Il est possible, néanmoins, de créer sur le même champ des index hash et des index simples.

Ils supportent le sharding.

db.collection.createIndex({ <champ>: "hashed" })

```
> db.articles.find()
{ "_id" : ObjectId("59cb8fdd00f42afa530efe67"), "code" : "xx1", "designation" : "l'article xx1" }
{ "_id" : ObjectId("59cb8fe800f42afa530efe68"), "code" : "xx2", "designation" : "l'article xx2" }
{ "_id" : ObjectId("59cb8ff200f42afa530efe69"), "code" : "xx3", "designation" : "l'article xx3" }
{ "_id" : ObjectId("59cb8ffa00f42afa530efe6a"), "code" : "xx4", "designation" : "l'article xx4" }
>
> db.articles.createIndex({designation: "hashed"})
{
        "createdCollectionAutomatically" : false,
        "numIndexesBefore" : 1,
        "numIndexesAfter" : 2,
        "ok" : 1
}
>
```

Les sparse Index

Ils indexent uniquement les documents qui contiennent les champs sur lesquels ils sont basés, même si le champ contient la valeur 'null'
Lorsque le serveur MongoDB détecte que l'index sparse risque de retourner un résultat incomplet, il ne le choisit pas.

db.collection.createIndex({ champ: 1 }, { sparse: true })

```
>  db.articles.find()
{ "_id" : ObjectId("59cb8fdd00f42afa530efe67"), "code" : "xx1", "designation" : "l'article xx1", "famille" : "fam_x" }
{ "_id" : ObjectId("59cb8fe800f42afa530efe68"), "code" : "xx2", "designation" : "l'article xx2", "famille" : "fam_y" }
{ "_id" : ObjectId("59cb8ff200f42afa530efe69"), "code" : "xx3", "designation" : "l'article xx3", "famille" : null }
{ "_id" : ObjectId("59cb8ffa00f42afa530efe6a"), "code" : "xx4", "designation" : "l'article xx4" }
{ "_id" : ObjectId("59cba08e00f42afa530efe6b"), "code" : "xx5" }
>
> db.articles.createIndex( { famille: 1 }, { sparse: true } )
{
        "createdCollectionAutomatically" : false,
        "numIndexesBefore" : 2,
        "numIndexesAfter" : 3,
        "ok" : 1
}
>
```

Les index text

Comme dans nombre de SGBD, les index text permettent d'effectuer des recherches textuelles sur les champs de type chaînes de caractères.
Un index text par collection. Tous les mots de la chaine de caractères constitueront une entrée d'index : Attention à l'espace disque.

```
db.collection.createIndex( { champ: "text" } )
```

```
> db.articles.find()
{ "_id" : ObjectId("59cb8fdd00f42afa530efe67"), "code" : "xx1", "designation" : "l'article xx1", "famille" : "fam_x", "description" : "Le formidable article xx1" }
{ "_id" : ObjectId("59cb8fe800f42afa530efe68"), "code" : "xx2", "designation" : "l'article xx2", "famille" : "fam_y", "description" : "L'indispensable machin" }
{ "_id" : ObjectId("59cb8ff200f42afa530efe69"), "code" : "xx3", "designation" : "l'article xx3", "famille" : null, "description" : "L'incontounable bidule xx3" }
{ "_id" : ObjectId("59cb8ffa00f42afa530efe6a"), "code" : "xx4", "designation" : "l'article xx4", "description" : "Vous adorerez le bijou en plastique xx4 à seulement 1799 euros" }
{ "_id" : ObjectId("59cba08e00f42afa530efe6b"), "code" : "xx5" }
>
> db.articles.createIndex( {designation: "text" } )
{
        "createdCollectionAutomatically" : false,
        "numIndexesBefore" : 3,
        "numIndexesAfter" : 4,
        "ok" : 1
}
>
```

Les index TTL (Time to live)

Ces index suppriment automatiquement les documents obsolètes.

```
db.<nomCollection>.createIndex( { <champ>: 1 }, { expireAfterSeconds: <num> } )
```

Le champ indexé doit être un champ ISODate.
expireAfterSeconds <num> : Nombre de secondes avant de supprimer les documents.

Ne pas hésiter à nommer ces index avec l'option name et à exprimer l'expireAfterSeconds de façon lisible.

```
> db.commandes.find()
{ "_id" : ObjectId("59c96b83b1a80c8004b35ee1"), "article" : "xx1", "qte" : 3, "px" : 2.5, "dat_cde" : ISODate("2017-09-27T15:52:11.277Z") }
{ "_id" : ObjectId("59c96b93b1a80c8004b35ee2"), "article" : "xx2", "qte" : 4, "px" : 6, "dat_cde" : ISODate("2017-09-27T15:52:16.748Z") }
{ "_id" : ObjectId("59c96ba1b1a80c8004b35ee3"), "article" : "xx3", "qte" : 5, "px" : 4.2, "dat_cde" : ISODate("2017-09-27T15:52:21.644Z") }
{ "_id" : ObjectId("59ca14761c2e338354c69870"), "article" : "xx4", "qte" : 10, "px" : 7.5, "dat_cde" : ISODate("2017-09-27T15:52:26.956Z") }
>
> 60*60*24*7 = 7 jours
>
> db.commandes.createIndex( { dat_cde: 1 }, { expireAfterSeconds: (60*60*24*7) }, { name: "ttl 7 jours" } )
{
        "createdCollectionAutomatically" : false,
        "numIndexesBefore" : 3,
        "numIndexesAfter" : 4,
        "ok" : 1
}
```

Les index - Compléments

Background

Le create index peut être réalisé en arrière plan avec l'option 'background'

```
db.<nomCollection>.createIndex({ <champ>: <1|-1> }, { background: true } )
```

Name

Tous les index peuvent être nommés. Par défaut, MongoDB génère un nom à partir des caractéristiques de l'index :

les noms des champs,
le sens (1 ou -1 pour ascendant ou descendant),
le type d'index pour les index text……

Le nom généré peut devenir un peu illisible. Exemple, soit une collection contenant 3 champs de type string. On créé l'index composé suivant :

```
db.collection.createIndex(
  {
   content: "text",
   "users.comments": "text",
   "users.profiles": "text"
  }
)
```

MongoDB génère le nom **"content_text_users.comments_text_users.profiles_text"**

Pour nommer explicitement un index :

```
db.<nomCollection>.createIndex({ <champ>: <1|-1> }, { name: "<un_nom_explicite>" } )
```

ensureIndex

L'ancienne commande pour créer des index était db.<NomCollection>.ensureIndex().

Désormais, c'est un alias (depuis la version 3.0).

getIndexes

Retourne un tableau des index pour une collection :

```
> db.articles.getIndexes()
[
        {
                "v" : 2,
                "key" : {
                        "_id" : 1
                },
                "name" : "_id_",
                "ns" : "test.articles"
        },
        {
                "v" : 2,
                "key" : {
                        "designation" : "hashed"
                },
                "name" : "designation_hashed",
                "ns" : "test.articles"
        }
]
```

Pour connaître tous les Index d'une base de données

db.system.indexes.find()

REINDEX

Cette fonction provoque la reconstruction des index d'une collection.

Supprime et recrée tous les index de la collection : l'opération peut donc être coûteuse… Comme pour tous les SGBD, à envisager si la collection a changé de façon significative (nombreuses mises à jour).

```
> db.commandes.reIndex()
{
        "nIndexesWas" : 4,
        "nIndexes" : 4,
        "indexes" : [
                {
                        "v" : 2,
                        "key" : {
                                "_id" : 1
                        },
                        "name" : "_id_",
                        "ns" : "test.commandes"
                },

../..

                {
                        "v" : 2,
                        "key" : {
                                "dat_cde" : 1
                        },
                        "name" : "dat_cde_1",
                        "ns" : "test.commandes",
                        "expireAfterSeconds" : 604800
                }
        ],
        "ok" : 1
}
>
```

Drop index

La méthode dropIndex() permet de supprimer l'index indiqué (à l'exception du champ _id!)

db.<nomCollection>.dropIndex(<NomIndex>)

Pour supprimer tous les index d'une collection

db.<nomCollection>.dropIndexes()

Modification d'un index

Il n'y a pas d'instruction pour modifier un index. Détruire et reconstruire.

Optimisation

db.<collection>.find().explain()

La méthode explain() peut être appelée avec des options (queryPlanner par défaut) :

- **queryPlanner** : détaille le plan sélectionné par l'optimiseur de requête (query optimizer)
- **executionStats** : détaille les informations chiffrées du meilleur plan (durée…)
- **allPlansExecution** : détaille les informations du meilleur plan et de tous les autres.

La méthode retourne le "winning Plan" et les "rejected Plans"
Le traitement est détaillé en "stage" qui peuvent avoir plusieurs valeurs :

- COLLSCAN : scan d'une collection
- IXSCAN : scan des clès d'index
- FETCH pour des documents recuperés
- SHARD_MERGE pour des resultats mergés de shards.

```
> db.communes.find({NOM_DEPT: 'LOIRE-ATLANTIQUE'}).explain()
{
        "queryPlanner" : {
                "plannerVersion" : 1,
                "namespace" : "communes.communes",
                "indexFilterSet" : false,
                "parsedQuery" : {
                        "NOM_DEPT" : {
                                "$eq" : "LOIRE-ATLANTIQUE"
                        }
                },
                "winningPlan" : {
                        "stage" : "COLLSCAN",
                        "filter" : {
                                "NOM_DEPT" : {
                                        "$eq" : "LOIRE-ATLANTIQUE"
                                }
                        },
                        "direction" : "forward"
                },
                "rejectedPlans" : [ ]
        },
        "serverInfo" : {
                "host" : "grouik.localdomain",
                "port" : 27017,
                "version" : "3.4.3",
                "gitVersion" : "f07437fb5a6cca07c10bafa78365456eb1d6d5e1"
        },
        "ok" : 1
}
```

Dans le plan ci dessus, on constate que MongoDB réalise un COLLSCAN, balayage séquentiel de la collection (environ 36600 documents). L'indexFilterSet est à False. MongoDB n'a trouvé qu'un "winningPlan" et le tableau des "rejectedPlans" est vide.

L'analyse du plan d'exécution permettra d'identifier les index à créer, et parfois guidera le développeur à forcer l'utilisation d'un index particulier (méthode hint(), voir plus loin)

On créé un index sur le champ "NOM_DEPT"

```
> db.communes.createIndex({NOM_DEPT: 1})
{
        "createdCollectionAutomatically" : false,
        "numIndexesBefore" : 2,
        "numIndexesAfter" : 3,
        "ok" : 1
}
```

On relance la requète :

```
> db.communes.find({NOM_DEPT: 'LOIRE-ATLANTIQUE'}).explain()
{
        "queryPlanner" : {
                "plannerVersion" : 1,
                "namespace" : "communes.communes",
                "indexFilterSet" : false,
                "parsedQuery" : {
                        "NOM_DEPT" : {
                                "$eq" : "LOIRE-ATLANTIQUE"
                        }
                },
                "winningPlan" : {
                        "stage" : "FETCH",
                        "inputStage" : {
                                "stage" : "IXSCAN",
                                "keyPattern" : {
                                        "NOM_DEPT" : 1
                                },
                                "indexName" : "NOM_DEPT_1",
                                "isMultiKey" : false,
                                "multiKeyPaths" : {
                                        "NOM_DEPT" : [ ]
                                },
                                "isUnique" : false,
                                "isSparse" : false,
                                "isPartial" : false,
                                "indexVersion" : 2,
                                "direction" : "forward",
                                "indexBounds" : {
                                        "NOM_DEPT" : [
                                                "[\"LOIRE-ATLANTIQUE\", \"LOIRE-ATLANTIQUE\"]"
                                        ]
                                }
                        }
                },
                "rejectedPlans" : [ ]
        },
        "serverInfo" : {
                "host" : "grouik.localdomain",
                "port" : 27017,
                "version" : "3.4.3",
                "gitVersion" : "f07437fb5a6cca07c10bafa78365456eb1d6d5e1"
        },
        "ok" : 1
}
>
```

Après la création de l'index, on constate que MongoDB réalise un 'IXSCAN', parcours de l'index. Rappel : au niveau le plus bas du B-TREE, les feuilles sont doublement chaînées. La descente de l'arbre permet de trouver la première ville de Loire Atlantique, ensuite, il n'y a plus qu'à suivre les pointeurs des blocs de feuilles.

On trouve par ailleurs dans ce query-plan les caractéristiques de l'index (isUnique : false, isSparse : false, etc..).
Les indexBounds expriment le fait qu'on commence l'index scan avec la Loire Atlantique et on termine avec.

Le tableau des rejectedPlans est vide car la solution au problème est sans ambiguité.

On peut utiliser db.<collection>.find.explain(executionStats) pour obtenir :

- durée d'exécution de la requête
- nombre de clés lues et nombre de documents examinés.

```
> db.communes.find({NOM_DEPT: 'LOIRE-ATLANTIQUE'}).explain("executionStats")
{
        "queryPlanner" :
        ../..

        "executionStats" : {
                "executionSuccess" : true,
                "nReturned" : 221,
                "executionTimeMillis" : 0,
                "totalKeysExamined" : 221,
                "totalDocsExamined" : 221,
                "executionStages" : {
                        "stage" : "FETCH",
                        "nReturned" : 221,
                        "executionTimeMillisEstimate" : 0,
                        "works" : 222,
                        "advanced" : 221,
                        "needTime" : 0,
                        "needYield" : 0,
                        "saveState" : 1,
                        "restoreState" : 1,
                        "isEOF" : 1,
                        "invalidates" : 0,
                        "docsExamined" : 221,
                        "alreadyHasObj" : 0,
                        "inputStage" : {
                                "stage" : "IXSCAN",
                                "nReturned" : 221,
                                "executionTimeMillisEstimate" : 0,
                                "works" : 222,
                                "advanced" : 221,
                                "needTime" : 0,
                                "needYield" : 0,
                                "saveState" : 1,
                                "restoreState" : 1,
                                "isEOF" : 1,
                                "invalidates" : 0,
                                "keyPattern" : {
                                        "NOM_DEPT" : 1
                                },
                                "indexName" : "NOM_DEPT_1",
                                "isMultiKey" : false,
                                "multiKeyPaths" : {
                                        "NOM_DEPT" : [ ]
                                },
                                "isUnique" : false,
                                "isSparse" : false,
                                "isPartial" : false,
                                "indexVersion" : 2,
                                "direction" : "forward",
                                "indexBounds" : {
                                        "NOM_DEPT" : [
                                                "[\"LOIRE-ATLANTIQUE\", \"LOIRE-ATLANTIQUE\"]"
                                        ]
                                },
                                "keysExamined" : 221,
                                "seeks" : 1,
                                "dupsTested" : 0,
                                "dupsDropped" : 0,
                                "seenInvalidated" : 0
                        }
                }
        },
        "serverInfo" :
        ../..
}
>
```

L'option explain peut être aussi utilisée avec la méthode aggregate().

