TP1 - Prise en main de la base de données NoSQL MongoDB

L'objectif de ce TP est de mettre en place une solution de stockage NoSQL via MongoDB et de manipuler les commandes de base (CRUD). L'accès au serveur se fera à travers Mongoshell.

Avant de commencer…

Il faut se connecter en ssh aux serveur gpueX (demander la valeur de X à l'enseignant) via le serveur skinner. Le serveur MongoDB tourne en continu sur la machine **node1** et attend les connections de vos clients, que vous lancerez ainsi depuis les machines **gpueX**.

Le jeu de données dataset.json est la base de données de type documents que vous allez manipuler avec MongoDB. Elle est normalement disponible dans votre répertoire. Si ce n'est pas le cas, elle est également disponible sur l'UMtice, il suffit de la télécharger puis de l'envoyer sur gpueX via scp.

Une très bonne documentation est disponible sur le site de MongoDB :
https://docs.mongodb.org/getting-started/shell/client/
Le client mongo est vraiment très simple à utiliser. Les commandes s'effectuent en JavaScript et les données sont en BSON (Binary JSON), mais s'affichent en JSON.

Attention, toutes les commandes doivent être copiées dans un fichier MonNom_Mongoshell.txt.

Premier pas avec MongoDB

1. Dans un shell Unix, créer la base de données **myDB_login** et y insérer une collection **restaurants** qui contiendra tous les documents du fichier dataset.json via la commande ci-dessous

```
Documentation Mongodb - Exemple

mongoimport --db <dbName> --collection <collectionName> --file <file.json>
–host node1
```

Connection via MongoShell

Le reste du TP se déroule dans un Shell Mongo. Le moyen le plus simple d'accéder au shell mongo est grâce à la commande : **mongo --host node1**.

2. Vérifier que la base de données a été créée, la sélectionner et s'assurer qu'elle contient la collection **restaurants**.

```
- Afficher la base de données courante : db
- Afficher la liste des bases de données : show dbs
- Sélectionner une base de données : use <name>
- Afficher les collections d'une base de données : show collections
```

3. Combien y-a-t-il de restaurants ?

```
Se référer à la documentation relative à db.collection.count()
```

4. Afficher de deux façons différentes les données du premier restaurant.

```
Se référer à la documentation relative à :
- db.collection.findOne()
- cursor.limit()
```

5. Afficher les informations concernant le restaurant dont le *name* est *Nordic Delicacies*

```
Se référer à la documentation relative à db.collection.find()
```

6. Afficher uniquement l'adresse du restaurant dont le *name* est *Nordic Delicacies*

```
Se référer à la documentation relative à db.collection.find()
```

Insertion avec Mongodb

7. Vous avez découvert un petit restaurant près de chez vous qui n'est pas encore référencé dans votre base de données. Il s'agit du resto 'Mon nouveau resto' qui se situe Avenue Laennec au Mans et dont la cuisine est 'trop bonne'.
   Insérez ce nouveau restaurant dans votre collection.

```
Se référer à la documentation relative à db.collection.insert()
```

8. Combien y-a-t-il de restaurants à présent ?

9. Afficher les données du nouveau restaurant, celui dont normalement le *borough* est ‘Le Mans’.

Requêtes multi-critères

10. Combien de restaurants font des *Hamburgers* ?

11. Parmi ceux-là, combien dans le *Borough* de *Brooklyn* ?

12. Sélectionner uniquement les *name* des restos de *Hamburgers* dans le *borough* de *Brooklyn* et ont dans leurs *grades* un *score* de 8?

13. Retourner la liste des restaurants dont le nom commence par un T, et sans afficher ni l’*address*, ni les *grades*.

```
Équivalence SQL
db.users.find({"name": /a/})  // Like '%a%'
db.users.find({"name": /^pa/}) // Like 'pa%'
db.users.find({"name": /ro$/}) // Like '%ro'
```

14. Retourner les informations concernant le restaurant de *restaurant_id* le plus faible

```
Se référer à la documentation relative à cursor.sort()
```

Mise à jour et suppression

15. Dans le nouveau restaurant, ajouter l’attribut « attribut_sup » qui a comme valeur 0.

```
Se référer à la documentation relative à db.collection.update()
```

16. Après avoir déjeuné dans ce nouveau restaurant, vous souhaitez y ajouter une note de *grade* D et *score* 1. Attention, il faut que cette note apparaisse dans l’array *grades*.

17. Finalement, vous souhaitez supprimer ce restaurant de votre base…

```
Se référer à la documentation relative à db.collection.remove()
```

Aggrégation

18. Créer une requête qui pour chaque *zipcode*, retourne le nombre de restaurants correspondants.

> Se référer à la documentation relative à ***db.collection.aggregate()***
>
> L'opérateur aggregate est une séquence d'opérations représentant une chaîne de pipeline entre chaque opérateur (le résultat d'un opérateur est donné à l'opérateur suivant). Il s'exprime de la sorte :
>
> ***aggregate([{$op1}, {$op2}, {$op3}])***
>
> *Voir l'opérateur* ***$group***

19. Modifier la requête précédente pour n'afficher que les 5 premiers résultats

> *Voir l'opérateur* ***$limit***

20. Modifier la requête précédente pour que les résultats soient triés par *zipcode* dans l'ordre croissant

> *Voir l'opérateur* ***$sort***

21. Modifiez cette requête pour que les résultats ne concernent que des restaurants dont la *cuisine* est *Hamburgers*

> *Voir l'opérateur* ***$match***

22. Modifiez cette requête (et supprimer l'opérateur *limit*) pour que son résultat soit stocké dans une nouvelle collection *comptage*

```
Voir l'opérateur $out
```

