UPEC
UNIVERSITÉ PARIS-EST CRÉTEIL

MONGODB

Daniel BADIATA

Consultant Data

Daniel.badiata-kanza@u-pecfr

Objectifs

1/

Big Data, NoSQL et leur rôle dans les applications d'entreprise modernes

2/ Prise en main de MongoDB « administration & Développement »

3/ Réplication et Sharding & performances et indexation de MongoDB

4/ Spring Data et MongoDB

Big Data, NoSQL et leur rôle dans les applications d'entreprise modernes,

CONSTATS

Estimation : il y aura 50 milliards d'objets connectés dans le monde en 2020 contre 12 milliards aujourd'hui

90 % de l'ensemble des données aujourd'hui disponibles ont été créées ces deux dernières années

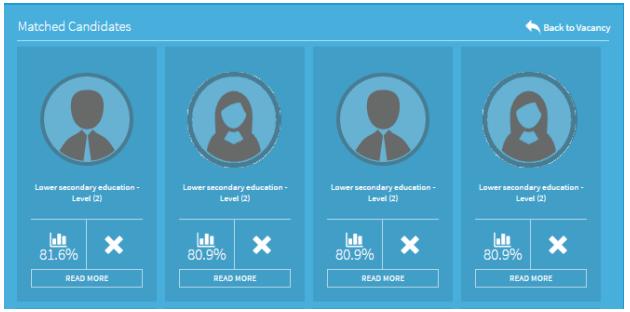
80 % des données générées aujourd'hui sont non structurées

CONSTATS

Exemples de problématiques mobilisant de grandes quantités de données

Text mining – Analyse des sentiments

Vue 360 degré du client



Moteurs de matching CV

CONSTATS

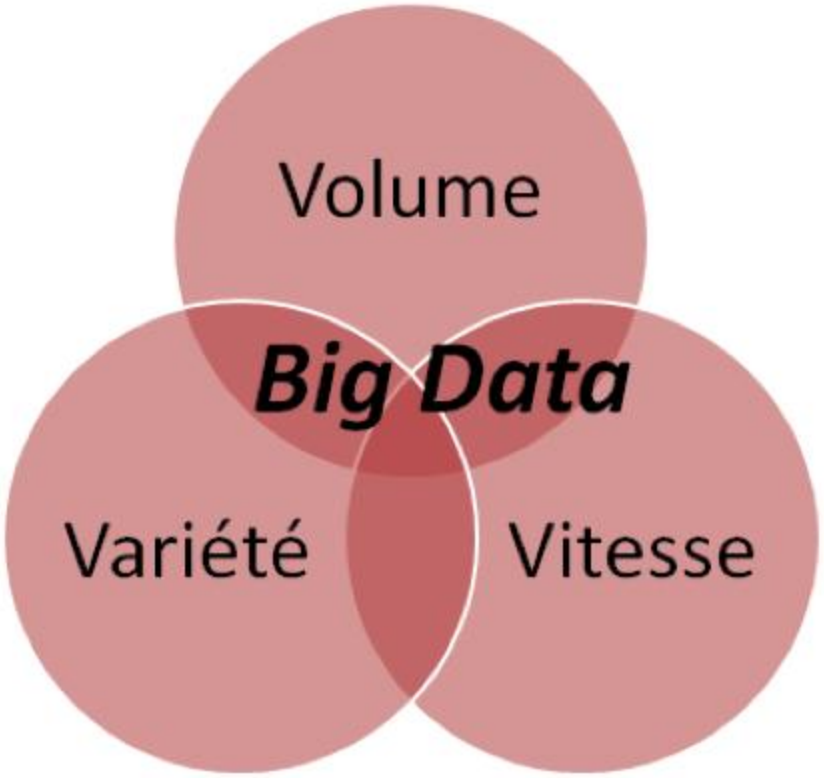
- vend de l'espace publicitaire très ciblé
- Analyses 3 milliards de bannières par jour

- 27 millions de clients particuliers
- 25 Péta octets à l'horizon 2022
- Compteurs linky dans tous les logements d'ici 2020
- Les compteurs communiquent les données toutes les 10 min -> 100 To par an.

