Rapport de TP - MongoDB

loïc divad

December 2014

1 Introduction

Ce TP est associé aux cours électifs de deuxième année et porte sur la base de donnée NoSQL orientée documents, MogoDB. Elle diffère en quelques points avec les outils habituellement utilisés par les DBA tel que les bases ORACLE vu dans le cours de base de donnée avancée. Ne pouvant assiter au cours je rédige ici une partie du TP.

2 Requêtage

- `db.media.find()`
  Cette requête trouve tous les éléments de la collection "media". Cette collection appartient à la base de donnée "library".
- `db.media.find({Artist:"Nirvana"})`
  Cette requête trouve tous les documents ayant pour valeur "Nirvana" associée à leur champ artiste.
- `db.media.find({Artist:"Nirvana"},{Title: 1})`
  Cette requête trouve tous les documents ayant pour valeur "Nirvana" dans leur champ artiste et n'affiche que le champ "Title" et "_id" grâce à une projection[1].
- `db.media.find({Artist:"Nirvana"},{Title:0})`
  Cette requête trouve tous les documents ayant pour valeur "Nirvana" dans leur champ artiste sans afficher leur champ "title".
- `db.media.find({"Tracklist.Title":"In Bloom"})`
  Cette requête trouve tous les documents ayant un sous document de titre "in Bloom".
- `db.media.findOne()`
  findOne retourne le premier document respectant les conditions passées en paramètre (aucune conditions dans ce cas). Elle renvoie donc le premier document inséré dans la collection.

Note: La projection[1] est le deuxième argument optionel de la méthode .find() il est de forme "field": 1/0 et permet de spécifier les champs que l'on veut récupérer. Par défaut le champ _id s'affiche si on ne préscise pas "_id": 0

- `db.media.find().sort({Title:1})`
  La méthode .find() retourne un objet de type "Object Cursor". En applicant la méthode sort on ordonne le résultat de la requête. Ici on affiche tous les documents de la collection media dans la base Library par ordre croissant.
- `db.media.find().sort({Title:-1})`
  Ici on affiche tous les documents de la collection media dans la base Library par ordre décroissant.

[1] *Usages and examples of $(projection)*

- `db.media.find().limit(10)`
  Ici on affiche les 10 premiers documents de la collection.

- `db.media.find().pretty()`
  La commande .pretty() permet d'agencer le JSON qui résulte de la requête pour une meilleur comprehension du résultat.

```
MongoDB shell version: 2.6.5
connecting to: test
Server has startup warnings:
2014-12-22T22:16:42.929+0100 [initandlisten]
2014-12-22T22:16:42.929+0100 [initandlisten] **
> use library
switched to db library
> db.media.find().pretty()
{
        "_id" : ObjectId("54988a9886af7e3c19b1c
        "Type" : "Book",
        "Title" : "Definitive Guide to MongoDB"
        "ISBN" : "987-1-4302-3051-9",
        "Publisher" : "Apress",
        "Author" : [
                "Membrey, Peter",
                "Plugge, Eelco",
                "Hawkins, Tim"
        ]
}
{
        "_id" : ObjectId("54988be196fc5d1018531
        "Type" : "CD",
        "Artist" : "Nirvana ",
        "Title" : "Nevermind",
        "Tracklist" : [
                {
                        "Track" : "1 ",
                        "Title" : "Smells like
                        "Length" : "5:02 "
                },
                {
                        "Track" : "2 ",
                        "Title" : "In Bloom ",
                        "Length" : "4:15 "
                }
        ]
}
>
```

Figure 1: Exemple d'utilisation de la méthode .pretty() en console

Ce comportement est automatiquement géré par l'interface RoboMongo[2]. Après cette première partie sur les requêtes nous continurons le TP sur RoboMongo et laisseront tomber la consol.

- `db.media.find().skip(20)`
  On recupère tous les documents à partir du 20 éléments du curseur renvoyé par .find(). La méthode .skip(n) permet de d'ignorer un certain nombre de résultats.

- `db.media.find().sort ({Title:-1}).limit(10).skip(20)`
  On obtient les 10 résultats à partir du 20ème sur la liste des documents par ordre décroissant des titres.

Remarque: On peut trouver cette dernière requête particulièrement contre intuitive. on pourrait penser que db.media.find().sort({Title:-1}).limit(m) renvoie une liste de m éléments et que l'on y applique .skip(n) pour passer les n premiers. Au lieu de celà on peut inverser les instructions .limite() et .skip(), on a le même résultat.

[2] *RobotMongo, the MongoDB management tool. http://robomongo.org*

3 Agregation

- `db.media.count()`
  Compte le nombre de documents dans la collection.
- `db.media.find({Publisher:"Apress",Type:"Book"}).count()`
  Compte le nombre de documents de type livre ET d'éditeur Apresss.
- `db.media.find({Publisher:"Apress",Type:"Book"}).skip(2).count(true)`
  Compte le nombre de documents de type livre ET d'éditeur Apresss sans compter les deux premiers resultats.

