TP MongoDB dans l'instance Openstack

L'objectif de ce TP est d'apprendre à utiliser un serveur de bases de données PaaS, MongoDB, que vous allez installer dans une instance Openstack.

Ce TP est noté et son compte-rendu avec la réponse aux questions, requêtes et les résultats respectifs (vous pouvez faire des copies d'écran de ces derniers) est à envoyer par email à l'enseignante **jusqu'au 6 avril 2019 à 20h**. Le titre du fichier à envoyer à l'enseignante doit contenir le nom des étudiants. Ce TP doit être réalisé en totale autonomie par les étudiants et peut être réalisé en binôme. Les comptes-rendus copiés entre étudiants ou binômes d'étudiants **seront sanctionnés, à la limite par un zéro**.

Documentation de MongoDB : http://docs.mongodb.org/manual/

1. *Étapes préliminaires*

Pour pouvoir communiquer avec le serveur MongoDB, vous devez ajouter une règle qui ouvre le port 27017 dans le groupe de sécurité de votre instance, dans l'interface Horizon d'Openstack. Cette règle n'est pas prédéfinie, vous devez sélectionner Custom TCP rule dans le champ Rule et CIDR dans le champs Distant.

Vous allez ensuite installer mongodb dans votre instance virtuelle.

1.1 Quelle est la commande à utiliser pour installer mongodb (en cas d'oubli, reportez-vous à l'énoncé du TP précédent) ?

1.2 Vérifiez que mongobd est bien installé. Quelle est la commande que vous pouvez utiliser ?

1.3 Créez un répertoire, dans votre instance virtuelle, qui va contenir les fichiers du serveur de votre base de données mongodb :

```
mkdir -p ~/db
```

2. *Lancer et terminer le serveur mongodb*

Pour lancer une instance du serveur mongodb, tapez dans l'invite de commandes de votre instance Openstack la commande suivante :

```
mongod --smallfiles --dbpath ~/db --port 27017
```

Après un temps, si tout se passe bien vous devez avoir le retour suivant :

```
<date>[initandlisten] waiting for connections on port 27017
```

Ce terminal doit rester ouvert, c'est celui du serveur qui doit toujours tourner lorsque que vous voulez interagir avec mongodb.

Notez qu'en fin de séance vous devez arrêter le serveur mongodb. Pour cela, faites les commandes suivantes (dans une autre invite de commandes, connecté au serveur mongodb, voir section 3.1) :

```
> use admin
switched to db admin
> db.shutdownServer()
```

3. *Connecter et interagir avec une instance mongodb*

3.1 Dans une autre invite de commandes connecté à l'instance openstack, connectez-vous à l'instance mongodb :

```
mongo --host localhost:27017
```

3.2 Créez la base de données **mydb**. Cette commande sert aussi à se connecter à cette base de données lorsqu'elle existe déjà :

```
use mydb                # Create the DB if not exists
```

Pour vérifier l'existence de la base de données :

```
show dbs
```

3.3 Créez les structures pour contenir les collections users et movies :

```
db.createCollection("users")

db.createCollection("movies")
```

Vérifiez l'existence de ces structures :

```
show collections
```

3.4 Dans le drive vous trouvez les fichiers users.json et movies.json. Profitez pour ouvrir ces fichiers dans un éditeur texte et comprendre la structure des données de ces fichiers, en comparant avec ce qui a été vu en cours.
Pour insérer des données dans les collections vous allez copier ces fichiers vers votre instance virtuelle.

Ensuite, dans une nouvelle invite de commandes de l'instance Openstack (pas celle de l'instance mongodb ouverte dans la section 3.1), vous allez importer ces fichiers vers votre base de données.
Dans l'exemple ci-dessous, on importe les fichiers users.json et movies.json depuis le répertoire `files` de l'instance Openstack vers la base de données `mydb`. A vous de modifier cette commande pour l'adapter à une autre situation.