Note : Si la collection est une collection shardée, le document montrera aussi le travail des shards et l'opération de merging et pour les requêtes filtrées, les shards ciblés. Voir chapitre "Le sharding"

La requète suivante réalise : somme de la population par département,
filtrage des département les moins peuplés et tri final.

```
root@127.0.0.1:27017:communes> db.communes.aggregate( [
...                                 { $group: { _id: "$NOM_DEPT", totalPop: { $sum: "$POPULATION" } } },
...                                 { $match: { totalPop: { $gte: 1500*1000 } } },
...                                 { $sort: { "totalPop" : -1 } }
...                                ])
{ "_id" : "NORD", "totalPop" : 2579208 }
{ "_id" : "PARIS", "totalPop" : 2249975 }
{ "_id" : "BOUCHES-DU-RHONE", "totalPop" : 1975896 }
{ "_id" : "RHONE", "totalPop" : 1744236 }
{ "_id" : "HAUTS-DE-SEINE", "totalPop" : 1581628 }
{ "_id" : "SEINE-SAINT-DENIS", "totalPop" : 1529928 }
root@127.0.0.1:27017:communes>
```

L'option explain met en évidence les étapes du 'pipe' aggregate :

```
root@127.0.0.1:27017:communes> db.communes.aggregate( [
                                                        { $group: { _id: "$NOM_DEPT", totalPop: { $sum: "$POPULATION" } } },
                                                        { $match: { totalPop: { $gte: 1500*1000 } } },
                                                        { $sort: { "totalPop" : -1 } }

                                                      ], {explain :true} )

{
        "stages" : [
                {
                        "$cursor" : {
                                "query" : {

                                },
                                "fields" : {
                                        "NOM_DEPT" : 1,
                                        "POPULATION" : 1,
                                        "_id" : 0
                                },
                                "queryPlanner" : {
                                        "plannerVersion" : 1,
                                        "namespace" : "communes.communes",
                                        "indexFilterSet" : false,
                                        "parsedQuery" : {

                                        },
                                        "winningPlan" : {
                                                "stage" : "COLLSCAN",
                                                "direction" : "forward"
                                        },
                                        "rejectedPlans" : [ ]
                                }
                        }
                },
                {
                        "$group" : {
                                "_id" : "$NOM_DEPT",
                                "totalPop" : {
                                        "$sum" : "$POPULATION"
                                }
                        }
                },
                {
                        "$match" : {
                                "totalPop" : {
                                        "$gte" : 1500000
                                }
                        }
                },
                {
                        "$sort" : {
                                "sortKey" : {
                                        "totalPop" : -1
                                }
                        }
                }
        ],
        "ok" : 1
}
root@127.0.0.1:27017:communes>
```

Note : L'aggregate précédent est l'équivalent de la requète SQL :

```
Select nom_dept, sum(population) totalPop
                                 from communes
                                 group by nom_dept
                                 having totalPop >= 1500*1000
                                 order by totalPop desc;
```

Le SQL possède sa méthode explain. On pourrait constater que, sur une question triviale comme celle là, SQL réaliserait les mêmes étapes. Le framework aggregate a quand même cet avantage de bien sérier les étapes.

Les Hints

La méthode hint() peut être utilisée pour forcer l'optimiseur à utiliser un index pour conduire l'exécution d'une requête find(). On précise le champ indexé à utiliser ou on désigne l'index par son nom (option { name: ….. } du createIndex)

db.<NomCollection>.find().hint({ Champ_Index: 1})
db.<NomCollection>.find().hint("NomIndex")

Cette commande incite l'optimiseur à utiliser l'index sur 'zipcode' pour exécuter la requête.

```
> db.people.find( { name: "John Doe", zipcode: { $in: "63000" } }).hint( { zipcode: 1 } )
```

Combinaison avec explain()

```
> db.<NomCollection>.explain("executionStats").find({ <query> }).hint( <index> )
```

Objectifs

Gérer les utilisateurs : créer, modifier, supprimer
Gérer les rôles

Introduction

Par défaut, après l'installation, MongoDB se repose sur l'OS pour l'authentification. Rapidement, la nécessité s'impose de créer des utilisateurs au sens MongoDB : chaque base de données possède sa liste d'utilisateurs habilités.

On créé les utilisateurs au niveau des bases et on active l'authentification au niveau de l'instance. La connexion est possible au vu du couple 'nom_utilisateur/mot_de_passe'.

Les droits sur les bases s'appellent des rôles. Il existe le rôle 'root' qui donne les droits sur toutes les bases. Avant d'activer la sécurité sur un serveur, il est nécessaire qu'au moins un 'superuser' soit créé et sécurisé avec son mot de passe.

Création d'un administrateur

```
> use admin
> db.createUser( { user: "root", pwd: "root", customData: { mail: "admin@orga.org" } , roles: [ "root" ] } )
Successfully added user: {
        "user" : "root",
        "customData" : {
                "mail" : "admin@orga.org",
                "bureau" : "1234"
        },
        "roles" : [
                "root"
        ]
}
>
```

Une fois créé un utilisateur 'admin', on peut redémarrer le serveur mongod en mode sécurisé

- en ajoutant l'option --auth lorsqu'on lance mongod depuis la ligne de commandes.
- en paramétrant le fichier /etc/mongod.conf : dé commenter la section 'security' et basculer l'option 'authorization' à 'enabled'

```
[root@grouik pierrick]# mongod ../.. --auth
[root@grouik pierrick]#
[root@grouik pierrick]#  # autre solution modifier /etc/mongod.conf  systemctl restart mongod.service
[root@grouik pierrick]# grep -C1 auth /etc/mongod.conf
security:
  # Private key for cluster authentication
  #keyFile: <string>

  # Run with/without security (enabled|disabled, disabled by default)
  authorization: enabled
```

Lorsque le serveur est démarré en mode sécurisé, on peut se connecter à mongo sans s'authentifier, mais, on ne peut rien faire....

```
[root@grouik pierrick]# mongo --quiet
> show dbs
2017-09-29T18:45:32.343+0200 E QUERY    [thread1] Error: listDatabases failed:{
        "ok" : 0,
        "errmsg" : "not authorized on admin to execute command { listDatabases: 1.0 }",
        "code" : 13,
        "codeName" : "Unauthorized"
} :
_getErrorWithCode@src/mongo/shell/utils.js:25:13
Mongo.prototype.getDBs@src/mongo/shell/mongo.js:62:1
shellHelper.show@src/mongo/shell/utils.js:769:19
shellHelper@src/mongo/shell/utils.js:659:15
@(shellhelp2):1:1
>
```

Création des utilisateurs

Pour ajouter un utilisateur, il faut avoir les droits createUser. Si l'utilisateur existe déjà, une erreur "duplicate user" est renvoyée.

```
>use <nomBase>
>db.createUser(util)
>
```

util est un document JSON qui contient les informations suivantes :

```
{ user: "nom",
        pwd: "mot_de_passe",
        customData: { "informations quelconques" },
        roles [ { rôle: "rôle", db: "base"}, {../..}  ]
}
```

Pour créer l'utilisateur 'jean' dans la base test avec une liste de rôles

```
> db
test
>
> db.createUser( {user: "jean", pwd: "jean", roles: [ {role: "readWrite", db: "test"},
                                                      {role: "read", db: "yougo"} ] } )
Successfully added user: {
        "user" : "jean",
        "roles" : [
                {
                        "role" : "readWrite",
                        "db" : "test"
                },
                {
                        "role" : "read",
                        "db" : "yougo"
                }
        ]
}
>
```

Pour modifier le mot de passe d'un utilisateur, il faut avoir les droits changeAnyPassword sur la base de données concernée.

```
>use <nomBase>
>db.changeUserPassword("nomUtil", "mdp")
>
```

Pour s'authentifier dès la connexion, se connecter à la base dans laquelle l'utilisateur est déclaré, ou utiliser l'option 'authentication Database'

```
[root@grouik pierrick]# mongo -u root -p root admin --quiet
>
> show collections
system.users
system.version
>
```

```
root@grouik pierrick]# mongo -u root -p root communes --authenticationDatabase admin --quiet
>
```

Pour s'authentifier après connexion :

```
> use admin
switched to db admin
>
> db.auth("root", "root")
1
>
> show dbs
admin     0.000GB
communes  0.033GB
local     0.000GB
test      0.000GB
villes    0.013GB
yougo     0.000GB
>
```

Liste des utilisateurs

Collection system.users

```
> use admin
switched to db admin
> db.system.users.find()
{ "_id" : "admin.pierrick", "user" : "pierrick", "db" : "admin", "credentials" : { "SCRAM-SHA-1" :
{ "iterationCount" : 10000, "salt" : "QzfQaoN1SlEA/NcuBn7Bvw==", "storedKey" :
"19462TINYB9mSG90bs60qQV8Ysw=", "serverKey" : "DRHaWlif8JO31UcLhz9Us/bQX0o=" } }, "roles" :
[ ] }
{ "_id" : "admin.root", "user" : "root", "db" : "admin", "credentials" : { "SCRAM-SHA-1" :
{ "iterationCount" : 10000, "salt" : "g8FEM3sE3rCz1+SqJTlgiQ==", "storedKey" :
"CgmcwaJBYqknsziGy7JZ4i7WATY=", "serverKey" : "1uaU7anCYLIXgY2w9KhO/0ekId4=" } }, "roles" :
[ { "role" : "root", "db" : "admin" } ] }
>
```

Methode show users

```
> use admin
switched to db admin
> show users
{
        "_id" : "admin.pierrick",
        "user" : "pierrick",
        "db" : "admin",
        "roles" : [ ]
}
{
        "_id" : "admin.root",
        "user" : "root",
        "db" : "admin",
        "roles" : [
                {
                        "role" : "root",
                        "db" : "admin"
                }
        ]
}
```

```
> use test
switched to db test
> show users
>
```

Modification d’un compte utilisateur

```
>use <nomBase>
>db.updateUser("nomUtil",update)
>
```

Où update est un document qui contient les informations suivantes :

```
{
        pwd: "<password en clair>",
        customData: { <any information> },
        roles: [
                { role: "<role>", db: "<baseconcerne>" } | "<role>", ...
        ]
}
```

Attention, pour ajouter ou retirer un rôle aux rôles déjà existants, il faut utiliser les méthodes db.grantRolesToUser() ou db.revokeRolesFromUser().
Ou utiliser updateUser() en reprenant la totalité des rôles dans la liste des rôles.

Suppression d'un compte utilisateur

```
> use <base>
> db.dropUser("nom_util")
>
```

Les rôles

Introduction

Un rôle est un ensemble de privilèges qui combinent une ressource (instance, base, collection, user) avec une (des) opération(s) autorisée(s). Ex : createUser, dropDatabase…

Chaque privilège est spécifié explicitement ou hérité d'un autre rôle.

A l'exception des rôles créés dans la base admin, un rôle inclut des privilèges qui s'appliquent sur la base de données d'attache ou hérite d'autres rôles de la même base.
Un rôle créé dans la base admin possède des privilèges qui s'appliquent sur toutes les bases et peut hériter de tous les rôles.

Un utilisateur est déclaré dans une (ou plusieurs) base(s). Un utilisateur bénéficiant d'un rôle reçoit tous les privilèges de ce rôle. Un utilisateur peut avoir de multiples rôles et des rôles différents sur différentes bases de données.

Les rôles autorisent toujours des privilèges et ne limitent jamais d'accès.
Par exemple, si un utilisateur a les rôles read et readWriteAnyDatabase, le plus permissif s'applique.

read	Autorise les utilisateurs à lire les données de toutes les collections de la base de données d'attache. La commande find() peut être utilisée.
readWrite	Autorise les utilisateurs à lire et écrire dans toute collection de la base de données d'attache. Find(), insert(), remove() et update()
dbAdmin	Tous les droits sur les objets de la base.
userAdmin	Autorise les utilisateurs à lire et écrire dans la collection system.user. Modification des utilisateurs existants et création de nouveaux. userAdmin est le superuser de la base concernée. L'utilisateur avec le droit userAdmin peut s'autoriser tous les privilèges. Néanmoins, l'utilisateur ne peut explicitement autoriser un utilisateur à avoir des privilèges au delà de l'administration de la base courante.
readAnyDatabase	autorise les utilisateurs à lire des données de toutes les bases de données de l'instance
userAdminAnyDatabase	autorise les opérations admin sur toutes les bases de données présentes.

Grant / revoke

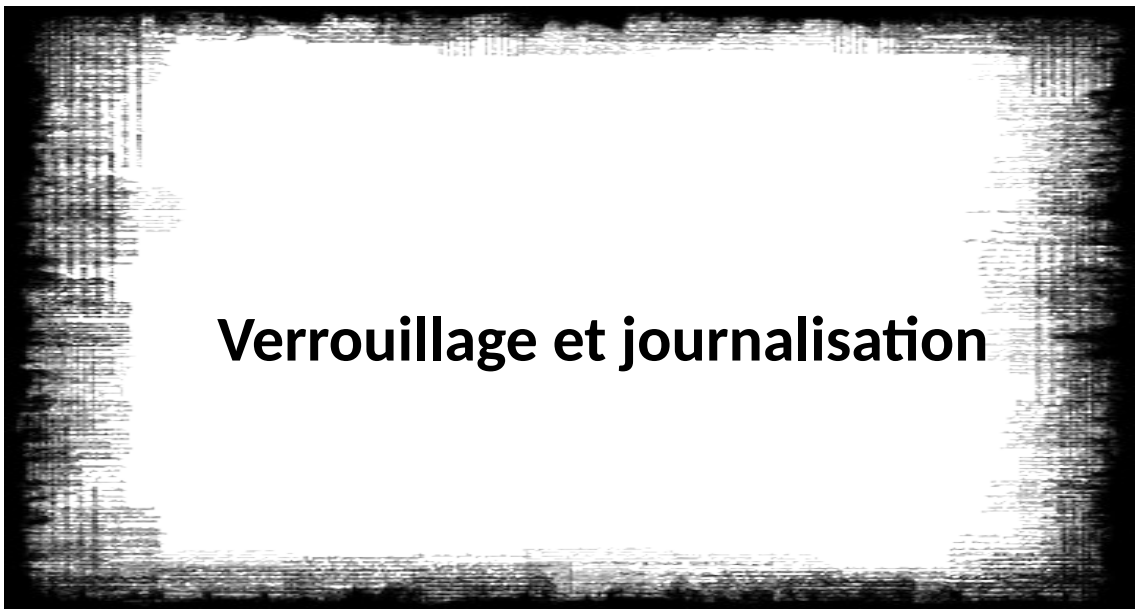
Pour pouvoir ajouter/enlever des rôles à un utilisateur, il faut avoir le droit grantRole/revokeRole.

