Travaux Pratiques : MongoDB TP 2

Intégration de données JSON

Rakib SHEIKH
NoSQL 2024-2025
rsheikh1@myges.fr

Pour rappel, le Github est disponible à cette adresse : https://github.com/Noobzik/TP_NoSQL

Dans ce TP, nous allons à la fois voir l'intégration de données à l'aide de Mongo Compas, mais aussi la version terminal de l'intégration de données. Nous en profiterons également pour rédiger quelque requêtes.

1. Prise en main de Mongo Compass

1. Téléchargez et installez Mongo Compass dans le lien suivant : https://www.mongodb.com/products/tools/compass
2. Lancez Mongo Compass et surtout, n'allez pas vite sur la page de connexion. En effet, nous avons fixé dans le docker compose, les identifiants de connexion pour accéder à la base de donnée. Par défaut, c'est `root` / `example`. Les users que vous avez créé dans le TP 1 fonctionnent également ici. Pour cela cliquez sur Authenication et renseignez les informations Username et Password.

Une fois la connexion établie, nous pouvons nous commencer l'ingestion des données.

3. Cliquer sur l'icone + qui correspond à la création d'une base de donnée. Notre base de donnée aura pour nom `test_compass` et de collection `restaurant`. Si tout se passe bien, nous avons `test_compass` qui devrait s'afficher.

Nous remarquons que nous n'avons pas de données qui sont disponible.

4. Cliquez sur Import Data et importez le fichier `restaurants.json` Vous devriez avoir 25359 documents importés.

Nous allons réaliser les actions via le shell intégré de compass. Pour l'afficher, cliquez sur _MONGOSH en fin de page de l'application.

5. Tapez la commande `show dbs`. Avez-vous remarqué que le shell intégré de mongo compass proposent l'auto-complétion ? Abusez la touche tabulation dès maintenant !
6. Nous utiliserons la commande `use test_compass` pour changer de base de donnée actuellement sélectionné.
7. Tapez la commande `show collections` pour voir les collections dans ces bases. (*Remarque :Il ne doit y avoir qu'une seule colletion*).
8. Tappez la commande `db.restaurant.countDocuments()` pour récupérer le nombre de docuents actuellement stocké dans cette collection.

2. Recherche de l'information

> **Remarque**
>
> **Quelque chose que vous n'avez peut-être pas remarqué** : Mongo utilise le langage JavaScript pour la programmation dans le shell, avec un ensemble de méthodes permettant la gestion de la base et des données (ce qui nous concerne peu), mais aussi bien évidemment pour la recherche d'information.

2.1. Dénombrement

Nous avons donc déjà vu comment obtenir le nombre de documents présents dans une collection, avec la fonction `countDocuments()` réalisé sur l'objet `db.nom_collection` (`nom_collection` étant donc à remplacer par le nom de la collection qui nous intéresse). Ceci est donc équivalent à `COUNT(*)` en SQL.

Il est aussi possible de se restreindre à certains documents, en réalisant donc une restriction. Pour cela, Mongo utilise un formalisme particulier. Nous allons indiquer les critères de recherche dans un objet (on dit aussi littéral) JSON. Par exemple, si l'on souhaite avoir le nombres de restaurants de Brooklyn, nous allons exécuter la commande suivante :

```
> db.restaurant.countDocuments({borough: "Brooklyn"})
< 6086
```
JavaScript

Nous verrons par la suite comment mettre plusieurs critères de recherche. Le formalisme est le même que dans les fonctions de recherche `findOne()` et `find()` que nous allons voir par la suite.

2.2. Recherche de documents

2.2.1. Premier documents vs tous les documents.

La première demande est souvent de recherche un ou plusieurs documents. Comme les documents n'ont pas de structure définies en amont, il existe une fonction (`findOne()`) permettant de renvoyer le premier document. Cela permet donc de visualiser un exemple de documents.

1. Tentez dès maintenant la commande `db.restaurant.findOne()`

Nous avons donc la structure du document suivant :

- Quelques champs simples : `_id` (clé primaire interne toujours présente), `borough`, `cuisine`, `name` et `restaurant_id`
- Un champs de type littéral (`address`), contenant des champs simples (`building`, `street` et `zipcode`) et un tableau à 2 valeurs `coord`)
- Un tableau `grades` (de 5 éléments ici - la taille n'est pas la même pour chaque restaurant).
  - chaque élément du tableau comprenant 3 champs simples (`date`, `grade` et `score`)
  - un élément du table = un contrôle sanitaire
    - `score` : nombre d'infractions sanitaires
    - `grade` : sorte de note du restaurant en fonction du score (A si peu, B si plus, C si encore plus)

Pour récupérer tous les documents, il faut utiliser la fonction `find()`. Puisqu'il y a beaucoup de résultats, et qu'ils en sont pas tous affichables, seulement les 20 sont affichés. Pour avoir les 20 suivants, vous pouvez taper `it` (pour iterate). Notez que l'affichage est moins lisible (pas de passage à la ligne, ni de tabulation).

