Big Data

MongoDB : syntaxe et exemples

Stéphane Vialle & **Gianluca Quercini**

1

MongoDB : syntaxe et exemples

- **Importation et exportation de documents**
- Gestion des bases et des collections
- Interrogation des données avec *find(…)*
- Application de *methods* aux collections
- Agrégation d'opérations avec *aggregate(…)*
- *JOIN* en MongoDB
- *Map-Reduce* en MongoDB
- Gestion des Index en MongoDB
- Comparaison des 3 mécanismes d'interrogation
- Annexe : Modifications d'enregistrements
- Annexe : Ecriture de fonctions Java Script

2

2

Lancement d'un client *MongoDB*

Dans un premier *command shell*

```
C:\Users\vialle> mkdir /…/mydata

C:\Users\vialle> mongod --dbpath /…/mydata
                        --port 10000
```

Crée un répertoire de stockage.

Lance le serveur MongoDB sur ce répertoire et sur le port 10000

Dans un *command shell*

Lancer un client **mongoimport**

Ou

Lancer un client **mongoexport**

Ou

Lancer un client **mongosh (le mongo shell)**

3

3

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 --port *port* --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)

4

4

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 --port *port* --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, et remplace ceux déjà

5

5

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 --port *port* --db *dbName* --collection *collectionName*
--mode *importMode* --file *fileName.json* --jsonArray

Exemple :

```
mongoimport --port 10000 --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
```

6

6

Exportation

Exportation de documents au format JSON

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

```
mongoexport  --port port  --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 enregistrements 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".

7

7

MongoDB : syntaxe et exemples

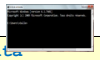
- Importation et exportation de documents
- **Gestion des bases et des collections**
- Interrogation des données avec *find(…)*
- Application de *methods* aux collections
- Agrégation d'opérations avec *aggregate(…)*
- *JOIN* en MongoDB
- *Map-Reduce* en MongoDB
- Gestion des Index en MongoDB
- Comparaison des 3 mécanismes d'interrogation
- Annexe : Modifications d'enregistrements
- Annexe : Ecriture de fonctions Java Script

8

8

Lancement du client *mongo*

Dans un premier *command shell*



```
C:\Users\vialle> mkdir /…/mydata

C:\Users\vialle> mongod --dbpath /…/mydata
                        --port 10000
```

Crée un répertoire de stockage.

Lance le serveur MongoDB sur ce répertoire et sur le port 10000

Dans un second *command shell*

```
C:\Users\vialle> mongosh --port 10000
MongoDB shell version v3.4.1
connecting to: mongodb://127.0.0.1:27017
MongoDB server version: 3.4.1
…
>
```

Lance un client **mongosh** (***mongo shell***)

9

9

Gestion des bases et des collections (1)

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
config  …..
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)

10

10

Gestion des bases et des collections (2)

Effacement complet d'une collection

```
…
> show collections
comedie

> db.comedie.drop()
true
> show collections
```

On crée une collection « comedie » (voir plus loin)

Elle est bien listée comme collection existante

On efface cette collection

11

11

Gestion des bases et des collections (3)

Effacement d'une base

```
> show dbs
admin    0.000GB
config   …..
db-films 0.000GB
local    0.000GB
```
Base courante : db-film

```
> db
db-films
```

```
> db.dropDatabase()
{ "dropped" : "db-films", "ok" : 1 }
```
On efface la base courante

```
> show dbs
admin  0.000GB
config   …..
local  0.000GB
```
Elle a bien disparu des enregistrements sur disque

```
> db
db-films
```
Mais on continue à la référencer (ne pas oublier de passer à une autre base)

12

12

Gestion des bases et des collections (4)

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

13

13

MongoDB : syntaxe et exemples

- Importation et exportation de documents
- Gestion des bases et des collections
- **Interrogation des données avec *find(…)***
- Application de *methods* aux collections
- Agrégation d'opérations avec *aggregate(…)*
- *JOIN* en MongoDB
- *Map-Reduce* en MongoDB
- Gestion des Index en MongoDB
- Comparaison des 3 mécanismes d'interrogation
- Annexe : Modifications d'enregistrements
- Annexe : Ecriture de fonctions Java Script

14

14

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)

15

15

Queries avec *find()*

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

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

findOne() : affiche « joliment » un(seul) enregistrement au hasard

16

16

Queries avec *find()*

Accès à des attributs composites/structurés

```
{entreprise : "Au bon produit",
 services : { direction : {etage : 5, effectifs : 6},
              comptabilite : {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

17

Opérateurs des *queries* (1)

Liste des divers types d'opérateurs

Comparaisons :