Pour ajouter un (ou des) rôle(s) à un utilisateur
db.grantRolesToUser(<util>, [<role>, ...], { <writeConcern> })

Pour enlever un (ou des) rôle(s) à un utilisateur
db.revokeRolesFromUser(<util>, [<role>, ...], { <writeConcern> })

Ex :Ajout du droit readWrite sur la base infos pour l'utilisateur util1

```
> db.grantRolesToUser("util1", [ {"role" : "readWrite", "db" : "infos" } ] )
```



Objectifs

Types de verrous
Journalisation
Recovery

Verrouillage

MongoDB offre différents moyens pour gérer la concurrence d'accès. Les maîtres mots sont verrouillage, isolation et MVCC (Multi Version Concurency Control).

Un verrou se caractérise par son type et sa granularité.

Types de verrous

Exclusif (X)	Un verrou exclusif est posé lors d'une opération d'écriture. Il interdit l'accès au document à toute autre session concurrente.
Partagé (S)	Un verrou partagé est posé lors d'une opération de lecture. Il autorise les accès concurrents en lecture mais pas en écriture.
Intention (IS / IX)	Un verrou d'intention est un verrou posé sur une structure plus importante que celle utilisée. Par exemple un verrou sur une base de données pour la modification d'une collection.

Opération	**Type de verrou**
Initier une requête	Verrou de lecture (S)
Obtenir plus de données d'un curseur	Verrou de lecture
Insérer des données	Verrou d'écriture (X)
Supprimer des données	Verrou d'écriture
Mettre à jour des données	Verrou d'écriture
Créer un index	Construire un index en avant plan (valeur par défaut) verrouille la base de données. Durée en fonction de la taille de la collection.
Aggregate	Verrou de lecture

Granularité
Verrous d'instance, de bases, de collections, de documents. Le niveau auquel joue le verrou dépend des opérations mais aussi du moteur utilisé.

MMAPv1 :
Depuis la version 3.0, les verrous sont posés au niveau de la collection. Avant, une opération d'écriture sur une collection verrouillait la base.

WiredTiger :
WiredTiger pose des verrous d'intention au niveau collection, et des verrous exclusifs au niveau document. En cas de conflit, l'opération est renouvelée. (tentée)

Les verrous posés à un moment donné sont identifiables à l'aide des outils mongotop et mongostat et des commandes db.serverStatus() et db.currentOp().
Pour mettre fin à une opération, utiliser db.killOp()

La journalisation

Comme dans nombre de SGBD, la journalisation assure le durabilité des mises à jour effectuées. MongoDB utilise la technique WAL 'Write Ahead Logging' :

Le système écrit les résultats des insert(), update() et delete() en mémoire avant de les écrire dans les data files. (Le sync intervient toutes les 30ms pour MMAPv1 ou 100ms pour wiredTiger).

En parallèle, le journal des 'transactions' est écrit (historique des opérations de mise à jour). Dès que le journal est écrit, le système rend la main à l'application. Cette technique permet une meilleure réactivité. L'écriture étant atomique (complète ou pas du tout), la cohérence est assurée.

La journalisation devrait être systématiquement active en production.

Les journaux de transactions sont supprimés lors d'un arrêt propre du serveur. Par contre, en cas de crash, les journaux persistent. Au redémarrage, ces journaux sont 'rejoués' et les bases reconstituées dans le dernier état cohérent connu.

La journalisation est activée par défaut : (extrait du fichier de configuration /etc/mongod.conf)

```
../..

# storage Options - How and Where to store data
storage:
  # Directory for datafiles (defaults to /data/db/)
  dbPath: /opt/mongod
  directoryPerDB: true

  #journal:
    # Enable/Disable journaling (journaling is on by default for 64 bit)
    #enabled: true

../..
```

Les journaux se trouvent dans le répertoire 'journal' du dbpath.

```
[root@grouik communes]# ls -l /opt/mongod/journal
total 214440
-rw-r--r--. 1 mongod mongod   9868672 28 sept. 13:33 WiredTigerLog.0000000009
-rw-r--r--. 1 mongod mongod 104857600 28 sept. 10:51 WiredTigerPreplog.0000000001
-rw-r--r--. 1 mongod mongod 104857600 28 sept. 10:51 WiredTigerPreplog.0000000002
[root@grouik communes]#
```

A l'initialisation d'un serveur MongoDB, les journaux sont initialisés, ce qui peut prendre un certain temps. Les fichiers journaux ont une taille de 1Go par défaut. Lorsque cette limite est atteinte, MongoDB créé un nouveau fichier. Les fichiers sont effacés au fur et à mesure des synchronisations des data files. Le nombre de fichiers présents dans le répertoire 'journal' est proportionnel à l'activité transactionnelle.

Recovery

Après un crash, lors du redémarrage, MongoDB rejoue les journaux. Si cette phase de 'recovery' échoue, tenter la commande db.repairDatabase(). Généralement la phase de recovery automatique suffit. Le repairDatabase() s'impose, par exemple, lors de corruption de file system Linux ou d'erreurs physiques au niveau disque.

Il est aussi possible d'exécuter mongod directement depuis le shell linux

```
$ mongod  --repair
```

Depuis mongo ou depuis le shell, la commande repairDatabase pose un verrou exclusif global. Pas de connexion pendant l'opération. La commande réclame de la place libre à hauteur du volume de la base + 2Go. Il est possible, si nécessaire de monter un volume dans l'arborescence Linux et de préciser :

```
$ mongod  --repair --repairPath <repertoire>
```

Ne pas utiliser 'db.repairDatabase()' sur un membre de replica set. Chercher plutôt à resynchroniser avec un autre replica set intact.

Avec MMAPv1, la commande db.repairDatabase() compacte les fichiers de données et permet, éventuellement, de récupérer de l'espace disque.
Pas avec WiredTiger.

db.repairDatabase est un alias pour la commande :

db.runCommand({ repairDatabase: 1, preserveClonedFilesOnFailure: <boolean>, backupOriginalFiles: <boolean> })

Options :
Ces deux options sont ± obsolètes. Elles ne concernent que le moteur MMAPv1.

Consulter la documentation pour plus d'informations :
https://docs.mongodb.com/manual/reference/command/repairDatabase/

preserveClonedFilesOnFailure: <boolean> :

Lorsque le boolean est "true", les fichiers temporaires dans le répertoire de sauvegarde ne sont pas supprimés.

backupOriginalFiles: <boolean> :

Lorsque le boolean est "true", les anciens fichiers de base de données sont déplacés dans le répertoire de sauvegarde au lieu de les supprimer.

Objectifs

currentOp, killOp
le profiler
mongostat, mongotop

Introduction

Deux méthodes disponibles : db.currentOp() et db.killOp(). Ces méthodes permettent de contrôler le déroulement des opérations en cours sur un serveur MongoDB.

Utiles, par exemple, dans le cas de créations d'index en arrière-plan.

currentOp

La commande retourne un document "inprog" :

```
> db.insee.createIndex( { Commune: 1 }, { background: true })
{
        "createdCollectionAutomatically" : false,
        "numIndexesBefore" : 1,
        "numIndexesAfter" : 2,
        "ok" : 1
}
>
> db.currentOp()
{
        "inprog" : [
                {
                        "desc" : "conn8",
                        "threadId" : "139818096727808",
                        "connectionId" : 8,
                        "client" : "127.0.0.1:37038",
                        "appName" : "MongoDB Shell",
                        "clientMetadata" : {
                                "application" : {
                                        "name" : "MongoDB Shell"
                                },
                                "driver" : {
                                        "name" : "MongoDB Internal Client",
                                        "version" : "3.4.9"
                                },
                                "os" : {
                                        "type" : "Linux",
                                        "name" : "Fedora release 25 (Twenty Five)",
                                        "architecture" : "x86_64",
                                        "version" : "Kernel 4.12.11-200.fc25.x86_64"
                                }
                        },
                        "active" : true,
                        "opid" : 26063,
                        "secs_running" : 0,
                        "microsecs_running" : NumberLong(13),
                        "op" : "command",
                        "ns" : "admin.$cmd",
                        "query" : {
                                "currentOp" : 1
                        },
                        "numYields" : 0,
                        "locks" : {

                        },
                        "waitingForLock" : false,
                        "lockStats" : {

                        }
                }
        ],
        "ok" : 1
}
>
```

Un tableau inprog vide signifie qu'il n'y a pas d'opérations actives.

Pour connaître l'ensemble des opérations en cours, l'option true est passée en paramètre.

```
> db.currentOp(true)
{
 "inprog" : [ { "desc" : "conn3",   "threadId" : "0xb48e2e0",   "connectionId" : 3,   "active" : false },
              { "desc" : "conn2",  "threadId" : "0xb48df80", "connectionId" : 2,
                "opid" : 361,   "active" : false,   "op" : "query",   "ns" : "",
                "query" : { "isMaster" : 1,  "forShell" : 1  },   "client" : "127.0.0.1:45042",
                "numYields" : 0,   "locks" : {   },
                "waitingForLock" : false,    "lockStats" : {     }
    },
....
 } ]}
```

db.currentOp(<query>) permet de filtrer les opérations recherchées :

opid	identifiant de l'opération
op	type de l'opération en cours, "none", "update", "insert", "query", "getmore", "remove", "killcursors"
active	true / false, si false opération terminée ou en attente de verrou
ns	le namespace cible de l'opération

KillOp

La commande termine l'opération identifiée par son "opid"

db.killOp(<opid>)

Note : La réponse peut ne pas être immédiate, en particulier dans les cas de création d'index.
Il est fortement déconseillé de terminer une opération système interne au serveur.
db.killOp() ne doit être utilisée que pour terminer des opérations utilisateurs.

Le profiler

Le profiler est un composant MongoDB qui enregistre les statistiques des opérations de lecture / écriture effectuées par le serveur.

Par défaut, le profiler n'est pas configuré car gros consommateur de ressources.
Il est donc conseillé de ne l'utiliser que de façon très ponctuelle pour tracer les bugs ou isoler les problèmes de performances.

L'activation du profiler se fait au niveau de la base. Le serveur créé une collection cappée nommée system.profile, de 1Mo par défaut.

Activation du profiler

Au lancement du serveur (pour les plateformes de développement ou de tests) :

```
[root@grouik MongoDB]# mongod .../... --profile <niveau> --slowms <durée>
```

En général, le profiler est activé au niveau session par la fonction :

db.setProfilingLevel(<level>, <slowOpThresholdMs>)

Cette fonction admet 2 arguments : le profile level et le seuil des opérations "lentes".

Profile Level	Comportement	
0	Désactivé	-
1	Activé	Opérations lentes
2	Activé	Toutes opérations

Le seuil des opérations "lentes" est défini en millisecondes (100, par défaut).

```
root@127.0.0.1:27017:communes> db.setProfilingLevel(1, 500)
{ "was" : 0, "slowms" : 100, "ok" : 1 }
root@127.0.0.1:27017:communes>
```

Ici, toutes les opérations qui dureront plus de 500 ms seront tracées par le profiler. (ie, un document par opération dans la collection system.profile de la base courante)

Consultation des statistiques

Tout simplement par la consultation de la collection 'system.profile'

db.system.profile.find().pretty()

```
root@127.0.0.1:27017:test> db.setProfilingLevel(2)
{ "was" : 0, "slowms" : 500, "ok" : 1 }
root@127.0.0.1:27017:test>  db.foo.insert({x:1})
WriteResult({ "nInserted" : 1 })
root@127.0.0.1:27017:test>  db.foo.update({},{$set:{x:2}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
root@127.0.0.1:27017:test> db.foo.remove({})
WriteResult({ "nRemoved" : 1 })
root@127.0.0.1:27017:test> db.system.profile.find().pretty()
{
        "op" : "insert",
        "ns" : "test.foo",
        "query" : {
                "insert" : "foo",
                "documents" : [
                        {
                                "_id" : ObjectId("59d267ad88fa0de23a2d4ed9"),
                                "x" : 1
                        }
                ],
../..

        "millis" : 222,
        "ts" : ISODate("2017-10-02T16:22:05.362Z"),
        "client" : "127.0.0.1",
        "appName" : "MongoDB Shell",
        "allUsers" : [
                {
                        "user" : "root",
                        "db" : "admin"
                }
        ],
        "user" : "root@admin"
}
{
        "op" : "update",
        "ns" : "test.foo",
        .../..
        "millis" : 0,
        "planSummary" : "COLLSCAN",
../..
root@127.0.0.1:27017:test>
```

Il est également possible d'utiliser la commande show profile.

```
root@127.0.0.1:27017:test> show profile

insert    test.foo 222ms Mon Oct 02 2017 18:22:05
query:{
        "insert" : "foo",
        "documents" : [
                {
                        "_id" : ObjectId("59d267ad88fa0de23a2d4ed9"),
                        "x" : 1
                }
        ],
        "ordered" : true
} ninserted:1 keysInserted:1 numYield:0 locks:{
        "Global" : {
                "acquireCount" : {
                        "r" : NumberLong(3),
                        "w" : NumberLong(3)
                }
        },
        "Database" : {
                "acquireCount" : {
                        "w" : NumberLong(2),
                        "W" : NumberLong(1)
                }
        },
        "Collection" : {
                "acquireCount" : {
                        "w" : NumberLong(2)
                }
        }
} responseLength:29 protocol:op_command client:127.0.0.1 appName:MongoDB Shell allUsers:[ { "user" : "root", "db" :
"admin" } ] user:root@admin


root@127.0.0.1:27017:test>
```

On peut évidemment filtrer la méthode find() :

```
root@127.0.0.1:27017:test> db.system.profile.find( { millis: { $gt: 100 } } ).pretty()
{
        "op" : "insert",
        "ns" : "test.foo",
        "query" : {
                "insert" : "foo",
                "documents" : [
                        {
                                "_id" : ObjectId("59d267ad88fa0de23a2d4ed9"),
                                "x" : 1
                        }
                ],
../..
```

La methode db.getProfilingLevel() permet d'afficher le niveau de profiling courant.

