mongoDB

CRUD

Mme Ines Slimene
Ines_slimene@yahoo.fr

Plan module

Introduction

Base de données orienté document : MongoDB

Base de données orienté clé/valeur : Redis

Base de données orienté colonne : Cassandra

Base de données orienté graphe : Neo4j

Plan

MongoDB Shell

Type de données

Insertion

Lecture

Mise à jour

Suppression

GridFS

Mongo Shell

Le meilleur moyen d'interroger MongoDB est d'utiliser le shell.

Toute l'administration de MongoDB se fait grâce au shell.

Le shell MongoDB est très simple à utiliser.

Mongo Shell

Démarrage : mongod

Connexion : mongo

Arrêter la base MongoDB : Utiliser la commande **shutdown**

```
> use admin    //connection a la base admin
> db.shutdownServer();   //arrêter le serveur
```

Mongo Shell

Afficher la base de données courante : db

Afficher la liste des bases de données : show dbs

Sélectionner une base de données : use <name>

Afficher les collections : show collections

Supprimer une base de données : db.runCommand({dropDatabase: 1})

Type de données

Type	Exemple
Int/double/..	{a:1}
boolean	{b:true}
String	{c:’hello’}
Array	{d:[1,2,3]}
Date	{e:ISODate(« 2012-12-19 »)}
Object	{g:{a:1,b:true}}

Mongo Shell

Les commandes ont la syntaxe suivante :

db.<collection>.<methode>

Exemple :

- db.inventory.find({ qty: { $gt: 20 } })
- db.val.insert({name: "Olivier", etude : "Master"})

CRUD

Create

- db.collection.insert(<document>)
- db.collection.update(<query>, <update>, { upsert: true })

Read

- db.collection.find(<query>, <projection>)
- db.collection.findOne(<query>, <projection>)

Update

- db.collection.update(<query>, <update>, <options>)

Delete

- db.collection.remove(<query>, <justOne>)

Insertion

```
> db.people.insert({nom:"smith",age:30,profession:"ingenieur"})
WriteResult({ "nInserted" : 1 })
> db.people.find()
{ "_id" : ObjectId("5444f7e49f88b7a61cd8d8b2"), "nom" : "smith", "age" : 30, "pr
ofession" : "ingenieur" }
>
```

on voit un champ de nom _id et de type *ObjectId* a été inséré.

- Tout document doit avoir un identifiant unique dans la collection,
- Si l'on ne le précise pas, MongoDB se charge de le créer à notre place.
- Cet identifiant ne peut pas être modifié et constitue la clé primaire de la collection.

On peut préciser le type d'un champ en utilisant :

```
db.people.insert({nom:"smith",age:NumberInt(12)})
```

Insertion : Garantie de l'écriture

Suite a une insertion ou mise a jours nous n'avons pas un accusée de réception de la modification

Pour voir le résultat de la requête on utilise :

- getLastError

Application : exécution de la requête en mode « safe »

Dans une application, la commande getLastError a deux paramètres:

- w : attendre un accusé de réception d'écriture, mais dans la mémoire et non pas l'écriture sur le disque. (valeur : 0, 1, N, majority) N:accuseé de réception pour N nœuds
- j : écrire dans un fichier de journalisation suite a l'écriture en mémoire.(valeur : 0, 1) (garantie la validation du journal du master uniquement)

Différents types de requêtes

la commande *findOne()* renvoi au plus un document. Sans argument, elle retourne un document quelconque de la collection

```
> db.people.insert({nom:"Jean",age:38,profession:"medecin"})
WriteResult({ "nInserted" : 1 })
> db.people.findOne()
{
        "_id" : ObjectId("5444f7e49f88b7a61cd8d8b2"),
        "nom" : "smith",
        "age" : 30,
        "profession" : "ingenieur"
}
> db.people.find()
{ "_id" : ObjectId("5444f7e49f88b7a61cd8d8b2"), "nom" : "smith", "age" : 30, "pr
ofession" : "ingenieur" }
{ "_id" : ObjectId("5444f8b29f88b7a61cd8d8b3"), "nom" : "Jean", "age" : 38, "pro
fession" : "medecin" }
>
```

Différents types de requêtes

La commande find peut prendre deux paramètres:

```
> db.people.find({nom:"smith"},{nom:true,age:true})
{ "_id" : ObjectId("5444f7e49f88b7a61cd8d8b2"), "nom" : "smith", "age" : 30 }
```

Le **premier** est un document qui constitue le **critère de recherche**. Le document retourné sera identique au document spécifié (sur les champs fournis bien sûr). Cette première partie correspond donc à la clause WHERE de SQL.