- Facebook génère 500 To par jour

CONSTATS

Une problématique est Big Data lorsqu'elle touche à au moins l'une des 3 dimensions



Volume : Facebook stocke environ 250 milliards d'images

Variété: Données stockées peuvent être à la fois des images, des sons, du texte…

Vitesse ou vélocité:
Vitesse à laquelle les données.
Les « like » sur Facebook, les tweets

CONSTATS

La réponse à cette problématique

1. Nouvelles formes de traitements et d'analyse de données

2. Nouveaux modes de stockage: Bases de données NoSQL : Haute disponibilité, Scalabilité

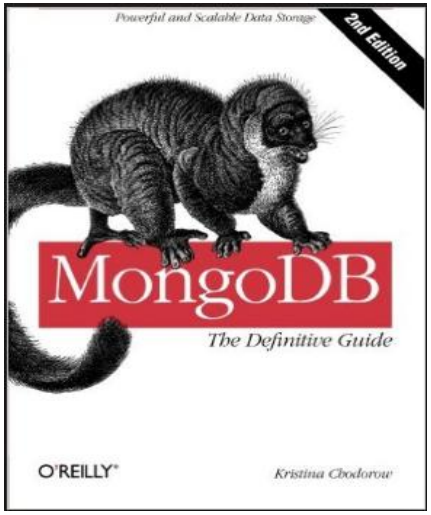
Clé/Valeur Colonne Graphe Document

Dans notre cas, nous nous intéressons à l'étude de mongoDB

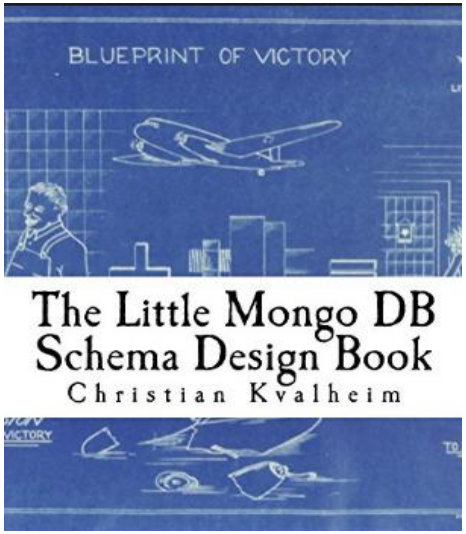
Prise en main de MongoDB

« administration & Développement »

Bibliographie



mongodb in action mongodb: the definitive guide



the little mongodb schema design

Site officiel: http://www.mongodb.org

Documentation: https://docs.mongodb.com/

Installation mongoDB Windows

1- Vérifier la version de son OS et son architecture, Taper les commandes suivantes dans l'invite de commandes windows:

> *wmic os get caption*

> *wmic os get osarchitecture*

2- Télécharger la version de mongoDB correspondant à l'OS (NB: Pour windows, choisir entre la version 32 bits et la version 64)

3- Lancer le fichier .exe et suivre les instructions

Installation

mongoDB

nécessite un répertoire .\data\db pour stocker des fichiers

On peut le spécifier :

- **soit lors du lancement du serveur :**

"C:\Program Files\MongoDB\Server\3.4\bin\mongod.exe" --dbpath d:\test\mongodb\data

- **soit dans le fichier config :**
- … \Server\3.4\ mongod.cfg

  Exemple de contenu de fichier config:

systemLog:

destination: file

path: C:\MongoDB_Donnees\data\log\mongod.log

storage:

- dbPath: C:\MongoDB_Donnees\data\db

Introduction - Bases NoSQL

- MongoDB fait partie d'une catégorie de systèmes de gestion de bases de données qu'on désigne sous le terme NoSQL.
- Ces systèmes, contrairement aux SGBD relationnels classiques, ne sont pas (pour la plupart) adressables par le biais de SQL.
- Ils ne structurent pas non plus les données sous une forme relationnelle « forte », mais utilisent des modèles alternatifs.
- Au delà de ces aspects, ils ont généralement pour caractéristiques d'être extrêmement **scalables**, et de pouvoir stocker de très larges quantités de données. Ils sont donc tout particulièrement adaptés aux problématiques relatives au big data.