Mongostat

Mongostat est un utilitaire à lancer depuis le shell linux. La commande affiche quelques statistiques du serveur, rafraichies à n secondes d'intervalle (n argument de la commande).

Mongostat peut être utilisé sur un replica set ou sur un cluster shardé.

https://docs.mongodb.com/manual/reference/program/mongostat/

Les informations retournées par mongostat :

insert, query, update, delete	Opérations CRUD – Nombre d'occurrences à la seconde
getmore	Balayages de curseur – Nbre d'occurrences à la seconde
command	Commandes serveur– Nbre d'occurrences à la seconde
dirty	Cache Wired Tiger. % Dirty pages
used	Cache Wired Tiger. % utilisation
flushes	Nbre de "flush" des données cachées sur le disque (checkpoints / fsync)
vsize	Quantite de mémoire virtuelle utilisée.
res	Quantité de mémoire que mongod utilise ± Mémoire totale du serveur.
idx miss %	Taux de défauts de page en lecture d'index.
qrw	Taille de la file d'attente pour les lectures / écritures
arw	Opérations de lectures / écritures actives
netIn	Trafic réseau entrant. (b)ytes, (k)ilos, (m)egas..)
netOut	Trafic réseau sortant.
conn	Nombre de connexions ouvertes.
time	Timestamp du relevé statistique

```
[root@grouik MongoDB]# mongostat -u root -p root --authenticationDatabase admin 5
insert query update delete getmore command  dirty used flushes vsize      res qrw arw  net_in net_out conn                 time
    *0    *0     *0     *0       0       0|0 0.0% 0.3%       0  974M 69.0M 0|0 1|0     31b    9.10k    2  Oct  2 19:43:54.768
    *0    *0     *0     *0       0       0|0 0.0% 0.3%       0  974M 69.0M 0|0 1|0     31b    9.10k    2  Oct  2 19:43:59.768
    *0    *0     *0     *0       0       0|0 0.0% 0.3%       0  975M 69.0M 0|0 1|0     31b    9.10k    2  Oct  2 19:44:04.767
    *0    *0     *0     *0       0       0|0 0.0% 0.3%       0  975M 69.0M 0|0 1|0     54b    9.14k    3  Oct  2 19:44:09.762
    *0    *0     *0     *0       0       0|0 0.0% 0.3%       0  975M 69.0M 0|0 1|0     31b    9.10k    3  Oct  2 19:44:14.765
    *0    *0     *0     *0       0       0|0 0.0% 0.3%       0  975M 69.0M 0|0 1|0     31b    9.10k    3  Oct  2 19:44:19.768
    *0    *0     *0     *0       0       1|0 0.0% 0.3%       0  975M 69.0M 0|0 1|0    154b    9.23k    3  Oct  2 19:44:24.774
```

Mongotop

Temps passé (en ms) par les opérations sur les collections toutes les n secondes.

```
[root@grouik MongoDB]# mongotop -u root -p root --authenticationDatabase admin 5
2017-10-02T20:14:53.709+0200    connected to: 127.0.0.1

ns                              total    read    write    2017-10-02T20:14:58+02:00
admin.system.roles               0ms     0ms      0ms
admin.system.users               0ms     0ms      0ms
admin.system.version             0ms     0ms      0ms
communes.commmunes               0ms     0ms      0ms
communes.communes                0ms     0ms      0ms
communes.insee                   0ms     0ms      0ms
communes.system.profile          0ms     0ms      0ms
../..

ns                              total    read    write    2017-10-02T20:15:03+02:00
admin.system.roles               0ms     0ms      0ms
admin.system.users               0ms     0ms      0ms
admin.system.version             0ms     0ms      0ms
../..

^C2017-10-02T20:15:06.074+0200          signal 'interrupt' received; forcefully terminating
[root@grouik MongoDB]#
```

https://docs.mongodb.com/manual/reference/program/mongotop/

Objectifs

Sauvegardes à froid, à chaud
Dump, restore
Exports, imports

Généralités

- Sauvegardes à froid
- Sauvegardes en ligne
- Export des données

La procédure de sauvegarde choisie induit la procédure de restauration.

Sauvegardes à froid

Comme pour tout SGBD, il est possible de :

- Arrêter l'instance
- Copier tous les fichiers du dbpath avec les outils de l'OS (cp, scp, tar…)
- Redémarrer

Sauvegardes en ligne

Sauvegarde dite à chaud,

- Verrouiller le serveur : db.fsyncLock() réalise un flush et met les écritures en attente.
- Copier tous les fichiers du dbpath avec les outils de l'OS (cp, scp, tar…)
- Relancer les écritures : db.fsyncUnlock()

Notes :

Attention : l'ensemble du serveur est verrouillé (toutes les bases…). Il se peut que les lectures soient elles même bloquées, notamment, les lectures nécessaires à l'authentification.
Conséquence : la sauvegarde en ligne d'une base peut empêcher la connexion à une autre.

Il est donc préférable de garder ouverte la session qui effectue le fsyncLock() pour réaliser le fsyncUnlock(). S'il y a une déconnexion, il se peut qu'il soit impossible de se reconnecter.

La commande fsyncLock() ne fonctionne pas avec WiredTiger avant la version 3.2.

Cette commande est un raccourci pour :

```
> use admin
> db.runCommand( { fsync : 1, lock : true })
>
```

Un redémarrage du serveur déverrouille le cluster (fsyncUnlock() + fsync synchrone implicite lors de l'arrêt)

Dump MongoDB

On utilise la commande mongodump. Son pendant est la commande mongorestore.

Dump possible aux niveaux instance, base ou collection. Il en résulte : un répertoire ‘dump’ qui contient un répertoire par base de données et un fichier par collection et index.

Dump de l’instance :

```
root@grouik MongoDB]# mongodump
2017-09-28T11:04:28.596+0200    writing admin.system.version to
2017-09-28T11:04:28.596+0200    done dumping admin.system.version (1 document)
2017-09-28T11:04:28.596+0200    writing communes.insee to
2017-09-28T11:04:28.596+0200    writing communes.communes to
2017-09-28T11:04:28.596+0200    writing yougo.users to
2017-09-28T11:04:28.596+0200    writing test.commandes to
2017-09-28T11:04:28.597+0200    done dumping test.commandes (4 documents)
2017-09-28T11:04:28.597+0200    writing yougo.user_type to
2017-09-28T11:04:28.597+0200    done dumping yougo.users (80 documents)
2017-09-28T11:04:28.597+0200    writing test.articles to
2017-09-28T11:04:28.597+0200    done dumping yougo.user_type (4 documents)
2017-09-28T11:04:28.597+0200    done dumping test.articles (1 document)
2017-09-28T11:04:28.698+0200    done dumping communes.insee (36742 documents)
2017-09-28T11:04:29.000+0200    done dumping communes.communes (36595 documents)

[root@grouik MongoDB]# ls -l dump
total 16
drwxr-xr-x. 2 root root 4096 28 sept. 11:04 admin
drwxr-xr-x. 2 root root 4096 28 sept. 11:04 communes
drwxr-xr-x. 2 root root 4096 28 sept. 11:04 test
drwxr-xr-x. 2 root root 4096 28 sept. 11:04 yougo
[root@grouik MongoDB]# ls -l dump/*
dump/admin:
total 8
-rw-r--r--. 1 root root  59 28 sept. 11:04 system.version.bson
-rw-r--r--. 1 root root 207 28 sept. 11:04 system.version.metadata.json

dump/communes:
total 16316
-rw-r--r--. 1 root root 13796550 28 sept. 11:04 communes.bson
-rw-r--r--. 1 root root      265 28 sept. 11:04 communes.metadata.json
-rw-r--r--. 1 root root  2896365 28 sept. 11:04 insee.bson
-rw-r--r--. 1 root root       86 28 sept. 11:04 insee.metadata.json

dump/test:
total 16
-rw-r--r--. 1 root root  36 28 sept. 11:04 articles.bson
-rw-r--r--. 1 root root 549 28 sept. 11:04 articles.metadata.json
-rw-r--r--. 1 root root 324 28 sept. 11:04 commandes.bson
-rw-r--r--. 1 root root 346 28 sept. 11:04 commandes.metadata.json

dump/yougo:
total 28
-rw-r--r--. 1 root root 15547 28 sept. 11:04 users.bson
-rw-r--r--. 1 root root    83 28 sept. 11:04 users.metadata.json
-rw-r--r--. 1 root root   283 28 sept. 11:04 user_type.bson
-rw-r--r--. 1 root root    87 28 sept. 11:04 user_type.metadata.json
```

La commande mongodump exporte les données des collections au format BSON (voir bsondump) et JSON pour les méta data (infos quant aux index)

mongodump <options>

--help	
--version	
-v | --verbose	Mode bavard. -vvv est plus bavard que -vv
--quiet	Silencieux
-h | --host	Le serveur cible
--port	Le port cible peut être --host <hostname>:<port>
-u | --username	Si l'authentification est actiée
-p | --password	Le mot de passe
-d|--db <nombase>	la base cible
-c | --collection <nomcollection>	la collection à dumper
--dumpDbUsersAndRoles	Pour une base, ajoute les définitions des users
-o | --out	Le répertoire destination (par défaut $PWD/dump)
--gzip	Comprime à la volée
--oplog	Le dump réalise un snapshot : sauvegarde les écritures réalisées pendant la sauvegarde

Restauration d'un dump MongoDB

On utilise la commande mongoretore. Attend des fichers .bson produits par la commande mongodump. Les options des deux commandes sont similaires.

Il est recommandé de supprimer la base cible avant de procéder à un mongoretsore

mongorestore <options>

Les principales options sont :
--db <nomBase> : base cible
--collection <nomCollection> : collection cible

Sans option, la commande mongorestore va chercher le répertoire dump du répertoire courant et restaurer toutes les bases qui s'y trouvent.

Remarques :
Si une base, les données seront ajoutées. Si les champs _id existent déjà, ils ne seront pas réécrit.
La commande mongorestore ne fait qu'insérer les données, il n'y a pas de contrôle dessus.

Bsondump

Bsondump est une commande qui convertit les fichiers binaires BSON au format JSON. Elle écrit le résultat de sa conversion sur la sortie standard, donc, prévoir une redirection pour créer un fichier de sortie.
Permet aussi de faire un contrôle des données.

bsondump <options> fichier.bson >fichier.json
Options :

--type arg	type de sortie : json / debug
--objcheck	valide l'objet BSON avant de le formater en JSON
--pretty	mise en page plus lisible

```
root@grouik MongoDB]# cd dump
[root@grouik dump]# cd test
[root@grouik test]# ls -l
total 16
-rw-r--r--. 1 root root  36 28 sept. 11:04 articles.bson
-rw-r--r--. 1 root root 549 28 sept. 11:04 articles.metadata.json
-rw-r--r--. 1 root root 324 28 sept. 11:04 commandes.bson
-rw-r--r--. 1 root root 346 28 sept. 11:04 commandes.metadata.json
[root@grouik test]# bsondump --pretty commandes.bson
{
        "_id": {
                "$oid": "59c96b83b1a80c8004b35ee1"
        },
        "article": "xx1",
        "qte": 3.0,
        "px": 2.5,
        "dat_cde": {
                "$date": "2017-09-27T15:52:11.277Z"
        }
}
{
        "_id": {
                "$oid": "59c96b93b1a80c8004b35ee2"
        },
        "article": "xx2",
        "qte": 4.0,
        "px": 6.0,
        "dat_cde": {
                "$date": "2017-09-27T15:52:16.748Z"
        }
}
{
        "_id": {
                "$oid": "59c96ba1b1a80c8004b35ee3"
        },
        "article": "xx3",
        "qte": 5.0,
        "px": 4.2,
        "dat_cde": {
                "$date": "2017-09-27T15:52:21.644Z"
        }
}
{
        "_id": {
                "$oid": "59ca14761c2e338354c69870"
        },
        "article": "xx4",
        "qte": 10.0,
        "px": 7.5,
        "dat_cde": {
                "$date": "2017-09-27T15:52:26.956Z"
        }
}
2017-09-28T12:10:16.733+0200    4 objects found
[root@grouik test]#
```

mongoexport

Cette commande exporte une instance, une base, une collection ou même une liste de champs d'une collection vers un fichier de type CSV ou JSON.

Note : les outils mongoexport et mongoimport ne sont pas des outils de sauvegarde. Ce sont plutôt des outils destinés à des transferts de données entre un SGBD quelconque et MongoDB. Pour les sauvegardes et restaurations, privilégier mongodump et mongorestore

mongoexport <options>

--db <nomBase>	la base de données cible
--collection <nomCollection>	la collection à exporter
--type	csv | json exportation sous format csv (par défaut, format JSON)
--fields <liste>	liste de champs à exporter séparés par une virgule
--out <fichier>	spécifie le fichier de sortie, par défaut <stdout>

```
[root@grouik dump]# mongoexport -d test -c commandes -o commandes.csv --type=csv -f 'article,qte,px,dat_cde'
2017-09-28T12:51:07.879+0200    connected to: localhost
2017-09-28T12:51:07.879+0200    exported 4 records
[root@grouik dump]#
[root@grouik dump]# ls -ltr
total 20
drwxr-xr-x. 2 root root 4096 28 sept. 11:04 admin
drwxr-xr-x. 2 root root 4096 28 sept. 11:04 yougo
drwxr-xr-x. 2 root root 4096 28 sept. 11:04 communes
drwxr-xr-x. 2 root root 4096 28 sept. 11:30 test
-rw-r--r--. 1 root root  162 28 sept. 12:51 commandes.csv
[root@grouik dump]#
[root@grouik dump]# more commandes.csv
article,qte,px,dat_cde
xx1,3,2.5,2017-09-27T15:52:11.277Z
xx2,4,6,2017-09-27T15:52:16.748Z
xx3,5,4.2,2017-09-27T15:52:21.644Z
xx4,10,7.5,2017-09-27T15:52:26.956Z
[root@grouik dump]#
```

mongoimport

La commande mongoimport permet d'importer des données au format CSV ou JSON
Les données peuvent avoir été préalablement 'sauvegardées' par la commande mongoexport ;
Elles peuvent tout à fait provenir d'une source quelconque : les formats CSV ou JSON sont connus de la majorité des SGBD.

