CRUD create/read/update/delete avec MongoDB avec la base de données ny

Création d'une base de données

Pour créer une nouvelle base de données taper la ligne de commande suivante :

```
mongo
> use school
```

Si vous avez installé Robo 3T vous pouvez également créer la base de données à l'aide de cet outils.

Insertion de données

Créez la collection authors dans la base de données school

```
db.createCollection("authors")
```

Insérer une donnée ou plusieurs données en même temps à l'aide de la méthode **insert** :

```
// Un seul document
db.authors.insert(
   {
    "name": "Alan",
    "grade": "master 5",
     "notes": [11, 20,18,19],
      "status": "A++"
    }
)

// Plusieurs dans un tableau
db.authors.insert([
   {
    "name": "Alan",
    "grade": "master 5",
     "notes": [11, 20,18,19],
      "status": "A++"
    },
    {
    "name": "Alice",
    "grade": "master 4",
     "notes": [11, 17,19, 13],
      "status": "A+"
    },
]
)
```

Remarques : si on ne précise pas de propriété _id dans le document, il sera automatiquement créé. Celui-ci est de type ObjectId (voir ci-après pour sa définition précise).

Les méthodes **insertMany** et **insertOne** permettent respectivement d'insérer plusieurs ou un document unique.

Création d'un document avec un ObjectId :

```
db.authors.insert(
    { "_id" : ObjectId("5063114bd386d8fadbd6b004"), "name" : "Naoudi", "grade" : "master 5"
)
```

Vous pouvez créer votre propre _id avec une valeur de type scalaire (non mutable).

- Précisions sur l'objet ObjectId

Il est codé sur 12 bytes :

- 4 bytes représentant le timestamp courant (nombre de secondes depuis epoch, naissance d'UNIX).
- 3 bytes pour idenfitication de la machine.
- 2 bytes pour représenter l'identifiant du processus.
- 3 bytes qui représentent un compteur qui démarre à un numéro aléatoire.

Tapez les lignes de code suivantes :

```
const _id = ObjectId()
print(_id)
// ObjectId("5eef0c14591a8edc333898dd")
print(_id.getTimestamp())
// Sun Jun 21 2020 09:28:20 GMT+0200 (CEST)
```

La méthode insertMany :

```
try {
   db.authors.insertMany( [
     {
    "name": "Alan",
    "grade": "master 5",
     "notes": [11, 20,18,19],
      "status": "A++"
    },
    {
    "name": "Alice",
    "grade": "master 3",
     "notes": [20, 18, 11, 13],
      "status": "A+"
    }
   ] );
```

```
} catch (e) {
   print(e);
}
```

Méthode insertOne :

```
db.authors.insertOne( { name: "Bernard", grade: "professor" } );
```

Méthode find lecture des données

Installez les données restaurants

Récupérez les données dans un dossier **DataExamples** :

https://raw.githubusercontent.com/mongodb/docs-assets/primer-dataset/primer-dataset.json

Dans le dossier **DataExamples** et dans un terminal ouvert dans ce dossier tapez la ligne de commande ci-dessous :

–db pour donner un nom à votre base de données. –collection indique le nom de votre collection –file indique le nom du fichier à intégrer dans la base de données –drop supprimera au préalable les collections existantes.

```
# Import de données csv dans une base de données que l'on va créer train
mongoimport --db ny --collection restaurants --file primer-dataset.json --drop
```

Vérifiez que vos données sont bien importées :

```
show dbs

use ny

show collections
restaurants

db.restaurants.count()
```

Pour faire une sauvegarde d'une collection au format BJSON tapez la ligne de commande suivante, la sauvegarde se fera dans un dossier dump, dans le dossier où votre terminal a été ouvert.

```
mongodump --collection restaurants --db ny
```

L'instruction suivante correspond à un SELECT * FROM restaurants en SQL :

```
db.restaurants.find( {} )
```

En SQL on peut faire des sélections précises à l'aide d'une restriction partie WHERE :

```
SELECT *
FROM restaurants
WHERE cuisine = "Delicatessen";
```

En MongoDB cela donnerait :

```
db.restaurants.find( { cuisine: "Delicatessen" } )
```