Présentation
MongoDB

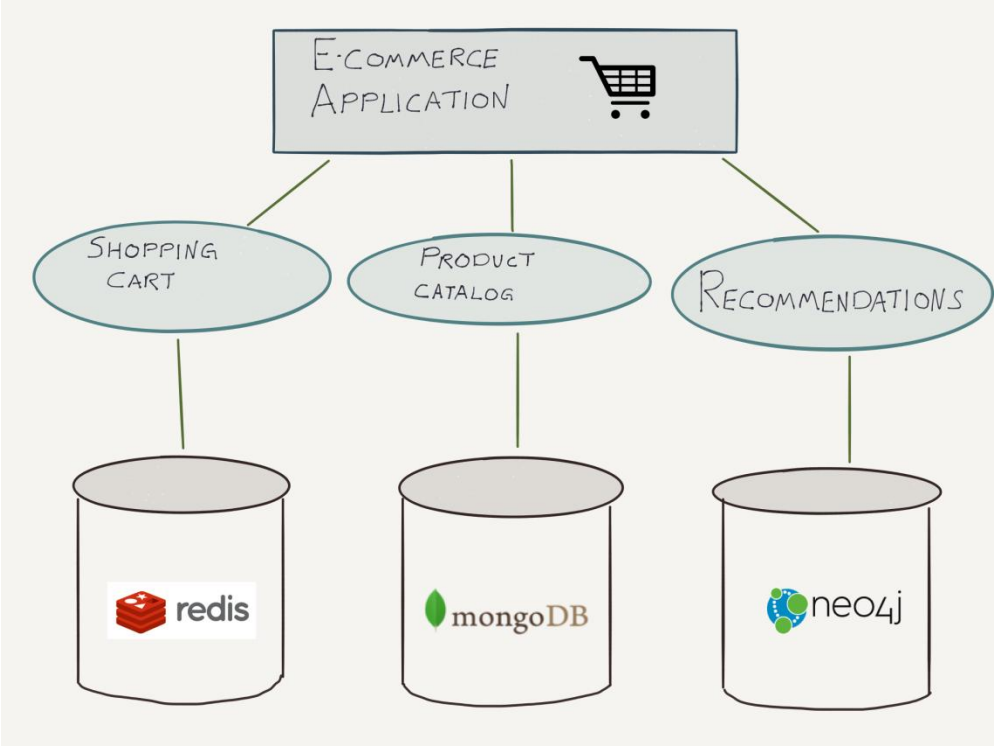
On les classe généralement dans 4 catégories distinctes:

- Les SGBD NoSQL **clef;valeur**: ils ont pour vocation de stocker de simples couples (clef;valeur); et sont donc pour des raisons évidentes tout particulièrement adaptés au map/reduce. Exemples: redis, riak.
- Les SGBD NoSQL **orientées document**: ils stockent les données sous la forme de documents au format variable mais structuré (généralement JSON ou XML); c'est à cette catégorie qu'appartient MongoDB.
- Les SGBD NoSQL **orientées graphe**: ils ont pour vocation de stocker les données sous la forme de graphes; et sont donc utilisés sont les use cases correspondants. Par exemple: Neo4J, OpenCog.
- Les SGBD NoSQL **orientées colonnes**: ils ont pour vocation de stocker les données sous la forme de colonnes. Par exemple: Cassandra
- NB : Il existe des bases de données relationnelles qui répondent également à des problématiques Big Data . Par exemple: HPE Vertica

Différents types de bases NoSql

Clé/Valeur | Colonne | Graphe | Document

Exemple d'utilisation Bases nosql



Source : https://neo4j.com/blog/neo4j-doc-manager-polyglot-persistence-mongodb/

Présentation
MongoDB

- MongoDB est un SGBD NoSQL document-based.
- Il s'agit d'un logiciel libre; initialement propriétaire (2007), il a été plus tard diffusé en licence AGPLv3 (2009).
- Il est développé et maintenu essentiellement par une entreprise, MongoDB Inc.; mais également par de nombreux bénévoles et entités tierces qui l'utilisent.
- Site officiel: http://www.mongodb.org
- Documentation: https://docs.mongodb.com/

Présentation
MongoDB

- MongoDB est utilisé aujourd'hui par de nombreux gros acteurs, comme par exemple: Ebay , Adobe, LinkedIn, ADP
- Ses performances sont excellentes; et sont estimées dans la plupart des benchmarks comme supérieures à la plupart des autres SGBD document-based équivalents (comme Cassandra).

Fonctionnement MongoDB

- MongoDB se présente sous la forme d'un serveur; un daemon/service système.
- Par défaut, pour y accéder, on se connectera en TCP/IP sur le port 27017.
- Comme la plupart des SGBD, il supporte la gestion d'utilisateurs/mots de passe pour s'identifier.
- Des APIs sont disponibles pour de nombreux langages: C/C++, Java, Python, PHP, PigLatin…
- Un connecteur Hadoop est par ailleurs disponible; permettant de lire les données d'entrée d'un programme Hadoop directement depuis MongoDB, et d'écrire les résultats d'un tel programme directement dans MongoDB également.