4 Distinct et regroupement

Après un bref passage par la documentation[3] on sait que la méthode .disctinct() s'applique sur un objet de type collection et renvoie dans un tableau de valeurs distinctes correspondants au champ passé en paramètre de la méthode. `db.media.distinct("ISBN")` renvoie alors une liste d"ISBN tous différents. La taille du tableau est également égale au nombre de livre car l"ISBN est unique.

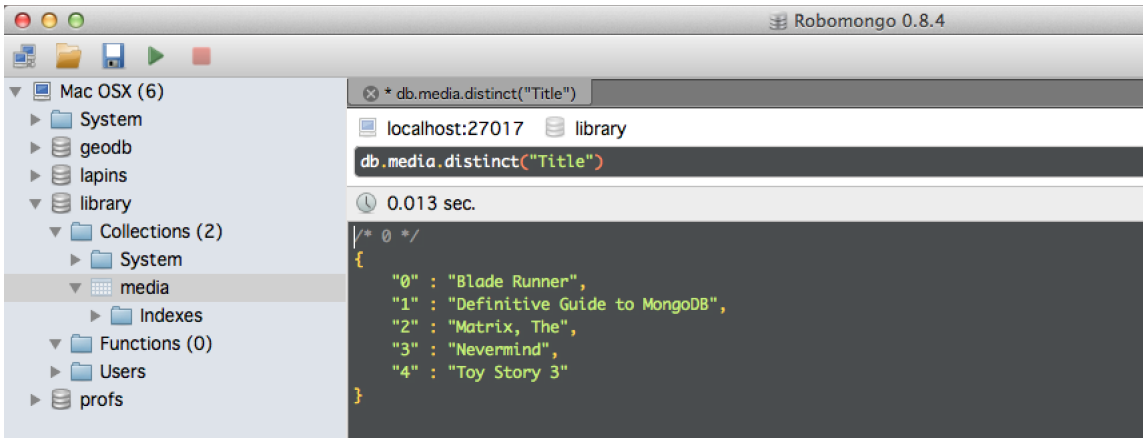


Figure 2: Aperçu du tableau Javascripte retourné par la méthode .distinct()

Il est également possible de filtrer les documents traités par un .distinct() à l'aide d'un argument optionnel **query**. La requête suivente contruit alors un tableau de titre distinct uniquement à partir des livres de prix supérieur à 5.

```
db.media.distinct("Title",{price:{$gt:5}})
```

A l'aide des indications sur la méthode .groupe() on essaie de résoudre l'exercice suivant:

4.1 Exercice, transformer cette requête SQL en requête mongo

select a,b,sum(c) csum from coll where active=1 group by a,b

```
db.coll.group({ key:  {a:  true, b:  true},
   cond:  { active:  1 },
   reduce:  function(curr, result){
      result.csum += curr.c;
   },
initial:  {csum:  0}
});
```

[3] *Examples of the db.collection.distinct() method:*

5 Opérateur de comparaison

- `db.media.find({Released:{$gt:2000}},{"Cast":0})`
  Renvoie tous les éléments réalisés après l"an 2000 sans afficher leur casting.
- `db.media.find({Released:{$gte:1990,$lt:2010}},{"Cast":0})`
  Renvoie tous les documents réalisés entre 1990 et 2010.
- `db.media.find({Type:"Book",Author:{$ne:"Plugge, Eelco"}})`
  Sélectionne tous les documents de type "book" dont les auteurs ne sont pas Plugge et Eelco.
- `db.media.find({Released:{$in:["1999","2008","2009"]}},{"Cast":0})`
  Renvoie les éléments ayant une date de réalisation dans la liste suivant "$in ".
- `db.media.find({Released:{$nin:["1999","2008","2009"]},Type:"DVD"},{"Cast":0})`
  Renvoie tous les dvd de 1999, 2008 et 2009.
- `db.media.find({$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"}]})`

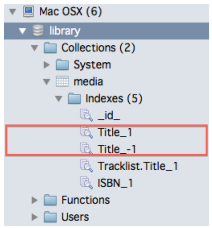
Les deux dernières requêtes renvoient les documents si le titre ou l"ISBN respecte la condition. La dernière est plus selective car elle ne concerne que les documents de type dvd. La condition "type :DVD" est en produit logique avec la condition préscédente.

L'argument **$slice** se place au sein d'une projection et peut prendre la forme d'un intervale [m,n] pour limiter les résultats de la requête.