Plus généralement la structure de la méthode find ressemble à :

```
db.collection.findOne(restriction, projection)
```

Par exemple on sélectionne les restaurants qui font de la cuisine Delicatessen en affichant que les champs : cuisine et address :

```
db.restaurants.find({ cuisine: "Delicatessen" }, {_id : 0, cuisine : 1, address : 1}).pretty
```

Opérateur IN

Vous pouvez également utiliser les query operators comme dans l'exemple suivant, ici on cherche à sélectionner les types de cuisines Delicatessen ou American dans la collection restaurants

```
db.restaurants.find( { cuisine: { $in : [ "Delicatessen", "American" ] } } )
```

Cet opérateur est similaire à l'opérateur IN de SQL.

Opérateurs AND et OR

- On peut également utiliser un opérateur logique ET comme suit :

```
db.restaurants.find( { "borough" : "Brooklyn", "cuisine" : "Hamburgers" } )

// De manière équivalente
db.restaurants.find( { $and : [ { "borough" : "Brookyn"}, { "cuisine" : "Hamburgers" } ] } )
```

- Syntaxe de l'opérateur or :

```
// { $or: [ { <expression1> }, { <expression2> }, ... , { <expressionN> } ] }

// Exemple sur la table authors
db.authors.find( { $or: [ { name: "Alan" }, { name: "Alice" } ] } )
```

Voici un exemple de condition logique en utilisant OR et AND. Remarquez le deuxième argument de la méthode find, il permet de faire une projection, c'est-à-dire de sélectionner uniquement certaine(s) propriété(s) du document :

```
db.restaurants.find( {
     borough: "Brooklyn",
     $or: [ { name: /^B/ }, { name : /^W/} ]
}, {"name" : 1, "borough" : 1} )
```

Cela correspondrait (...) en SQL à la requête suivante :

```
SELECT
`name`,
borough
FROM restaurants
WHERE borough = "Brooklyn"
AND ( `name` LIKE '/^B/' OR `name` LIKE '/^W/')
```

1. Exercice compter le nombre de restaurants

Sans utiliser la méthode count dans un premier temps comptez le nombre de restaurants dans le quartier de Brooklyn.

Pour itérer sur une requête vous utiliserez l'une des deux syntaxes suivantes :

```
// 1
db.collection.find().forEach(doc => print(tojson(doc)))

// 2
const myCursor = db.users.find( restriction );
while (myCursor.hasNext()) {
   print(tojson(myCursor.next()));
}
```

Puis comparez le résultat avec la méthode count :

```
db.collection.findOne(query, restriction).count()
```

2. Exercices sur la notion de filtrage

Exemple de filtres classiques :

```
// plus grand que
$gt, $gte

// Plus petit que
$lt, $lte

// collection inventory  quantité < 10
db.inventory.find( { quantity : { $lt: 20 } } )
```

D'autres filtres :

```
// différent de
$ne
"number" : {"$ne" : 10}

// fait partie de ...
$in, $nin
```

```
"notes" : {"$in" : [10, 12, 15, 18] }
"notes" : {"$nin" : [10, 12, 15, 18] }

// Ou
$or
"notes : { "$or": [{"$gt" : 10}, {"$lt" : 5} ] }
// and
$and

"notes : { "$and": [{"$gt" : 10}, {"$lt" : 5} ] }

// négation
$not
"notes" : {"$not" : {"$lt" : 10} }

// existe
$exists
"notes" : {"$exists" : true}

// tous les documents qui possède(nt) la propriété level
db.inventory.find( { level : { $exists: true } } )

// tous les documents qui ne possède(nt) pas la propriété level
db.inventory.find( { level : { $exists: false } } )

// test sur la taille d'une liste
$size
"notes" : {"$size" : 4}

// element match

/*
{
    "content" : [
        { "name" : <string>, year: <number>, by: <string> }
        ...
    ]
}
*/


{ "content": { $elemMatch: { "name": "Turing Award", "year": { $gt: 1980 } } } }

// recherche avec une Regex
$regex
{ "name": { $regex: /^A/ } }
```

- 1. Combien y a t il de restaurants qui font de la cuisine italienne et qui ont eu un score de 10 au moins ? Affichez également le nom, les scores et les coordonnées GPS de ces restaurants. Ordonnez les résultats par ordre décroissant sur les noms des restaurants.

