

bases de données
avancées
SASUCGG2

Les bases non relationnelles: NoSQL

Présenté par:

Amal HALFAOUI (Epse GHERNAOUT)

amal.halfaoui@univ-tlemcen.dz

amal.halfaoui@gmail.com

Commandes MongoDB

- Interrogation des données-

La commande find()

Syntaxe

`‘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 :

- Tous les livres de l'auteur Christian Soutou

```
> db.livres.find({auteur : "Christian Soutou"})
```

Equivalent à : `SELECT* FROM livres WHERE auteur ="Christian Soutou"`

- Liste des titres des livres de l'auteur Christian Soutou

```
> db.livres.find({auteur : "Christian Soutou"},{titre:1,_id:0})
```

Equivalent à : `SELECT titre FROM livres WHERE auteur ="Christian Soutou"`

Opérateurs de la commande find()

Liste des opérateurs (par catégorie)

Logiques :

\$and, \$or AND, OR

\$not, \$nor NOT, NOR

Comparaisons :

\$eq, \$ne ==, !=

\$gt, \$gte, \$lt, \$lte >, >=, <, <=

Op de test sur élément :

\$exists Existence d'un champ ?

\$type Teste le type d'un champ

Op de test sur tableau :

\$all Test sur le contenu

\$elemMatch d'un tableau

\$size Taille du tableau

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

Liste des opérateurs (exemples)

Opérateur	équivalent	signification	exemple
\$gt, \$gte	$>, \geq$	Plus grand que	"a" : { "\$gt" : 10 }
\$lt, \$lte	$<, \leq$	Plus petit que	"a" : { "\$lt" : 10 }
\$ne	\neq	Différent de	"a" : { "\$ne" : 10 }
\$in, \$nin	\in, \notin	Fait parti de	"a" : { "\$in" : [10, 12, 15, 18] }
\$or	\vee	OU logique	"a" : { "\$or" : [{ "\$gt" : 10 }, { "\$lt" : 5 }] }
\$and	\wedge	ET logique	"a" : { "\$and" : [{ "\$lt" : 10 }, { "\$gt" : 5 }] }
\$not	\neg	Négation	"a" : { "\$not" : { "\$lt" : 10 } }
\$exists	\exists	Le champ existe dans le document	"a" : { "\$exists" : 1 }
\$size		test sur la taille d'une liste (uniquement par égalité)	"a" : { "\$size" : 5 }

Opérateur And

1- De manière implicite (sur des attributs différents)

- Préciser plusieurs valeurs d'attributs séparées par ' , ' dans le premier argument

```
> db.livres.find({auteur : "Christian Soutou", annee:2012},{titre:1,_id:0})
```

→ les titres des livres de l'auteur soutou **et** année sorties en 2012

Equivalent à : `SELECT titre FROM livres WHERE auteur="Christian Soutou " and annee = 2012`

2- De manière implicite (sur des attributs groupés)

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

```
> db.livres.find({langue : "français", annee:{$gte : 2000, $lt : 2010}},{titre:1,_id:0})
```

→ Tous les livres (titres) écrits en français **et** sortis entre : 2000 (inclus) **et** 2010 (exclus)

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

Opérateur And

3- De manière explicite

Utiliser un opérateur **\$and** explicite

```
➤ db.livres.find({$and:[{langue : "Français"},  
                        {annee:{$gte: 2010}},  
                        {annee:{$lt:2012}}]},  
                {titre:1,_id:0})
```

→ Tous les livres (titres) écrits en français **et** sortis entre : 2010 (inclus) **et** 2012 (exclus)

Opérateur Or

Deux manières d'exprimer le choix:

1- l'opérateur \$In

```
➤ db.livres.find({annee:{$in : [2000 , 2005, 2010]}}, {titre:1,_id:0})
```

→ les titres des livres où l'année sorties est soit 2000 ou 2005, ou 2010

Equivalent à : `SELECT titre FROM livres WHERE annee in (2000, 2005, 2010)`

2- l'opérateur \$Or

```
➤ db.livres.find( { $or: [ { annee:{$gte : "2012"}}, { auteur: "Bruchez Rudi"} ] } )
```

Equivalent à `SELECT *`
`FROM livres`
`WHERE (annee >= 2012 Or auteur = "Bruchez Rudi")`

```
➤ db.livres.find( { editeur:"Eyrolles", $or: [ { annee:{$gte : "2012"}}, { auteur: "Bruchez Rudi"} ] } )
```

