Big Data

Requêtes MongoDB

Thomas Bourgeois

03/12/20

Table des matières

I - Introduction à MongoDB

MongoDB est un logiciel de gestion de base de données orienté documents, faisant partie de la mouvance NoSQL. Cet outil a été développé depuis 2007 par l'entreprise du même nom. Ce système est écrit en C++, possède un serveur et des outils sous licence open source SSPL (Server Side Public Licence) et des pilotes sous Apache.

MongoDB est devenu l'un des SGBD (Système de Gestion de Base de Données) les plus utilisés, notamment par eBay, le New York Times etc.

Il est possible de manipuler les données d'une BDD MongoDB à l'aide de plusieurs langages de programmation. En effet, le logiciel est compatible avec les langages C, C++, Java, Python, PHP, etc. Il est possible de les manipuler en ligne de commandes mais également via des applications, notamment Robo3T, qui est une application cliente de ce genre de SGBD.

NoSQL est une famille de bases de données, qui s'écartent de la façon conventionnelle de gérer des bases de données, souvent relationnelles.

II - Queries and projection operators

Voici un lien utile vers la documentation concernant les requêtes pour manipuler une base de données MongoDB : https://docs.mongodb.com/v3.2/reference/operator/query/

Ce logiciel fonctionne en mode Client/Serveur. Ainsi, pour démarrer le serveur, il faut utiliser la commande : **./mongodb**

De même, pour lancer un client, on doit utiliser la commande : **./mong** (sans le d qui signifie daemon).

Comme dit précédemment, Robo3T est un client de base de données, tout comme PhpMyAdmin.

Afin de créer une nouvelle BDD et d'y importer des données contenues dans un fichier, il faut utiliser cette commande : **./mongoimport -d « le nom de la base» -c « nom de la collection : équivalent d'une table en MySQL » --file « fichier de données à importer »**

db.restaurants.find() : permet de renvoyer tous les documents d'une collection
db.restaurants.findOne() : permet de renvoyer un seul document contrairement à find()

III - Filtre et projection

Afin de faire des filtres de recherche, il faut utiliser une requête .find() ou .findOne(), et leur ajouter un pattern en langage JSON). Un pattern est un patron utilisé pour transformer une expression dans un autre langage.

Exemple: **db.restaurants.find({"borough" : "Manhattan"}).count() :**
On passe un paramètre (pattern) à la méthode find, ici **{"borough" : "Manhattan"}** afin de faire une recherche dans la BDD restaurants, de trouver uniquement les documents qui satisfont le pattern, et de nous renvoyer leur nombre grâce à **.count()**.

Remarque : le pattern peut être un objet complexe (il peut s'agir de n'importe quel document JSON) :
db.restaurants.find({"borough" : "Manhattan", "cuisine" : "Irish"}).
Ici on récupère tous les documents de la collection qui se trouvent à Manhattan et proposent un type de cuisine : Irish.

La commande suivante permet de renvoyer tous les restaurants du quartier Manhattan, qui propose la cuisine Irish situé à l'adresse 10019 et qui contiennent le mot pub (insensible à la casse)

```
db.restaurants.find({
   "borough" : "Manhattan",
   "cuisine" : "Irish",
   "address.zipcode" : "10019" ,
   "name" : /pub/i
   })
```

Quelques exemples sur le fonctionnement de la projection avec MongoDB :

Requête	Description

db.restaurants.find(
{
"borough" : "Manhattan",
"cuisine" : "Irish",
"address.zipcode" : "10019" ,
"name" : /pub/i
},
 	
{
"name":1,
"_id":0
}
)
	La requête suivante permet de ne renvoyer que le nom des restaurants qui satisfont les autres critères suivants : quartier Manhattan, cuisine irish, addresse 10019.
	
	Remarque : l'id unique est renvoyé par défaut. Si on veut ne pas le renvoyer, on le précise dans le deuxième paramètre de la fonction find comme suit

db.restaurants.find({ "borough" : "Manhattan", "cuisine" : "Irish", "address.zipcode" : "10019" , "name" : /pub/i }, { "name":1, "_id":0, "cuisine":1 })	Même requête que précédemment mais qui renvoie en plus du nom le type de cuisine

IV - Filtrage avec des comparateurs

Requête	Description

db.restaurants.find(
{
"borough" : "Manhattan",
"grades.score" : {
$lt : 5,
$not:{$gte:5}
 	
}
},
{
"name":1, "_id":0, "grades.score":1
}
)
	Renvoie les noms et score des restaurants de Manhattan ayants un score <5. Sans $not:{$gte:5}, dès qu'un restaurant a un score inférieur à 5, la requête nous ramène tous les autres score.

db.restaurants.find(
{
"grades" :
{
$elemMatch :
{
"grade" : "A",
"score" :
{
$lt:5
}
}
}
},
{
"grades.grade":1, "grades.score":1 , "_id":0
}
)
	Renvoie tous les restaurants dont le grade est A et le score est inférieur à 5.

db.restaurants.findOne(
{
"grades.0.grade": "B"
},
{
"name" : 1, "grades.grade":1
}
)
	Renvoyer tous les restaurants dont le dernier grade est B.

On peut également effectuer des recherches sur un champ particulier et renvoyer toutes les valeurs disponibles lui correspondant. Par exemple : **db.restaurants.distinct("cuisine")** permet de renvoyer tous les types de cuisine disponibles.

V - Agrégations

$match : permet de correspondre à des documents

$project : permet de remplir les valeurs de champs spécifiques

$group : permet de regrouper les documents par champs spécifiques

$sort : permet de trier

$set : permet de modifier des valeurs

Exemple 1 :

```
db.restaurants.aggregate([
{ $match :{
    "grades.0.grade" : "B"
}},
{ $project : {
    "name":1
}}
])
Ici, c'est l'équivalent de db.restaurants.find() :
match = {$match : {"grades.0.grade" : "B"}};// db.restaurants.find()
project = {$project : {"_id":0} }; // db.restaurants.find({}, {}), deux paramètres avec projection ....

db.restaurants.aggregate([match, project]); // aggrégation ....

sort = {$sort : {"address.zipcode":1}};

db.restaurants.aggregate([match, project, sort]);   //  comand_1 | commande_2
```

Exemple 2 : Ecrire une requête qui permet de renvoyer le nombre de restaurants ayant dans la dernière évaluation un B

```
match = {$match : {"grades.0.grade" : "B"}}; // db.restaurants.find() ....
//match_1 = {$match : {"cuisine" : "Italian"}};
project = {$project : {"_id":0} }; // db.restaurants.find({}, {}), deux paramètres avec projection ....
db.restaurants.aggregate([match, project]); // aggrégation ....
sort = {$sort : {"address.zipcode":1}};
db.restaurants.aggregate([match_1, project, sort]);   //  comand_1 | commande_2

// sans utiliser le $group ....
db.restaurants.count({"grades.0.grade":"B"});  // appliquer un filtre
// écrire la requête autrement ....
db.restaurants.find({"grades.0.grade":"B"}).count();
```

Exemple 3 : Nombre total de restaurants dans un quartier

```
match_c = {$match:{}};

groupe = {$group : {"_id":null,  "somme": {$sum: 1} }};

db.restaurants.aggregate([match_c, groupe]);
```

Exemple de mise à jour d'une valeur dans un champ de la base de données :

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

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

On peut également importer deux collections et créer une jointure entre les deux