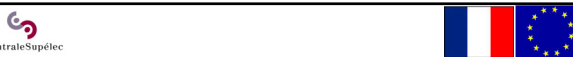


CentraleSupélec



Big Data : Informatique pour les données et calculs massifs

## 10 – Utilisation de MongoDB

Stéphane Vialle

Stephane.Vialle@centralesupelec.fr  
http://www.metz.supelec.fr/~vialle

CentraleSupélec

- Premiers pas en MongoDB
- Importation et exportation de documents
- Interrogation des données avec *find(...)*
- Application de méthodes aux collections
- Agrégation d'opérations avec *aggregate(...)*
- *Map-Reduce* en MongoDB
- JOIN en MongoDB
- Gestion des Index en MongoDB
- Comparaison des 3 mécanismes

2

CentraleSupélec

## Premiers pas en MongoDB

3

CentraleSupélec

## Premiers pas... (1)

Dans un premier *command shell*

```
C:\Users\vialle> mkdir \data0
```

Crée un répertoire de stockage.

```
C:\Users\vialle> mongod -dbpath \data0
```

Lance le serveur MongoDB sur ce répertoire.

Dans un second *command shell*

```
C:\Users\vialle> mongo
```

```
MongoDB shell version v3.4.1
connecting to: mongod://127.0.0.1:27017
MongoDB server version: 3.4.1
...
>
```

Lance un client MongoDB sur ce répertoire de stockage.

4

CentraleSupélec

## Premiers pas... (2)

Juste après la création de la BdD et la connexion d'un client :

```
C:\Users\vialle> dir \data0
```

07/05/2017	16:14	4	096	collection-0-8154577603626997227.wt
07/05/2017	16:14	4	096	collection-2-8154577603626997227.wt
07/05/2017	16:14	<REP>		diagnostic.data
07/05/2017	16:14	4	096	index-1-8154577603626997227.wt
07/05/2017	16:14	4	096	index-3-8154577603626997227.wt
07/05/2017	16:14	4	096	index-4-8154577603626997227.wt
07/05/2017	16:14	<REP>		journal
07/05/2017	16:14	5		mongod.lock
07/05/2017	16:14	4	096	sizeStorer.wt
07/05/2017	16:14	95		storage.bson
07/05/2017	16:14	46		WiredTiger
07/05/2017	16:14	0		WiredTiger.lock
07/05/2017	16:14	814		WiredTiger.turtle
07/05/2017	16:14	4	096	WiredTiger.wt
07/05/2017	16:14	4	096	WiredTigerLAS.wt
07/05/2017	16:14	4	096	_mdb_catalog.wt

5

CentraleSupélec

## Premiers pas... (3)

Changement de BdD, et création de la BdD et d'une de ses collection

```
>db
test
```

Par défaut on référence la base « test »...

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

Qui n'existe pas réellement, car elle est vide !  
Mais deux autres bases existent (toujours)

```
>use db-films
switched to db-films
```

On indique d'utiliser une autre base, qui n'existe pas encore

```
>db
db-films
```

On référence bien cette nouvelle base, qui n'existe pas...

```
>show collections
```

... et qui ne contient encore aucune « collection » (logique)

6

### Premiers pas... (4)

Insertion de deux enregistrements de structures différentes

```
>db.comedie.insert({titre : "Shrek", animation : true})
WriteResult({ "nInserted" : 1 })
>db.comedie.insert({titre : "Les visiteurs", acteurs : ["Clavier", "Reno"]})
WriteResult({ "nInserted" : 1 })
>db.comedie.find()
{ "_id" : ObjectId("590f3bc1ae2448d877bbf8ea"),
  "titre" : "Shrek", "animation" : true }
{ "_id" : ObjectId("590f3bcaae2448d877bbf8eb"),
  "titre" : "Les visiteurs", "acteurs" : [ "Clavier", "Reno" ] }
```

7

### Premiers pas... (5)

Affichage "confortable"

```
> db.comedie.find()
{ "_id" : ObjectId("590f79cd646823f59510e489"),
  "titre" : "Les visiteurs", "acteurs" : [ "Clavier", "Reno" ] }
> db.comedie.find().pretty()
{
  "_id" : ObjectId("590f79cd646823f59510e489"),
  "titre" : "Les visiteurs",
  "acteurs" : [
    "Clavier",
    "Reno"
  ]
}
```

Ajouter ".pretty()" permet de mieux afficher les données structurées

8

### Premiers pas... (6)

Mise à jour d'un enregistrement

```
>db.comedie.find()
{ "_id" : ObjectId("590f3bc1ae2448d877bbf8ea"),
  "titre" : "Shrek", "animation" : true }
{ "_id" : ObjectId("590f3bcaae2448d877bbf8eb"),
  "titre" : "Les visiteurs", "acteurs" : [ "Clavier", "Reno" ] }
> db.comedie.update({titre : "Shrek"}, {date : 2001})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.comedie.find()
{ "_id" : ObjectId("590f3bc1ae2448d877bbf8ea"), "date" : 2001 }
{ "_id" : ObjectId("590f3bcaae2448d877bbf8eb"),
  "titre" : "Les visiteurs", "acteurs" : [ "Clavier", "Reno" ] }
```

- On a écrasé tout l'enregistrement !
- Mais l'id est resté le même (c'est bien un update)