$eq, $ne	==, !=
$gt, $gte, $lt,$lte	>, >=, <, <=
$in, $nin	∈, ∉

Logiques :

$and, $or	AND, OR
$not, $nor	NOT, NOR

Op de test sur tableau :

$all	Test sur le contenu d'un tableau
$elemMatch	
$size	Taille du tableau

Op de test sur élément :

$exists	Existence d'un champ ?
$type	Teste le type d'un champ

Op d'évaluation :

$mod	Calcule et teste le résultat d'un modulo
$regex	Recherche d'expression régulière
$text	Analyse d'un texte
$where	Test de sélectionne des enregistrements

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

18

18

Opérateurs des *queries* (2.1)

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 !!

19

19

Opérateurs des *queries* (2.2)

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** sortis entre : 2000 (inclus) **et** 2010 (exclus)

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

20

20

Opérateurs des *queries* (2.3)

Opérateur(s) AND

Trois façons de faire un AND :

Utiliser un opérateur $and explicite :

```
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** sortis après 2000 (inclus) **et** sortis avant 2010 (exclus)

21

21

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)
```

22

22

Opérateurs des *queries* (4)

Opérateur de test sur un élément

$exists** { *fieldName* : {**$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*

23

23

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 !

24

24

Opérateurs des *queries* (6)

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 **{cours : { $all : ["info", "anglais"] } }** renverra *true*

25

25

Opérateurs des *queries* (7)

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 champ *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 *theSize*.

Rmq : dans un ***aggregate*** (voir + loin) : **…{$size : "$notes"}…** permettra de récupérer la taille du tableau dans une opération de projection…

Et encore bien d'autres opérateurs ….

26

26

Opérateurs des *queries* (8)

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

27

27

MongoDB : syntaxe et exemples

- Importation et exportation de documents
- Gestion des bases et des collections
- Interrogation des données avec *find(…)*
- **Application de *methods* aux collections**
- Agrégation d'opérations avec *aggregate(…)*
- *JOIN* en MongoDB
- *Map-Reduce* en MongoDB
- Gestion des Index en MongoDB
- Comparaison des 3 mécanismes d'interrogation
- Annexe : Modifications d'enregistrements
- Annexe : Ecriture de fonctions Java Script

28

28

Application de méthodes aux collections

Syntaxe générale :
db.*laCollection*.find(…).methode(…)

Parfois aussi :
db.*laCollection*.methode(…)

- **sort(…)**
- **forEach(…)**
- **…**
- **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})~~

29

29

Application de méthodes aux collections

Syntaxe générale :
db.*laCollection*.find(…).methode(…)

Parfois aussi :
db.*laCollection*.methode(…)

- **sort(…)**
- **forEach(…)**
- **…**
- **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

30

30

Application de méthodes aux collections

Méthode forEach(*définition d'une fonction (en java script)*) :

Définit et applique une fonction à chaque enregistrement d'une collection

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

Ex :
```
db.laCollection.find().forEach(function(doc) {
                                  if (doc.x.y > 0)
                                    print(doc.a.b.c.d);
                                })
```

En cas de tests et actions complexes ce peut être plus simple que par les queries.

31

31

Application de méthodes aux collections

Méthode forEach(*définition d'une fonction (en java script)*) :

Définit et applique une fonction à chaque enregistrement d'une collection

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

Ex :
```
db.laCollection.find().forEach(function(doc) {
                                  var n = doc.x.y;
                                  if (n < 0)
                                    db.laCollection.updateOne(
                                        {_id : doc._id},
                                        { $set : {"x.y" : 0} }) ;
                                })
```

On peut même réaliser un update de la collection depuis la fonction en java script

32

32

MongoDB : syntaxe et exemples

- Importation et exportation de documents
- Gestion des bases et des collections
- Interrogation des données avec *find(…)*
- Application de *methods* aux collections
- **Agrégation d'opérations avec *aggregate(…)***
- *JOIN* en MongoDB
- *Map-Reduce* en MongoDB
- Gestion des Index en MongoDB
- Comparaison des 3 mécanismes d'interrogation
- Annexe : Modifications d'enregistrements
- Annexe : Ecriture de fonctions Java Script

33

33

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**
- **$lookup**
- **$out**

34

34

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
```

… TOUT D'ABORD 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

35

35

Framework d'agrégation (3)

$match :

Permet de sélectionner/filtrer les enregistrements d'entrée

Ex : On ne s'intéresse qu'aux études statistiques après 2007

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

36

36

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.pub_an" : "}}
])
```