L'option **$size** concerne les tableaux, c'est une condition sur le nombre délements. Ainsi la requête suivante renvoie tous les documents ayant un champ tracklist composé de 2 titres: `db.media.find({Tracklist:{$size:2}})`

L'option **$exist** ici est assez différente de WHERE EXIST que l'on connait en SQL. Elle porte sur l'existence du champs dans un document et non sur la valeur. Le format NoSQL nous libère du traditionel schéma. Il n'y a pas de tables avec des champs obligatoires laissés à Null si une valeur est manquante. Il se peut donc que certain documents, en fonction de leur type (CD, DVD, Livre) présente ou pas certain champs comme Tracklist. La requête suivante retourne donc tous les documents susceptibles d'avoir un auteur:
`db.media.find({Author:{$exists:true}})`

6 Création d'un index



Après la création de l'index avec la méthode ensureIndex un dossier arrive sous média indexes, l'index est définie au niveau de la collection associée. On y retrouve les objets index_1 et index_-1

Remarque: Sans index les recherches de type .find() ou .group() scan toute la collection. Elles font également appel, pour un large volume de data, au démon de mongodb : mongod. Ce qui est particulièrement lent. On créer alors des index pour faciliter la recherche sur les collections.

les indexes de mongodb respectent une organisation B-Tree[4]...

Note sur le model de donnée d'arbre en B:
C'est un model pour organiser des données de manière ordonnée. Il est sous forme d'arbre et grossit par la racine, c'est à dire que L'insertion des données (les clefs) se fait depuis le neud parent. Les clefs sont alors dirigées vers les neuds inférieures en fonction de leur valeur de sorte que le bout des branches soit ordonnée.

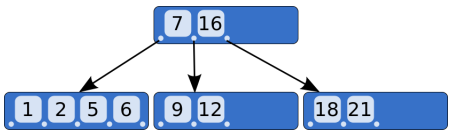


Figure 3: Représentation d"un arbre en B d"ordre 2.

Pour faire le raprochement avec nos indexes mongodb, lindex Title_1 correspond à la dernière ligne de l'arbre. c'est une suite de titres rangés dans l'ordre alphabétique dont chaqu'un a associé à un _id.

Figure 4: Binding entre l"index et la collection.[5]

La méthode .hint() force la recherche à l'aide d'un index. En indiquant des arguments on désigne un l'index approprié. Dans un premier temps lorsque l'index n'existe pas encore la ligne: `error:"$err": "bad hint", "code": 10113` nous indique une erreur. Une fois le l'index crée mongodb fait lien entre l'_id indexé et le document de manière à pouvoir restituer des documents entiers.

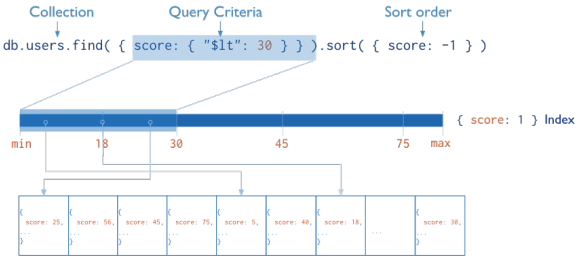


Figure 5: Binding entre l'index et les documents de la collection[5]

[4] *http://en.wikipedia.org/wiki/B-tree*
[5] *Ressource images: http://mongodb.org*

Résultat de la méthode .getIndexes() est une liste des indexes avec les propriétés v version d'index (cette version dépend de la version de mongodb). key est le champ d'indexation, name le nom de l'index et ns pour name space, base de donnée [point] la collection.

```
{
    "0" : {
        "v" : 1,
        "key" : {
            "id" : 1
        },
        "name" : "id",
        "ns" : "library.media"
    },
    "1" : {
        "v" : 1,
        "key" : {
            "Title" : 1
        },
        "name" : "Title1",
        "ns" : "library.media"
    },
    "2" : {
        "v" : 1,
        "key" : {
            "Title" : -1
        },
        "name" : "Title -1",
        "ns" : "library.media"
    },
    "3" : {
        "v" : 1,
        "key" : {
            "Tracklist.Title" : 1
        },
        "name" : "Tracklist.Title 1",
        "ns" : "library.media"
    }
}
```

Listing 1: Résultat de la méthode .getIndexes()

7 Mise à jour des données - le U de CRUD

Pour faire la mise à jour d'un ou plusieur documents on a à notre disposition la fonction **.update()**. C'est une méthode d'objet collection et non celle d'un curseur renvoyé par .find() ou .group(). Cette fonction prend les arguments suivants: **query**, **object** et **option**.

```
1 db.collection.update(
2     { type: "",
3       size: {$gt : 30}
4     },
5     {
6         multi: true, // modifie tous les documents trouve par query
7         upsert: true, // creer le document si il n"existe pas.
8     }
9 );
```

Listing 2: fonction update().

