Chapitre 3

Indexation, 2DSphere & Lookup

Il est possible d'indexer des clés sous MongoDB. Nous allons voir comment les créer, et comment constater les modifications sur les plans d'exécution (explain() à partir de la version 2.6).

3.1 Indexation simple (type scalaire)

3.1.1 Pour la requête ci-dessous, regarder le plan d'exécution généré avec "`.explain()`" à la fin de la requête :

```
db.paris.find({"services" : "chambres non-fumeurs", "reviews.rating" : {$gte : 4}}).explain();
```

3.1.2 On remarquera qu'une opération de, "type" `COLSCAN` est appliqué (WinningPlan). Cela correspond à un parcours intégral de la collection.

3.1.3 Créer un index sur l'attribut année " `db.paris.createIndex({"services":1});` ";

3.1.4 Exécuter à nouveau la requête en regardant le plan d'exécution généré ;

3.1.5 Nous pourrons alors constater qu'une opération `IXSCAN` est appliquée sur la clé "*services*". Le nouvel index est utilisé. Créer un index sur la clé "reviews.rating". Vous pourrez alors constater que le plan d'exécution est identique, toutefois, un plan d'exécution a été rejeté "rejectedPlans" (celui sur le "reviews.rating")

3.1.6 Il est également possible de consulter le plan d'exécution pour les agrégats avec l'oxption (2° paramètre de "aggregate") {explain :true} :

```
db.paris.aggregate([{$match:{"services" : "chambres non-fumeurs"}},
 {$group:{_id:"$type", total : { $sum : 1}}}], {explain:true});
```

3.1.7 Pour MapReduce, il n'est pas possible de consulter le plan d'exécution. Toutefois, on peut constater l'utilisation de l'index via le nombre d'éléments en 'input' :

```
var mapFunction = function () {
        for(var i=0 ; i < this.reviews.length ; i++){
                if(this.reviews[i].rating > 4)
                        emit(this.reviews[i].language, 1);
        }
};
var reduceFunction = function (key, values) {
        return Array.sum(values);};
var queryParam = {"query":{}, "out":"result_set"};
db.paris.mapReduce(mapFunction, reduceFunction, queryParam);
```

3.1.8 Vous pourrez constater que l'ensemble des documents sont interrogés (input :15439). En effet, le prédicat "reviews" est présent dans le map. L'optimiseur de MongoDB ne peut analyser la fonction map (appliquée à chaque document), il faut pour cela utiliser le paramètre "query" du queryParam pour que l'index soit pris en compte :

```
var mapFunction = function () {
        for(var i=0 ; i < this.reviews.length ; i++){
                if(this.reviews[i].rating > 4)
                        emit(this.reviews[i].language, 1);
}
};
var reduceFunction = function (key, values) {
        return Array.sum(values);};
var queryParam = {"query":{"reviews.rating" : {$gt : 4}}, "out":"result_set"};
db.paris.mapReduce(mapFunction, reduceFunction, queryParam);
```

Seul 7909 documents sont interrogés, ceux qui nous intéresse.

3.2 Indexation 2D avec 2DSphere

Nous pouvons indexer les données avec 2DSphere qui permet de faire des recherches en 2 dimensions. Le schéma des coordonnées doit être structuré de la manière suivante [1] :

```
"location" : {
 "coord" : {
       "type" : "Point",
       "coordinates" : [
           1.53414,
           42.50729
       ]
   }
}
```

La clé de localisation dans ce document est "`location.coord`".

L'attribut `location` sera alors indexé :

```
db.paris.ensureIndex( { "location.coord" : "2dsphere" } );
```

Pour interroger l'index, il faut utiliser un opérateur 2D et l'utiliser sur "location.coord"
Documentation : `http://docs.mongodb.org/manual/tutorial/query-a-2d-index/`.