Le **second** est un document qui permet de **restreindre les champs** du document retourné, à la manière de la clause SELECT de SQL. En son absence, tous les champs sont retournés. On note aussi que l'identifiant est toujours retourné, à moins que l'on dise explicitement le contraire.

Lecture : Utilisation d'opérateurs

Opérateur	Signification	Exemple
$gt, $gte, $lt, $lte	>, >=, <, <=	score: { $gt: 95, $lte: 98 }
$exists	test sur l'existence d'un champ	profession: { $exists: true }
$type	test sur le type d'un champ	name: { $type: 2 }
$regex	recherche de pattern dans une chaine	name: { $regex: "e$" }
$or	OU logique sur les clauses fournies dans le tableau	$or: [{ name: { $regex: "e$"} }, age: { $exists: true}]
$and	ET logique sur les clauses fournies dans le tableau	$and: [{ name: { $regex: "e$"} }, age: { $exists: true}]

Lecture : Utilisation d'opérateurs

db.people.find()	Select * from people
db.people.find({"age" : 27})	Select * from people where age=27
db.people.find({}, {“nom" : 1, “age" : 1})	Select nom,age from people
db.people.find({}, {“nom" : 0})	Select age,profession from people
db.people.find({"age" : {"$gte" : 18, "$lte" : 30}})	Select * from people where age between 18 and 30
db.raffle.find({"ticket_no" : {"$in" : [725, 542, 390]}})	Select * from raffle where ticket_no in (725,542,390)

Lecture : Utilisation d'opérateurs

Si l'on donne plusieurs conditions pour le même champ, c'est la dernière qui va gagner car le shell va construire un premier objet correspondant à la première condition, puis écraser cet objet quand il va lire la seconde condition.

```
> db.people.find({"age" : {"$lte" : 32},age:{ "$gte" : 35}})
{ "_id" : ObjectId("5444f8b29f88b7a61cd8d8b3"), "nom" : "Jean", "age" : 38, "pro
fession" : "medecin" }
> db.people.find({"age" : {"$lte" : 32,"$gte" : 35}})
```

Lecture : Utilisation d'opérateurs

Afficher les collections qui contiennent le champ **age**

```
db.people.find({age:{$exists:true}})
```

Afficher les collections dont le type du champ nom est String

```
db.people.find({nom:{$type:2}})
```

Nom qui contient la lettre i : `db.people.find({nom:{$regex:"i"}})`

Nom qui se termine par h :`db.people.find({nom:{$regex:"h$"}})`

Nom qui commence par : `db.people.find({nom:{$regex:"^s"}})`

Lecture : Utilisation d'opérateurs

Operateur OR :

```
db.people.find({$or:[{name:{$regex:"e$"}},{age:{$exists:true}}]})
```

Operateur AND :

```
db.people.find({$and:[{name:{$regex:"e$"}},{age:{$exists:true}}]})
```

Lecture : Utilisation d'opérateurs

Afficher les utilisateurs dont le nom est Smith ou Bob

```
db.users.find( { nom : { $in : [ "smith" , "Bob" ] }})
```

Afficher les utilisateurs dont leurs préférences sont running et pickles.

```
db.users.find({favorites:{$all:["running","pickles"] }})
```