8 Suppression des données - le D de CRUD

Dans mongoDB la suppression peut être effectuée à différent niveaux.

- Il est possible de supprimer un document.
  `db.media.remove({"Title":"Different Title"})`
- Pour vider toute une collection.
  `db.media.remove({})`
- Pour supprimer une collection.
  `db.media.drop()`

9 Exercice i

L'exercice tourne autour d'une base de donnée contenant des lapins. Pour la création de la base de données et l'insertion je propose un script js qui sera joué de la façon suivante.
`mongo exo_lapin.js`
les autres requêtes seront exécutées dans le shell mongo. Après la création de la base de donnée on demande l'affichage de tous les documents

```
1  var db = connect("lapins");
2  // Connection a la base mongo.
3
4  //creation en dure de la base.
5  var all_rabbits = [
6
7    {nom: "leny", genre: "f", ville: "Lyon", regime: ["carotte",
8                                                      "courgette"
9                                                      ],
10    poids: 4, taille: 20 },
11
12   {nom: "bunny", genre: "h", ville: "Paris", regime: [ ],
13    poids: "3", taille: ""},
14
15   {nom: "olto", genre: "h", ville: "Paris", regime: ["raisin",
16                                                      "carotte",
17                                                      "salade"
18                                                      ],
19    poids: 5, taille: 25 }
20 ];
21
22 // Pour charque lapin on insert une ligne du JSON
23 all_rabbits.forEach(function(rabbit){
24   db.france.insert(rabbit);
25 })
26
27 print("insertion terminee.");
28
29 //affichage en console du contenut de la base.
30 var lapin_fr = db.france.find();
31
32 lapin_fr.forEach(function(m){
33   printjson(m);
34 });
```

Listing 3: Création de la base de donnée "lapin"

```
~/Documents/mongo workspace loicMDIVAD \$ mongo exo_lapins.js
MongoDB shell version: 2.6.6
connecting to: test
connecting to: lapins
insertion terminee.
{
  "_id" : ObjectId("54a025fca35fa2e9d3a4dac2"),
  "nom" : "leny",
  "genre" : "f",
  "ville" : "Lyon",
  "regime" : [
    "carotte",
    "courgette"
  ],
  "poids" : 4,
  "taille" : 20
}
{
  "_id" : ObjectId("54a025fca35fa2e9d3a4dac3"),
  "nom" : "bunny",
  "genre" : "h",
  "ville" : "Paris",
  "regime" : [ ],
  "poids" : "3",
  "taille" : ""
}
{
  "_id" : ObjectId("54a025fca35fa2e9d3a4dac4"),
  "nom" : "olto",
  "genre" : "h",
  "ville" : "Paris",
  "regime" : [
    "raisin",
    "carotte",
    "salade"
  ],
  "poids" : 5,
  "taille" : 25
}
```

Listing 4: lapin.france

On applique quelques requêtes sur cette base.

- Trouvez tous les lapins mâles?
  `db.france.find({"genre":"m"});`
- Nombre de lapins qui aiment les carottes et qui pèsent plus de 4kg
  `db.france.find({regime :{$all:['carotte']},poids:{$gt:4}).count();`
  Le compte donne 1, car ici on interprète plus par strictement supérieur. Si on remplace $gt par $gte on obtient 2.
- Les lapins qui aiment les courgettes ou les raisins ou qui n'ont pas de champ « ville »
  `db.france.find({$or:[regime:{$in:["salade","courgette"]}},{ville:null}]});`
- Tous les lapins qui n'aiment pas la salade.
  `db.france.find({regime:{$nin:["salade"]}});`
  Leny et bunny font les difficiles.
- Nous savons que Bunny se trouve en France, rajouter un champ pays à ce document.
  `db.france.update({"nom":"bunny"},{$set:{"departement":"Île-de-France"}});`
  N'ayant pas de nom de collection je l'ai appelé france. On traite donc cette question avec le nom du département.
- Supprimer le champ « taille », s'il existe, de tous les documents.
  `db.france.update({},{$unset:{taille:""}},false,true);`

- Supprimer la base de données.
  `db.france.drop()`
  ... plus de petits lapins.

10 Exercice ii - Base de donnée géographique.

On procède à l'insertion:
`> mongoimport -type -json -d geodb -c earthquakes -file eartquakes.json`
Puis on applique les modifications :

```
1 db.earthquakes.find().forEach(
2     function(eq){
3         eq.properties.iso_date = new Date(eq.properties.time);
4         db.earthquakes.save(eq);
5     }
6 );
```

Le sujet propose de convertir la chaîne de caractère du champ properties.types en tableau et le mettre dans un champ types_as_array.

```
1     db.earthquakes.find().forEach( function(eq){
2         eq.properties.types_as_array= eq.properties.types.split(",");
3         eq.properties.types_as_array.shift();
4         eq.properties.types_as_array.pop();
5         db.earthquakes.save(eq);
6         }
7     );
```

Listing 5: Chaine de carractères vers tableau.