37

37

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"}}},
    {"$project" : {"fields.contributeurs" : 1,
                   "fields.pub_an" : "$fields.publication_date_annee"}},
    {"$group"  : {_id : "$fields.contributeurs",
                  "debut" : {$min : "$fields.pub_an"},
                  "occurences" : {$sum : 1}}}
])
```

38

38

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 (après 2007)

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

39

39

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 après 2007

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

40

40

Framework d'agrégation (8)

$out :

Permet de spécifier une collection de sortie où stocker les résultats

Ex : on ne s'intéresse qu'aux 5 équipes de contributeurs les plus productives après 2007

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

41

41

Framework d'agrégation (9)

Résultats :

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

42

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

Ex : **{ "_id" : 10, "item" : "ABC", "sizes": ["S", "M", "L"] }**

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

→ **{ "_id" : 10, "item" : "ABC", "sizes" : "S" }**
{ "_id" : 10, "item" : "ABC", "sizes" : "M" }
{ "_id" : 10, "item" : "ABC", "sizes" : "L" }

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

43

43

Framework d'agrégation (11)

$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" : 10, "item" : "EFG", "sizes": [ ] }**

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

→ **{ "_id" : 10, "item" : "EFG" }**

44

44

Framework d'agrégation (12)

$unwind :

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

Ex : **{ "_id" : 10, "item" : "ABC", "sizes": ["S", "M", "L"] }**

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

→ **{ "_id" : 10, "item" : "ABC", "sizes" : "S" }**
{ "_id" : 10, "item" : "ABC", "sizes" : "M" }
{ "_id" : 10, "item" : "ABC", "sizes" : "L" }

Crée 3 documents avec le même _id !!

→ Toute création d'une collection pour conserver les résultats sera impossible (avec {$out : "xxxxx"}) !!

45

45

Framework d'agrégation (13)

$unwind :

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

Ex : **{ "_id" : 10, "item" : "ABC", "sizes": ["S", "M", "L"] }**

db.*theCollection*.aggregate([{ $project: {"_id" : 0}},
{ $unwind: "$sizes" }])

→ **{"item" : "ABC", "sizes" : "S" }**
{"item" : "ABC", "sizes" : "M" }
{"item" : "ABC", "sizes" : "L" }

On a supprimé le champ « _id » (!), ce qui ne pose pas de pb au pileline de l'aggregate.

→ Un _id sera alors recréé automatiquement pour chaque document lors de l'écriture dans une nouvelle collection (avec {$out : "xxxxx"})

46

46

Framework d'agrégation (14)

$lookup:

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

→ voir plus loin la réalisation des « JOIN »

47

47

Bilan de l'agrégation (1)

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

Si 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 requêtes soumise
→ On applique des opérations à une seule collection à la fois (''à une seule table'')

Sinon : on doit faire un join explicite avec un **$lookup**

48

48

Bilan de l'agrégation (2)

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 perfs)
- MongoDB refuse une agrégation s'il estime qu'elle nécessitera plus de 20% de la mémoire disponible


49

49

MongoDB : syntaxe et exemples

- Importation et exportation de documents
- Gestion des bases et des collections
- Interrogation des données avec *find(…)*
- Application de *methods* aux collections
- Agrégation d'opérations avec *aggregate(…)*
- ***JOIN* en MongoDB**
- *Map-Reduce* en MongoDB
- Gestion des Index en MongoDB
- Comparaison des 3 mécanismes d'interrogation
- Annexe : Modifications d'enregistrements
- Annexe : Ecriture de fonctions Java Script

50

50

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"}
]);
```

- Autre collection de la même base

- "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"

51

51

JOIN en MongoDB (1)

Utilisation de l'opérateur $lookup :

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

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

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

Collection resJoin

```
{
{ "_id" : …..,
  "title" : "Cours de Big Data",
  "editeur" : "CentraleSupelec",
  "annee" : "2016-2917",
  "joined" : [
    { "_id" : ….
      "title_book" : "Cours de Big Data",
      "thematique" : "informatique"
      "prerequis" : [ "programmation",
                      "SQL" ]
    },
    …
  ]
}
…
}
```

52

52

MongoDB : syntaxe et exemples

- Importation et exportation de documents
- Gestion des bases et des collections
- Interrogation des données avec *find(…)*
- Application de *methods* aux collections
- Agrégation d'opérations avec *aggregate(…)*
- *JOIN* en MongoDB
- ***Map-Reduce* en MongoDB**
- Gestion des Index en MongoDB
- Comparaison des 3 mécanismes d'interrogation
- Annexe : Modifications d'enregistrements
- Annexe : Ecriture de fonctions Java Script