Fonctionnement - BSON

- MongoDB stocke les données non pas sous la forme de tables à la structure statique, mais sous la forme de documents rédigés dans un langage proche de JSON: BSON (Binary jSON).
- En interne, les données sont stockées sous un format binaire; mais elles sont évidemment la plupart du temps lues, écrites et manipulées sous un format textuel (c'est le SGBD qui compile le format texte).
- BSON est en réalité un *subset* de JSON qui inclut quelques fonctionnalités additionnelles (notamment des types et fonctions non disponibles en JSON).

Fonctionnement - BSON

Un exemple de document MongoDB :

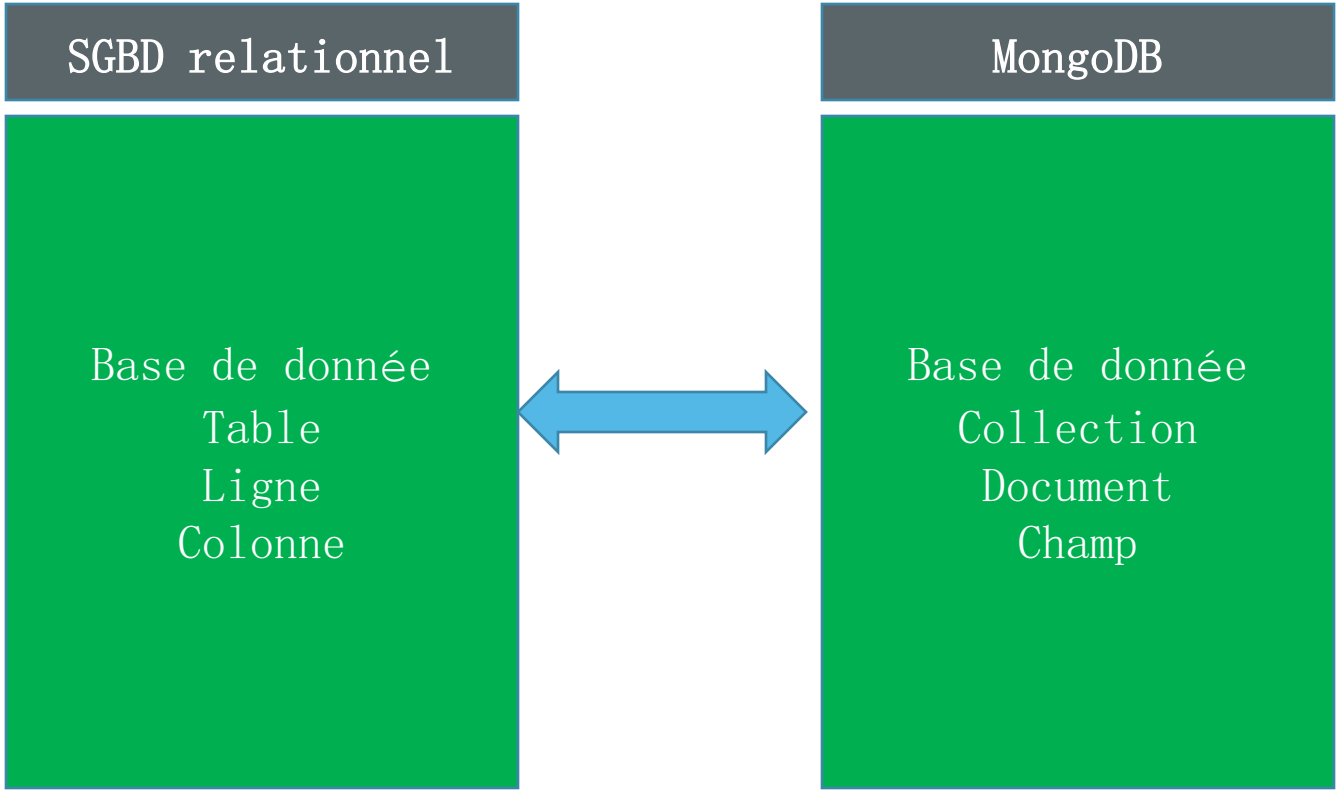
```
{
    prenom: 'John',
    nom: 'Smith',
    datenaissance: ISODate("1984-12-11T00:00:00Z"),
    "notes": [
            {
                "matiere" : 'Big Data',
                "note" : 12.0
            },
            {
                "matiere" : 'Gestion de projets',
                "note" : 14.0
            }
        ]
}
```

Fonctionnement - BSON

- Les documents sont par ailleurs insérés au sein de collections. Il s'agit de groupes de documents, généralement similaires.
- Il n'ont cependant pas l'obligation de respecter le même format, et peuvent comporter des champs différents.
- Au delà de ce concept, on retrouve également dans MongoDB le concept de base de données.
- Une base de données pourra ainsi contenir plusieurs collections, elles-mêmes contenant chacune de nombreux documents (ayant potentiellement un format variable même au sein d'une même collection).