Reamrque: Si on observe un ou deux documents au hasard on remarque que tous les champs types commencent et finissent par un séparateur. Une case vide se créer au début et à la fin du tableau, c'est assez dommage. De mémoire il existe <array>.shift() en javascript. Impossible de retrouver cette fonction dans la doc mongodb, on l'essai donc directement sur l'attribut du document aux lignes 3 et 4.

Comme en JavaScript, la méthode ne retourne pas un tableau mais affecte directement le tableau en question et donc, ici, le document. On affiche un document pour tester la requête.

```
1 db.earthquakes.findOne({},
2     {
3         id:1,
4         "properties.types":1,
5         "properties.types_as_array":1
6     }
7 );
```

Listing 6: Accès aux tableau Types_as_array d"un document.

```
    :/* 0 */
{
    "_id" : ObjectId("54a038b2237aba92e1072616"),
    "properties" : {
        "types" : ",general-link,geoserve,nearby-cities,origin,phase-data,
            scitech-link,",
        "types_as_array" : [
            "general-link",
            "geoserve",
            "nearby-cities",
            "origin",
            "phase-data",
            "scitech-link"
        ]
    },
    "id" : "nc72001620"
}
```

Listing 7: Tableau de types et la chaine de carractère.

La question suivante conserne le nettoyage des données.
Ah zut ... Avec la remarque précédente on a grillé les étapes :(. On l'a fait quand même? Après avoir rechargé la base de données, on confirme le bon fonctionnement de la requête suivante.

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

Listing 8: Nettoyage des données.

10.1 requêtage

- Donnez le nombre de documents dont la liste de type contient "geoserves" et "tectonic-summary »

```
    db.earthquakes.find({
        "properties.types_as_array":{$all:["geoserve",
        "tectonic-summary"]
        }
    }).count()
```

le résultat retourné est 2954

Remarque: L'énoncé indique entre crochet «geoserves» avec un S, ce qui donne zéro résultats. Attention aux copier/coller.

- Ecrire une requête qui donne le nombre de tremblements de terre en Californie (Indice : RegExp)

```
    db.earthquakes.find({
        "properties.place": { $regex: /California$/, $options: "i"}
    }).count()
```

Explication: on remarque que le l'état est placée à la fin du champ properties.place. La requête précédente compte tous les document dont le champ properties.place termine par California (Sans prendre en compte les Majuscules).
le résultat retourné est 3535.

10.2 Coordonnées et dimensions

Pour cette question on proprose un code directement tapé dans l'interface RoboMongo. On rapelle que ce code peut être écrit dans un fichier JavaScript et passé en console par la commande mongo.

```
var nbDocs = db.earthquakes.find().count()
var nbComplying = db.earthquakes.find({"geometry.coordinates":{$size :3}}).
    count()
if (nbComplying ==  nbDocs) { // si tous les docs respectent le meme schema
    allDocs = db.earthquakes.find()
    var j = 0
    for (var i = 0; i < nbDocs; i++) {
        var value = allDocs[i].geometry.coordinates[2]; // recuperation de
            la valeur
        allDocs[i].depth = value; // creation du champ depth
        j = j+1;
        db.earthquakes.save(allDocs[i]); //svgarde du doc courant
    }
    print(j +" Documents ont ete modifies et inseres.");

    db.earthquakes.update({},{$pop: {"geometry.coordinates": 1}}, {multi:
        true});
}
```

Listing 9: Création du champ "depth".

Explication: Avant de supprimer le dernier élément de chaque tableau, "geometry.coordinates" on vérifie qu'ils aient tous les 3 éléments (l.4). Puis on boucle sur tous les éléments présents en leur ajoutant le champ depth. (l.9). Une fois tous les documents édités on modifie directement la collection en suppriment le dernier élément du tableau "coordinates" (l.15).

Le code retourne l'inscription : 7669 Documents ont été modifiés et insérés.

Pour vérifier le bon déroulement du script on passe la commande suivante:

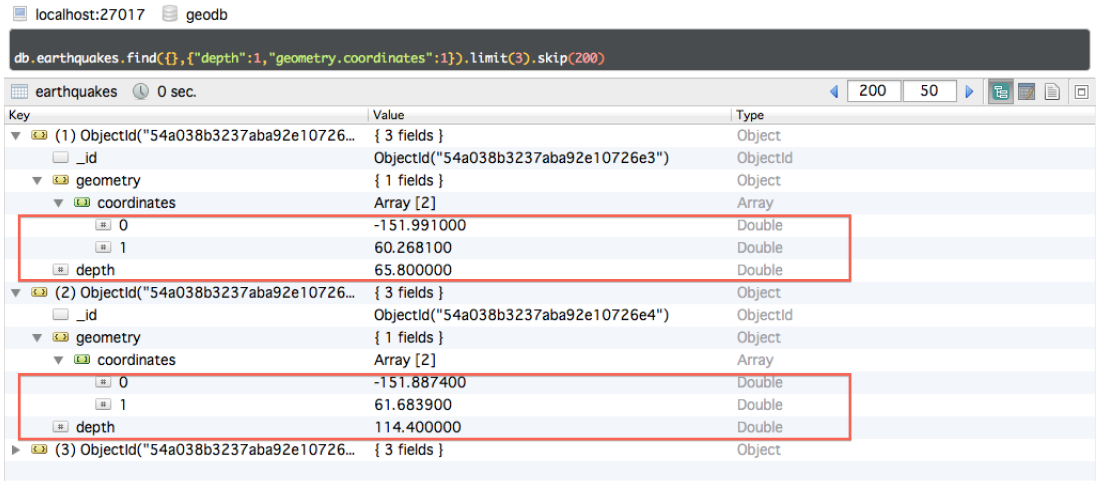


Figure 6: ...find(,"depth": 1, "geometry.coordinates": 1).limit(3).skip(200)

10.3 Indexation géographique

MongoDB propose plusieurs types d'index, dont les "géospacial Indexes". Ils permettent de gérer des informations sur plusieurs dimensions (coordonnées, distance, position...). Il y en a notament un index pour les coordonées cartésienne et un autre pour les sphériques. L'index demandé est de type 2d.

Ce type d'index prend en charge les coordonnées plannes et ne supporte pas l'utilisation d"objet GeoJSON (contrairement à 2dsphere). Il prend comme argument le champ support de l'index et les options min, max bits. Par défaut mongodb considère les valeurs commes des longitudes et latitudes, elles vont donc de -190 à 190. Si une valeur sort de se cette fourchette mongodb renvoie un erreur.

Avant de créer l'index on vérifie donc que le jeux de données respecte bien cette convention avec les reqêtes suivantes :

```
db.earthquakes.find("geometry.coordinates": $gt: 10).count()
db.earthquakes.find("geometry.coordinates": $lt: 10).count()
```

Le résultat est zero, les données sont adaptées, on ne préscise donc pas d"option.

```
1    db.earthquakes.ensureIndex({"geometry.coordinates": "2dsphere"})
```

On peut alors requêter sur ce champs en utilisant des fonctions géométriques proposées par mongodb comme: $geoIntersects, $geoWithin, $near. Exemple, exécuter une requête qui cherche les tremblements de terre proche de la position -3.984,48.724

```
1     db.earthquakes.find({
2         "geometry.coordinates":
3         {
4             $near : [-3.984, 48.724],
5             $maxDistance: 1000
6         }
7     });
```

11 Replication

11.1 un maitre avec un ou plusieur esclaves

L'une des porpriétés de mongodb est sa capacité à gérer des duplicas pour offrire une base de donnée plus persistante. Une copie des données est alors faites sur un autre server. Mongodb est un système actif/passif[6], ses différnents noeuds portent un statut maître ou esclave.

Pour simmuler les différents noeuds on utilise les ports d'une machine locale que l'on passe au démon. `mongo` tout en indiquant le statu du noeud. On récupère quelques lignes de prompte intéressantes.

- Lancement du maître:
  `mongod -master -dbpath .../data/master -port 27016`
- Premier esclace:
  `mongod -slave -source localhost:27016 -dbpath .../data/slave -port 27018`
- Deuxième esclave:
  `mongod -slave -source localhost:27016 -dbpath .../data/slave2 -port 27020`

```
mongod --master --dbpath ~/Lmdapp/data/mongodb/data/master -port 27016
2015-01-17 [initandlisten] db version v2.6.6
[ ... ]
2015-01-17 [initandlisten] waiting for connections on port 27016
[ ... ]
2015-01-17 [clientcursormon]  connections:0
2015-01-17 [initandlisten] connection accepted from 127.0.0.1:50358 #1 (1
    connection now open)
2015-01-17 [slaveTracking] build index on: local.slaves properties: { v: 1,
     key: { _id: 1 }, name: "_id_", ns: "local.slaves" }
[ ... ]
2015-01-17 [clientcursormon]  connections:1
2015-01-17 [initandlisten] connection accepted from 127.0.0.1:50731 #2 (2
    connections now open)
2015-01-17 [slaveTracking] build index on: local.slaves properties: { v: 2,
     key: { _id: 2 }, name: "_id_", ns: "local.slaves" }
[ ... ]
```

