# cours-2020-2021

Documents de cours 2020-2021 - FX Jollois

View the Project on GitHub fxjollois/cours-2020-2021

---

# TP1 : Premiers pas sur Mongo

> *NB : Pour réaliser ce TP chez vous, vous devez avoir installé le serveur Mongo, ainsi que Compass et le Shell. Les instructions sont disponibles sur cette page.*

## Intégration de données de type `JSON`

### Création d'une base et d'une collection

Nous allons ici créer une premi!ère base de données, que l'on nommera `test`, contenant une seule collection pour le moment, nommée `essais`.

1. Ouvrir Compass
2. Cliquer directement sur *Connect* (votre serveur étant local, il n'y a pas besoin de le spécifier)
    - Vous devriez voir 3 bases de données déjà existantes : `admin`, `config` et `local`
3. Cliquer sur *CREATE DATABASE* affiché en haut
4. Il faut maintenant nommer la nouvelle base de données, et la collection dans laquelle nous allons mettre les données :
    - *Database Name* : `test`
    - *Collection Name*: `restaurants`
5. Cliquer maintenant sur *CREATE DATABASE*
6. Vous devriez voir apparaître la base `test` à gauche (et la collection `restaurants` lorsque vous cliquez sur la petite flèche à droite de `test`)

Lorsque vous cliquez sur la collection, dans la base, vous voyez le détail de son contenu. Pour le moment, notre collection `restaurants` est vide. Il n'y a donc rien. Il est possible d'importer des documents directement.

### Importation des données

Nous allons importer maintenant les données dans notre collection ainsi créée. Vous devez avoir téléchargé le fichier `restaurants.json` (11.9 Mo - cela peut prendre un peu de temps).

Une fois téléchargé, suivez la procédure suivante pour l'ajouter dans Mongo :

1. Cliquer sur `restaurants` pour voir le contenu de la collection (vide donc pour le moment)
2. Cliquer sur *ADD DATA* en haut et choisissez *Import File* (ou cliquer sur *Import data* directement)
3. Sélectionner le fichier téléchargé, et choisir aussi *JSON*
4. Cliquer sur *IMPORT*
5. Une fois l'opération terminé (25359 documents ajoutés), cliquer sur *DONE*

Votre première base est maitenant créée avec une collection de 25359 restaurants donc.

### Connexion à Mongo via le shell

Nous allons maintenant vérifier que l'opération s'est bien déroulé et qu'on peut accéder à ces données dans le terminal Mongo.

1. Lancer le shell Mongo
2. Taper la commande `show dbs` pour afficher les bases de données existantes
    - Vous devriez donc voir 4 bases (et leur taille)

3. Pour utiliser la base `test`, taper la commande `use test`
4. Taper la commande `show collections` pour voir les collections dans ces bases
    - Il ne doit y en avoir qu'une seule donc

5. Pour voir le nombre de documents de cette collection, taper la commande `db.restaurants.count()`

Vous devriez donc avoir un résultat comme ci-dessous.

```
> show dbs
admin   0.000GB
config  0.000GB
local   0.000GB
test   0.004GB
> use test
switched to db test
> show collections
restaurants
> db.restaurants.count()
25359
```

## Recherche d'information

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.

### Dénombrements

Nous avons donc déjà vu comment obtenir le nombre de documents présents dans une collection, avec la fonction `count()` 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.restaurants.count({ borough: "Brooklyn" })
6086
```

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.

## Recherche de documents

### Premier document 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.

```
db.restaurants.findOne()
```

Noter la structure du document :

- 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 tableau = 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 20 sont affichés.

Pour avoir les 20 suivants, vous pouvez taper `it` (pour *iterate*). Noter que l'affichage est moins *lisible* (pas de passage à la ligne, ni de tabulation).

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

### Restriction

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.

```
db.restaurants.findOne({borough: "Brooklyn"})
```

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.

```
db.restaurants.findOne({borough: "Brooklyn", cuisine: "French"})
```

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).

```
db.restaurants.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.

```
db.restaurants.findOne({"grades.score": 0})
```

### 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 (`{}`).

```
db.restaurants.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).

```
db.restaurants.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.

```
db.restaurants.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.

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

#### 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.restaurants.findOne(
    { "grades.score": { $lt: 5 }},
    { _id: 0, name: 1, borough: 1}
)
```

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 de

```
db.restaurants.findOne(
    { "cuisine": { $in: [ "French", "Italian" ]} },
    { _id: 0, name: 1, borough: 1, cuisine: 1}
)
```

### 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.restaurants.distinct("borough")
```

# A faire

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”

## Questions complémentaires

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

---