Fonctionnement - BSON

- Chaque document comporte par ailleurs quoiqu'il arrive un champs interne à MongoDB: _id, qui est garanti comme étant unique dans la collection contenant le document.
- Cela veut dire que le nom de champs « _id » est en théorie réservé; on peut cependant l'utiliser si on souhaite utiliser son propre format d'identifiant (mais cela est cependant déconseillé).
- Par défaut, lorsqu'on insère un nouveau document, si aucun _id n'est spécifié, alors MongoDB en générera un par lui même.
- On peut également le spécifier explicitement; il va alors de soit que la valeur spécifiée ne doit pas déjà exister au sein de la collection (si c'est le cas, l'insertion échouera).


SGBD relationnel
MongoDB
Base de donnée
Table
Ligne
Colonne
Base de donnée
Collection
Document
Champ

Fonctionnement - BSON

- Par défaut, si on ne spécifie pas de valeur explicite pour le champs _id, alors MongoDB en générera un de 24 caractères alpha-numériques (en réalité 12 octets binaires, représentés sous une forme hexadécimale).

Par exemple: 56563b19f4c8c7c9597d7d95

- Par défaut, cet identifiant est composé:
- D'un timestamp sur 4 octets (indiquant la date/heure de création).
- D'un identifiant de machine sur 3 octets.
- D'un identifiant de processus sur 2 octets.
- D'un compteur sur 3 octets, démarrant à une valeur aléatoire.

Cela veut dire que, pour tout document, on dispose d'ores et déjà d'une date de création sans avoir à créer de champs explicite.

Fonctionnement - BSON - Types

- Les types principaux suivants sont disponibles au sein de BSON (il y en a d'autres):
- String
- Integer (32 ou 64 bits selon l'architecture)
- Boolean
- Double
- Tableaux (structure JSON classique)
- Timestamp Date (au format ISO)
- ObjectID (un identifiant unique binaire de document MongoDB)
- Binary

Fonctionnement - BSON

- Lorsqu'on insère un champs dans un document, par défaut, MongoDB choisira le type qui lui semble le plus approprié par rapport à la valeur spécifiée.
- On peut également explicitement spécifier le type, par le biais d'appel à des fonctions.
- Par exemple:

```
{
    prenom: 'John',
    nom: String('Smith'),
    Note: NumberInt(5)
}
```

Collection - Documents

Exemple :

Imaginons qu'on souhaite stocker la liste des élèves d'une promotion avec, pour chacun d'entre eux:

- Des informations générales (nom, prénom, date de naissance, etc.).
- Une liste de notes obtenues à des examens.
- Une liste de remarques liées au dossier scolaire.

Comment (dans combien de tables) serait stockées ces informations sur un SGBD relationnel classique ?

Collection - Documents

Exemple :

Imaginons qu'on souhaite stocker la liste des élèves d'une promotion avec, pour chacun d'entre eux:

- Des informations générales (nom, prénom, date de naissance, etc.).
- Une liste de notes obtenues à des examens.
- Une liste de remarques liées au dossier scolaire.

Comment (dans combien de tables) serait stockées ces informations sur un SGBD relationnel classique ?

Sur un SGBD classique, on stockerait ces informations par le biais d'au moins trois tables, par exemple « eleves », « notes », et « remarques »

Avec une jointure assurée par exemple par un identifiant numérique situé dans la table « eleves » et référencé au sein des deux tables « notes » et « remarques » (par exemple via un champs id_eleve).

Collection - Documents

Exemple :

Si on souhaite en revanche stocker ces données au sein de MongoDB (et d'une manière générale dans un SGBD NoSQL document-based), on utilisera à la place probablement simplement une seule collection « eleves »; dans laquelle on inclura un document pour chacun des élèves et qui contiendra toutes les informations (y compris les notes et les remarques, à quantité variable).