9

### Premiers pas... (7)

Mise à jour d'un enregistrement

```
> db.comedie.find()
{ "_id" : ObjectId("590f3bc1ae2448d877bbf8ea"), "date" : 2001 }
{ "_id" : ObjectId("590f3bcaae2448d877bbf8eb"),
  "titre" : "Les visiteurs", "acteurs" : [ "Clavier", "Reno" ] }
> db.comedie.update(
  { "_id" : ObjectId("590f3bc1ae2448d877bbf8ea") },
  { titre : "Shrek", animation : true, date : 2000 })
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.comedie.find()
{ "_id" : ObjectId("590f3bc1ae2448d877bbf8ea"),
  "titre" : "Shrek", "animation" : true, "date" : 2000 }
{ "_id" : ObjectId("590f3bcaae2448d877bbf8eb"),
  "titre" : "Les visiteurs", "acteurs" : [ "Clavier", "Reno" ] }
```

On corrige avec un nouvel update de tout l'enregistrement

→ On peut remplacer un enregistrement dans une collection d'une base

10

### Premiers pas... (8)

Mise à jour et ajout de champs d'un enregistrement

```
> db.comedie.find()
{ "_id" : ObjectId("590f3bc1ae2448d877bbf8ea"),
  "titre" : "Shrek", "animation" : true, "date" : 2000 }
{ "_id" : ObjectId("590f3bcaae2448d877bbf8eb"),
  "titre" : "Les visiteurs", "acteurs" : [ "Clavier", "Reno" ] }
> db.comedie.update({titre : "Shrek"},
  {$set : {date : 2001, avis : "*****"}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.comedie.find()
{ "_id" : ObjectId("590f3bc1ae2448d877bbf8ea"),
  "titre" : "Shrek", "animation" : true, "date" : 2001, "avis" : "*****" }
{ "_id" : ObjectId("590f3bcaae2448d877bbf8eb"), "titre" :
  "Les visiteurs", "acteurs" : [ "Clavier", "Reno" ] }
```

→ On peut modifier champ, ou ajouter un champ à un enregistrement

11

### Premiers pas... (9)

Effacement du contenu d'une collection

```
> db.comedie.find()
{ "_id" : ObjectId("590f3bc1ae2448d877bbf8ea"),
  "titre" : "Shrek", "animation" : true, "date" : 2001, "avis" : "*****" }
{ "_id" : ObjectId("590f3bcaae2448d877bbf8eb"), "titre" :
  "Les visiteurs", "acteurs" : [ "Clavier", "Reno" ] }
> db.comedie.remove({titre : "Shrek"})
WriteResult({ "nRemoved" : 1 })
> db.comedie.find()
{ "_id" : ObjectId("590f3bcaae2448d877bbf8eb"),
  "titre" : "Les visiteurs", "acteurs" : [ "Clavier", "Reno" ] }
> db.comedie.remove({})
WriteResult({ "nRemoved" : 1 })
> db.comedie.find()
> show collections
comedie
```

Effacement sélectif d'un enregistrement

Effacement de toute la collection

La collection est vide, mais existe encore !

12


### Premiers pas... (10)

#### Effacement complet d'une collection

```
> show collections
comedie

> db.comedie.drop()
true

> show collections
```



On peut supprimer :

- Certains éléments d'une collection
- Tous les éléments d'une collection
- La collection entière (ses éléments et elle-même)

13

### Premiers pas... (11)

#### Effacement d'une base

```
> show dbs
admin 0.000GB
db-films 0.000GB
local 0.000GB

> db
db-films

> db.dropDatabase()
{"dropped": "db-films", "ok": 1}

> show dbs
admin 0.000GB
local 0.000GB

> db
db-films
```

Base courante : db-film

On efface la base courante

Elle a bien disparu des enregistrements sur disque

Mais on continue à la référencer (ne pas oublier de passer à une autre base)

14

### Premiers pas... (12)

#### Renommage d'une base

On ne peut pas renommer une base de MongoDB. Il faut :

- la copier avec :  
`db.copyDatabase(fromdb, todb, fromhost, username, password, mechanism)`
- puis effacer la première version avec :  
`db.dropDatabase()`

```
> db.copyDatabase("db_to_rename", "db_renamed", "localhost")
> use db_to_rename
> db.dropDatabase();
```

#### Renommage d'une collection

```
> db.collection.renameCollection(newCollectionName, dropTarget)
```

- Ne fonctionne pas sur les collections « sharded »
- Ne peut pas déplacer une collection d'une base à une autre

15

### Premiers pas... (13)

#### Le client MongoDB permet de définir et d'exécuter du code Java Script

```
> function fact(n) {
... if (n == 1) return 1
... else return (n * fact(n-1))
... }

> fact (2)
2

> prompt = function() { return (new Date() + ">") }
Sun May 07 2017 22:42:39 GMT+0200>
```

Le client MongoDB contient un interpréteur de Java Script permettant de définir des fonctions et des variables

On peut associer chaque affichage de prompt à un calcul de fonction

La définition de fonctions de calcul en Java Script sera très utile/nécessaire pour l'écriture des fonctions `map()` et `reduce()` des **Map-Reduce** (voir plus loin)