On importe une base, une collection ou une liste de champs.

mongoimport <options>

--db <nomBase>	base cible
--collection <nomCollection>	collection à importer
--type <typefichier>	type du fichier à importer : json, csv, tsv
--file <nomfichier>	fichier d'importation, par défaut c'est stdin

L'import ci dessous créé la collection 'communes' dans la base villes. Si la collection existe, elle est supprimée (option --drop), le fichier en entrée est un CSV dont la première ligne contient la liste des champs à générer dans la collection :

```
[root@grouik communes]# head -5 communes.csv
ID_GEOFLA,CODE_COM,INSEE_COM,NOM_COM,STATUT,X_CHF_LIEU,Y_CHF_LIEU,X_CENTROID,Y_CENTROID,Z_MOYEN,SU
PERFICIE,POPULATION,CODE_CANT,CODE_ARR,CODE_DEPT,NOM_DEPT,CODE_REG,NOM_REG
COMMUNE0000000000000001,216,32216,LOURTIES-MONBRUN,Commune
simple,500820,6264958,500515,6265413,248,966,138,15,3,32,GERS,73,MIDI-PYRENEES
COMMUNE0000000000000002,033,47033,BOUDY-DE-BEAUREGARD,Commune
simple,516424,6384852,515575,6385938,112,1019,406,06,3,47,LOT-ET-GARONNE,72,AQUITAINE
COMMUNE0000000000000003,009,32009,ARMOUS-ET-CAU,Commune
simple,472979,6278963,473004,6278937,217,932,95,20,3,32,GERS,73,MIDI-PYRENEES
COMMUNE0000000000000004,225,38225,MEAUDRE,Commune
simple,898640,6450689,898625,6451597,1184,3371,1378,41,1,38,ISERE,82,RHONE-ALPES
[root@grouik communes]#
[root@grouik communes]# mongoimport -d villes -c communes --type csv --headerline --file communes.csv --drop -v
2017-09-28T13:30:56.858+0200    filesize: 5142951 bytes
2017-09-28T13:30:56.858+0200    using fields:
ID_GEOFLA,CODE_COM,INSEE_COM,NOM_COM,STATUT,X_CHF_LIEU,Y_CHF_LIEU,X_CENTROID,Y_CENTROID,Z_MOYEN,SU
PERFICIE,POPULATION,CODE_CANT,CODE_ARR,CODE_DEPT,NOM_DEPT,CODE_REG,NOM_REG
2017-09-28T13:30:56.859+0200    connected to: localhost
2017-09-28T13:30:56.859+0200    ns: villes.communes
2017-09-28T13:30:56.859+0200    connected to node type: standalone
2017-09-28T13:30:56.859+0200    using write concern: w='1', j=false, fsync=false, wtimeout=0
2017-09-28T13:30:56.859+0200    dropping: villes.communes
2017-09-28T13:30:56.859+0200    using write concern: w='1', j=false, fsync=false, wtimeout=0
2017-09-28T13:30:57.731+0200    imported 36595 documents
[root@grouik communes]#
```

```
> use villes
switched to db villes
>
> db.communes.findOne()
{
        "_id" : ObjectId("59ccdd70a636cb2fc83997b2"),
        "ID_GEOFLA" : "COMMUNE00000000000000003",
        "CODE_COM" : 9,
        "INSEE_COM" : 32009,
        "NOM_COM" : "ARMOUS-ET-CAU",
        "STATUT" : "Commune simple",
        "X_CHF_LIEU" : 472979,
        "Y_CHF_LIEU" : 6278963,
        "X_CENTROID" : 473004,
        "Y_CENTROID" : 6278937,
        "Z_MOYEN" : 217,
        "SUPERFICIE" : 932,
        "POPULATION" : 95,
        "CODE_CANT" : 20,
        "CODE_ARR" : 3,
        "CODE_DEPT" : 32,
        "NOM_DEPT" : "GERS",
        "CODE_REG" : 73,
        "NOM_REG" : "MIDI-PYRENEES"
}
>
```

Objectifs

Préparation des serveurs
Initialisation du replica set, rs.initiate()
Etat : rs.conf()
Ajout / suppression de serveurs au replica set
Serveur arbitre

Introduction

La réplication permet la disponibilité des données en les multipliant sur deux serveurs ou plus.
Les modifications apportées sur le master sont reproduites sur le ou les slave(s).

Cette redondance assure la continuité de service, un des objectifs étant que les requêtes aboutissent quel que soit l'état du système. Si un serveur 'tombe', un autre prend le relai.

La réplication est également utilisée pour assurer la performance en affectant certains serveurs à des tâches bien précises telles que les sauvegardes ou le reporting.

Un replica set est une grappe de serveurs qui font partie du même dispositif de réplication.
Un serveur primaire reçoit les requêtes de mise à jour des clients et les serveurs secondaires rejouent ces modifications.

Un 'master' est ouvert en lecture/écriture. Les 'slaves' sont, au mieux, ouverts en lecture.

Un seul serveur primaire au temps t. En cas de défaillance, le replica set procède à l'élection d'un nouveau primaire.

Pour assurer le bon fonctionnement de l'élection, on peut adjoindre des membres votants au replica set. Ces serveurs n'hébergent pas de données. Ils sont juste là pour le quorum. On les appelle 'serveurs arbitres'. (Utile particulièrement pour avoir un nombre impair de votants).

Un replica set peut avoir 50 membres et 7 membres votants. (12 membres avant la version 3.0)

Le(s) serveur(s) secondaire(s) rejoue(nt) les mises à jour depuis le serveur primaire.
Par défaut les serveurs secondaires sont inaccessibles.
Ils peuvent être ouverts en lecture avec la méthode 'setSalveOk()'.

Pré requis

On configure un replica set avec :

- 3 instances,
- Chaque instance écoute un port différent,
- Chaque instance possède son dbpath,
- Soit 3 configurations.

La démarche :

- Préparation et définition des configurations (3 fichiers .conf)
- Démarrage des instances
- Initialisation des instances

L'option --replSet <nomReplica> de la commande mongod permet de baptiser le replica set.

Chacune des instances démarre avec le même nom ; Elle se reconnaissent et se synchronisent.

Préparation des répertoires (sous root)

```
[root@grouik MongoDB]# mkdir -p /opt/mongo/rs0-1 /opt/mongo/rs0-2 /opt/mongo/rs0-3
[root@grouik MongoDB]# chown mongod.  /opt/mongo/rs0-1 /opt/mongo/rs0-2 /opt/mongo/rs0-3
[root@grouik MongoDB]# ls -l /opt/mongo
total 12
drwxr-xr-x. 2 mongod mongod 4096 30 sept. 10:25 rs0-1
drwxr-xr-x. 2 mongod mongod 4096 30 sept. 10:25 rs0-2
drwxr-xr-x. 2 mongod mongod 4096 30 sept. 10:25 rs0-3
```

Dans un premier temps, on démarre les instances depuis la ligne de commandes

```
$ mongod --port 27017 --dbpath /opt/mongo/rs0-1 --replSet rs0 --fork --logpath /tmp/mongo-rs0-1.log
$ mongod --port 27018 --dbpath /opt/mongo/rs0-2 --replSet rs0 --fork --logpath /tmp/mongo-rs0-2.log
$ mongod --port 27019 --dbpath /opt/mongo/rs0-3 --replSet rs0 --fork --logpath /tmp/mongo-rs0-3.log
```

Les serveurs fonctionnent chacun de manière autonome.

```
$ ps -ef | grep mongod
root     4036    1  0 11:57 ?        00:00:00 mongod --port 27017 --dbpath /opt/mongo/rs0-1 --replSet rs0 --fork --logpath /tmp/mongo-rs0-1.log
root     4108    1  0 11:58 ?        00:00:00 mongod --port 27018 --dbpath /opt/mongo/rs0-2 --replSet rs0 --fork --logpath /tmp/mongo-rs0-2.log
root     4141    1  0 11:58 ?        00:00:00 mongod --port 27019 --dbpath /opt/mongo/rs0-3 --replSet rs0 --fork --logpath /tmp/mongo-rs0-3.log
```

Initialisation du replica set

Connexion à un des serveurs, déclaration du document JSON de configuration et initialisation du replica set avec ce document.

```
[root@grouik MongoDB]# mongo –quiet
>
> rsconf={
... ...       _id: "rs0",
... ...       members: [
...                   { _id:1, host: "localhost:27017" },
...                   { _id:2, host: "localhost:27018" },
...                   { _id:3, host: "localhost:27019" }
...           ]
... }
{
        "_id" : "rs0",
        "members" : [
                {
                        "_id" : 1,
                        "host" : "localhost:27017"
                },
                {
                        "_id" : 2,
                        "host" : "localhost:27018"
                },
                {
                        "_id" : 3,
                        "host" : "localhost:27019"
                }
        ]
}
>
```

Le document de configuration contient le nom du replica set et la liste de ses membres. Ils vont se synchroniser lors de l'initialisation.

Note : Chaque id et ports des membres doivent être différents ; Le champ _id: "nom_rs" doit être le nom du replica set utilisé lors du démarrage des instances ; Le nom du document de configuration (ici rsconf) est indifférent. L'initialisation se fait avec la commande rs.initiate(rsconf)

```
> rs.initiate(rsconf)
{ "ok" : 1 }
rs0:SECONDARY>
```

Note : Les commandes relatives aux replica sets sont préfixées par 'rs.', de la même manière que les commandes de manipulation des bases et collections sont préfixées par 'db.' ; On verra, dans le chapitre sur le sharding, que les commandes de gestion sont préfixées par 'sh.'

Ces préfixes sont à la fois des alias pour les bases, replica sets ou shards courants et la précision d'un contexte pour une action :

- *initiate est une méthode rs*
- *find est une méthode db.*

Si la synchronisation réussit, la méthode retourne '`{ "ok" : 1 }`' et affiche un prompt modifié en '`rs0:PRIMARY>`' ou '`rs0:SECONDARY>`'.
La synchronisation peut échouer : problème de configuration, serveur membre inaccessible…..

Cette synchonisation consiste notamment en l'élection du 'PRIMARY'. C'est à lui qu'il faut adresser les requetes de mises à jour. On peut se déplacer d'une instance à l'autre en se connectant sur l'adresse cible (new Mongo()). On peut connaitre l'état du replica set, la liste des membres, le master et l'adresse locale avec la fonction isMaster()

```
rs0:SECONDARY> conn=new Mongo("localhost:27019")
connection to localhost:27019
rs0:SECONDARY> db=conn.getDB("admin")
admin
rs0:SECONDARY> db.isMaster()
{
        "hosts" : [
                "localhost:27017",
                "localhost:27018",
                "localhost:27019"
        ],
        "setName" : "rs0",
        "setVersion" : 1,
        "ismaster" : false,
        "secondary" : true,
        "primary" : "localhost:27017",
        "me" : "localhost:27019",
        "lastWrite" : {
                "opTime" : {
                        "ts" : Timestamp(1506772072, 1),
                        "t" : NumberLong(1)
                },
                "lastWriteDate" : ISODate("2017-09-30T11:47:52Z")
        },
        "maxBsonObjectSize" : 16777216,
        "maxMessageSizeBytes" : 48000000,
        "maxWriteBatchSize" : 1000,
        "localTime" : ISODate("2017-09-30T11:47:55.125Z"),
        "maxWireVersion" : 5,
        "minWireVersion" : 0,
        "readOnly" : false,
        "ok" : 1
}
rs0:SECONDARY>
```

Etat du replica set – Reconfiguration

rs.conf()

La méthode rs.conf() permet d'afficher la configuration de la réplication en cours :

```
root@localhost:27019:admin> rs.conf()
{
        "_id" : "rs0",
        "version" : 1,
        "protocolVersion" : NumberLong(1),
        "members" : [
                {
                        "_id" : 1,
                        "host" : "localhost:27017",
                        "arbiterOnly" : false,
                        "buildIndexes" : true,
                        "hidden" : false,
                        "priority" : 1,
                        "tags" : {

                        },
                        "slaveDelay" : NumberLong(0),
                        "votes" : 1
                },
                {
                        "_id" : 2,
                        "host" : "localhost:27018",
                        "arbiterOnly" : false,
                        "buildIndexes" : true,
                        "hidden" : false,
                        "priority" : 1,
                        "tags" : {

                        },
                        "slaveDelay" : NumberLong(0),
                        "votes" : 1
                },
                {
                        "_id" : 3,
                        "host" : "localhost:27019",
                        "arbiterOnly" : false,
                        "buildIndexes" : true,
                        "hidden" : false,
                        "priority" : 1,
                        "tags" : {

                        },
                        "slaveDelay" : NumberLong(0),
                        "votes" : 1
                }
        ],
        "settings" : {
                "chainingAllowed" : true,
                "heartbeatIntervalMillis" : 2000,
                "heartbeatTimeoutSecs" : 10,
                "electionTimeoutMillis" : 10000,
                "catchUpTimeoutMillis" : 60000,
                "getLastErrorModes" : {

                },
                "getLastErrorDefaults" : {
                        "w" : 1,
                        "wtimeout" : 0
                },
                "replicaSetId" : ObjectId("59cf6e8f7cee66af65276213")
        }
}
root@localhost:27019:admin>
```

SetSlaveOk()

Par défaut les serveurs secondaires sont inaccessibles. On peut les ouvrir en lecture avec la méthode setSlaveOk() :

```
root@localhost:27018:admin> conn=new Mongo("localhost:27019")
connection to localhost:27019
root@localhost:27018:admin>
root@localhost:27018:admin> // La connexion est établie sur le serveur 27019.
root@localhost:27018:admin> // On sélectionne la base
root@localhost:27018:admin>
root@localhost:27018:admin> db=conn.getDB("admin")
admin
anon@localhost:27019:admin> // On active la lecture sur le secondary
anon@localhost:27019:admin> conn.setSlaveOk()
anon@localhost:27019:admin> db.auth("root", "root")
1
root@localhost:27018:admin>
root@localhost:27019:admin> show collections
system.users
system.version
root@localhost:27019:admin>
```

Reconfiguration d'un replica set

La commande rs.reconfig() permet de reconfigurer un replica set. L'opération s'effectue sur le serveur primaire.

La démarche :
- récupérer la configuration en cours avec rs.conf(),
- modifier les valeurs souhaitées,
- appeler rs.reconfig() avec le document modifié.