Les documents auraient alors probablement par exemple une structure de ce type:

```
{
    prenom: 'John',
    nom: 'Smith',
    ...
    notes: [12, 14.5, 8.5, 17],
    remarques: ["retard le 12/10", "redoublement envisagé"]
}
```

Documents - Relations

- L'approche montrée dans l'exemple précédent constitue déjà une forme d'implémentation d'une relation (ici 1:N entre un élève et ses notes/ses remarques).
- On peut également au sein d'un document référencer un autre document. Pour ce faire, on pourra plutôt que d'inclure les données directement au sein du même document spécifier une série d'ObjectIDs: ceux des adresses au sein d'une autre collection adresse_eleves, par exemple.

Shell MongoDB

- Un peu comme MySQL (et beaucoup d'autres SGBD), MongoDB est livré avec un logiciel client console: le shell MongoDB.
- C'est cet outil qu'on utilisera souvent pour créer des collections, les consulter, effectuer des tâches de maintenance, consulter l'état actuel des bases de données et des collections qu'elles contiennent, etc.
- Ce shell est accessible par le biais de la commande: ***mongo***
- (on peut par ailleurs spécifier des options variées indiquant comment se connecter à l'instance MongoDB visée, avec quel utilisateur, etc.)

Lancement du Shell MongoDB

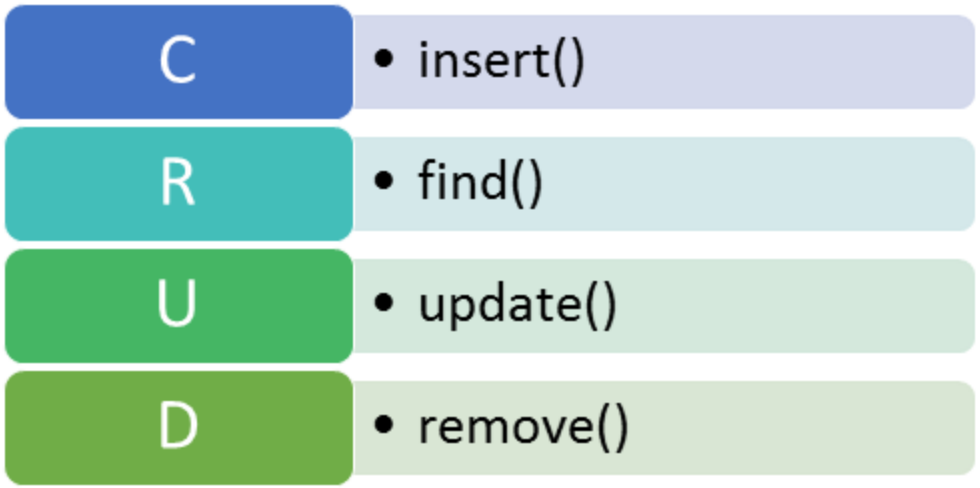
- Lancer le serveur mongo dans une fenêtre : **mongod**

"C:\Program Files\MongoDB\Server\3.4\bin\mongod.exe"

- Lancer dans une fenêtre le client : **mongo**

"C:\Program Files\MongoDB\Server\3.4\bin\mongo.exe"

OPERATIONS DE BASE



Instructions

- *use nom_de_la_Base* : Pour utiliser une base dont le nom est *nom_de_la_Base* si elle existe. Si la base n'existe pas, elle est créée.
- *db* : Permet de connaître la base courante
- show databases : Affiche les bases de données du serveur
- *show collections*: Affiche les collections d'une base de donnée

Les commandes peuvent être utilisées sur une base ou sur une collection:

- *db.help()*: Affiche une liste des commandes qui peut être appliquée sur une base

db.getCollectionNames( ): Affiche la liste des collections de la base

Instructions

Création d'une collection

Une collection est créée lors de l'insertion du premier document

> db.exemple1.insert({nom: 'Philippe', sexe: 'm', poids: 70})

Cette instruction va créer la collection exemple1 et va insérer un document JSON

> db.exemple1.find() : affiche tous les documents contenus dans la collection exemple1

> db.exemple1.find().pretty() : affiche tous les documents contenus dans la collection exemple1 de façon plus lisible

```
> db.exemple1.insert({nom: 'Jeanne', sexe: 'f', poids: 70, age:25})
```

Cette instruction va insérer un nouveau document dans la collection exemple1.

```
> db.exemple1.find().pretty()
```