16

### Premiers pas... (14)

#### Accès à des attributs composites/structurés

```
{entreprise : "Au bon produit",
  services : { direction : { etage : 5, effectifs : 6 }
              compatibilite : { etage : 5, effectif : 4 }
              innovation : { etage : 4, effectif : 12 }
              ....
            },
  adresse : "Metz Plage",
}
```

Un document structuré avec des sous-documents structurés

Queries

```
> find({"entreprise" : "Au bon produit"}, {"adresse" : 1}) ✓
> find({"entreprise" : "Au bon produit"}, {"adresse" : 1}) ✓
> find ({"entreprise" : "Au bon produit"}, {"services.innovation.etage" : 1}) ✓
> find ({"entreprise" : "Au bon produit"}, {"services.innovation.etage : 1}) ✗
```

Les chemins sur plusieurs niveaux doivent être entre " "

17

### Importation et exportation de documents

18

## Importation (1)

Importation de documents au format **JSON**

On utilise l'outil **mongoimport** : un exécutable à part entière, pas une commande du mongo shell

```
mongoimport --db dbName --collection collectionName
            --mode importMode --file fileName.json --jsonArray
```

Importe le contenu du fichier JSON indiqué, dans la base et la collection spécifiées, et les ajoute aux données déjà présentes.

- Le démon **mongod** doit être lancé, et **mongoimport** s'y connectera
- L'option **--mode** permet de préciser si on ajoute/mélange/remplace les données déjà présente dans la collection
- L'option **--jsonArray** permet d'importer des tableaux de documents JSON
- La commande **mongoimport** est riche en options de fonctionnement (voir la doc technique de MongoDB)

19

## Importation (2)

Importation de documents au format **JSON**

On utilise l'outil **mongoimport** : un exécutable à part entière, pas une commande du mongo shell

```
mongoimport --db dbName --collection collectionName
            --mode importMode --file fileName.json --jsonArray
```

Gestion de l'ajout des nouvelles données :

- Si la donnée lue ne possède pas de "**\_id**" : alors un "**\_id**" sera créé et lui sera affecté.
- Si la donnée lue possède un "**\_id**" déjà utilisé dans une donnée de la base :
  - mode insert** : indique une erreur
  - mode upsert** : remplace l'ancienne donnée par la donnée lue
  - mode merge** : ajoute les nouveaux champs présents dans le fichier JSON, remplace ceux déjà existants et redéfinis dans la donnée lue.

20

## Importation (3)

Importation de documents au format **JSON**

On utilise l'outil **mongoimport** : un exécutable à part entière, pas une commande du mongo shell

```
mongoimport --db dbName --collection collectionName
            --mode importMode --file fileName.json --jsonArray
```

Exemple :

```
mongoimport --db etablissements --collection statesr --mode upsert
            --file \arch-json\fr-esr-publications-statistiques.json
            --jsonArray
```

```
2017-05-11T09:33:22.530+0200  connected to: localhost
2017-05-11T09:33:22.694+0200  imported 420 documents
```

21

## Exportation

Exportation de documents au format **JSON**

On utilise l'outil **mongoexport** : un exécutable à part entière, pas une commande du mongo shell

```
mongoexport --db dbName --collection collectionName
            --out fileName.json --query theQuery
```

Sauve la collection **collectionName** de la base **dbName** dans le fichier **fileName.json**.

- Le démon **mongod** doit être lancé, et **mongoexport** s'y connectera
- Tout fichier cible préexistant est écrasé (pas de **merge** à l'écriture).
- On peut spécifier un filtrage sur les enregistrement avec l'option **--query**, et ne sauvegarder que ceux satisfaisant certains critères :
  - Ex : **mongoexport .... --query {titre : "Shrek"}**
  - Ne sauvera que les enregistrements dont le titre est "Shrek".

22

## Interrogation des données avec *find(...)*

23

## Queries avec *find()*

Commande "**db.collectionName.find(...){...}**"

- Le premier argument est la condition de sélection/filtrage :
  - Seuls les enregistrements satisfaisant cette condition seront retenus
- Le second argument (optionnel) décrit la projection :
  - L'information voulue de chaque enregistrement retenu après filtrage

Exemples :

```
db.comedie.find({date : 2000}) :
```

- Tous les films de la collection comedie sortie en 2000
- Equivalent à : **SELECT\* FROM comedie WHERE (date == 2000)**

```
db.comedie.find({date : 2000},{titre : 1, _id : 0}) :
```

- Tous les films de la collection comedie sortie en 2000
- Affiche le titre (uniquement) des films retenus
- Equivalent à : **SELECT titre FROM comedie WHERE (date == 2000)**

24

## Opérateurs des *queries* (1)

Liste des divers types d'opérateurs

<b>Comparaisons :</b>	<b>Op de test sur élément :</b>
\$eq, \$ne	\$exists Existence d'un champ ?
\$gt, \$gte, \$lt, \$lte	\$type Teste le type d'un champ
\$in, \$nin	
<b>Logiques :</b>	<b>Op d'évaluation :</b>
\$and, \$or	\$mod Calcule et teste le résultat d'un modulo
\$not, \$nor	\$regex Recherche d'expression régulière
	\$text Analyse d'un texte
	\$where Test de sélectionne des enregistrements
<b>Op de test sur tableau :</b>	
\$all	\$test Test sur le contenu d'un tableau
\$elemMatch	\$size Taille du tableau
\$size	

