TP - MongoDB

Bases de données avancée

Larbi Boubchir

Partie1: INSERTION DE DONNEES (INSERT DATA)

```
>use library


>document = ( { Type : "Book", Title : "Definitive Guide to MongoDB", ISBN :
"987-1-4302-3051-9", Publisher : "Apress", Author: ["Membrey, Peter",
"Plugge, Eelco", "Hawkins, Tim" ] } )


>db.media.insert(document)


>db.media.insert( { Type : "CD" ,Artist : "Nirvana",Title : "Nevermind",
Tracklist : [
{ Track : "1 ", Title : "Smells like teen spirit", Length : "5:02 "}, { Track : "2 ",
Title : "In Bloom", Length : "4:15 " }
]} )
```

REQUETES:

QUE FAIT CES COMMANDES?

```
>db.media.find()
>db.media.find ( { Artist : "Nirvana" } )
>db.media.find ( {Artist : "Nirvana"}, {Title: 1} )
>db.media.find ( {Artist : "Nirvana"}, {Title: 0} )
>db.media.find( { "Tracklist.Title" : "In Bloom" } )
>db.media.findOne()
```

Add the function pretty() for the indentation

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

FONCTIONS: SORT, LIMIT et SKIP

QUE FAIT CES COMMANDES?

```
>db.media.find().sort( { Title: 1 })
>db.media.find().sort( { Title: -1 })
>db.media.find().limit( 10 )
>db.media.find().skip( 20 )
>db.media.find().sort ( { Title : -1 } ).limit ( 10 ).skip ( 20 )
```

AGGREGATION

QUE FAIT CES COMMANDES?

```
>db.media.count()
>db.media.find( { Publisher : "Apress", Type: "Book" } ).count()
>db.media.find( { Publisher: "Apress", Type: "Book" }).skip(2).count(true)
```

DISTINCT()

QUE FAIT CES COMMANDES?

Add a new record

>document = ({ Type : "Book",Title : "Definitive Guide to MongoDB", ISBN: "1-4302-3051-7", Publisher : "Apress", Author : ["Membrey, Peter","Plugge, Eelco","Hawkins, Tim"] })

>db.media.insert (document)

>db.media.distinct("Title")

>db.media.distinct ("ISBN")

>db.media.distinct ("Tracklist.Title")

AGGREGATION

QUE FAIT CETTE COMMANDE?

```
>db.media.group ( { key: {Title : true},
initial: {Total : 0},
reduce : function (items,prev) {
prev.Total += 1 }
})
```

- Key: grouping parameter
- Initial: initial value (0 by default)
- Reduce: takes 2 arguments, the document (items) and the counter (prev) and
- performs aggregation
- Cond: condition that the attributes of the document must respect

ADD MORE RECORDS

```
>dvd = ( { Type : "DVD", Title : "Matrix, The", Released : 1999, Cast: ["Keanu Reeves","Carry-Anne Moss","Laurence Fishburne","Hugo Weaving","Gloria Foster","Joe Pantoliano"] } )
>db.media.insert(dvd)

>dvd = ( { "Type" : "DVD", "Title" : "Toy Story 3", "Released" : 2010 } )
>db.media.insert(dvd)
```

Insert with JavaScript

```
>function insertMedia( type, title, released ){
db.media.insert({
"Type": type,
"Title": title,
"Released": released
}); }

>insertMedia("DVD", "Blade Runner", 1982 )
```

COMPARISON OPERATORS

$gt, $lt, $gte, $lte, $ne, $in, $nin (resp. >,<,>=,<=,!=,IN, NOT IN)

What do these queries do?

```
>db.media.find ( { Released : {$gt : 2000} }, { "Cast" : 0 } )
>db.media.find( {Released : {$gte: 1990, $lt : 2010}}, { "Cast" : 0 })
>db.media.find( { Type : "Book", Author: {$ne : "Plugge, Eelco"}})
>db.media.find( {Released : {$in : ["1999","2008","2009"] } }, { "Cast" : 0})
>db.media.find( {Released : {$nin : ["1999","2008","2009"] },Type : "DVD" }, {
"Cast" : 0 } )
```

$or

What do these queries do?

```
>db.media.find({ $or : [ { "Title" : "Toy Story 3" }, { "ISBN" : "987-1-4302-3051-9" }
] } )
>db.media.find({ "Type" : "DVD", $or : [ { "Title" : "Toy Story 3" }, { "ISBN" : "987-1-
4302-3051-9" } ] })
```