Comparer les documents « Philippe » et « Jeanne »

Shell MongoDB

Exemples

```
>db.exemple1.insert({nom:'Christian',prenom: 'Romain', sexe: 'm', poids: 70, age:43,
   Niveau:'ING2' })

>db.exemple1.insert({nom:'Teresa',prenom:'Nadia', sexe: 'f', poids: 70, age:30,
   Niveau:'ING3',ville: 'Créteil'})

>db.exemple1.insert({nom:'Antoine', sexe: 'm', poids: 70, age:43, Niveau:'ING1'

>db.exemple1.insert({nom:'karima', sexe: 'f', poids: 70, age:30,
   Niveau:'ING2',ville:'Lognes'})
```

```
> db.exemple1.find({age: {$gte:30}}) …
```

pour sélectionner les élèves ayant 30 ans ou plus, ou encore

```
> db.exemple1.find({$or: [{age:{$lt:20}}, {Niveau:{$ne:'ING2'}} ]})
```

- Pour sélectionner les élèves ayant moins de 21 ans OU tous ceux n'étant pas du niveau « ING2 ».

Plutôt que d'indiquer une valeur fixe dans ces instructions conditionnelles, on peut également utiliser des opérateurs tel que « supérieur à », « inférieur à », etc. Ils respectent la syntaxe suivante: {CHAMP: {OPERATEUR:VALEUR}}

Et on dispose des opérateurs suivantes:

- $gt: plus grand que.
- $gte: plus grand ou égal à.
- $lt: plus petit que.
- $lte: plus petit ou égal à.
- $ne: différent de.

Shell MongoDB

- On peut également indiquer, par le biais d'un second argument à find(), quels champs on souhaite sélectionner
- Si ce second argument n'est pas spécifié, tous les champs du document sont renvoyés
- La syntaxe: .find({…},{FIELD1:[0|1],FIELD2:[0|1],…})

(par défaut, tous les champs sont à 1

Exemple:

```
db.eleves.find({}, {prenom:0,nom:0})
```

Pour sélectionner tous les élèves mais sans afficher leurs noms et prénoms.

- On peut également appliquer un équivalent de l'opérateur LIMIT par le biais des fonctions limit() et skip() à appliquer respectivement à find() et à limit().
- **Syntaxe:**

.find(…).limit(N) … pour sélectionner les N premiers documents.

.find(…).limit(N).skip(M) … pour sélectionner les N premier documents à partir du Mième document.

Exemple:

```
db.eleves.find().limit(10).skip(5)
```

Shell MongoDB

- On peut également trier les documents résultant de notre recherche.
- Pour ce faire, on va utiliser la fonction .sort() à appliquer après find().
- Syntaxe: .find(…).sort({FIELD1:[1|-1], FIELD2:[1|-1], …) …où 1 designe un ordre ascendant et -1 un ordre descendant.

Exemple:

db.exemple1.find().sort({prenom:1})

qui serait l'équivalent d'un « ORDER BY prenom ASC » en SQL

Shell MongoDB

Quelques opérateurs de find

- $exists : Pour exprimer la présence ou l'absence d'un champ

db.exemple1.find({ ville: {$exists: false}})→ Champ absent

- $in Pour rechercher une des valeurs présentes dans un tableau
- $or et $and : pour exprimer des conditions sur plusieurs champs

Exemples:

db.exemple1.find({sexe: 'f', $or: [{Niveau: 'ING2'}, {age: {$lt: 40}}]})

db.exemple1.find({ $and: [{Niveau: 'ING2'}, {age: {$gt: 18}}]})

Shell MongoDB :
update

// **Sans $set**

```
db.exemple1.update({nom:'Christian'},{Niveau:'ING2'})
```

Quel est le résultat de la requête?

// **Avec $set**

```
>db.exemple1.insert({nom:'Christelle',prenom: 'Romain', sexe:
   'f', poids: 40, age:21, Niveau:'ING2'})
db.exemple1.update({nom:'Christelle'},{$set:{Niveau:'ING2'}})
```

Remarque: $unset permet de supprimer un champ dans le document.