+ d'autres types d'opérateurs .... (voir doc MongoDB)

## Opérateurs des *queries* (2)

### Opérateur(s) AND

Trois façons de faire un AND :

Préciser plusieurs valeurs d'attributs dans le premier argument constitue un « AND »

```
db.comedie.find({pays : "France", date : 2000},{titre : 1, _id : 0})
```

→ Tous les films de la collection comedie tournés en France **et** sorties en 2000

→ Equivalent à :

```
SELECT titre FROM comedie WHERE (pays = "France" AND date == 2000)
```

**En fait, l'opérateur AND est implicite A CONDITION DE PORTER SUR DES ATTRIBUTS DIFFERENTS !!**

## Opérateurs des *queries* (2)

### Opérateur(s) AND

Trois façons de faire un AND :

Pour réaliser plusieurs tests numériques sur un même attribut il faut les regrouper :

```
db.comedie.find({pays : "France", date : {$gte : 2000, $lt : 2010}}, {titre : 1, _id : 0})
```

→ Tous les films de la collection comedie tournés en France **et** sorties entre : 2000 (inclus) **et** 2010 (exclus)

**Si les deux tests numériques sur le même attributs ne sont pas regroupés, alors MongoDB ne retiendra que le résultat du dernier**

## Opérateurs des *queries* (2)

### Opérateur(s) AND

Trois façons de faire un AND :

Utiliser un opérateur \$and explicite, mais il ne faut pas le mélanger avec une « , » :

```
db.comedie.find({$and : [{pays : "France"}, {date : {$gte : 2000}, {date : {$lt : 2010}}]}, {titre : 1, _id : 0})
```

→ Tous les films de la collection comedie tournés en France **et** sorties après 2000 (inclus) **et** sorties avant 2010 (exclus)

## Opérateurs des *queries* (3)

### Opérateur(s) OR

Deux solutions pour exprimer un 'OR' :

- Avec l'opérateur "\$in" :
 

```
db.comedie.find({date : {$in [2000, 2002, 2004]}},{titre : 1, _id : 0})
```

 → Equivalent à :
 

```
SELECT titre FROM comedie WHERE date in (2000, 2002, 2004)
```
- Avec l'opérateur "\$or" :
 

```
db.comedie.find({date : 2000, $or : [{budget : {$lt : 100000}}, {nbEntrees : {$lt : 10000}}]}, {titre : 1, _id : 0})
```

 → Equivalent à :
 

```
SELECT titre FROM comedie WHERE date = 2000 AND (budget < 100000 OR nbEntrees < 10000)
```

## Opérateurs des *queries* (4)

### Opérateur de test sur un élément

**\$exist** { fieldName : { \$exists : boolean } }

```
{nbEntrees : {$exists : true, $lt 10000}}
```

→ Si le champ nbEntrees existe, alors teste s'il est < 10000

```
{nbEntrees : {$exists : false}}
```

→ Si le champ nbEntrees n'existe pas alors retourne true

**\$type** { fieldName : { \$type : BSON type number | String alias } }

```
{nbEntrees : {$type : 16}}
```

Rmq : type BSON numéro 16 (!) ↔ "int"

```
{nbEntrees : {$type : "int"}}
```

→ Si le champ nbEntrees est de type int, alors retourne true

## Opérateurs des *queries* (5)

### Opérateurs d'évaluation

**\$mod** { *fieldName* : { **\$mod** : [ *divisor*, *remainder* ] } }

**{ nbEtudiants : { \$mod : [3, 0] } }**

→ Si le nombre d'étudiants est un multiple de 3, retourne *true*  
(ex : pour savoir si on peut constituer des trinômes...)

**\$regex** { *fieldName* : { **\$regex** : *regExp*, **\$options** : *options* } }

**{ regionName : { \$regex : /^lor/, \$options : 'i' } }**

**/^lor/** : commence par "lor"

**\$options : 'i'** : insensible à la casse

→ si le nom de la région commence par *lor*, ou *Lor*, ou *loR*, ou *LOR* ... alors retourne *true*

**{ regionName : { \$regex : /lor/, \$options : 'i' } }**

→ si le nom de la région contient *lor*, ou *Lor*, ou *loR*, ou *LOR* ... alors retourne *true*

**Il y a beaucoup de façons d'écrire des expressions régulières ... voir la doc!**

## Opérateurs des *queries* (6)

### Opérateurs d'évaluation

**\$text**

- Créer un index "text" sur le champs à analyser  
`db.collection.createIndex("a.b.c.d" : "text")`
- Lancer un filtrage sur le contenu de ce champ (et sur d'autres)  
`db.collection.find({"a.e.f" : "haha", $text : {$search : "houba"}}`  
→ a.e.f doit valoir "haha" ET "a.b.c.d" doit contenir "houba"

- { \$text : { \$search : <string>, \$language : <string>, \$caseSensitive : <boolean>, \$diacriticSensitive : <boolean> } }**

Rmq : non diacritique → *Etudiant*, *étudiant* et *etudiant* seront acceptés

Rmq : spécifier le langage permet de faciliter les recherches.

**MongoDB contient des facilités pour analyser du texte sur le web**

## Opérateurs des *queries* (7)

### Opérateurs d'évaluation

**\$where** { *fieldName* : { **\$where** : *JavaScript expression* } }

Quand les opérateurs natifs de MongoDB ne suffisent plus, on peut introduire du code JavaScript (mais c'est plus lent).