L'exemple suivant permet de changer la priorité d'un serveur (droit de vote => probabilité d'être elu 'PRIMARY' :

```
root@localhost:27017:admin> // on récupère la configuration dans une variable
root@localhost:27017:admin> rsconf=rs.conf()
root@localhost:27017:admin> rsconf.members[0]     // affiche les caractéristiques du premier serveur
{
        "_id" : 1,
        "host" : "localhost:27017",
        "arbiterOnly" : false,
        "buildIndexes" : true,
        "hidden" : false,
        "priority" : 1,
        "tags" : {

        },
        "slaveDelay" : NumberLong(0),
        "votes" : 1
}
root@localhost:27017:admin> rsconf.members[0].priority=3    // affecte une priorité supérieure
3
root@localhost:27017:admin> rs.reconfig(rsconf)                 // reconfigure
root@localhost:27017:admin>
```

Attention, le fait de changer de configuration peut provoquer une réélection, ce qui peut entrainer une suspension des sessions clientes.

Ajout / suppression de serveur

Ces opérations s'effectuent sur le primaire. Quelle que soit l'opération, un nouveau serveur primaire sera choisi.

Ajout :

```
rs0:PRIMARY> rs.add("localhost:27021")
```

Suppression :

Suivre la procédure suivante :

- Arrêt du serveur à supprimer :
  db.shutdownServer() / db.adminCommand({"shutdown":1})

- Connexion au serveur primaire :
  db = (new Mongo("serveur:port")).getDB("admin")

- Suppression du serveur souhaité :
  rs.remove("serveur")

stepDown()

Cette méthode force le primaire à se rétrograder pendant un nombre de secondes passé en argument.

```
root@localhost:27019:admin> db = (new Mongo("127.0.0.1:27017")).getDB("admin")
admin
anon@127.0.0.1:27017:admin>
anon@127.0.0.1:27017:admin> rs.stepDown(10)
2017-09-30T16:58:51.346+0200 E QUERY    [thread1] Error: error doing query: failed: network error while attempting
to run command 'replSetStepDown' on host '127.0.0.1:27017'  :
DB.prototype.runCommand@src/mongo/shell/db.js:132:1
DB.prototype.adminCommand@src/mongo/shell/db.js:149:1
rs.stepDown@src/mongo/shell/utils.js:1261:12
@(shell):1:1
2017-09-30T16:58:51.348+0200 I NETWORK  [thread1] trying reconnect to 127.0.0.1:27017 (127.0.0.1) failed
2017-09-30T16:58:51.349+0200 I NETWORK  [thread1] reconnect 127.0.0.1:27017 (127.0.0.1) ok
anon@127.0.0.1:27017:admin>
```

Pendant ce temps, le serveur primaire ne peut être réélu. La commande ne fonctionne que s'il est possible d'avoir un serveur secondaire éligible. Passé 10 secondes, la commande échoue et le primaire reste primaire.

Dans l'exemple qui suit, on provoque un stepDown de 20 secondes. On appelle une première fois la méthode isMaster(), il n'y a pas de serveur primaire et le 27017 est devenu un serveur secondaire. Lors d'un deuxième appel isMaster(), le serveur 27018 a été promu master.

```
anon@127.0.0.1:27017:admin> rs.stepDown(20)
2017-09-30T17:05:43.345+0200 E QUERY    [thread1] Error: error doing query: failed: network error while attempting to run command
'replSetStepDown' on host '127.0.0.1:27017'  :
DB.prototype.runCommand@src/mongo/shell/db.js:132:1
DB.prototype.adminCommand@src/mongo/shell/db.js:149:1
rs.stepDown@src/mongo/shell/utils.js:1261:12
@(shell):1:1
2017-09-30T17:05:43.346+0200 I NETWORK  [thread1] trying reconnect to 127.0.0.1:27017 (127.0.0.1) failed
2017-09-30T17:05:43.347+0200 I NETWORK  [thread1] reconnect 127.0.0.1:27017 (127.0.0.1) ok
anon@127.0.0.1:27017:admin> db.isMaster()
{
        "hosts" : [
                "localhost:27017",
                "localhost:27018",
                "localhost:27019"
        ],
        "setName" : "rs0",
        "setVersion" : 75531,
        "ismaster" : false,
        "secondary" : true,
        "me" : "localhost:27017",
        "lastWrite" : {
                "opTime" : {
                        "ts" : Timestamp(1506783933, 1),
                        "t" : NumberLong(7)
                },
                "lastWriteDate" : ISODate("2017-09-30T15:05:33Z")
        },
        "maxBsonObjectSize" : 16777216,
        "maxMessageSizeBytes" : 48000000,
        "maxWriteBatchSize" : 1000,
        "localTime" : ISODate("2017-09-30T15:05:47.208Z"),
        "maxWireVersion" : 5,
        "minWireVersion" : 0,
        "readOnly" : false,
        "ok" : 1
}
anon@127.0.0.1:27017:admin> db.isMaster()
{
        "hosts" : [
                "localhost:27017",
                "localhost:27018",
                "localhost:27019"
        ],
        "setName" : "rs0",
        "setVersion" : 75531,
        "ismaster" : false,
        "secondary" : true,
        "primary" : "localhost:27018",
        "me" : "localhost:27017",
        "lastWrite" : {
                "opTime" : {
                        "ts" : Timestamp(1506783933, 1),
                        "t" : NumberLong(7)
                },
                "lastWriteDate" : ISODate("2017-09-30T15:05:33Z")
        },
        "maxBsonObjectSize" : 16777216,
        "maxMessageSizeBytes" : 48000000,
        "maxWriteBatchSize" : 1000,
        "localTime" : ISODate("2017-09-30T15:05:59.400Z"),
        "maxWireVersion" : 5,
        "minWireVersion" : 0,
        "readOnly" : false,
        "ok" : 1
}
anon@127.0.0.1:27017:admin>
```

Si la commande réussit, elle force tous les clients à se déconnecter. Les écritures sont bloquées le temps de sa durée.

Surveillance de la réplication

La méthode rs.status() retourne les informations courantes des serveurs du replica set.

```
anon@127.0.0.1:27017:admin> rs.status()
{
        "set" : "rs0",
        "date" : ISODate("2017-09-30T15:43:17.025Z"),
        "myState" : 1,
        "term" : NumberLong(9),
        "heartbeatIntervalMillis" : NumberLong(2000),
        "optimes" : {
                "lastCommittedOpTime" : {
                        "ts" : Timestamp(1506786187, 1),
                        "t" : NumberLong(9)
                },
                "appliedOpTime" : {
                        "ts" : Timestamp(1506786187, 1),
                        "t" : NumberLong(9)
                },
                "durableOpTime" : {
                        "ts" : Timestamp(1506786187, 1),
                        "t" : NumberLong(9)
                }
        },
        "members" : [
                {
                        "_id" : 1,
                        "name" : "localhost:27017",
                        "health" : 1,
                        "state" : 1,
                        "stateStr" : "PRIMARY",
                        "uptime" : 20748,
                        "optime" : {
                                "ts" : Timestamp(1506786187, 1),
                                "t" : NumberLong(9)
                        },
                        "optimeDate" : ISODate("2017-09-30T15:43:07Z"),
                        "electionTime" : Timestamp(1506783965, 2),
                        "electionDate" : ISODate("2017-09-30T15:06:05Z"),
                        "configVersion" : 75531,
                        "self" : true
                },
                {
                        "_id" : 2,
                        "name" : "localhost:27018",
                        "health" : 1,
                        "state" : 2,
                        "stateStr" : "SECONDARY",
                        "uptime" : 19716,
                        "optime" : {
                                "ts" : Timestamp(1506786187, 1),
                                "t" : NumberLong(9)
                        },
                        "optimeDurable" : {
                                "ts" : Timestamp(1506786187, 1),
                                "t" : NumberLong(9)
                        },
                        "optimeDate" : ISODate("2017-09-30T15:43:07Z"),
                        "optimeDurableDate" : ISODate("2017-09-30T15:43:07Z"),
                        "lastHeartbeat" : ISODate("2017-09-30T15:43:16.517Z"),
                        "lastHeartbeatRecv" : ISODate("2017-09-30T15:43:15.254Z"),
                        "pingMs" : NumberLong(0),
                        "syncingTo" : "localhost:27019",
                        "configVersion" : 75531
                },
                {
                        "_id" : 3,
                        "name" : "localhost:27019",
                        "health" : 1,
                        "state" : 2,
                        "stateStr" : "SECONDARY",
                        "uptime" : 19716,
                        "optime" : {
                                "ts" : Timestamp(1506786187, 1),
                                "t" : NumberLong(9)
                        },
                        "optimeDurable" : {
                                "ts" : Timestamp(1506786187, 1),
                                "t" : NumberLong(9)
                        },
                        "optimeDate" : ISODate("2017-09-30T15:43:07Z"),
                        "optimeDurableDate" : ISODate("2017-09-30T15:43:07Z"),
                        "lastHeartbeat" : ISODate("2017-09-30T15:43:16.517Z"),
                        "lastHeartbeatRecv" : ISODate("2017-09-30T15:43:16.516Z"),
                        "pingMs" : NumberLong(0),
                        "syncingTo" : "localhost:27017",
                        "configVersion" : 75531
                }
        ],
        "ok" : 1
}
anon@127.0.0.1:27017:admin>
```

Quelques champs utiles pour comprendre le document retourné :

self : indique le serveur ou la commande rs.status() est lancée
stateStr : status du serveur (primaire, secondaire)
Uptime : temps de fonctionnement du serveur .
syncingTo : présent uniquement sur les membres secondaires, indique sur quel serveur le membre secondaire est synchronisé

Serveur arbitre

L'objectif est d'arrêter un des serveurs secondaires et d'intégrer un arbitre pour résoudre les problèmes de vote. On aura deux serveurs hébergeant des données (chacun d'eux pouvant être primaire ou secondaire) et un simple serveur votant.

```
[root@grouik MongoDB]# mkdir /opt/mongo/rs0-arb
[root@grouik MongoDB]# chown mongodb. /opt/mongo/rs0-arb
[root@grouik MongoDB]# ls -l /opt/mongo
total 16
drwxr-xr-x. 4 mongod mongod 4096 30 sept. 18:52 rs0-1
drwxr-xr-x. 4 mongod mongod 4096 30 sept. 18:53 rs0-2
drwxr-xr-x. 4 mongod mongod 4096 30 sept. 18:53 rs0-3
drwxr-xr-x. 2 mongod mongod 4096 30 sept. 18:53 rs0-arb
[root@grouik MongoDB]#
```

Le serveur arbitre est démarré comme n'importe quel autre serveur :

```
$ mongod --port 27020 --dbpath /opt/mongo/rs0-arb --replSet rs0 --fork --logpath /tmp/mongo-rs0-arb.log
```

L'arbitre rejoint le replica set en exécutant la commande addArb() sur le primaire :

```
root@grouik MongoDB]# mongo --quiet
anon@127.0.0.1:27017:test> db.isMaster()
{
        "hosts" : [
                "localhost:27017",
                "localhost:27018",
                "localhost:27019"
        ],
        "setName" : "rs0",
        "setVersion" : 75531,
        "ismaster" : true,
        "secondary" : false,
        "primary" : "localhost:27017",
        "me" : "localhost:27017",
        "electionId" : ObjectId("7fffffff0000000000000009"),
        "lastWrite" : {
                "opTime" : {
                        "ts" : Timestamp(1506791128, 1),
                        "t" : NumberLong(9)
                },
                "lastWriteDate" : ISODate("2017-09-30T17:05:28Z")
        },
        "maxBsonObjectSize" : 16777216,
        "maxMessageSizeBytes" : 48000000,
        "maxWriteBatchSize" : 1000,
        "localTime" : ISODate("2017-09-30T17:05:36.259Z"),
        "maxWireVersion" : 5,
        "minWireVersion" : 0,
        "readOnly" : false,
        "ok" : 1
}
anon@127.0.0.1:27017:test>
anon@127.0.0.1:27017:test> rs.addArb("127.0.0.1:27020")
{ "ok" : 1 }
anon@127.0.0.1:27017:test>
```

On peut constater l'intégration de l'arbitre avec isMaster :

```
anon@127.0.0.1:27017:test> db.isMaster()
{
        "hosts" : [
                "localhost:27017",
                "localhost:27018",
                "localhost:27019"
        ],
        "arbiters" : [
                "127.0.0.1:27020"
        ],
        "setName" : "rs0",
../..
```

Il ne reste plus qu'à arrêter le serveur 27019 :

```
[root@grouik MongoDB]# mongod --dbpath /opt/mongo/rs0-3 --shutdown
killing process with pid: 4141
[root@grouik MongoDB]#
```

et le soustraire du replica set :

```
anon@127.0.0.1:27017:test> rs.remove("localhost:27019")
{ "ok" : 1 }
anon@127.0.0.1:27017:test> db.isMaster()
{
        "hosts" : [
                "localhost:27017",
                "localhost:27018"
        ],
        "arbiters" : [
                "127.0.0.1:27020"
        ],
        "setName" : "rs0",
```

Objectifs

Préparation du replica set de configuration
Préparation des replica sets de données
Démarrage mongos
Le balancing - splitting

Introduction

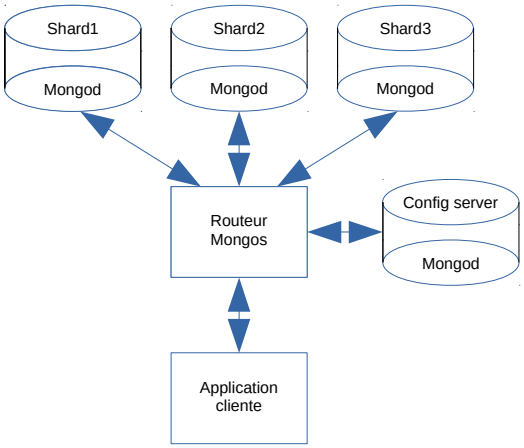
Le sharding est une fonctionnalité qui permet de répartir les documents sur plusieurs serveurs, appelés shards.

Chaque shard est un nœud qui contient un replica set ou une instance mongod standalone.

Le sharding permet de mettre en place une architecture de haute performance en utilisant plusieurs machines peu onéreuses.

Gains de performance liés à la répartition :

- La mémoire disponible est constituée de celle de toutes les machines,
- La charge de chaque serveur individuel est plus faible



Un cluster shardé est constitué de :

shards : les serveurs Mongod constituant le cluster contenant chacun une partie des documents.
config server : serveurs MongoDB stockant la configuration du cluster.
mongos : le serveur de routage des connexions et requêtes vers les shards.