Listing 10: Prompt renvoyé par le noeud maître

[6] Attention, le model mongo est assez controversé, exemple:
Evaluation sévère de mongo sur le post: *Call me maybe: MongoDB by aphyr*
"...Mongo, un choix plus User firendly que résistant.": *Tugdual Grall, au Talk BigDataHebdo.*

La base local donne des infos sur l'intence mongodb. l'instruction db.slaves.find() nous renvoies les propriétés des esclaves stokés dans mongodb.

```
> db.slaves.find().pretty()
{
  "_id" : ObjectId("54ba60891fd9ebd9f64c30ef"),
  "config" : {
    "host" : "127.0.0.1:50358",
    "upgradeNeeded" : true
  },
  "ns" : "local.oplog.$main",
  "syncedTo" : Timestamp(1421501959, 1)
},
{
  "_id" : ObjectId("54ba6df131f0c24be9fe3bb0"),
  "config" : {
    "host" : "127.0.0.1:51059",
    "upgradeNeeded" : true
  },
  "ns" : "local.oplog.$main",
  "syncedTo" : Timestamp(1421504048, 1)
}
```

1	"host"	Nom et port du noeud esclave.
2	"upgradeNeeded"	Etat de la réplication des neuds. "False" si le neud est à jour.
3	"ns"	Collection contenant les logs liés aux réplication
4	"syncedTo"	Dernière date de sychronisation.

Pour tester la réplication, sur le noeud maître on insert un nouveau document dans une nouvelle collection d'une base encore inexistante. Mongodb créer automatiquement la base puis la réplique sur les autres noeuds. On la retrouve en se connectant aux esclaves et en appliquant la méthode show dbs.

11.2 Plusieur maîtres et un esclave

Dans cette organisation le travaille s'effectue dans le sens inverse. Au lieu de lancer des slaves avec une adresse sources et attendre l'écriture des documents local.slaves on lance les maîtres et on insert leur adresse dans la collection local.sources d'un slave: `db.sources.insert(host:"ip.a.dre.ss:port")`
Une fois insérée la connection s'effectue et les documents se complètent avec l'_id et la date du dernier update. On retrouve alors les collections des maîtres sur le neud esclave. Il est également possible de présciser dans les documents local.sources les collections à récupérer avec l"option "only".

On retrouve en suite l'adresse slave dans la collection local.slave des maîtres. Chaque document ajouté est recopié.

12 Le sharding ou partitionement

MongoDB en plus de dupliquer les données propose également de les fragmenter et les répartire sur des noeuds distincts. Il est alors possible séparer une base en méttant les collections sur des machines différentes (scaling vertical). Ou disposer d'une partie des documents de toutes les collestions sur chaque noeuds (scaling horizontale).
Pour cette partie on va créer un server de configuration et deux servers de shard:

```
mongod -port 27022 -dbpath .../data/config -configsvr
mongod -configdb localhost:27022 -port 27021 -chunkSize 1
mongod -port 27023 -dbpath .../data/shard0 -shardsvr
mongod -port 27024 -dbpath .../data/shard1 -shardsvr
```

L"oprtion –chunkSize donne une taille maximal aux collections. par défauts elles sont de 64 bits. En réduisant les tailles des collections on partionne plus rapidement.

En se connnectant au controlleur on accède à un shell mongos (mongodb Shards). Une instance mongo qui permet la configuration des shards. Après la connection à admin, et

le lancement de la commande:
`db.runCommand({ addshard:"adress:port",allowLocal:true})`
On retrouve les shards que l'on a lancé dans la collection: config.shards:

```
mongos> use config
switched to db config
mongos> db.shards.find()
{ "_id" : "shard0000", "host" : "localhost:27023" }
{ "_id" : "shard0001", "host" : "localhost:27024" }
```

Pour la suite du TP on créer la base de donnée phone et on autorise le sharding de cette base avec la commande `db.runCommande(enablesharding:"base")`. On définit en suite la fonction qui génère les documents de la bases phones.

Remarque: Pour bien comprendre la diffusions des actions depuis le controlleur on se plug au mongos via RoboMongo. En faisant un .count() on retrouve 200 000. Puis en cosole on constate l'exitance des bases et collections sur les serveurs shards. On trouve respectivement sur les ports 27023 et 27024: 94627 et 105373 docs.
(105373 + 94627 = 200 000)

Vérificatio: Les testes demandés reviennent exactement à la remarque préscédente.

13 Administration et sécurité

13.1 Backup

Dans cette partie on découvre les commandes `mongodump` et `mongorestore` qui permettent respectivement de sauvegader et rétablir une ou plusieur bases de données (celon les options). Un dump de l'instence mongo donne une collection et son system.indexes au format .bson et fichier de metta données au format .json.

Note: Le démon mongod doit être lancé pour passer ces commandes.