3.2.1 Tout d'abord récupérer les coordonnées de :

- "Eiffel Tower Paris France"
- "Pyramide du Louvre"
- "Boulevard Saint-Michel"

Affecter des variables tourEiffel, louvre et saintMichel à ces coordonnées pour les réutiliser par ailleurs ;
Correction :

```
db.paris.find({"name":{$in:["Eiffel Tower Paris France",
                        "Pyramide du Louvre",
                        "Boulevard Saint-Michel"]}},
              {"name":1,"location.coord.coordinates":1,"_id":0});
>>
{ "name" : "Pyramide du Louvre",
        "location" : { "coord" : { "coordinates" : [ 2.3358714580536, 48.861018076911 ] } } }
{ "name" : "Boulevard Saint-Michel",
        "location" : { "coord" : { "coordinates" : [ 2.3421263694763, 48.849368645992 ] } } }
{ "name" : "Eiffel Tower Paris France",
        "location" : { "coord" : { "coordinates" : [ 2.3516704899184, 48.857770855496 ] } } }

tourEiffel = [ 2.3516704899184, 48.857770855496 ];
louvre = [ 2.3358714580536, 48.861018076911 ];
saintMichel = [ 2.3421263694763, 48.849368645992 ];
```

3.2.2 Afficher les noms et adresses des restaurants autour de saint-Michel dans un rayon de 200m. Pour cela, utiliser la syntaxe suivant :
`{"location.coord":{$near:{$geometry:{"type":"Point","coordinates":[LAT,LONG]},$maxDistance:XXX}}}` ;
(distance en mètres)
Correction :

1. Documentation : `http://docs.mongodb.org/manual/applications/geospatial-indexes/`

```
dist = 200;
near = {$near : {$geometry : {"type" : "Point", "coordinates" : saintMichel},
                          $maxDistance : dist}};
db.paris.find({"location.coord":near, "category":"restaurant"},
        {"name":1, "location.address":1, "_id":0});
```

3.2.3 Calculer la note moyenne des restaurants de cette zone. Dans un `aggregate` il faut utiliser l'opérateur **$geoNear** :
`{"$geoNear": {"near":{"type":"Point", "coordinates" : [ LAT, LONG ]},"maxDistance":XXX, "distanceField" : "outputDistance", "spherical":true}};`
Par ailleurs il faut activer la version "3.4" : `db.adminCommand( { setFeatureCompatibilityVersion: "3.4" } )`

Correction : Il faut d'abord filtrer sur l'entourage, désimbriquer les "reviews", puis grouper par restaurant (moyenne par restaurant), puis la moyenne des restaurants.

```
var geoNear = {"near":{"type":"Point", "coordinates" : saintMichel},
        "maxDistance":dist,        "distanceField" : "outputDistance", "spherical":true};
db.paris.aggregate([
        {$geoNear: geoNear},
        {$match: {"category":"restaurant"}},
        {$unwind: "$reviews"},
        {$match: {"reviews.rating":{$gt:0}}},
        {$group:{"_id":"$_id", "avg_restaurant" : { "$avg" : "$reviews.rating"}}},
        {$group:{"_id":null, "avg" : { "$avg" : "$avg_restaurant"}}}]);
```

Attention, l'index ne peut être utilisé avec *aggregate* que si "$geoNear" est en première position dans la séquence d'opératrs.

3.2.4 idem avec la Tour Eiffel et le Louvre
Correction :

```
var geoNear = {"near":{"type":"Point", "coordinates" : tourEiffel},"maxDistance":dist,
        "distanceField" : "outputDistance", "spherical":true};
var geoNear = {"near":{"type":"Point", "coordinates" : louvre},"maxDistance":dist,
        "distanceField" : "outputDistance", "spherical":true};
```

3.2.5 Afficher le nom des points d'intérêts (poi) compris dans le triangle "tour Eiffel - Louvre - Saint-Michel" avec l'opérateur **$geoWithin** :
`{$geoWithin:{$geometry:{"type":"Polygon","coordinates": [triangle]}}}`
Le triangle est une liste de points avec retour au point de départ. La requête est une liste de polygones.
Correction :

```
var triangle = [tourEiffel, saintMichel, louvre, tourEiffel];
var polygon = {$geoWithin : {$geometry : {"type" : "Polygon", "coordinates" : [triangle]}}}
db.paris.find({"location.coord":polygon, "category":"poi"}, {"name":1, "_id":0});
```