Lecture : Utilisation d'opérateurs

limit(): pour récupérer les n premiers résultats uniquement

```
> db.people.find().limit(1)
{ "_id" : ObjectId("5444f7e49f88b7a61cd8d8b2"), "nom" : "smith", "age" : 30, "pr
ofession" : "ingenieur" }
```

sort(): pour trier les résultats

```
> db.people.find().sort({nom:1})
{ "_id" : ObjectId("5444f8b29f88b7a61cd8d8b3"), "nom" : "Jean",
fession" : "medecin" }
{ "_id" : ObjectId("5444f7e
ofession" : "ingenieur" }
```

```
> db.people.find().sort({nom:-1})
{ "_id" : ObjectId("5444f7e49f88b7a61cd8d8b2"), "nom" : "smith"
ofession" : "ingenieur" }
{ "_id" : ObjectId("5444f8b29f88b7a61cd8d8b3"), "nom" : "Jean",
fession" : "medecin" }
```

count(): retourne le nombre de documents satisfaisant la requête.

```
> db.people.find().count()
2
```

Requêtes et sous-documents

Sous document spécifique

```
> db.article.insert({titre : "azerty",auteur: {prénom:'David', nom : 'Wursteisen
' }})
WriteResult({ "nInserted" : 1 })
> db.article.find()
{ "_id" : ObjectId("544501819f88b7a61cd8d8b4"), "titre" : "azerty", "auteur" : {
 "prénom" : "David", "nom" : "Wursteisen" } }
```

Afficher l'article dont le nom de l'auteur est Wursteisen :

```
> db.article.find({"auteur.nom":"Wursteisen"})
{ "_id" : ObjectId("544501819f88b7a61cd8d8b4"), "titre" : "azerty", "auteur" : {
 "prénom" : "David", "nom" : "Wursteisen" } }
```

Mettre à jour les documents

La méthode *update()* de MongoDB possède quatre variantes:

- La première permet la **mise à jour globale** des documents

```
> db.people.update({nom:'smith'},{nom:'Alain',salaire:5000})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.people.find()
{ "_id" : ObjectId("5444f7e49f88b7a61cd8d8b2"), "nom" : "Alain", "salaire" : 500
0 }
```

- La seconde variante permet une **mise à jour sélective** des champs. On utilise pour cela de nouveaux opérateurs. Le premier et probablement le plus utile est l'opérateur *$set*:

```
> db.people.update({nom:'Jean'},{$set:{salaire:5000}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.people.find()
{ "_id" : ObjectId("5444f7e49f88b7a61cd8d8b2"), "nom" : "Alain", "salaire" : 500
0 }
{ "_id" : ObjectId("5444f8b29f88b7a61cd8d8b3"), "nom" : "Jean", "age" : 38, "pro
fession" : "medecin", "salaire" : 5000 }
```

Mettre à jour les documents

- On peut aussi utiliser l'opérateur *$inc* sur les champs entiers:

```
> db.people.update({nom:'Jean'},{$inc:{age:1}})
```

```
{ "_id" : ObjectId("5444f8b29f88b7a61cd8d8b3"), "nom" : "Jean", "age" : 39, "pro
fession" : "medecin", "salaire" : 5000 }
```

- L'opérateur suivant, *$unset*, permet de supprimer un champ d'un document:

```
{ "_id" : ObjectId("5444f8b29f88b7a61cd8d8b3"), "nom" : "Jean", "age" : 39, "pro
fession" : "medecin", "salaire" : 5000 }
> db.people.update({nom:'Jean'},{$unset:{salaire:1}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.people.find({nom:'Jean'})
{ "_id" : ObjectId("5444f8b29f88b7a61cd8d8b3"), "nom" : "Jean", "age" : 39, "pro
fession" : "medecin" }
```

Mises à jour de plusieurs documents

Les requêtes de mise à jour ne touchent qu'un seul document.