13.2 Authetification

Dans cette partie la démarche indiquée est dépréciée. Connection à la base de donnée admin (use admin) et utilisation de la méthode .addUser(). A la place il est conseillé de créer un document de la manière suivnate.

```
use admin
db.createUser(
  {
    user: "siteUserAdmin",
    pwd: "password",
    roles: [ { role: "userAdminAnyDatabase", db: "admin" } ]
  }
)
```

Listing 11: Création d"un user

Annexe

SQL to Mongodb, avec Node.js[7]

Une des problématiques apportées par les technos NoSQL est la conversion des anciennes base rellationnelles. Pour les petits acteurs du web qui passent sur des architechtures scalables avec des languages serveur comme node, la migration s'accompagne souvent d'un changement de base de données.

Bien que mongoDB ai son propre module node[8] je propose ici un module concurrent **mongoose** [9] qui lui nécessite la création d'un schéma. Le cas d'utilisation est simple: Si on recopie une une base SQL, il est plus simple de crée un shéma avant de la pousser dans Mongo.

Je propose donc le code suivant qui peut être lancé de la manière suivante:
`> Node bank.js` Le but est de pouvoir insérer dans mongoDB un export au format .Json d'une base MySQL, exemple:

```
{"Bid":"982","account_number":982,"balance":16511,"firstname":"Buck","
    lastname":"Robinson","age":24,...,"email":"buckrobinson@calcu.com"}
```

Listing 12: Une ligne de l'export.

```
1  mongoose = require('mongoose'); //import du module mongoose
2  fs = require('fs');             //
3
4  mongoose.connection.on('error', function(){// si il a une erreur.
5    console.log('Failed to Connect to the BDD');
6  });
7
8  mongoose.connection.on('open', function(){// a la connection
9    console.log('Connection to the BDD');
10 });
11
12 mongoose.connect('mongodb://localhost/bank'); // mongo> use bank
13
14 var count_model = mongoose.Schema({ //definition du schema
15   'Bid':String,
16   'account_number':Number,
17   'balance':Number,
18   'firstname':String,
19   'age':Number,
20   '...'
21   'state':String
22 });
23
24 //Array ---> mongoose.model.Object
25 var account = mongoose.model('account', count_model);
26 console.log('Building Schema');
27
28 fs.exists('./accounts.json', function (exists) {
29   //Si le fichier existe, L'ouvrire et Le lire
30   fs.readFile('./accounts.json', function(err, data){
31     var bank = JSON.parse(data);
32     bank.forEach(function(d){
33       var row = new account(d);
34       row.save(function (err) {
35         if(err)console.log('Inserting Error!');
36       });
37     });
38   });
39 });
```

Listing 13: SQL to MongoDb avec le module Mongoose.

[7] *http://nodejs.org*
[8] *Native Node module for MongoDB*
[9] *Mongoose: mongodb object modeling for node.js*
[10] *Retrouvez l'exemple complet **ici**.*

Map / Reduce avec Pymongo[11]

Pymongo est une dépendence python qui permet de s'adrésser à mogodb. Une fois L'instence mongo lancée L'import de ce module permet la connection et donne accès à des fonctions mongo.

```
>>> Import mongo
```

Pour faire un exemple on propose de reprendre la fonction map reduce de pymongo appliquée à la base de lapin de la partie 9. Pour lancer le script:

```
> python mapredbunny.py
```

```
1  ############################################################
2  #     EXEMPLE DE MAP / REDUCE SUR MONGODB                  #
3  ############################################################
4
5  # Import du connecteur et du traducteur bson #
6  import pymongo
7  from bson.code import Code # Traduction du code Js compris par mongodb #
8
9  # Connection a la bonne BDD #
10 db = pymongo.MongoClient().lapins
11
12 # ecriture du mappeur #
13 mapp = Code("""
14              function() {
15                  this.regime.forEach(function(r){
16                      emit(r,1);
17                  });
18              }
19              """)
20
21 # ecriture du reduceur #
22 red = Code("""
23              function(key, tab) {
24                  var sum = 0;
25                  for(var i; i < tab.length - 1; i++){
26                      sum += values[i];
27                  }
28                  return sum;
29              }
30
31              """)
32
33 # Lancement de l'agregation #
34 res = db.france.map_reduce(mapp, red, "mapred_bunny", query={"genre":"h"})
35 #
36 Le resultat est disponible
37 dans la collection "mapred_bunny"
38 #
```

La collection retour est une suite de document contenant: un nom d'alliment et le nombre de lapins mâles qui en mange. Ce n'est pas très util car ce n'est qu'un exemple. Il faut s'imaginer que la donnée est maintenant bien agrégée pour réaliser de super grafiques sur le régime des lapins de la collection france.

Sources

Le code source de ce rapport est normalement disponible sur: *https://github.com/DivLoic/TP-Mongodb.*

[11] *API du connecteur "pymongo".*