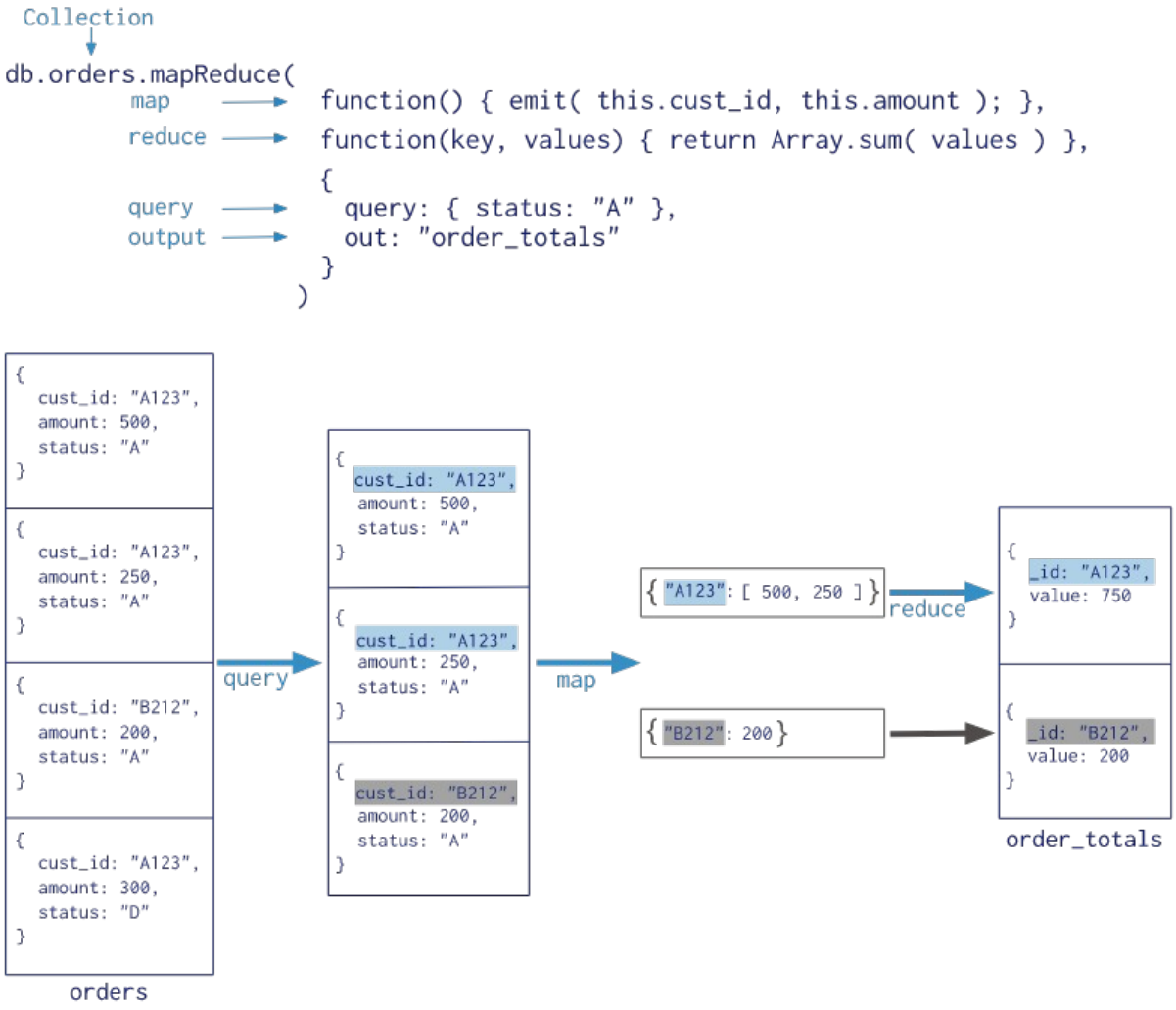
23. Affichez les 10 *zipcode* les plus fréquents en utilisant la collection *comptage*. Proposer une formulation avec *find* et une avec un pipeline *aggregate.*

24. Affichez les *zipcode* dont le nombre d'occurrence est supérieur à 5. Proposer une formulation avec *find* et une avec *aggregate.*

```
Voir l'opérateur $gt
```

MapReduce

Dans la documentation MongoDB, on trouve cet exemple




```
Se référer à mapReduce avec MongoDB. Les exemples sont bien conçus.
db.collection.mapReduce(
      function() { <TODO> ;emit(<TODO>) }, // map
      function(key,values) { <TODO> ;return <TODO>;}, // reduce
      { out: "newCollection" } // collection où stocker le résultat
)
```

25. La requête mapReduce suivante retourne le nombre de restaurants pour chaque type de *Cuisine* et la stocke dans la collection *comptage*. Exécutez-là et observez les résultats.

```
db.restaurants.mapReduce(
      function() { emit(this.cuisine, 1);},
```

```
      function(key, values) {return Array.sum(values)},
      {
            out: "comptage"
      }
)
```

26. En modifiant le fonction map de la requête précédente, retourner le nombre de restaurants pour chaque bo*rough*.

27. En modifiant le fonction map de la requête précédente, retourner le nombre de restaurants pour chaque zipcode.

28. Modifiez la requête précédente pour que les résultats ne concernent que les restaurants de Hamburgers (similaire 21)

```
Voir query dans la documentation sur mapReduce
```

29. Modifier la fonction map de la question 25 et retourner la somme des *score* des restaurants pour chaque *zipcode*.

```
Dans les fonctions map et reduce, écrivez autant de code Javascript que
vous le souhaitez, le langage MongoDB est conçu ainsi.
```