L'ensemble des shards réunis contient toutes les données du cluster.
Idéalement, chaque shard est un replica set. Le replica set fournit la redondance et la haute disponibilité pour les données de chaque shard.

Les collections des bases peuvent être shardées ou pas. (Le sharding n'est pas rentable pour les 'petites' collections).
Chaque base de données a un shard "primaire" qui contient ces collections non-shardées.

Un shard est composé de chunks. Un chunk est un ensemble de documents contigus partageant la même clé shard dans un shard spécifique. Un chunk peut être découpé s'il atteint la taille définie. Par défaut la taille du chunk est de 64Mo, taille variable de 1 Mo à 1Go.

Les serveurs de configuration

Un config server est une instance mongod spéciale qui contient les méta data du cluster. Les méta data décrivent l'état et la structure des shards et de leurs chunks. (Liste des chunks et des plages de valeur des clés). Le config server est un composant essentiel du dispositif. En effet, il doit être disponible pour que le cluster fonctionne.

Jusqu'en 3.2, un dispositif de production basé sur le sharding peut comporter n configs servers pour la redondance des données.
A partir de la version 3.2, le config server peut se déployer sur un replica set.
A partir de la version 3.4, le config server est impérativement un replica set.

Chaque config server appartient à un et un seul cluster.
Un config server doit être journalisé.

Mongos

L' instance (ou les instances) mongos oriente(nt) les requêtes vers les shards du cluster (lectures et mises à jour). Il n'y a pas de limite au nombre de serveurs mongos.

Généralement il est recommandé de faire tourner l'application et son instance mongos sur la même machine cliente.

Mongos est le seul point d'entrée du cluster pour les applications. Elles ne sont pas autorisées à se connecter directement sur les shards.Les instances mongos gardent en cache la localisation des données dans les shards. Cette localisation est un des objets des méta data des configs servers. Le dialogue est permanent entre les mongos et ses config servers.

Les instances mongos consommant peu de ressources système, la procédure la plus commune est de les faire tourner sur les mêmes systèmes que les applications clientes. Elles peuvent aussi bien tourner sur les shards ou d'autres ressources dédiées.

Distribution des données

MongoDB distribue les documents des collections sur les différents shards par la répartition des données selon la valeur d'un champ dit 'shard key'.

MongoDB maintient une distribution équilibrée des données entre les différents shards par l'intermédiaire du balancer et du splitting.

L'ajout de nouvelles données ou de nouveaux serveurs peuvent entraîner un déséquilibrage dans le cluster : un shard contient plus de chunks qu'un autre shard ou un chunk est plus grand que les autres chunks.

MongoDB assure le rééquilibrage des données avec 2 méthodes d'arrière plan : 'splitting' (découpage) et 'balancer' (balance)

Splitting

Le splitting est un processus qui permet de gérer le volume des chunks. Quand un chunk dépasse un volume spécifié, MongoDB coupe le chunk en 2.

Balancer

Le balancer est un processus en arrière plan qui gère la migration des chunks. Il est géré par les méthodes sh.isBalancerRunning() et sh.setBalancerState().

Shard key

La clé de shard détermine la distribution des documents de la collection entre les shards.

Le clé est soit un index simple (sur un seul champ) soit composé (l'index est sur n champs du document). La clé doit exister dans tous les documents de la collection.

Chaque chunk représente un intervalle de valeurs de clé distinctes.

Les clefs shards ne sont pas modifiables et ne peuvent être changées après insertion.

Note : La clef shard ne peut reposer sur un index de type multi-key (index sur les entrées de tableau).

Initialisation du sharding

Les étapes :

- démarrage du (des) config server(s)
- démarrage du (des) mongod et mongos
- ajout des shards
- activation du sharding au niveau base
- activation du sharding au niveau collection

Il faut définir au préalable le replica set des config servers, le serveur mongos et les shards.
Il faut aussi définir quelles bases et collections sont concernées et les clés shard à utiliser.

Les config servers sont à démarrer en premier.

Préparer les répertoires pour les 3 membres du replica set de configuration (mkdir… chown…) :

```
[root@grouik MongoDB]# ls -l /opt/mongo-sh/
total 12
drwxr-xr-x. 4 mongod mongod 4096  1 oct.  14:48 conf.rs1
drwxr-xr-x. 4 mongod mongod 4096  1 oct.  14:48 conf.rs2
drwxr-xr-x. 4 mongod mongod 4096  1 oct.  14:48 conf.rs3
```

Lancer un mongod pour chaque membre avec les options

--configsvr	: initialise une instance de type 'serveur de configuration de cluster de shards'
--replSet	: précise que les mongod vont fonctionner en replica set. Le nom doit être identique
--dbpath	: chacun des répertoires créés précédemment (vides, appartenant à l'utilisateur mongod)
--port	: respectivement 27517, 27518 et 27519. Numéros arbitraires, mais uniques.
--fork, --logpath	: lancement en tache de fond et redirection des traces

```
[root@grouik MongoDB]# mongod --configsvr --replSet rs-sh-conf --dbpath /opt/mongo-sh/conf.rs1 --port 27517 --fork --logpath /tmp/mongo-sh-conf.rs1
about to fork child process, waiting until server is ready for connections.
forked process: 7547
child process started successfully, parent exiting

[root@grouik MongoDB]# mongod --configsvr --replSet rs-sh-conf --dbpath /opt/mongo-sh/conf.rs2 --port 27518 --fork --logpath /tmp/mongo-sh-conf.rs2
about to fork child process, waiting until server is ready for connections.
forked process: 7593
child process started successfully, parent exiting

[root@grouik MongoDB]# mongod --configsvr --replSet rs-sh-conf --dbpath /opt/mongo-sh/conf.rs3 --port 27519 --fork --logpath /tmp/mongo-sh-conf.rs3
about to fork child process, waiting until server is ready for connections.
forked process: 7640
child process started successfully, parent exiting
```

En cas d'utilisation de fichiers de configuration, les options dbpath, port, fork et logpath sont usuelles. Ajouter les options suivantes (en fin de fichier) :

```
../..

sharding:
  clusterRole: configsvr
replication:
  replSetName: <setname>
```

Comme pour tout replica set, il faut l'initialiser. Dans le document d'initialisation, on ajoute l'option 'configsvr: true'. Le nom du replica set est le nom utilisé dans les démarrages des serveurs.

```
[pierrick@grouik MongoDB]$ mongo --port 27517
../..

anon@127.0.0.1:27517:test> rsconf={
                "_id" : "rs-sh-conf",
                "configsvr" : true,
                "members" : [
                                {
                                                "_id" : 0,
                                                "host" : "localhost:27517"
                                },
                                {
                                                "_id" : 1,
                                                "host" : "localhost:27518"
                                },
                                {
                                                "_id" : 2,
                                                "host" : "localhost:27519"
                                }
                ]
}
../..

anon@127.0.0.1:27517:test> rs.initiate(rsconf)
{ "ok" : 1 }
anon@127.0.0.1:27517:test>
```

Préparation des replica sets pour chaque shard. Création des répertoires

```
[root@grouik MongoDB]# mkdir /opt/mongo-sh/rs0-1
[root@grouik MongoDB]# mkdir /opt/mongo-sh/rs0-2
[root@grouik MongoDB]# mkdir /opt/mongo-sh/rs0-arb
[root@grouik MongoDB]# mkdir /opt/mongo-sh/rs1-1
[root@grouik MongoDB]# mkdir /opt/mongo-sh/rs1-2
[root@grouik MongoDB]# mkdir /opt/mongo-sh/rs1-arb
[root@grouik MongoDB]# mkdir /opt/mongo-sh/rs2-1
[root@grouik MongoDB]# mkdir /opt/mongo-sh/rs2-2
[root@grouik MongoDB]# mkdir /opt/mongo-sh/rs2-arb
../..
[root@grouik MongoDB]# chown mongod. /opt/mongo-sh/rs*
../..
```

```
[root@grouik MongoDB]# ls -l /opt/mongo-sh/
total 48
drwxr-xr-x. 4 mongod mongod 4096  1 oct.  17:05 conf.rs1
drwxr-xr-x. 4 mongod mongod 4096  1 oct.  17:05 conf.rs2
drwxr-xr-x. 4 mongod mongod 4096  1 oct.  17:05 conf.rs3
drwxr-xr-x. 2 mongod mongod 4096  1 oct.  17:02 rs0-1
drwxr-xr-x. 2 mongod mongod 4096  1 oct.  17:02 rs0-2
drwxr-xr-x. 2 mongod mongod 4096  1 oct.  17:02 rs0-arb
drwxr-xr-x. 2 mongod mongod 4096  1 oct.  17:03 rs1-1
drwxr-xr-x. 2 mongod mongod 4096  1 oct.  17:03 rs1-2
drwxr-xr-x. 2 mongod mongod 4096  1 oct.  17:03 rs1-arb
drwxr-xr-x. 2 mongod mongod 4096  1 oct.  17:03 rs2-1
drwxr-xr-x. 2 mongod mongod 4096  1 oct.  17:03 rs2-2
drwxr-xr-x. 2 mongod mongod 4096  1 oct.  17:03 rs2-arb
[root@grouik MongoDB]#
```

Démarrage des serveurs shards, 3 replicas sets avec un primaire et un secondaire :

```
#
# démarrage du primaire du replica set sh-rs0.
#
[root@grouik MongoDB]# mongod --shardsvr --replSet sh-rs0 --dbpath /opt/mongo-sh/rs0-1 -port 27117 --fork --logpath /tmp/sh-rs0-1.log
about to fork child process, waiting until server is ready for connections.
forked process: 9397
child process started successfully, parent exiting
#
# démarrage du secondaire du replica set sh-rs0.
#
[root@grouik MongoDB]# mongod --shardsvr --replSet sh-rs0 --dbpath /opt/mongo-sh/rs0-2 -port 27118 --fork --logpath /tmp/sh-rs0-2.log
about to fork child process, waiting until server is ready for connections.
forked process: 9431
child process started successfully, parent exiting
#
[root@grouik MongoDB]# mongod --shardsvr --replSet sh-rs1 --dbpath /opt/mongo-sh/rs1-1 -port 27217 --fork --logpath /tmp/sh-rs1-1.log
about to fork child process, waiting until server is ready for connections.
forked process: 9511
child process started successfully, parent exiting
[root@grouik MongoDB]# mongod --shardsvr --replSet sh-rs1 --dbpath /opt/mongo-sh/rs1-2 -port 27218 --fork --logpath /tmp/sh-rs1-2.log
about to fork child process, waiting until server is ready for connections.
forked process: 9547
child process started successfully, parent exiting
#
[root@grouik MongoDB]# mongod --shardsvr --replSet sh-rs2 --dbpath /opt/mongo-sh/rs2-1 -port 27317 --fork --logpath /tmp/sh-rs2-1.log
about to fork child process, waiting until server is ready for connections.
forked process: 9591
child process started successfully, parent exiting
[root@grouik MongoDB]# mongod --shardsvr --replSet sh-rs2 --dbpath /opt/mongo-sh/rs2-2 -port 27318 --fork --logpath /tmp/sh-rs2-2.log
about to fork child process, waiting until server is ready for connections.
forked process: 9637
child process started successfully, parent exiting
[root@grouik MongoDB]#
```

Démarrage des arbitres sur les 3 replica sets

```
[root@grouik MongoDB]# mongod --shardsvr --replSet sh-rs0 --dbpath /opt/mongo-sh/rs0-arb -port 27120 --fork --logpath /tmp/sh-rs0-arb.log
about to fork child process, waiting until server is ready for connections.
forked process: 9970
child process started successfully, parent exiting
[root@grouik MongoDB]# mongod --shardsvr --replSet sh-rs1 --dbpath /opt/mongo-sh/rs1-arb -port 27220 --fork --logpath /tmp/sh-rs1-arb.log
about to fork child process, waiting until server is ready for connections.
forked process: 10007
child process started successfully, parent exiting
[root@grouik MongoDB]# mongod --shardsvr --replSet sh-rs2 --dbpath /opt/mongo-sh/rs2-arb -port 27320 --fork --logpath /tmp/sh-rs2-arb.log
about to fork child process, waiting until server is ready for connections.
forked process: 10063
child process started successfully, parent exiting
```

Il en résulte la grappe de processus suivante :

```
[root@grouik MongoDB]# ps -ef | grep mongod
root      7547     1  0 14:46 ?        00:00:53 mongod --configsvr --replSet rs-sh-conf --dbpath /opt/mongo-sh/conf.rs1 --port 27517 --fork --logpath /tmp/mongo-sh-conf.rs1
root      7593     1  0 14:47 ?        00:00:53 mongod --configsvr --replSet rs-sh-conf --dbpath /opt/mongo-sh/conf.rs2 --port 27518 --fork --logpath /tmp/mongo-sh-conf.rs2
root      7640     1  0 14:47 ?        00:00:52 mongod --configsvr --replSet rs-sh-conf --dbpath /opt/mongo-sh/conf.rs3 --port 27519 --fork --logpath /tmp/mongo-sh-conf.rs3
root      9397     1  0 17:16 ?        00:00:05 mongod --shardsvr --replSet sh-rs0 --dbpath /opt/mongo-sh/rs0-1 -port 27117 --fork --logpath /tmp/sh-rs0-1.log
root      9431     1  0 17:16 ?        00:00:05 mongod --shardsvr --replSet sh-rs0 --dbpath /opt/mongo-sh/rs0-2 -port 27118 --fork --logpath /tmp/sh-rs0-2.log
root      9511     1  0 17:20 ?        00:00:05 mongod --shardsvr --replSet sh-rs1 --dbpath /opt/mongo-sh/rs1-1 -port 27217 --fork --logpath /tmp/sh-rs1-1.log
root      9547     1  0 17:20 ?        00:00:05 mongod --shardsvr --replSet sh-rs1 --dbpath /opt/mongo-sh/rs1-2 -port 27218 --fork --logpath /tmp/sh-rs1-2.log
root      9591     1  0 17:21 ?        00:00:05 mongod --shardsvr --replSet sh-rs2 --dbpath /opt/mongo-sh/rs2-1 -port 27317 --fork --logpath /tmp/sh-rs2-1.log
root      9637     1  0 17:21 ?        00:00:04 mongod --shardsvr --replSet sh-rs2 --dbpath /opt/mongo-sh/rs2-2 -port 27318 --fork --logpath /tmp/sh-rs2-2.log
root      9970     1  0 17:38 ?        00:00:00 mongod --shardsvr --replSet sh-rs0 --dbpath /opt/mongo-sh/rs0-arb -port 27120 --fork --logpath /tmp/sh-rs0-arb.log
root     10007     1  0 17:39 ?        00:00:00 mongod --shardsvr --replSet sh-rs1 --dbpath /opt/mongo-sh/rs1-arb -port 27220 --fork --logpath /tmp/sh-rs1-arb.log
root     10063     1  0 17:39 ?        00:00:00 mongod --shardsvr --replSet sh-rs2 --dbpath /opt/mongo-sh/rs2-arb -port 27320 --fork --logpath /tmp/sh-rs2-arb.log
root     10157  5771  0 17:42 pts/2    00:00:00 grep --color=auto mongod
[root@grouik MongoDB]#
```

Initialiser les replicas sets

```
anon@localhost:27117:admin> rsconf={
        "_id" : "sh-rs0",
        "members" : [
                {
                        "_id" : 0,
                        "host" : "localhost:27117"
                }
        ]
}
anon@localhost:27117:admin> rs.initiate(rsconf)
{ "ok" : 1 }
anon@localhost:27117:admin> rs.add("localhost:27118")
{ "ok" : 1 }
anon@localhost:27117:admin> rs.addArb("localhost:27120")
{ "ok" : 1 }
anon@localhost:27117:admin>
```

Répéter les opérations pour les replica sets sh-rs1 et sh-rs2. Les serveurs de shard sont complets.