Deux syntaxes possibles :

```
db.collectionName.find({ active: true,
  $where: function() {
    return obj.credits - obj.debits < 0;
  }
})
db.collectionName.find({ active: true,
  $where: "this.credits - this.debits < 0"
})
```

## Opérateurs des *queries* (8)

### Opérateurs de test sur tableau

**\$all** { *arrayFieldName* : { **\$all** : [ *val1*, *val2*, *val3*... ] } }

Renverra *true* (retiendra l'enregistrement) si le champ *arrayFieldName* contient toutes les valeurs listées ensuite (*val1*, *val2* et *val3*).

**Ex :**

- Si le champs *cours* vaut ["info", "elec", "auto", "anglais"]
- Alors **{ \$all : ["info", "anglais"] }** renverra *true*

## Opérateurs des *queries* (9)

### Opérateurs de test sur tableau

**\$elemMatch** { *arrayFieldName* : { **\$elemMatch** : { *query1*, *query2*... } } }

Renverra *true* (retiendra l'enregistrement) si le champ *arrayFieldName* contient au moins un élément satisfaisant toutes les requêtes.

**Ex :** « on cherche s'il y a au moins une note dans l'intervalle [7, 10] »

- Si le champs *notes* vaut [18, 8, 17, 11]
- Alors **{ notes : { \$elemMatch : { \$gte : 7, \$lt : 10 } } }** renverra *true*

**\$size** { *arrayFieldName* : { **\$size** : *theSize* } }

Renverra *true* (retiendra l'enregistrement) si le champ *arrayFieldName* a la taille indiquée.

**Mais ... comment tester si un tableau a une taille > n ?? ...**

**Et encore bien d'autres opérateurs ....**

## Application de méthodes sur une collection

CentraleSupélec

## Application de méthodes aux collections

**Syntaxe générale :**

```
db.laCollection.find(...).methode(...)
```

- `sort(...)`
- `forEach(...)`
- ...

Parfois aussi :

```
db.laCollection.methode(...)
```

- `count(...)`
- ...

**Méthode `sort` :**

Ordonne les enregistrements d'une collection selon un ou plusieurs champs, dans l'ordre croissant ou décroissant

```
db.laCollection.find(...).sort({"a.b.c.d" : 1, "a.b.x.y" : -1})
```

→ trie selon le champ a.b.c.d dans l'ordre croissant, et pour un même champ a.b.c.d, trie selon a.b.x.y dans l'ordre décroissant

Rmq : `db.laCollection.sort({"a.b.c.d" : 1, "a.b.x.y" : -1})`

37

CentraleSupélec

## Application de méthodes aux collections

**Syntaxe générale :**

```
db.laCollection.find(...).methode(...)
```

- `sort(...)`
- `forEach(...)`
- ...

Parfois aussi :

```
db.laCollection.methode(...)
```

- `count(...)`
- ...

**Méthode `count` :**

Compte les enregistrements d'une collection

```
db.laCollection.find(...).count()
```

Ou bien :

```
db.laCollection.count()
```

→ Retourne le nombre d'enregistrements (de documents JSON) dans la collection

38

CentraleSupélec

## Application de méthodes aux collections

**Méthode `forEach( définition d'une fonction )` :**

Définit et applique une fonction à chaque enregistrement (ou document JSON) d'une collection


Syntaxe : `db.laCollection.find(...).forEach(function(doc) {...})`

- La fonction appliquée peut être sans action sur la collection

Ex : `db.laCollection.find().forEach(function(doc) {print(doc.a.b.c.d);})`

- Ou au contraire agir sur la collection (ex : la modifier).

Ex : `db.laCollection.find().forEach(function(doc) { db.laCollection.update({_id : doc._id}, { nouveau document : valeur }); })`

→ **Ecrase tout l'enregistrement ciblé** 

39

CentraleSupélec

## Application de méthodes aux collections

**Méthode `forEach( définition d'une fonction )` :**

Définit et applique une fonction à chaque enregistrement (ou document JSON) d'une collection

Syntaxe : `db.laCollection.find(...).forEach(function(doc) {...})`

- La fonction appliquée peut être sans action sur la collection

Ex : `db.laCollection.find().forEach(function(doc) {print(doc.a.b.c.d);})`

- Ou au contraire agir sur la collection (ex : la modifier).

Ex : `db.laCollection.find().forEach(function(doc) { db.laCollection.update({_id : doc._id}, { $set : {document : valeur} }); })`

→ **Modifie un enregistrement (ne l'écrase pas complètement)**

40

CentraleSupélec

## Agrégations d'opérations avec `aggregate(...)`

41

CentraleSupélec

## Framework d'agrégation (1)

**Principe :**

MongoDB propose une autre façon de coder des traitements : en formant un **pipeline d'opérations**

```
db.collectionName.aggregate({op1}, {op2}, {op3}...)
```

La **sortie d'une opération est l'entrée de la suivante**, ou bien la sortie finale de l'agrégation