Equivalent à `SELECT *`
`FROM livres`
`WHERE editeur = "Eyrolles"`
`and (annee >= 2012 Or auteur = "Bruchez Rudi")`

Opérateur de test sur un élément

- l'opérateur **\$exists**

```
> db.livres.find({annee:{$exists: true}})
```

→ retourne que les documents qui ont le champ année

```
> db.livres.find({annee:{$exists: true, $gte: 2012}})
```

→ Si le champ *année* existe, alors on teste s'il est ≥ 2012

- l'opérateur **\$type**

```
> db.livres.find({annee:{$type: 2}})
```

→ Retourne les documents de la collection *livres* pour lesquels le type du champ *annee* est un **String**
peut utiliser le numéro ou l'alias

Type	Number	Alias
Double	1	"double"
String	2	"string"
Object	3	"object"
Array	4	"array"
Binary data	5	"binData"

Voir les autres correspondances des types bson
<https://docs.mongodb.com/manual/reference/operator/query/type/>

Opérateur d'évaluation

- l'opérateur **\$mod**

```
> db.promo.find({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...)

- Les expression régulières: **\$ regex**

```
> db.livres.find({titre : {$regex : "sql" , $options : "i"} })
```

→ Retourne les livres qui contiennent *sql, ou Sql, ou SQL,...* insensible à la casse
Il y a plusieurs façons d'écrire les expressions régulières ... voir la doc de Mongo

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

Opérateur d'évaluation

- l'opérateur **\$text**:

```
$text { fieldName : { $text : { $search: <string>,  
$language: <string>,  
$caseSensitive: <boolean>,  
$diacriticSensitive: <boolean> } } }
```

```
> db.livres.find({resume : { $text : { " requêtes" , "fr", false, false } } })
```

→ cherche si le mot **requêtes** apparaît dans livres, sans être sensible à la casse, ni aux accents (Requetes, requêtes et requetes seront acceptés).

Rmq: spécifier le langage permet de définir automatiquement les mots de fin de texte et les racines des mots, pour faciliter les recherches.

MongoDB facilite l'analyse du texte sur le web

Opérateur sur les tableaux

1- Operateur : \$all

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

Retourne les documents si le champ *arrayFieldName* contient toutes les valeurs listées

```
> db.livres.find({auteurs : { $all : [ "Christain Soutou" , "Bruchez Rudi" ] } })
```

→ Le document qui contient le sous ensemble sera retourné exp: { **id_**, ..., **auteurs** : [" Christain Soutou" , "Bruchez Rudi", "Gardarin "] }

2- Operateur : \$elemMatch

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

Retourne les documents qui contiennent un champ de tableau avec au moins un élément qui correspond à tous les critères des conditions indiquées

EXP1 (tableau d'elements simples):

soit la collection notation : { **_id**: 1, **scores**: [82, 85, 88] }
{ **_id**: 2, **scores**: [75, 88, 89] }

```
> db.notation.find( { scores: { $elemMatch: { $gte: 80, $lt: 85 } } } )
```

→ Le document **_id**:1 sera retourné car il contient l'élément 82

Rq: si une **seule** condition est exprimée pas besoin de **\$elemMatch**

Opérateur sur les tableaux

2- Operateur : \$ elemMatch

EXP2 (tableau d'objets imbriqués): Soit la collection notation qui contient les documents :

```
{ _id: 1, notes: [ { produit: "abc", score: 10 }, { produit: "xyz", score: 5 } ] }  
{ _id: 2, notes : [ {produit : "abc", score: 8 }, {produit : "xyz", score: 7 } ] }  
{ _id: 3, notes : [ {produit : "abc", score: 7 }, {produit : "xyz", score: 8 } ] }
```

```
> db.notation.find( { notes: { $elemMatch: { produit: "xyz", score: { $gte: 8 } } } } )
```

→ Le document `_id: 3` sera retourné

Rq : s'il y a une seule condition **pas besoin de \$elemMatch**

```
> db.notation.find( { notes: { $elemMatch: { produit: "xyz" } } } )
```



```
> db.notation.find( { "notes.produit" : "xyz" } )
```

Méthodes appliquées aux collections

Méthodes

Syntaxe :

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

Méthodes = **sort(...)**, **forEach(...)**,

Pour certaines méthodes on peut aussi utiliser

db.laCollection.methode(...)

Méthodes = **count(...)**, ...

La méthode sort :

```
> db.laCollection.find(...).sort({"champ1": 1, "champ2": -1})
```

Champ1 : croissant, ensuite pour les mêmes valeurs, ordonner selon le champ2: décroissant

La méthode count :

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

Ou bien :

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

Le nombre de documents de la collection

Méthodes

La méthode `forEach(function())` :

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

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

Exp 1:

```
> db.livres.find({"nbrVentes":{$gt:500000}}).forEach( function(doc)
    {db.livres.update( {_id:doc._id}, {$set:{"mention":"Top_ventes"}} ); } )
```

→ Ajoute la mention aux livres qui ont vendu plus de 500000 exemplaires du livre.

Exp 2:

```
> db.livres.find().forEach(function(doc) {print(doc.titre,doc.isbn); } )
```

→ Affiche le titre et lbn de tous les livres équivalents à:

```
db.livres.find({}, {titre:1, isbn:1 _id:0})
```

Agrégation d'opérations: **aggregate (..)**

Framework d'aggregation

Principe

Depuis la version 2.2, MongoDB propose un **framework d'agrégation** (aggregation framework)

- pour réaliser des opérations complexes d'analyse
- qui permet de récupérer et manipuler les données en pipeline dans mongoDB Sans utiliser des traitements complexes en batch avec **Map/Reduce** (une deuxième façon de faire des aggregations)

Syntaxe

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

- Pour regrouper les données et les manipuler en créant un "pipeline" d'agrégation: La sortie d'une opération est l'entrée de la suivante, ou la sortie finale de l'agrégation

Operations

\$group, \$limit, \$match, \$sort, \$unwind, \$project, \$skip, \$out, \$redact, \$lookup, etc.

<https://docs.mongodb.com/manual/reference/operator/aggregation/index.html>

Framework d'aggregation

\$Group

Permet de regrouper les enregistrements selon l'_id (qui peut être redéfini), et d'appliquer des fonctions de groupe aux autres attributs projetés.

Exemple:

Soit la collection **test** qui contient les documents suivants: **db.test.find()**

```
{ "_id" : 1, "item" : "a", "prix" : 10, "quantite" : 2, "date" : ISODate("2014-03-01T08:00:00Z") }
{ "_id" : 2, "item" : "z", "prix" : 10, "quantite" : 2, "date" : ISODate("2014-03-01T08:00:00Z") }
{ "_id" : 3, "item" : "b", "prix" : 10, "quantite" : 2, "date" : ISODate("2015-03-01T08:00:00Z") }
```

→ On souhaite avoir le prix total des ventes par mois de chaque année

```
> db.test.aggregate( [
{ $group : { _id : { mois: { $month: "$date" }, an: { $year: "$date" } },
  PrixTotal: { $sum: { $multiply: [ "$prix", "$quantite" ] } } }
}
])
```

Resultat

```
{ "_id" : { "mois" : 3, "an" : 2015 }, "PrixTotal" : 20 }
{ "_id" : { "mois" : 3, "an" : 2014 }, "PrixTotal" : 40 }
```

Framework d'aggregation

\$match :

Permet de sélectionner/filtrer les document à passer au pipeline, selon une ou des conditions

```
db.test.aggregate( [ { $match : { item : "a" } } ] );
```

```
db.test.aggregate( [ { $match: { $or: [ { prix: { $gt: 70, $lt: 90 } } ] } } ] )
```

\$sort :

Permet de trier les documents et de les transmettre de façon ordonnée au pipeline