Lorsque l'import est terminé, **fermez cette invite de comandes.**

```
mongoimport --host localhost:27017 --db mydb --collection users --file ~/files/users.json

mongoimport --host localhost:27017 --db mydb --collection movies --file ~/files/movies.json
```

Combien d'objets ont été importés pour chacune des collections ?

Maintenant, retournez à l'invite de commandes connecté à l'instance de mongodb (celle crée dans la section 3.1) et vérifiez l'existence des données dans la base, en utilisant quelques requêtes :

```
db.movies.find()        # Equivalent to SELECT * FROM mydb.movies

db.movies.count()

db.users.find()

help                    # Show information about the most important commands
```

À partir d'ici l'énoncé de ce TP est en anglais.

4. Database schema

Documents within a collection do not necessarily follow the same schema. However, in the example of the json files used for this work documents of each collection have a similar data structure. We give an example of each file content hereafter.

The collection movies contains information about the movies, including their id, title (which includes the year of production) and genres. This is not the best representation, as you may discover while answering the questions. In section 8 you will take care of this.

```
> db.movies.findOne();
{
        "_id" : 1,
        "title" : "Toy Story (1995)",
        "genres" : "Animation|Children's|Comedy"
}
```

The collection users contains information about the users and the ratings they have given to the films. User information includes id, name, age, occupation and sex. The ratings are included as an array of embedded documents, each one having a movie id (which references one of the movies in the other collection), a rating and a timestamp which indicates when the rating was included). Note that the timestamp does not follow JavaScript standard, as it indicates the number of seconds since the Unix epoch, instead of the number of milliseconds as in Javascript. You will modify this representation in section 8.

4.1 Explain what the following query does.

```
> db.users.findOne({},{movies : {$slice : 2}});


# results:

{
          "_id" : 6040,
          "name" : "Barry Erin",
          "gender" : "M",
          "age" : 32,
          "occupation" : "doctor/health care",
          "movies" : [
                 {
                        "movieid" : 573,
                        "rating" : 4,
                        "timestamp" : 956704056
                 },
                 {
                        "movieid" : 589,
                        "rating" : 4,
                        "timestamp" : 956704996
                 }
          ]
}
```

5. *Basic recall*

MongoDB is a NoSQL, document-oriented database system. As such, it moves away from traditional relational databases. A document-oriented database consists of a set of collections which, in turn, store documents with a dynamic, not-predefined schema.
A document is an ordered set of field-value pairs, where fields are strings and values can be any of the defined data types (null, boolean, number, string, date, regular expression, array, object id, binary data and code), or another document itself, called embedded document. Every document has a special key `_id` which is unique in the collection. However, documents within a collection may follow different schemas, i.e., consist of different sets of field-value pairs.
A comparison between MongoDB and SQL-based databases is available at https://docs.mongodb.com/manual/reference/sql-comparison/

The MongoDB shell provided by the MongoDB instance is in fact a full-featured JavaScript interpreter, where you can run arbitrary JavaScript code.

You can explore the documentation about the MongoDB database commands at https://docs.mongodb.com/manual/reference/command/

6. *Basic Operations*

6.1. *Read queries*

6.1.1

How many movies are there in the database?
Ref: http://docs.mongodb.org/manual/reference/command/count/

6.1.2

How many users are there in the database of the feminine genre?

6.1.3

Which is the occupation of *Clifford Johnathan*? In the answer show only his name and occupation.
Ref: http://docs.mongodb.org/manual/reference/method/db.collection.find/

6.1.4

How many users are there between 18 **and** 30 years old (both included)?

6.1.5

How many users are artist or scientist?

6.1.6

Who are the 10 oldest woman writers?

6.1.7

Show all the distinct occupations in the database.

6.2 *Inserts, Updates and Deletes*

6.2.1

Insert yourself in the `users` collection, with your name, gender and occupation. Do not include the field movies. Do not worry about privacy, you are not obliged to use real information.