**On peut pipeliner successivement autant d'opérations que l'on veut**

**Plus lent que des commandes natives, car les données pipelinées ne profitent pas des index !**

**Opérations pour l'agrégation :**

- `$match`
- `$project`
- `$group`
- `$unwind`
- `$sort`
- `$limit`
- `$lookup`

42

### Framework d'agrégation (2)

On travaille sur la Bdd des publications statistiques de l'enseignement supérieur :

```
mongoimport --db etablissements --collection statesr --jsonArray
--mode upsert
--file \arch-json\fr-esr-publications-statistiques.json
```

... On visualise UN enregistrement pour voir ce que contient la base :

```
> db.statesr.findOne()
{
  "_id" : ObjectId("5911b9e9df7184e1c0778a8a"),
  "fields" : {
    "contributeurs" : "Isabelle Kabla-Langlois, Florian Lezec",
    "publication_date_annee" : "2017"
  },
  "record_timestamp" : "2017-04-14T16:04:46+02:00"
}
```

Champs intéressants

43

### Framework d'agrégation (3)

**\$match :**

Permet de sélectionner/filtrer les enregistrements d'entrée

Ex : On ne s'intéresse qu'aux études statistiques de moins de 10 ans

```
db.statesr.aggregate(
  { "$match" : { "fields.publication_date_annee" : { "$gt" : "2007" } } }
)
```

44

### Framework d'agrégation (4)

**\$project :**

Permet de projeter des attributs : seulement ceux voulus, et au besoin en les renommant ou en créant de nouveaux attributs

Ex : On ne retient que les contributeurs à l'étude et l'année de la publication

```
db.statesr.aggregate(
  { "$match" : { "fields.publication_date_annee" : { "$gt" : "2007" } } },
  { "$project" : { "fields.contributeurs" : 1,
    "fields.publication_date_annee" : 1 } }
)
```

45

### Framework d'agrégation (5)

**\$group :**

Permet de regrouper les enregistrements retenus selon l'\_id (que l'on peut redéfinir au passage), et d'appliquer des fonctions de groupe au autres attributs projetés.

Ex : les contributeurs deviennent l'\_id, et on calcule la date de début de leur collaboration, et leur nombre de publications

```
db.statesr.aggregate(
  { "$match" : { "fields.publication_date_annee" : { "$gt" : "2007" } } },
  { "$project" : { "fields.contributeurs" : 1,
    "fields.publication_date_annee" : 1 } },
  { "$group" : { "_id" : "$fields.contributeurs",
    "debut" : { "$min" : "$fields.publication_date_annee" },
    "occurrences" : { "$sum" : 1 } } }
)
```

46

### Framework d'agrégation (6)

**\$sort :**

Permet d'ordonner les enregistrements selon un des attributs, par ordre croissant (+1) ou décroissant (-1).

Ex : on ordonne les équipes de contributeurs par ordre décroissant de publications produites (ces 10 dernières années)

```
db.statesr.aggregate(
  { "$match" : { "fields.publication_date_annee" : { "$gt" : "2007" } } },
  { "$project" : { "fields.contributeurs" : 1,
    "fields.publication_date_annee" : 1 } },
  { "$group" : { "_id" : "$fields.contributeurs",
    "debut" : { "$min" : "$fields.publication_date_annee" },
    "occurrences" : { "$sum" : 1 } } },
  { "$sort" : { "occurrences" : -1 } }
)
```

47

### Framework d'agrégation (7)

**\$limit :**

Permet de limiter le nombre de réponses aux plus importantes

Ex : on ne s'intéresse qu'aux 5 équipes de contributeurs les plus productives ces 10 dernières années

```
db.statesr.aggregate(
  { "$match" : { "fields.publication_date_annee" : { "$gt" : "2007" } } },
  { "$project" : { "fields.contributeurs" : 1,
    "fields.publication_date_annee" : 1 } },
  { "$group" : { "_id" : "$fields.contributeurs",
    "debut" : { "$min" : "$fields.publication_date_annee" },
    "occurrences" : { "$sum" : 1 } } },
  { "$sort" : { "occurrences" : -1 } },
  { "$limit" : 5 }
),pretty()
```

48



### Framework d'agrégation (8)

Résultats :

```
{ "_id" : "Joëlle Grille",
  "debut" : "2008",
  "occurrences" : 7 }
{ "_id" : "Isabelle Kabla-Langlois, Louis-Alexandre Erb",
  "debut" : "2015",
  "occurrences" : 6 }
{ "_id" : "Isabelle Kabla-Langlois, Mathias Denjean",
  "debut" : "2016",
  "occurrences" : 5 }
{ "_id" : "Annie Le Roux",
  "debut" : "2009",
  "occurrences" : 5 }
{ "_id" : "Isabelle Kabla-Langlois, Claudette-Vincent Nisslé, Laurent Perrain",
  "debut" : "2015",
  "occurrences" : 5 }
```

### Framework d'agrégation (9)

**\$unwind :**

Permet de remplacer un enregistrement contenant un tableau par une suite d'enregistrements contenant chacun un seul élément du tableau

Ex : `{ "_id" : 1, "item" : "ABC", "sizes" : [ "S", "M", "L" ] }`

```
db.theCollection.aggregate( [ { $unwind : "$sizes" } ] )
```

ou bien:

```
db.theCollection.aggregate( [ { $unwind : { path : "$sizes" } } ] )
```

→ `{ "_id" : 1, "item" : "ABC", "sizes" : "S" }`  
`{ "_id" : 1, "item" : "ABC", "sizes" : "M" }`  
`{ "_id" : 1, "item" : "ABC", "sizes" : "L" }`

Rmq : en cas de tableau vide [], ou de champ null, ou inexistant, l'enregistrement initial disparaît de la sortie du \$unwind (!)