Shell MongoDB : Mise à jour

- Pour mettre à jour un document, on utilisera la méthode update().
- Syntaxe:

db.COLLECTION.update({CRITERIA}, {$set: {FIEL1:VAL1, FIEL2:VAL2, ...}})

Par exemple:

- db.eleves.update({eleve_id:'781638'},{$set:{prenom: 'Smith'}})
- Ou encore:
  db.eleves.update({$or:[{nom:'Smith'},{prenom:'John'}]},
  {$set:{notes:[12, 13, 14.5]}})

Shell MongoDB: update

- Par défaut, si le critère spécifié correspond à plusieurs enregistrements, MongoDB n'en mettra à jour qu'un seul.
- Si on souhaite qu'il mette à jour tous les documents correspondant au critère, on devra définir un troisième argument « multi » à true:
- *db.COLLECTION.update({CRITERIA}, {$set: {FIEL1:VAL1, FIEL2:VAL2, …}}, {multi:true})*
- Exemple: db.eleves.update({year:'2015'},{$set:{section: 'MBDS'}}, {multi:true})

Shell MongoDB: update

- Il est également possible de faire de telle sorte que si aucun document ne correspond aux critères pour la mise à jour, que mongoDB le crée. Pour ce faire, on utilise **{upsert:true}**
- Si on souhaite qu'il mette à jour tous les documents correspondant au critère, on devra définir un troisième argument « multi » à true:
- *db.COLLECTION.update({CRITERIA}, {$set: {FIEL1:VAL1, FIEL2:VAL2, …}}, {upsert:true})*
- Exemple: db.eleves.update({year:'2015'},{$set:{section: 'MBDS'}}, {upsert:true})

Shell MongoDB:
update

Différents opérateurs pour update

- $inc : pour incrémenter la valeur du champ
- $mul Multiplie la valeur du champ par le nombre présenté
- $rename : renomme un champ
- $set : mets la valeur dans le champ du document
- $unset : supprime le champ correspondant dans le document

Shell MongoDB

- On peut également remplacer un document en utilisant la méthode save().
- syntaxe: db.COLLECTION.save(DOCUMENT_BSON)
- … si le document spécifié contient un identifiant _id et que celui-ci existe déjà dans la collection, alors le document ayant cet identifiant sera remplacé par le document spécifié.
- Si en revanche aucun identifiant _id n'est spécifié ou si il est spécifié mais que cet identifiant n'existe pas dans la collection, alors un nouveau document sera inséré.

Shell MongoDB : remove

- Enfin, pour supprimer un élément, on utilisera la fonction remove(). Syntaxe:

db.COLLECTION.remove({CRITERE},[0|1])

- … Avec le premier argument un critère de suppression, et le second indiquant s'il est positionné à 1 qu'un seul enregistrement doit être supprimé même si plusieurs enregistrements correspondent au critère.
- **Exemple:**

db.eleves.remove({nom:'John',prenom:'Smith'},0)

Shell MongoDB

- ● Enfin, on peut également stocker les résultats de nos requêtes dans des variables, par le biais de la syntaxe:
- var MAVARIABLE=db.COLLECTION.find(…)
- Et ensuite accéder aux champs du résultat par le biais de la syntaxe: MAVRIABLE[X][CHAMP]
- … pour obtenir le champs CHAMP de la ligne de résultat X.

```
{
        "prenom" : "John",
        "nom" : "Smith",
        ...
        "examen_ids": [ObjectId("565662cbf4c8c7c9597d7d99"),
                       ObjectId("565333cbf4c8c7c9597d7d99") ]
}
```

Shell MongoDB

- On peut d'ailleurs utiliser cette possibilité pour référencer au sein de nouvelles requêtes des champs issus de résultats de requêtes précédentes.
- Imaginons qu'on ait une collection « eleves » dont les documents contiennent un tableau « examen_ids » référençant des identifiants d'une seconde collection «examens ».

Shell MongoDB

- On pourrait alors faire par exemple:

```
var data=db.eleves.find({nom: 'Antoine', prenom: 'Rudy'})
```

- Suivi de:

```
var exams=db.examens.find({_id:{"$in":data['examen_ids']}})
```

… pour obtenir les documents de la collection « examens » pour l'élève « Antoine Rudy », et les stocker dans une variable « exams ».

Shell MongoDB: quelques commandes

- *db.dropDatabase()*

Pour supprimer la base de données courante.

- *db.createCollection(COLLECTION)*

Pour créer une nouvelle collection.

- *db.COLLECTION.drop()*

Pour supprimer la collection COLLECTION.

- *db.COLLECTION.insert(DOCUMENT_BSON)*

Pour insérer le document spécifié dans la collection COLLECTION.

Si la collection n'existe pas, elle sera créée. On peut donc créer une collection également de cette manière.