Remarque pour la dernière partie de la question utilisez la méthode sort :

```
db.collection.findOne(query, restriction).sort({ key : 1 }) // 1 pour ordre croissant et -1
```

- 2. Quels sont les restaurants qui ont un grade A avec un score supérieur ou égal à 20 ? Affichez uniquement les noms et ordonnez les par ordre décroissant. Affichez le nombre de résultat.

Remarque pour la dernière partie de la question utilisez la méthode count :

```
db.collection.findOne(query, restriction).count()
```

- 3. A l'aide de la méthode distinct trouvez tous les quartiers distincts de NY.
- 4. Trouvez tous les types de restaurants dans le quartiers du Bronx. Vous pouvez là encore utiliser distinct et un deuxième paramètre pour préciser sur quel ensemble vous voulez appliquer cette close :

```
db.restaurants.distinct('field', {"key" : "value" })
```

- 5. Sélectionnez les restaurants dont le grade est A ou B dans le Bronx.
- 6. Même question mais, on aimerait récupérer les restaurants qui ont eu à la dernière inspection un A ou B. Vous pouvez utilisez la notion d'indice sur la clé grade :

```
"grades.2.grade"
```

- 7. Sélectionnez maintenant tous les restaurants qui ont le mot "Coffee" ou "coffee" dans la propriété name du document. Puis, même question mais uniquement dans le quartier du Bronx.
- 8. Trouvez tous les restaurants avec les mots Coffee ou Restaurant et qui ne contiennent pas le mot Starbucks. Puis, même question mais uniquement dans le quartier du Bronx.
- 9. Nouvelle question : Trouvez tous les restaurants qui ont dans leur nom le mot clé coffee, qui sont dans le bronx ou dans Brooklyn, qui ont eu exactement 4 appréciations (grades), qui ont eu au moins un A en dernière notation et qui ont été évalués à une date supérieur ou égale à 2012-10-24 mais pas avant.
- Affichez tous les noms de ces restaurants en majuscule avec leur dernière date et permière date d'évaluation.
- Précisez également le quartier dans lequel ce restaurent se trouve.

Recherche de restaurents à proximité d'un lieu

MongoDB permet de gérér des points GPS. Dans la collection restaurants nous avons un champ grades.coord qui correspond à des coordonnées GPS (longitude & latitude).

Nous allons utiliser les coordonnées sphériques de MongoDB. Pour l'implémenter dans la collection vous devez créer un index particulier sur le champ coord :

```
db.restaurants.createIndex({ "address.coord" : "2dsphere" })
```

Trouvez tous les restaurants qui sont à 5 miles autour du point GPS suivant, donnez leurs noms, leur quartier ainsi que les coordonnées GPS :

```
const coordinate = [
  -73.961704,
  40.662942
];
```

Vous utiliserez la syntaxe suivante avec les opérateurs MongoDB :

```
{ $nearSphere: { $geometry: { type: "Point", coordinates: coordinate }, $maxDistance: VOTRE_
```

Recherche par rapport à la date

Exécutez la requête suivante, qu'affiche-t-elle ?

```
db.restaurants.find({
    "grades.0.date" : ISODate("2013-12-30T00:00:00Z")

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

Lire un document entièrement

La méthode find permet de lire les documents dans une collection, par défaut elle ne retournera que 20 documents au maximum.

```
db.restaurants.find()
```

Dans le terminal vous pouvez utiliser la commande it pour avancer dans la lecture du document.

- Utilisation d'un curseur pour lire le document :

```
const resCursor1 = db.restaurants.find();

while (resCursor1.hasNext()) {
   print(tojson(resCursor1.next()));
}
```

Avec la méthode **foreEach** :

```
const resCursor2 =  db.restaurants.find();

resCursor2.forEach(printjson);
```

Vous pouvez également récupérez l'ensemble des documents dans un array :

```
const resCursor3 = db.restaurants.find();
const resArray = resCursor3.toArray();
print( resArray[3].name );
```