### Framework d'agrégation (10)

**\$unwind :**

Permet de remplacer un enregistrement contenant un tableau par une suite d'enregistrements contenant chacun un seul élément du tableau

Si on veut conserver les enregistrements des tableaux vides : on utilise une spécification de \$unwind :

Ex : `{ "_id" : 1, "item" : "EFG", "sizes" : [ ] }`

```
db.theCollection.aggregate( [
  { $unwind : { path : "$sizes",
              preserveNullAndEmptyArrays : true } }
] )
```

→ `{ "_id" : 2, "item" : "EFG" }`

### Framework d'agrégation (11)

**\$lookup :**

\$lookup est LA SOLUTION pour réaliser des « JOIN » sur plusieurs collections...

→ voir plus loin la réalisation des « JOIN »

### Bilan de l'agrégation (8)

On travaille sur une seule collection à la fois :

- `db.collectionName.find(...)` : est ce qui ressemble à un SELECT
- `db.collectionName.aggregate(...)` : permet de pipeliner des opérations et de créer de "petits programmes"

Mais jusqu'ici on n'applique ces opérations qu'à une seule collection à la fois : "à une seule table".



→ Il n'y a pas de jointure !!

On considère que la jointure a été faite à l'écriture/à la conception de la collection et de ses documents internes

→ La collection traitée est supposée autoporteuse pour la requête soumise.

### Bilan de l'agrégation (9)

Limite d'efficacité de l'agrégation (de MongoDB) :

- Les opérations du pipeline de l'agrégation ne prennent pas leur données directement dans les collections...  
... elles ne peuvent pas profiter des index ! 
- Les agrégations restent lentes
- Essayer de filtrer les données au maximum en entrée (un match très sélectif améliorera les perf)
- MongoDB refuse une agrégation s'il estime qu'elle nécessitera plus de 20% de la mémoire disponible 

## Map-Reduce mapReduce en MongoDB

55

## mapReduce de MongoDB (1)

**MongoDB possède son propre « Map-Reduce » :**

- Ses propres règles de fonctionnement (un peu différentes d'Hadoop)
- Ses propres règles de déploiement des tâches
- Et son propre middleware sous-jacent (il n'est pas bâti au dessus d'Hadoop)
- Fonctionne sur des bases distribuées (*sharded*)
- N'exploite pas les index !

**Principes du *mapReduce* de MongoDB :**

- Une *query* pour pré-filtrer la collection traitée
- Une fonction *map()*, en Java Script et qui accède à la base
- Une fonction *reduce()*, en Java Script et qui ne doit PAS accéder à la base qui doit être *commutative, associative et idempotente* (!!)
- Une fonction *finalize()*, en Java Script et qui ne doit pas accéder à la base
- La possibilité de définir un ensemble de variables globales aux 3 fonctions *map()*, *reduce()* et *finalize()*

56

## mapReduce de MongoDB (2)

**Principes du *mapReduce* de MongoDB :**

input data → query() → map() → Shuffle & Sort → reduce() → finalize() → output data

Identical < key, val > scheme

- *map()* fonctionne comme en Hadoop, mais sur une seule collection
- pour éviter d'implanter un filtrage en JS, une *query* exprimée en Mongo shell est applicable en amont (plus pratique et plus rapide)
- *reduce()* est plus contraint qu'en Hadoop car MongoDB applique *reduce()* sur des < key, [val] > de sortie de *map()*, et sur des sorties précédentes de *reduce()* :
  - le format de sortie de *reduce()* doit être celui de sortie de *map()*
  - La fonction *reduce()* doit être commutative, associative et idempotente
- *finalize()* permet de modifier la sortie finale de *reduce()* sans contrainte

57

## mapReduce de MongoDB (3)

**Principes du *mapReduce* de MongoDB :**

input data → query() → map() → Shuffle & Sort → reduce() → finalize() → output data

Identical < key, val > scheme

- Pas de *Combiner*
- Pas d'optimisation possible du *Shuffle & Sort*

**Les contraintes sur la fonction *reduce()* :**

- évite les débordements mémoire, en traitant chaque paire < clé, liste de val > en plusieurs passes si la liste de valeurs est « grande »
- limitent fortement les algorithmes possibles en une seule opération *mapReduce*.