$SLICE

$slice: combines limit() and skip()

- $slice: [20, 10] // skip 20, limit 10
- $slice: 5 // The first 5
- $slice:-5 //The last 5

```
>db.media.find({"Title" : "Matrix, The"}, {"Cast" : {$slice: 3}})
>db.media.find({"Title" : "Matrix, The"}, {"Cast" : {$slice: -3}})
```

$SIZE AND $EXISTS

```
>db.media.find ( { Tracklist : {$size : 2} } )

>db.media.find ( { Author : {$exists : true } } )

>db.media.find ( { Author : {$exists : false } } )
```

INDEX CREATION

Ascending index

```
>db.media.ensureIndex( { Title :1 } )
```

Descending index

```
>db.media.ensureIndex( { Title :-1 } )
```

Index for enbed objects

```
>db.media.ensureIndex( { "Tracklist.Title" : 1 } )
```

Force the index usage: hint()

```
>db.media.find( { ISBN: "987-1-4302-3051-9"} ) . hint ( { ISBN: -1 } )
      error: { "$err" : "bad hint", "code" : 10113 }
>db.media.ensureIndex({ISBN: 1})
>db.media.find( { ISBN: "987-1-4302-3051-9"} ) . hint ( { ISBN: 1 }  )
>db.media.getIndexes()
```

DATA UPDATE

Update (condition, newObject, upsert, multi)

- upsert = true //create the object if does not exist
- Multi Specifies whether the change is made on a single object (default) or on all objects that meet the condition

>db.media.update({ "Title" : "Matrix, the"}, {"Type" : "DVD", "Title" : "Matrix, the", "Released" : "1999", "Genre" : "Action"}, true)

Add/delete an attribute

>db.media.update ({ "Title" : "Matrix, the" }, {$set : { Genre : "Sci-Fi" } })

>db.media.update ({"Title": "Matrix, the"}, {$unset : { "Genre" : 1 } })

Delete

- Documents meeting a condition: >db.media.remove({ "Title" : "Different Title" })
- All documents: >db.media.remove({})
- All the collection: >db.media.drop()

Partie 2:
BASE DE DONNEES GEOGRAPHIQUE

Import data

>./bin/mongoimport --type json -d geodb -c earthquakes --file earthquakes.json

Quelques requêtes de base:

1) Count the number of documents
2) Show first 5
3) View the 6th document
4) How many separate statuses exist in the DB?

MODIFICATION DES DONNÉES

- Combien de documents contient la propriété felt (propriety.felt! = Null) ?
- Supprimer ce champ pour les documents pour lesquels il est Null
- Ajouter une colonne iso_date dont la valeur est la conversion de timestamp contenu dans properties.time

```
> db.earthquakes.find().forEach(
      function(eq){
      eq.properties.iso_date = new Date(eq.properties.time);
      db.earthquakes.save(eq);
}
 );
```

NETTOYAGE DES DONNÉES (DATA CLEANING)

Convert the string from the properties.types field to an array and put it in a field types_as_array

Use the function ch.split (",") to separate a string ch into several words according to the separator ‘,’

```
db.earthquakes.find().forEach( function(eq){
    var str = new String(eq.properties.types);
    eq.properties.types_as_array = str.split(",");
    db.earthquakes.save(eq); } );
```

Vérifier en montrant le 1er document

NETTOYAGE DES DONNÉES (DATA CLEANING)

Clean the empty elements ("") from the array properties.types_as_array

```
>db.earthquakes.update(
        {},
        { $pullAll: { "properties.types_as_array" : [""] } },
        { multi: true }
)
```

Vérifier en montrant le 1er document

REQUETES

- Indiquez le nombre de documents dont la liste de types (properties.type_as_array) contient "geoserve" et "tectonic- summary"

- Indiquez le nombre de documents dont la liste de types (properties.type_as_array) contient "geoserve" ou "tectonic- summary"

INDEXATION GÉOGRAPHIQUE

Nous allons maintenant modifier les données afin d'adapter les coordonnées géographiques au format qui nous permettra de construire un index 2dsphere.

Normalisez les données en supprimant le dernier élément de la table 'geometry.coordinates' et en le copiant dans un champ 'profondeur'.