2. Tentez dès maintenant la commande `db.restaurant.find()`

2.2.2. Restrictions

On peut aussi chercher le premier document respectant un critère (par exemple, situé à Brooklyn). Ce sera le premier paramètre de la fonction. Nous réalisons donc ici une restriction sur un critère simple.

3. A l'aide de la fonction `findOne` utilisez `Brooklyn` en tant que critère de filtre.

On peut combiner les critères pour avoir le premier restaurant de Brooklyn, proposant de la cuisine française par exemple. Ici, les critères sont mis dans le littéral passé en paramètre, et l'outil effectue un ET entre les deux à l'aide d'une virgule « , ».

4. Reprennez la requête de la question 3 ci-dessus, et ajoutez un deuxième critère basé sur la cuisine française.

Les critères ici sont définis sur des champs simples. Si on veut faire une recherche sur un littéral (par exemple, on veut le premier restaurant sur "Franklin Street"), il faut procéder comme ci-dessous. Attention, il faut bien mettre les "" pour un champ intégré dans un autre (comme « address.street » ici).

```JavaScript
db.restaurant.findOne({"address.street": "Franklin Street"})
```

On peut aussi chercher dans un tableau. Par exemple, on veut avoir le premier restaurant ayant eu un score de 0 (aucune infraction sanitaire lors de l'inspection donc). Noter qu'il renvoie tous les scores.

5. En reprenant l'exemple précédent, filtrez le score des grade des restaurants qui ont eu 0.

2.2.3. Projection

Ici, nous avons à chaque fois récupéré le document en entier. Mais il est parfois utile de ne récupérer que certains éléments (et donc de faire une projection). Pour cela, nous allons passer en paramètre un deuxième littéral, indiquant les champs que l'on souhaite garder (1) ou ne pas garder (0).

Par exemple, nous ne voulons que le `nom` et le `quartier`. Noter ici que pour ne pas faire de restriction, il faut mettre un littéral vide `({})`.

```js
db.restaurant.findOne({}, { name: 1, borough: 1 })
```

Noter que l'identifiant `_id` est présent par défaut. Pour ne pas l'avoir, il faut l'indiquer précisemment. C'est la seule façon de mélanger un choix d'attributs à afficher (1) et un attribut à ne pas afficher (0).

```js
db.restaurant.findOne({}, { _id: 0, name: 1, borough: 1 })
```

On peut aussi afficher que certains éléments des champs complexes (littéral ou tableau).

Par exemple, on souhaite n'avoir que la rue du restaurant en plus, ainsi que la liste des grades obtenus.

```js
db.restaurant.findOne(
  {},
  {
    _id: 0, name: 1, borough: 1,
    "address.street": 1, "grades.grade": 1
  }
)
```

Si au contraire, on ne veut pas afficher certains éléments, on peut aussi utiliser ce formalisme. Ici, on n'affiche pas l'adresse et les visites.

```js
db.restaurant.findOne({}, {address: 0, grades: 0})
```

2.2.4. Restrictions complexes

Malheureusement, il n'est pas possible d'utiliser les opérateurs classiques pour effectuer des restrictions plus complexes que l'égalité stricte.

Par exemple, pour comparer avec une valeur (infériorité ou supérioté), nous devons utiliser des opérateurs spécifiques :

- `$lt` : less than
- `$lte` : less than or equal
- `$gt` : greater than
- `$gte` : greater than or equal

- `$ne` : note equal).

Par exemple, nous cherchons les restaurants avec un score inférieur à 5.

```
> db.restaurant.findOne(                                                    js
  {
    "grades.score": { $lt: 5 }
    },
    { _id: 0, name: 1, borough: 1}
)
< {
  borough: 'Bronx',
  name: 'Morris Park Bake Shop'
}
```

De même, si on souhaite tester si un champs a une valeur dans un ensemble donnée, il faut utiliser l'opérateur `$in`. Ici, nous cherchons les restaurants française et italien :

```
> db.restaurant.findOne(                                                    js
  { "cuisine": { $in: [ "French", "Italian" ]} },
  { _id: 0, name: 1, borough: 1, cuisine: 1}
)
< {
  borough: 'Brooklyn',
  cuisine: 'Italian',
  name: 'Philadelhia Grille Express'
}
```