53

53

mapReduce de MongoDB (1)

MongoDB possède son propre « Map-Reduce » :

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

Principes du *mapReduce* de MongoDB :

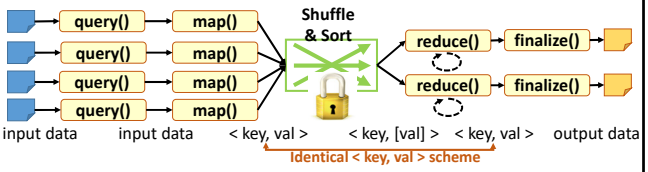
- Une **query** pour pré-filtrer la collection traitée
- Une fonction **map()**, en Java Script et qui accède à la base
- Une fonction **reduce()**, en Java Script et qui ne doit PAS accéder à la base qui doit être commutative, associative et idempotente (!!)
- Une fonction **finalize()**, en Java Script 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()

54

54

mapReduce de MongoDB (2)

Principes du *mapReduce* de MongoDB :



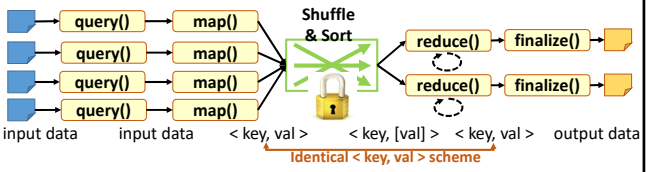
- ***map()*** fonctionne classiquement **sur une collection**
- pour éviter d'implanter un filtrage en JS, une ***query*** exprimée en Mongo shell est applicable en amont (plus pratique et plus rapide)
- ***reduce()*** est **plus contraint qu'en Hadoop ou Spark** car MongoDB applique *reduce()* sur des < 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 indempotente***
- ***finalize()*** permet de modifier la sortie finale de *reduce()* sans contrainte

55

55

mapReduce de MongoDB (3)

Principes du *mapReduce* de MongoDB :



- **Pas** de ***Combiner*** (contrairement à Hadoop)
- **Pas d'optimisation** possible du ***Shuffle & Sort*** (contrairement à Hadoop)

Les contraintes sur la fonction *reduce()* :

- **évitent 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** à ceux travaillant avec une liste partielle des données de même clé

56

56

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, ListVals) {
   var s = 0;
   for (var i in ListVals)
      s += ListVals[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})
```

57

57

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 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})
```

58

58

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 }
.....
```

59

59

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"} }
       )
```

60

60

MongoDB : syntaxe et exemples

- Importation et exportation de documents
- Gestion des bases et des collections
- Interrogation des données avec *find(…)*
- Application de *methods* aux collections
- Agrégation d'opérations avec *aggregate(…)*
- *JOIN* en MongoDB
- *Map-Reduce* en MongoDB
- **Gestion des Index en MongoDB**
- Comparaison des 3 mécanismes d'interrogation
- Annexe : Modifications d'enregistrements
- Annexe : Ecriture de fonctions Java Script

61

61

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 !
```

62

62

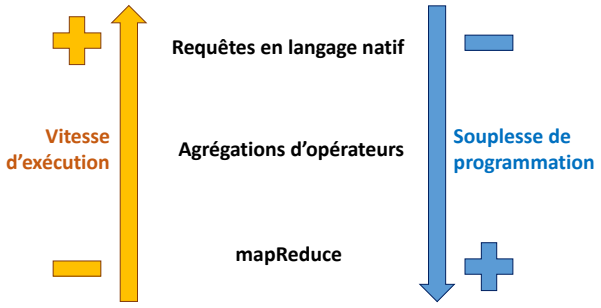
MongoDB : syntaxe et exemples

- Importation et exportation de documents
- Gestion des bases et des collections
- Interrogation des données avec *find(…)*
- Application de *methods* aux collections
- Agrégation d'opérations avec *aggregate(…)*
- *JOIN* en MongoDB
- *Map-Reduce* en MongoDB
- Gestion des Index en MongoDB
- **Comparaison des 3 mécanismes d'interrogation**
- Annexe : Modifications d'enregistrements
- Annexe : Ecriture de fonctions Java Script

63

63

Comparaison des 3 mécanismes



64

64

MongoDB : syntaxe et exemples

65

65

Annexes

66

66

Modifications d'enregistrements (1)

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" ] }
```


Modifications d'enregistrements (2)

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*)

Modifications d'enregistrements (3)

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**

Modifications d'enregistrements (4)

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**

Modifications d'enregistrements (5)

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 !

Ecriture de fonctions Java Script

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
```

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

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

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)