Exemple:

```
geometry : {
    "type" : "Point",
    "coordinates" : [-147.35, 63.59, 0.1]
}

// devient ....

depth : 0.1
geometry : {
    "type" : "Point",
    "coordinates" : [-147.35, 63.59]
```

INDEX CREATION

Créer un index de type 2dsphere sur les attributs «geometry»

Requête:

Exécuter une requête qui recherche à partir de "earthquakes" near -74, 40.74 (dans un rayon de 1000 m) (utilisez $geoWithin and center)

Documentation :
http://docs.mongodb.org/manual/reference/operator/query- geospatial/

REQUETE

Trouvez les "earthquakesde" qui sont autour de la place "8km NW of Cobb, California", avec une distance maximale de 500 km.

AGGREGATE

Séquence ordonnée des opérateurs

- **Commande :**

  > db.earthquakes.aggregate([{$op1 : {}}, {$op2 : {}}, ...]);

- **Opérateurs :** //equivalent SQL

$match : Simple filter // where
$project : Projection //select
$sort : Sorting //order by
$unwind : normalisation 1NF //group by + fn
$group : grouping + aggregate function
$lookup : left join(from 3.2) //left outer-join
$out : storing the result (from 3.2)
$redact : conditional pruning (nested documents) + $sample, $limit, $skip,

AGGREGATE: EXAMPLES

Sequence: le résultat d'une opération sert comme entrée pour la prochaine

```
> db.earthquakes.aggregate([{$match : { "properties.type" : "quarry"}},
{$project : { "_id" : 1, "geometry" : 1}},
{$sort : { "depth" : -1}}
]);
```

$unwind

Créer un document pour chaque instance

```
>db.earthquakes.aggregate([ {$unwind :"$properties.types_as_array"},{$limit:5} ])
```

AGGREGATE: EXEMPLES

Group : key (_id) + aggregate ($sum / $avg / ...)

No group: null

```
> db.users.aggregate([ {$group : {"_id" : null, "res": {$sum : 1}}} ]);
```

Groupe by value : $key

```
> db.users.aggregate([ {$group:{"_id" : "$age", "res": {$sum : 1}}} ]);
```

Average: $key

```
> db.users.aggregate([{$group:{"_id":"$address.city", "moy": {$avg: "$age"}}} ]);
```

Exemple: sequence

```
> db.movies.aggregate([
      {$match: { "year" : {$gt : 1995}}},
      {$unwind : "$genres_as_array"},
      {$group : {"_id" : "$year", "count": {$sum: 1}}},
      {$match : {"count" : {$gt : 2}}},
      {$sort : { "count" : -1} ]);
```

ADMINISTRATION

Backup

>mkdir testmongobackup

>cd testmongobackup

>../mongodb/bin/mongodump --help

>../mongo/bin/mongodump

>../mongodump --db library --collection media

→ ./dump/[databasename]/[collectionname].bson

Restore

>cd testmongobackup

>../mongo/bin/mongorestore –help

- Tout restaurer

  >../mongo/bin/mongorestore –drop

- Restaurer une seule collection

  >../mongo/bin/mongorestore –d library -c media –drop

SECURITY

Authentification

- Client side

  > use admin

  >db.addUser(“admin”, “adminpassword”)

- Server side

  > use admin

  >db.addUser(“admin”, “adminpassword”)

- Shell (Restart the server)

  >sudo service mongodb restart

  or

  >db.shutdownServer()

- Authenticate

  >use admin

  >db.auth("admin","adminpassword")

>use library
>db.addUser(“ronaldo”, “ronaldopassword”)
>db.addUser(“messi”, “messipassword”,true) //read only
>db.removeUser(“ronaldo”)

SECURITY: USER, PRIVILEGE, RESOURCE

```
db.createUser( {
user: "reportsUser",
pwd: "12345678",
roles: [
{ role: "read", db: "reporting" },
{ role: "read", db: "products" },
{ role: "read", db: "sales" },
{ role: "readWrite", db: "accounts" }
]
})

db.dropUser("reportsUser")
db.dropAllUsers( )
```

CREATE ROLE

```
db.createRole({
role: "<name>",
privileges: [ { resource: { <resource> }, actions: [ "<action>", ... ] }, ... ],
roles: [ { role: "<role>", db: "<database>" } | "<role>", ... ] })

db.updateRole("< rolename >",
...)

db.dropRole("< rolename >")

db.grantPrivilegesToRole( "< rolename >",
[
        { resource: { <resource> }, actions: [ "<action>", ... ] },
        ... ],
        )
db.grantRolesToRole( "<rolename>", [ <roles> ],)
```