2.2.5. Valeurs distinctes

Enfin, on peut vouloir connaître les valeurs possibles prises par un champs dans les documents. Par exemple, si on veut la liste des quartiers présents dans la base, on fait comme ci-dessous.

```
db.restaurant.distinct("borough")                                           js
```

2.3. Au boulot !

1. Donner les styles de cuisine présent dans la collection
2. Donner tous les grades possibles dans la base
3. Compter le nombre de restaurants proposant de la cuisine fraçaise (“French”)
4. Compter le nombre de restaurants situé sur la rue “Central Avenue”
5. Compter le nombre de restaurants ayant eu une note supérieure à 50
6. Lister tous les restaurants, en n'affichant que le nom, l'immeuble et la rue
7. Lister tous les restaurants nommés “Burger King” (nom et quartier uniquement)
8. Lister les restaurants situés sur les rues “Union Street” ou “Union Square”
9. Lister les restaurants situés au-dessus de la lattitude 40.90
10. Lister les restaurants ayant eu un score de 0 et un grade “A”

2.3.1. Pour les plus balèzes :

Nécessitent une recherche sur la toile pour compléter ce qu'on a déjà vu dans ce TP.

1. Lister les restaurants (nom et rue uniquement) situés sur une rue ayant le terme “Union” dans le nom

2. Lister les restaurants ayant eu une visite le 1er février 2014
3. Lister les restaurants situés entre les longitudes -74.2 et -74.1 et les lattitudes 40.1 et 40.2

2. 4. Allez dans le vrai terminal maintenant !

2. 4. 1. Premiers pas

1. Récupérer le fichier users.json et l'importer dans la collection `myUsers` de la base `test`. Voir la commande mongoimport. **ATTENTION: la commande mongoimport est à lancer dans le terminal de commande, pas dans le mongo shell**
   - `mongoimport --host localhost --db tp-mongo --collection myUsers < sample_data/users.json`
   - ou
     `mongoimport --host localhost --db tp-mongo --collection myUsers --file sample_data/users.json`

   > **Note**
   >
   > Assurez vous que le fichier users.json est bien visible du container docker, sinon c'est mission impossible.

2. Finalement j'ai changé d'avis, je voulais l'importer dans la collection `users`. Supprimer la collection `myUsers` et réimporter dans la collection `users`,

> **Note**
>
> Il est également possible de simplement renommer la collection

3. Utiliser le shell mongo pour compter le nombre d'éléments dans la collection `users`.
   Voir la documentation du shell mongoDB

Astuce : utiliser `.pretty()` avec un `.find()` permet de rendre le résultat bien plus lisible.

2. 4. 2. Créer un élément

4. Ajouter l'utilisateur dans la collection `users` :
   - Chuck Norris
   - 77 ans
   - hobbies : [« Karate », « Kung-fu », « Ruling the world »]

2. 4. 3. Lecture d'un élément

5. Afficher Chuck Norris (si il le permet).
6. Afficher Chuck sans le champs `_id`.
7. Afficher les utilisateurs qui ont entre 20 et 25 ans.
8. Afficher uniquement les hommes entre 30 et 40 ans.
9. Afficher les utilisateurs habitant l'état de Louisianne (**Louisiana**)
10. Afficher les 20 premiers utilisateurs triés par ordre décroissant d'age.
11. Combien y'a-t-il de femmes agées de 30 ans?

2. 4. 4. Modifier/Supprimer un élément

12. Nos juristes nous ont dit que nous ne pouvions plus garder les numéro de téléphones de nos utilisateurs : supprimer le champ `phone` de tous les enregistrements.
13. Chuck Norris est venu nous dire que le temps ne marquait pas Chuck Norris, mais que Chuck Norris marquait le temps : changer l'age de Chuck Norris à **infinity**
14. Ajoutons un hobby à tous nos utilisateurs de plus de 50 ans : **jardinage**

2. 4. 5. Agrégation

15. Je souhaite savoir combien d'utilisateur j'ai dans chaque état, lister les états en indiquant pour chacun, le nombre d'utilisateurs présents.
16. Je souhaite savoir l'âge moyen des utilisateurs de chaque état.
17. Je veux connaître la liste de tous les hobbies de chaque ville.
18. Utiliser les projections (**$project**) pour lister les utilisateurs en n'affichant que leur nom (en minuscule) et leur age.
19. Je souhaite savoir l'âge moyen des hommes de chaque état.

Ouf ! Nous avons enfin fini ce TP !