58

## mapReduce de MongoDB (4)

**Exemple de mapReduce en MongoDB (compteur de livres) :**

```
Code JavaScript entré dans un mongo shell
```

```
map1 = function() {
  var k = this.book.title;
  var v = {"count" : 1};
  emit(k,v);
}

reduce1 = function(key,values) {
  var s = 0;
  for (var i in values)
    s += values[i].count;
  return({"count" : s});
}

finalize1 = function(k,v) {
  return({"title" : k, "nb" : v.count});
}
```

Deux syntaxes possibles d'exécution d'un mapReduce dans un mongo shell :

- 1 > res = db.runCommand({"mapreduce" : "theCollection", "map" : map1, "reduce" : reduce1, "out" : "res1", "finalize" : finalize1})
- 2 > db.theCollection.mapReduce(map1, reduce1, {"out" : "res1", "finalize" : finalize1})

59

## mapReduce de MongoDB (4)

**Exemple de mapReduce en MongoDB (compteur de livres) :**

```
Code JavaScript entré dans un mongo shell
```

```
Allure d'un Map :
function() {
  ...
  emit (this.a, val);
}
→ <key, val >
```

```
Allure d'un Reduce :
function(key, ListVals) {
  ...
  return ( val );
}
→ < key, val >
```

```
Allure d'un finalize :
function (k,v) {
  ...
  return ({...});
}
→ Sortie JSON : { "id" : k ..., "value" : {...} }
```

Deux syntaxes possibles :

- 1 > res = db.runCommand({"mapreduce" : "theCollection", "map" : map1, "reduce" : reduce1, "out" : "res1", "finalize" : finalize1})
- 2 > db.theCollection.mapReduce(map1, reduce1, {"out" : "res1", "finalize" : finalize1})

60

### mapReduce de MongoDB (5)

Exemple de mapReduce en MongoDB :

SANS finalize() :

```
> b.res1.find()
{ "_id": "titre1", "value": { "count": 10 } }
{ "_id": "titre2", "value": { "count": 2 } }
....
```

AVEC finalize() : on peut modifier le document "value"

```
> b.res1.find()
{ "_id": "titre1", "value": { "title": "titre1", "nb": 10 } }
{ "_id": "titre2", "value": { "title": "titre2", "nb": 2 } }
....
```

La sortie d'un mapReduce est toujours un document JSON de format :

```
{ "_id" : xxxx, "value" : yyyy }
....
```

61

### mapReduce de MongoDB (6)

Ajout d'une query sur les données d'entrée :

```
1 > res = db.runCommand(
  {"mapreduce": "theCollection",
   "map": map1,
   "reduce": reduce1,
   "out": "res1",
   "finalize": finalize1,
   "query": {"booktitle": "Cours de Big Data"}
)

2 > db.theCollection.mapReduce(
  map1,
  reduce1,
  {"out": "res1", "finalize": finalize,
   "query": {"booktitle": "Cours de Big Data"}
)

```

62

## JOIN en MongoDB

63

### JOIN en MongoDB (1)

Utilisation de l'opérateur \$lookup :

Toutes les opérations précédentes se focalisaient sur 1 collection

Pour faire un JOIN entre deux collections il faut utiliser :

- Une agrégation
- Avec l'opérateur \$lookup

```
db.co1.aggregate([
  {"$lookup": {
    "localField": "title",
    "from": "co2",
    "foreignField": "title_book",
    "as": "joined"
  }},
  {"$out": "resJoin"}
]);
```

- "title" : champ de la collection co1
- "title\_book" : champ de la collection co2

Dans le pipeline, on génère un document égal aux enregistrements de co1 enrichis d'un champ "joined" contenant un tableau de tous les enregistrements de co2 de même clé

- Le document généré est sauvé dans la collection "resJoin"

64

### JOIN en MongoDB (1)

Utilisation de l'opérateur \$lookup :

```
db.co1.aggregate([
  {"$lookup": {
    "localField": "title",
    "from": "co2",
    "foreignField": "title_book",
    "as": "joined"
  }},
  {"$out": "resJoin"}
]);
```

Collection resJoin

```
{ "_id": ...,
  "title": "Cours de Big Data",
  "editeur": "CentraleSupelec",
  "annee": "2016-2017",
  "joined": [
    { "_id": ...,
      "title_book": "Cours de Big Data",
      "thematique": "informatique",
      "prerequis": [ "programmation", "SQL" ]
    },
    ...
  ]
},
...
]
```

Une transformation du document obtenu est nécessaire pour une exploitation confortable !

→ Insérer des opérateurs \$unwind et \$project dans l'agrégation

65

## Gestion des index en MongoDB

66

**Gestion des index**

Les index permettent d'accélérer les requêtes, mais ne sont pas exploités par les pipelines des agrégations ni par les mapReduce

L'utilisateur peut créer des index explicitement sur une collection et selon un champ :

- Pour accélérer les requêtes sur cette collection et sur ce champ
- Pour rendre possible des requêtes demandant beaucoup de RAM  
→ très efficace sur les requêtes de tri

```
> res = db.inscription.find().sort({"fields.diplome_lib" : -1})
Error: ... pas assez de RAM !
> db.inscription.createIndex({"fields.diplome_lib" : -1})
Ok !
> res = db.inscription.find().sort({"fields.diplome_lib" : -1})
Ok !
```

67

**Comparaison des 3 mécanismes**

68

**Comparaison des 3 mécanismes**

The diagram compares three query mechanisms based on two criteria:

- Vitesse d'exécution (Execution Speed):** Indicated by a yellow arrow pointing upwards. Native language queries (Requêtes en langage natif) are marked with a plus sign (+), indicating high speed. Aggregations of operators (Agrégations d'opérateurs) and mapReduce are marked with minus signs (-), indicating lower speed.
- Souplesse de programmation (Programming Flexibility):** Indicated by a blue arrow pointing downwards. Native language queries and Aggregations of operators are marked with minus signs (-), indicating low flexibility. mapReduce is marked with a plus sign (+), indicating high flexibility.

69

**Utilisation de MongoDB**

70