```
db.test.aggregate( [ { $sort : { prix : -1 } } ] )
```

1 = ordre croissant, -1 décroissant

\$project

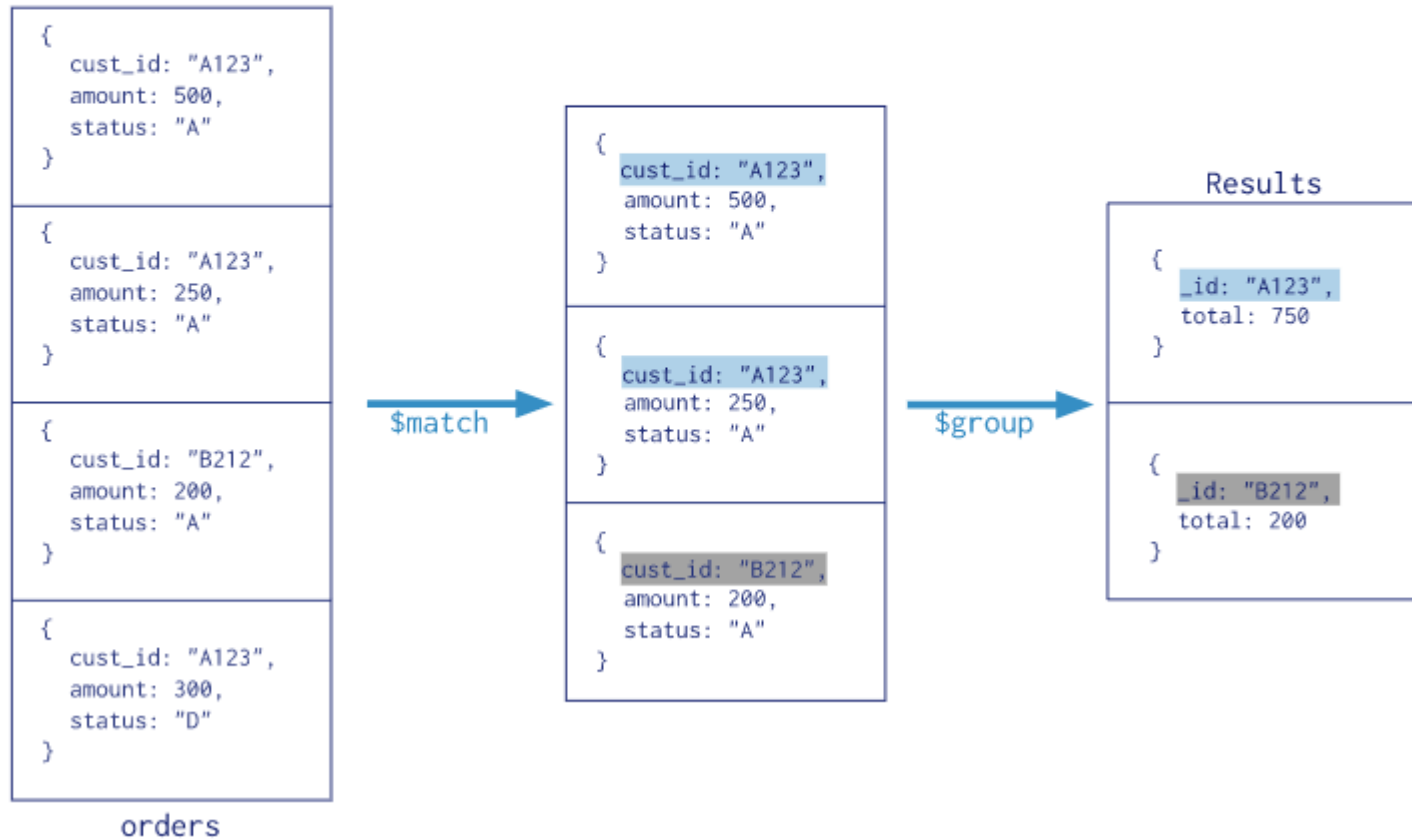
permet de mettre en forme les documents avec les champs existants ou de nouveaux champs, avant de les passer au pipeline

```
db.livres.aggregate( [ { $project : { titre : 1 , auteurs : 1 } } ] )
```

Framework d'aggregation

Exemple de combinaison des operateurs:

Collection
↓
`db.orders.aggregate([`
 `$match stage → { $match: { status: "A" } },`
 `$group stage → { $group: { _id: "$cust_id", total: { $sum: "$amount" } } }`
 `])`



Framework d'aggregation

Exemple de combinaison des opérateurs:

```
{ "_id" : 1, "item" : "a", "prix" : 10, "quantite" : 2, "date" : ISODate("2014-03-01T08:00:00Z") }
{ "_id" : 2, "item" : "z", "prix" : 10, "quantite" : 2, "date" : ISODate("2014-03-01T08:00:00Z") }
{ "_id" : 3, "item" : "b", "prix" : 10, "quantite" : 2, "date" : ISODate("2015-03-01T08:00:00Z") }
{ "_id" : 4, "item" : "a", "prix" : 10, "quantite" : 2, "date" : ISODate("2014-03-01T08:00:00Z") }
```

1

```
db.test.aggregate( [
  { $count : "nombre" }
])
```

2

```
db.test.aggregate( [
  { $group: { _id : null, nombre: { $sum : 1 } } },
  { $project: { _id: 0 } }
])
```

Equivalent à `db.test.count()` ou `db.test.find().count()`

3

```
db.test.aggregate( [
  { $group: { _id : "$_id" } }, { $count : "nombre" }
])
```

4

```
db.test.aggregate( [
  { $group: { _id : "$_id", nombre: { $sum : 1 } } }
])
```

Attention: retourne le nombre de groupe
et vu que `_id` est unique donc retourne la
meme chose que 1

5