Démarrage du routeur mongos

```
[root@grouik MongoDB]# mongos --configdb rs-sh-conf/localhost:27517,localhost:27518,localhost:27519 --fork --logpath /tmp/mongos.log
```

Par défaut, le serveur mongos écoute le port 27017. L'inspection de la trace montre que mongos se connecte sur chacun des config servers.

Ajout des shards au cluster

Le mongos tourne sur localhost et écoute le port 27017. Pour ajouter le replica set, on utilise la commande 'addShard' à laquelle on précise le nom du replica set et l'adresse d'un des membres.

```
[pierrick@grouik MongoDB]$ mongo  --quiet
anon@127.0.0.1:27017:test> sh.addShard("sh-rs0/localhost:27117")
{ "shardAdded" : "sh-rs0", "ok" : 1 }
anon@127.0.0.1:27017:test> sh.addShard("sh-rs1/localhost:27217")
{ "shardAdded" : "sh-rs1", "ok" : 1 }
anon@127.0.0.1:27017:test> sh.addShard("sh-rs2/localhost:27317")
{ "shardAdded" : "sh-rs2", "ok" : 1 }
anon@127.0.0.1:27017:test>
```

Activation du sharding

On active le sharding au niveau base, puis au niveau collection.

Activer le sharding d'une base ne redistribue pas les données.
Un shard primaire sera attribué pour pouvoir accueillir les collections non shardées de la base.

L'opération se réalise depuis le client mongo connecté à mongos.

```
anon@127.0.0.1:27017:test> sh.enableSharding("test")
{ "ok" : 1 }
anon@127.0.0.1:27017:test> db.xx.insert({xx: "xx"})
WriteResult({ "nInserted" : 1 })
```

Note : sh.enableSharding("<nomBase>") est un alias pour db.runCommand({ enableSharding: "<nomBase>" })

Il est impératif d'activer le sharding sur une base si on veut y écrire

```
anon@127.0.0.1:27017:test> use xx
switched to db xx
anon@127.0.0.1:27017:xx> db.xx.insert({xx: "xx"})
WriteResult({
        "writeError" : {
                "code" : 70,
                "errmsg" : "unable to target write op for collection xx.xx :: caused by :: ShardNotFound: Database xx
not found due to No shards found"
        }
})
anon@127.0.0.1:27017:xx>
```

Activation au niveau collection

La collection shardée devrait théoriquement s'équilibrer en distribuant 1/n documents par shard dans un cluster de n shards. Il faut déterminer la clé qui va servir à répartir les documents. L'_id des documents est souvent une bonne candidate.

Si la collection contient déjà des données, il faut obligatoirement créer un index sur la clé.
Si la collection est vide, l'index sera créé lors de l'activation du sharding.
Si la collection n'existe pas encore, la clé de shard et l'index seront créés en même temps.

```
anon@127.0.0.1:27017:test> use communes                                  // connexion à la base communes, inexistante
switched to db communes
anon@127.0.0.1:27017:communes> db.createCollection("communes")   // création de la collection
{ "ok" : 1 }
anon@127.0.0.1:27017:communes> show collections
communes
anon@127.0.0.1:27017:communes>
```

```
anon@127.0.0.1:27017:communes> sh.enableSharding("communes")          // activation du sharding au niveau base
{ "ok" : 1 }
anon@127.0.0.1:27017:communes> //
anon@127.0.0.1:27017:communes> //     définition de la clé de shard sur le champ INSEE_COM
anon@127.0.0.1:27017:communes> //
anon@127.0.0.1:27017:communes> sh.shardCollection("communes.communes", {"INSEE_COM": 1})
{ "collectionsharded" : "communes.communes", "ok" : 1 }
anon@127.0.0.1:27017:communes>
```

Au niveau du shell Linux

```
[root@grouik communes]# mongoimport -d communes -c communes --file communes.csv --type csv --headerline
2017-10-01T23:05:10.742+0200    connected to: localhost
2017-10-01T23:05:13.720+0200    [#########.............] communes.communes  2.02MB/4.90MB (41.2%)
2017-10-01T23:05:16.720+0200    [###################.....] communes.communes  4.03MB/4.90MB (82.1%)
2017-10-01T23:05:18.198+0200    [########################] communes.communes  4.90MB/4.90MB (100.0%)
2017-10-01T23:05:18.198+0200    imported 36595 documents
[root@grouik communes]#
```

```
anon@127.0.0.1:27017:communes> db.communes.find().count()
36595
anon@127.0.0.1:27017:communes>  // Bingo !
```

Statut du cluster

La méthode sh.status() permet de connaître la configuration du cluster.
Elle informe sur les shards ainsi que sur la distribution des chunks dans les shards.
Au dela de 20 chunks, les informations ne s'affichent plus, à moins d'utiliser l'option verbose.

```
anon@127.0.0.1:27017:communes> sh.status()
--- Sharding Status ---
  sharding version: {
          "_id" : 1,
          "minCompatibleVersion" : 5,
          "currentVersion" : 6,
          "clusterId" : ObjectId("59d0f2f7a48f2a4d381458d7")
}
  shards:
          {  "_id" : "sh-rs0",  "host" : "sh-rs0/localhost:27117,localhost:27118",  "state" : 1 }
          {  "_id" : "sh-rs1",  "host" : "sh-rs1/localhost:27217,localhost:27218",  "state" : 1 }
          {  "_id" : "sh-rs2",  "host" : "sh-rs2/localhost:27317,localhost:27318",  "state" : 1 }
  active mongoses:
          "3.4.9" : 1
 autosplit:
          Currently enabled: yes
  balancer:
          Currently enabled:  yes
          Currently running:  no
                  Balancer lock taken at Sun Oct 01 2017 15:51:57 GMT+0200 (CEST) by ConfigServer:Balancer
          Failed balancer rounds in last 5 attempts:  0
          Migration Results for the last 24 hours:
                  No recent migrations
  databases:
          {  "_id" : "test",  "primary" : "sh-rs1",  "partitioned" : true }
          {  "_id" : "communes",  "primary" : "sh-rs0",  "partitioned" : true }
                  communes.communes
                          shard key: { "INSEE_COM" : 1 }
                          unique: false
                          balancing: true
                          chunks:
                                  sh-rs0    1
                          { "INSEE_COM" : { "$minKey" : 1 } } -->> { "INSEE_COM" : { "$maxKey" : 1 } } on : sh-rs0
Timestamp(1, 0)

anon@127.0.0.1:27017:communes> // tous les documents sont dans le même shard
```

Autre exemple

```
mongos> sh.status()
--- Sharding Status ---
  sharding version: {
      "_id" : 1, "version" : 4,
      "minCompatibleVersion" : 4, "currentVersion" : 5,
      "clusterId" : ObjectId("55c1c9385cfb5ce62d971e2e")
}
```

```
 shards:
      {  "_id" : "shard0000",  "host" : "localhost:30000" }
      {  "_id" : "shard0001",  "host" : "localhost:30001" }
      {  "_id" : "shard0002",  "host" : "localhost:30002" }
 databases:
      {  "_id" : "admin",  "partitioned" : false,  "primary" : "config" }
      {  "_id" : "test",  "partitioned" : true,  "primary" : "shard0001" }
            test.users
                  shard key: { "username" : 1 }
                  chunks:
                        shard0000      1
                        shard0001      1
                        shard0002      1
                  { "username" : { "$minKey" : 1 } } -->> { "username" : "user0" } on : shard0000 Timestamp(2, 0)
                  { "username" : "user0" } -->> { "username" : "user999" } on : shard0001 Timestamp(3, 1)
                  { "username" : "user999" } -->> { "username" : { "$maxKey" : 1 } } on : shard0002 Timestamp(3, 0)
mongos>
mongos> // les documents sont répartis sur les trois shards
```

Ajout d'un shard

La méthode permet d'ajouter une instance ou un replica set au cluster.
Elle doit être effectuée sur un serveur mongos.

Ajouter un shard à un cluster crée une instabilité puisque le nouveau shard n'a pas de chunks. L'opération prend du temps, car MongoDB redistribue les données dans le nouveau shard.

sh.addShard("<host>")

où <host> peut être :
[hostname]
[hostname]:[port]
[nomReplicaSet]/[hostname]
[nomReplicaSet]/[hostname]:port

Suppression d'un shard

db.adminCommand({ "removeShard": "nomshard" })

Quand on souhaite supprimer un shard, le processus balancer migre tous les chunks du shard dans les autres shards. Après migration de tous les documents et mise à jour des méta données, le shard peut être supprimé.

Attention, une fois l'opération de suppression lancée, elle ne peut être arrêtée.

Supprimer un shard est une opération délicate sur plusieurs points :

Pour supprimer un shard, se connecter sur la base 'admin' d'un des mongos puis :

- vérifier que le balancer est activé : **sh.getBalancerState()** (voir plus loin).
- vérifier le nom du shard : **db.adminCommand({ listShards: 1 }).**
- lancer la commande **db.adminCommand({ "removeShard": "nomshard" }),** une première fois.

Le balancer va commencer à déplacer toutes les données du shard à supprimer dans les autres shards via un processus appelé draining.

Le lancement de la commande rend la main et retourne le document suivant :

```
{
  "msg" : "draining started successfully",
  "state" : "started",
  "shard" : "mongodb0",
  "note" : "you need to drop or movePrimary these databases",
  "dbsToMove" : [
    "fiz",
    "buzz"
  ],
  "ok" : 1
}
```

Pour vérifier l'avancement du processus, relancer la commande 'removeShard'.
(Elle peut être lancée autant de fois souhaité.)

```
{
  "msg" : "draining ongoing",
  "state" : "ongoing",
  "remaining" : {
    "chunks" : NumberLong(2),
    "dbs" : NumberLong(2)
  },
  "note" : "you need to drop or movePrimary these databases",
  "dbsToMove" : [
    "fizz",
    "buzz"
  ],
  "ok" : 1
}
```

La section 'remaining' liste le nombre de shards/bases restant à migrer.

Il se peut que le shard dont on demande la suppression soit le shard primaire d'une base. (ie domiciliation des collections non shardées de la base).
Dans ce cas, un message "note" apparaît lors de la surveillance et demande de déplacer la (es) base(s).

Utiliser la commande suivante :
db.adminCommand({ "movePrimary": "*base-x*","to": "*shard-y*" })

ATTENTION : cette migration du shard primaire doit s'effectuer en dernier lieu.

Cette commande ne rend pas la main tant que l'opération n'a pas abouti. Elle finit par retourner un document de succès :

```
{ "primary" : "shard-y", "ok" : 1 }
```

Relancer enfin la commande **db.adminCommand({ "removeShard":"nomshard" })** qui devrait confirmer le succès de la suppression.

Gestion du balancer

La méthode sh.getBalancerState() permet de connaître l'état en cours du balancer : elle retourne la valeur true si il est activé, ou false s'il est désactivé.
Elle ne permet pas de savoir si des opérations sont en cours.

La méthode sh.isBalancerRunning() permet de savoir si le balancer est activé et migre des chunks (retourne la valeur true), ou s'il est désactivé (retourne la valeur false).

Il est conseillé lors d'opérations de maintenance sur le cluster, de désactiver le balancer.
La méthode sh.setBalancerState(<bool>) active ou désactive le balancer. <bool> a pour valeur true pour l'activation et false pour la désactivation.

Il est possible pour des activités de maintenance telles que l'export ou l'import de données, de désactiver ou ré-activer le balancer au niveau d'une collection.

La méthode sh.disableBalancing("<nom_base>.<nom_collection>") désactive le balancer sur une collection particulière.
La méthode sh.enableBalancing() ré-active le balancer de la collection.

Paramétrer la taille des chunks

Quand le premier serveur mongos se connecte à un ensemble de config server, il initialise le cluster avec une taille de chunk à 64Mo par défaut.
Cette taille par défaut fonctionne bien pour la plupart des déploiements ; Toutefois, si les migrations automatiques génèrent plus d'I/O que le matériel peut en gérer, la taille du chunk peut être réduite. Pour les migrations, une petite taille de chunk conduit à des migrations plus rapides et plus fréquentes. La taille autorisée peut varier de 1 Mo à 1024 Mo.

Pour modifier la taille des chunks :

Se connecter au serveur mongos du cluster avec le mongo shell
Utiliser la commande « use config » pour se connecter à la base de configuration

Lancer la commande suivante pour modifier la taille du chunk

db.settings.save({ _id:"chunksize", value: <tailleEnMo> })

Remarque

L' options chunkSize n'affecte pas la taille du chunk lors d'un redémarrage de mongos.
Cette option ne s'utilise qu'au moment de l'initialisation du sharding. (1er démarrage de mongos avec un cluster nouvellement installé).
Utiliser la procédure ci-dessus pour des modifications ultérieures.

Modifier la taille du chunk a quelques limites :
-le découpage automatique des chunks ne se produit que sur les insert et update
-si la taille du chunk diminue, cela peut prendre du temps pour découper les chunks à leur nouvelle taille
-le découpage ne peut être annulé
-si la taille du chunk augmente, les chunks existants augmenteront uniquement via les insert ou update jusqu'à atteindre la nouvelle taille.


Index détaillé