3.2.6 Calculer le nombre de lieux par catégorie dans cette zone ;
Correction :

```
db.paris.aggregate([
   {$match : {"location.coord":polygon}},
   {"$group":{"_id":"$category", "total" : { "$sum":1}}}]);
```

3.3 $Lookup

Cette fonctionnalité est un opérateur "aggregate" disponible depuis la version 3.4, permet de faire des jointures "gauches". C'est à dire que pour chaque document *source* il vérifie l'existance de la clé de jointure dans une collection

externe. Si c'est le cas, il imbrique le(s) document(s) correspondant(s) dans une liste imbriquée dans le document source, sinon, le document n'est pas modifié (ni filtré).

Il faut pour cela préciser la clé à filtrer, puis la clé à vérifier dans la collection externe :

```
{
        $lookup: {
                "from": "<collection destination>",
                "localField": "<clé à filtrer>",
                "foreignField": "<clé à aller vérifier (provenant du from)>",
                "as": "<nom de la clé contenant la liste des documents joints>"
        }
}
```

Pour illustrer cela, nous allons créer une collection de données personnalisées. Nous allons y stocker tous vos commentaires sur les destinations que vous avez visité. Pour cela, il faudra y associer l'identifiant du lieu que vous commentez. Exemple :

```
db.comments.save({
        "user":1, "idLocation": 231445, "date":"2017-02-03",
        "comment" : "Impressive place, especially at sunset"
});
```

3.3.1 Créer 5 documents pour la collection "comments" sur 5 lieux différents (ou non).

Correction :

```
db.paris.find({"name":{$regex:/eiffel/, $options:"i"}}, {"name":1});

db.comments.save({"user":3, "idLocation": 231445, "date":"2016-12-15",
        "comment" : "Par jour de bouchon, c'est assez amusant"});
db.comments.save({"user":1, "idLocation": 215501, "date":"2015-07-12",
        "comment" : "Incontournable, mais à prendre avec coupe fil"});
db.comments.save({"user":1, "idLocation": 84005, "date":"2016-05-18",
        "comment" : "Super quand il fait beau en terasse"});
db.comments.save({"user":2, "idLocation": 84005, "date":"2016-12-15",
        "comment" : "Pas pour les gros mangeurs"});
```

3.3.2 Testons la jointure :

```
db.paris.aggregate([
        {$lookup : {
                "from" : "comments",
                "localField":"_id",
                "foreignField" : "idLocation",
                "as" : "my_comments"
        }}
]);
```

3.3.3 Quelle est la taille du résultat produit ?

Correction :

```
db.paris.aggregate([
        {$lookup : {"from" : "comments","localField":"_id",
                "foreignField" : "idLocation", "as" : "my_comments"}},
        {$group : {"_id" : null, "tot" : {$sum : 1}}}
]);
```

56361 documents en sortie, sur 56361 en entrée. Pas de changement sur le nombre de documents.

3.3.4 Quelle est la taille du résultat après filtrage sur l'existance de la clé "my_comments" ?

Correction :

```
db.paris.aggregate([
        {$lookup : {"from" : "comments","localField":"_id",
                "foreignField" : "idLocation", "as" : "my_comments"}},
        {$match : {"my_comments.0":{$exists : 1}}},
        {$group : {"_id" : null, "tot" : {$sum : 1}}}
]);
```

3 documents en sortie. Cette fois-ci, on voit bien que la collection est jointe avec "comments". Attention, tous les documents ont maintenant une clé "my_comments" (liste vide pour la plupart).

3.3.5 Pour être plus efficace dans cette opération, il est recommandé de créer un index ou de "sharder" la collection "comments" sur la clé "idLocation". Créer un index et tester à nouveau la requête.

Correction :

```
db.comments.createIndex({"idLocation":1});
```