Si on souhaite mettre à jour **plusieurs** documents, il faut passer un troisième argument à la méthode update()

```
> db.people.update({nom:'Jean'},{$set:{salaire:5000}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.people.find({nom:'Jean'})
{ "_id" : ObjectId("5444f8b29f88b7a61cd8d8b3"), "nom" : "Jean", "age" : 39, "pro
fession" : "medecin", "salaire" : 5000 }
{ "_id" : ObjectId("544512089f88b7a61cd8d8b5"), "nom" : "Jean", "age" : 34 }
```

```
> db.people.update({nom:'Jean'},{$set:{salaire:5000}} {multi:true})
WriteResult({ "nMatched" : 2, "nUpserted" : 0, "nModified" : 2 })
> db.people.find({nom:'Jean'})
{ "_id" : ObjectId("5444f8b29f88b7a61cd8d8b3"), "nom" : "Jean", "age" : 39, "pro
fession" : "medecin", "salaire" : 5000 }
{ "_id" : ObjectId("544512089f88b7a61cd8d8b5"), "nom" : "Jean", "age" : 34, "sal
aire" : 5000 }
```

Mettre à jour un champ de type tableau

Création champ de type tableau:

```
> db.people.insert({nom:'Smith',age:34,activite:['sport','musique']})
```

modifier la deuxième valeur du tableau "activité" pour lui attribuer la valeur "camping".

```
> db.people.update({nom:'Smith'},{$set:{"activite.1":'camping'}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.people.find({nom:'Smith'})
{ "_id" : ObjectId("544518359f88b7a61cd8d8b7"), "nom" : "Smith", "age" : 34, "ac
tivite" : [ "sport", "camping" ] }
```

Ajouter une valeur au tableau des activités (*$push)*

```
> db.people.update({nom:'Smith'},{$push:{"activite":'musique'}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.people.find({nom:'Smith'})
{ "_id" : ObjectId("544518359f88b7a61cd8d8b7"), "nom" : "Smith", "age" : 34, "ac
tivite" : [ "sport", "camping", "musique" ] }
```

Mettre à jour un champ de type tableau

L'opérateur *$pop* permet de retirer une valeur du tableau.

- Si la valeur passée à l'opérateur est 1, ce sera la dernière valeur qui sera supprimée,
- si c'est -1 ce sera la première valeur.

```
> db.people.update({nom:'Smith'},{$pop:{"activite":1}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.people.find({nom:'Smith'})
{ "_id" : ObjectId("544518359f88b7a61cd8d8b7"), "nom" : "Smith", "age" : 34, "ac
tivite" : [ "sport", "camping" ] }
```

Pour spécifier l'élément a retirer (*$pull)*

```
> db.people.update({nom:'Smith'},{$pull:{"activite":'camping'}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.people.find({nom:'Smith'})
{ "_id" : ObjectId("544518359f88b7a61cd8d8b7"), "nom" : "Smith", "age" : 34, "ac
tivite" : [ "sport", "musique" ] }
```

Suppression de documents

Delete :

db.people.remove({})

```
> db.people.remove({nom:"Jean"})
WriteResult({ "nRemoved" : 3 })
```

Drop collection :

db.article.drop()

```
> show collections
article
people
system.indexes
val
> db.article.drop()
true
> show collections
people
system.indexes
val
```

GridFS

GridFS est une spécification MongoDB qui permet le stockage et la lecture des fichiers a grande taille comme les images, les fichiers audio et vidéo, etc.

C’est une sorte de système de fichier. Mais le stockage se fait dans des collections MongoDB.

GridFS offre la possibilité de stocker des fichiers même si leurs tailles sont supérieures a 16M.

GridFS

GridFS divise un fichier en morceaux (chunks) et enregistre chaque morceaux de données dans un document, chacun de taille max 255k.

GridFS utilise par défaut deux collections **fs.files** et **fs.chunks** pour enregistrer les métadonnées des fichiers et les morceaux.

Chaque morceau est identifié par un unique _id.

le fs.files présente le document père. Le champ **files_id** dans le document fs.chunks relie les morceaux a leur père.

GridFS

Insérer la sequence audio song.mp3 dans mongoDB

- $ mongofiles -d **gridfs** put **song.mp3**

gridfs est le nom de la base dans laquelle le fichier serait inséré.

