Quentin LEBARD

TP- MongoDB

Travailler avec MongoDB

MongoDB est un système de gestion de base de données orienté documents, répartissable sur un nombre quelconque d'ordinateurs et ne nécessitant pas de schéma prédéfini des données. Il fait partie de la mouvance NoSQL.

Pour utiliser, MongoDB nous avons commencé par installer le serveur sur une machine (pour ma part je l'ai installé sur une machine virtuelle sous Ubuntu 18.04). J'ai ensuite créé un répertoire sur le bureau pour stocker les données. Pour lancer le serveur j'utilise la commande : « `./mongod` »

Nous avons utilisé l'interface utilisateur Robot3T en établissant une connexion en localhost sur le port 27017. Au préalable, j'ai importé une base de données, elle contient des documents au format «. json»

La base de données :

La base de données est composée de restaurant qui on chacun un nom un quartier, le type de cuisine, une adresse, et un ensemble de notes.

Filtre et projection

Une commande basique de filtrage pour une requête de filtrage :

```
db.restaurants.find({"borough" : "Manhattan"}).count()
```

Cette commande permet de renvoyer les restaurants dans le quartier de Manhattan. Le `.count()` renverra lui le nombre de ces documents.

Remarque : il est possible de complexifier la recherche pour affiner celle-ci par exemple : `db.restaurants.find({"borough" : "Manhattan", "cuisine" : "Irish"}).count()`

- ⇨ Récupère tous les documents de la collection qui se trouvent à Manhattan et proposent un type de cuise : Irish

Ci-dessous je vous montrer quelque exemple sur le fonctionnement de la projection avec MongoDB :

```
db.restaurants.find(
{
    "borough" : "Manhattan",
    "cuisine" : "Irish",
    "address.zipcode" : "10019" ,
```

```
    "name" : /pub/i
},

{
    "name":1,
    "_id":0
}
    )
```

Cette requête permet de renvoyer uniquement le nom des restaurants qui satisferont les critères donner. L'id unique est renvoyer par défaut, si on ne souhait pas le renvoyer il suffit de le préciser dans le deuxième paramètre de la fonction `find()`.

Pour en plus renvoyer le type de cuisine il suffit d'ajouter une virgule après l'id 0 et la ligne `"cuisine":1`

Filtrage avec des comparateurs :

Il est possible d'utiliser des opérateurs arithmétiques :

`$eq`	Matches values that are equal to a specified value.
`$gt`	Matches values that are greater than a specified value.
`$gte`	Matches values that are greater than or equal to a specified value.
`$lt`	Matches values that are less than a specified value.
`$lte`	Matches values that are less than or equal to a specified value.
`$ne`	Matches all values that are not equal to a specified value.
`$in`	Matches any of the values specified in an array.
`$nin`	Matches none of the values specified in an array.

Ci-dessous nous renvoyons tous les restaurant qui se trouve dans le quartier de Manhattan ayant un score inférieur à 5.

```
db.restaurants.find(
{
    "borough" : "Manhattan",
```

```
     "grades.score" : {
         $lt : 5,
         $not:{$gte:5}

         }
},
{
    "name":1, "_id":0, "grades.score":1
}
)
```

Attention, si nous souhaitons avoir plusieurs conditions qui combine comme par exemple ci-dessous, nous souhaitons renvoyer tous les restaurant dont le grade est A et des score inférieur à 5. A ce moment il faut utiliser le `$elemMatch :`

```
db.restaurants.find(
{
    "grades" :
    {
        $elemMatch :
        {
         "grade" : "A",
          "score" :
          {
           $lt:5
          }
        }
    }
},
{
   "grades.grade":1, "grades.score":1 , "_id":0
}

)
```

Pour renvoyer tous les restaurants dont le dernier score est B nous utilisons la fonction `findOne()`, comme ci-dessous.

```
db.restaurants.findOne(
{
    "grades.0.grade": "B"
},
{
    "name" : 1, "grades.grade":1
}
)
```

Pour rechercher sur un champ et retourner toutes les valeur disponible nous utilisons

la commande `distinct()`

```
db.restaurants.distinct("cuisine")
db.restaurants.distinct("borough")
db.restaurants.distinct("grades.score")
```

Agrégation

Nous allons utiliser la fonction `aggregate()` pour des manipulation plus complexes. Cette fonction va pouvoir prendre en paramètre plusieurs opérateur. Les principaux opérateurs sont : `{$match : {} }` , `{$project : {} }` et `{$sort : {} }`

Nous allons comparer les 2 requêtes :

```
db.restaurants.find({
      "grades.0.grade":"B"
},
{"name":1, "_id":0}
);
```

=>

```
db.restaurants.find({
"grades.0.grade":"B"
},
{"name":1, "_id":0}
);
```

On peut constater que la structure est différente, ici nous avons dans un document les opérateurs et la valeur dans des documents.

Mise à jour

Ici nous allons mettre à jour des documents avec la fonction `update()`, avec comme paramètre `$set :{}`, la commande ci-dessous permettra de mettre à jour le champ cuisine du restaurant ayant pour ID `5fc4cc48c914b0ee370fbc0e.`

```
bb.restaurants.update(
{
    "_id" : ObjectId("5fc4cc48c914b0ee370fbc0e")
},

{
    $set : {"cuisine" : "Lorraine"}
}
);
```

Ici nous allons tous d'abord ajouter un champs tester et ensuite le retirer avec le `$unset : {}`

```
db.restaurants.update(
{
    "_id" : ObjectId("5fc4cc48c914b0ee370fbc0e")
},

{
    $set : {"tester" : "Lorraine"}
```

```
}
);

db.restaurants.update(
{
    "_id" : ObjectId("5fc4cc48c914b0ee370fbc0e")
 },
{
        $unset : {"tester":1}
}
);
```