```
db.test.aggregate( [
  { $group: { _id : "$item", nombre: { $sum : 1 } } },
  { $match: { nombre : { $gt: 3 } } }
])
```

retourne le nombre de document pour
chaque groupe d'item (select item, count()
from test
group by item)
Having count(*) > 3

Framework d'aggregation

\$unwind:

permet de réorganiser les champs d'un tableau, avec un document pour chaque élément du tableau

Exemple de document

```
{ "_id":1, "isbn" : "9782212134131", "titre" : "Modélisation des bases de données",  
  "auteur" : [ "Christian Soutou", "Frédéric Brouard" ] }
```

➤ `db.livres.aggregate([{$unwind:"$auteur"}])`

Résultat:

```
{ "_id" : 1, "isbn" : "9782212134131", "titre" : "Modélisation des bases de données",  
  "auteur" : "Christian Soutou" }
```

```
{ "_id" : 1, "isbn" : "9782212134131", "titre" : "Modélisation des bases de données",  
  "auteur" : "Frédéric Brouard" }
```

➔ Un document est créé pour chaque auteur

Framework d'aggregation

\$out :

permet de créer une collection avec le résultat d'un pipeline

Exemple

Soient les documents de la collection **livres**

```
{ "_id" : 1, "isbn" : "9782212134131", "titre" : "Modélisation des bases de données", "auteur" : [ "Christian Soutou", "Frédéric Brouard" ], "editeur" : "Eyrolles" }
{ "_id" : 2, "isbn" : "2212141556", "titre" : "Les bases de données NoSQL et le big data", "auteur" : [ "Rudi Bruchez" ], "editeur" : "Eyrolles" }
{ "_id" : 3, "isbn" : "208134761X", "titre" : "Maktub", "auteur" : [ "Paulo Cohelo" ], "editeur" : "ellipses" }
```

→ Pour chaque Editeur, l'ensemble (titre) des livres publiés

```
➤ db.livres.aggregate( [ { $group : { _id : "$editeur", "livres" : { $push : "$titre" } } }, { $out : "editeurs" } ] )
```

Résultat: Création de la collection **editeurs** qui contient les documents

```
{ "_id" : "ellipses", "livres" : [ "Maktub" ] }
{ "_id" : "Eyrolles", "livres" : [ "Modélisation des bases de données", "Les bases de données NoSQL et le big data" ] }
```

Framework d'aggregation

Toutes les opérations précédentes se focalisaient sur **1 collection**

\$lookup :

permet de faire une "jointure" entre des objets de **deux collections**,

Exp

Collection livres

```
{ "_id" : ObjectId("5a99bded567089e00a38a0e4"), "isbn" : "9782212134131", "titre" :  
"Modélisation des bases de données", "auteur" : 2, "langue" : "Français" }
```

Collection Auteurs

```
{ "_id" : ObjectId("5a94824c664976bfc818548c"), "num_auteur" : 2, "nom" : "Soutou", "fonction"  
: "Maitre assistant" }
```

```
> db.livres.aggregate([ { $lookup:  
{ from: "auteurs",  
  localField: "auteur",  
  foreignField: "num_auteur",  
  as: "Livre_auteur"  
} } ])
```

- la collection référencée qui contient le champ de la jointure
- le champ de la collection actuelle
- Le champ référencé
- Dans le pipeline, on génère un document égal au enregistrements de collection actuelle (livre) enrichie d'un champ "Livre_auteur" contenant un tableau tous les enregistrements de la collection référencée (auteur) de même clé

Le document généré peut être sauvegardé dans la collection "resJoin" {"\$out" : "resJoin"}

Framework d'aggregation

Toutes les opérations précédentes se focalisaient sur **1 collection**

\$lookup :


permet de faire une "jointure" entre des objets de **deux collections**,

```
> db.livres.aggregate([ { $lookup:  
  { from: "auteur",  
    localField: "auteur",  
    foreignField: "num_auteur",  
    as: "Livre_auteur"},  
  {"$out" : "resJoin"}  
]);
```

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

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

Collection: resJoin



```
{  
  "_id" : ObjectId("5a99bded567089e00a38a0e4"),  
  "isbn" : "9782212134131",  
  "titre" : "Modélisation des bases de données",  
  "auteur" : 2,  
  "langue" : "Français",  
  "Livre_auteur" : [  
    {  
      "_id" : ObjectId("5a94824c664976bfc818548c"),  
      "num_auteur" : 2,  
      "nom" : "Soutou",  
      "fonction" : "Maitre assistant"  
    }  
  ]  
}
```