$ mongo gridfs

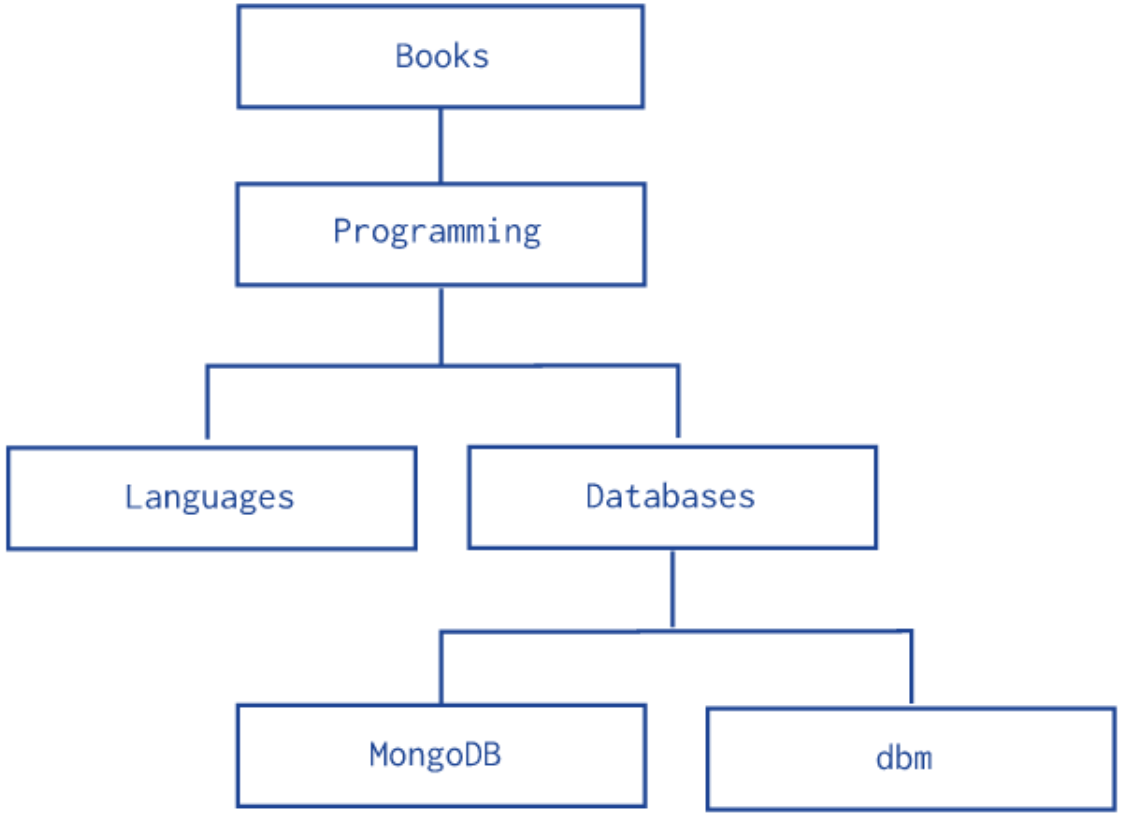
> db.fs.files.find()

```
> db.fs.files.find().pretty()
{
        "_id" : ObjectId("5458ea27cd2a7c2f3fbaf181"),
        "filename" : "song.mp3",
        "chunkSize" : 261120,
        "uploadDate" : ISODate("2014-11-04T15:00:56.241Z"),
        "md5" : "f0fb6c525cc9b53ca8b047763abb1220",
        "length" : 4001281
}
```

> db.fs.chunks.count({files_id:ObjectId('5458ea27cd2a7c2f3fbaf181')})

```
> db.fs.chunks.count({files_id:ObjectId('5458ea27cd2a7c2f3fbaf181')})
16
```

Structure en arbre



Structure en arbre

```
db.categories.insert( { _id: "MongoDB", children: [] } )
db.categories.insert( { _id: "dbm", children: [] } )
db.categories.insert( { _id: "Databases", children: [ "MongoDB", "dbm" ] } )
db.categories.insert( { _id: "Languages", children: [] } )
db.categories.insert( { _id: "Programming", children: [ "Databases", "Languages" ] } )
db.categories.insert( { _id: "Books", children: [ "Programming" ] } )
db.categories.findOne( { _id: "Databases" } ).children
db.categories.find( { children: "MongoDB" } )
```