6.2.2

Select one movie from the collection `movies` and update your entry by adding the field movies, which is an array, including a review of the film.
Hint: You can use the current timestamp:

`Math.round(new Date().getTime() / 1000).`
Hint: You can use the db.collection.update() method to update a document.
https://docs.mongodb.com/manual/reference/method/db.collection.update

6.2.3

Remove your entry from the collection. Do not worry if you accidentally removed other entries from the collection `users`. You can always recreate the original one by removing all its documents and reimporting them with the command mongoimport.

6.2.4

Change in all users the occupation programmer by developer.

7. Regular Expressions

7.1

How many movies have been released in the 80s?
Hint: The titles include the released date in parentheses.
Ref: http://docs.mongodb.org/manual/reference/operator/query/regex/

7.2

How many movies have been released between 1984 and 1992?

7.3

How many horror movies are there?

7.4

How many movies are **both** *Musical* **and** *Romance*?

8. Cursor ForEach

8.1

As you have just seen, having the year integrated in the title name is not very practical. Modify the collection movies by creating a new field called `year` and removing it from the title.
Ref: http://docs.mongodb.org/manual/reference/method/cursor.forEach/
Hint: Hint: You can use the `db.collection.update()` method to update a document.
https://docs.mongodb.com/manual/reference/method/db.collection.update

8.2

Modify the collection `movies` by replacing the string field genres with an array that contains all the genres.
Hint: You can use the `db.collection.update()` method to update a document.

8.3

Modify the collection `users` to create a new field called `date` of type Date, with the data values from each field timestamp.
Hint: Field timestamp in the original dataset is in seconds since the Unix epoch, but dates in Javascript can be created by using the number of milliseconds since the Unix epoch.

After executing your query, try

```
db.users.findOne()
```

It should return, for instance

```
{
    "_id" : 6038,
    "name" : "Yaeko Hassan",
    "gender" : "F",
    "age" : 95,
    "occupation" : "academic/educator",
    "movies" : [
           {
                  "movieid" : 1419,
                  "rating" : 4,
                  "timestamp" : 956714815,
                  "date" : ISODate("2000-04-26T02:06:55Z")
           },
…
}
```

9. Queries on Arrays

9.1 Read Queries

9.1.1

How many users have seen the movie with id 1196 (*Star Wars: Episode V - The Empire Strikes Back (1980)*)?

9.1.2

How many users have seen all the movies from the old Star Wars trilogy (IV,V,VI) with ids: 260, 1196, 1210?
Ref: http://docs.mongodb.org/manual/reference/operator/query/all/
You can use the `$elemMatch` operator, which matches documents that contain an array field with at least one element that matches all the specified query criteria.

9.1.3

How many users have rated exactly 48 movies?
Ref: http://docs.mongodb.org/manual/reference/operator/query/size/

The `$size` operator can only match exact number (48 movies in this query). Selecting users that have rated more than a specified number of movies has to be done in two steps which will be the subject of the following questions.

9.1.4
For each user create a field **`num_ratings`** that display the number of ratings he has given.

9.1.5
How many users have rated more than 90 movies?

9.1.6
How many ratings have been submitted after 1st January 2001?

9.1.7
Which are the last 5 films rated by *Jayson Brad*. Do not use the date but the order in which they have been added to the array **`movies`** (assume that ratings are added to the end).

9.1.8
Return only the information about *Tracy Edward* regarding the rating of the film *Star Wars: Episode VI - Return of the Jedi* (whose id is 1210).

9.1.9
Find how many users have rated the movie « *Untouchables, The* » (whose id is 2194) with a note of 5.

9.2 Update Queries

9.2.1
Barry Erin has just seen the movie Nixon with id 14. Add the movie with a rating of 4 in the array movies of Barry Erin. Recall that the field **`num_ratings`** should reflect the size of the movies' array.
Hint: http://docs.mongodb.org/manual/reference/method/db.collection.update

9.2.2
Marquis Billie does not assume having seen the movie *Santa with Muscles* (whose id is 1311). For her dignity, please remove this movie from her movies' list.

9.2.3
The genres for the film *Cinderella* should be *Animation*, *Children's* and *Musical*. Modify the document so that it has the three genres with no duplicates in just one query.