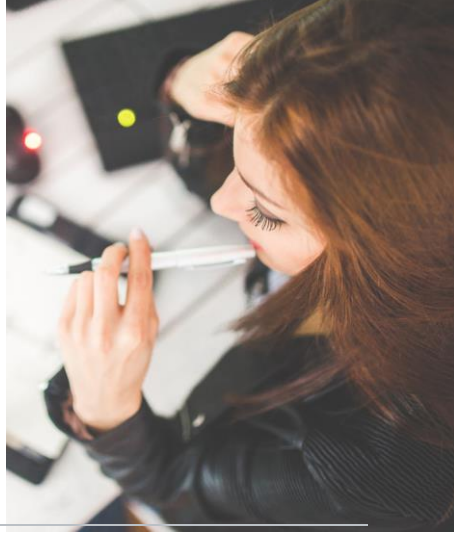
TP 3 : LES BASES DE DONNEES NOSQL - MONGODB



Le but de ce TP3 est de vous entraîner à interroger une base de données MongoDB

1. IMPORTATION DU FICHIER JSON

1. Téléchargez le fichier JSON à partir du lien suivant : https://s3-eu-west-1.amazonaws.com/course.oc-static.com/courses/4462426/restaurants.json_.zip

2. Exportez le fichier **restaurants.json** dans une nouvelle base de données et dans une collection qui porte le même nom « resto ».

3. Interrogez cette collection afin de voir à quoi ressemble un document de la collection restaurants. Utilisez la méthode "**findOne()**".

4. Dressez un schéma approximatif (MCD) de cette base de données.

En effet cette base regroupe des informations sur les restaurants de la ville de New York. Chaque restaurant a un nom, un quartier (borough), le type de cuisine, une adresse (document imbriqué, avec coordonnées GPS, rue et code postale), et un ensemble de notes (résultats des inspections).

2. FILTRER ET PROJETER LES DONNEES

1. Récupérez les informations sur tous les restaurants dans le quartier (borough) de Manhattan

2. Cherchez parmi ces restaurants ceux qui font de la cuisine Marocaine.

3. Même chose, mais n'afficher que les noms des restaurants.

4. Cherchez parmi les restaurants dans le quartier de Manhattan qui sont spécialisés dans les fruits de mer « Seafood »

5. Si l'on souhaite ne récupérer les restaurants de cuisine « Italian » dont le nom contient Pizza, qui ont des scores inférieurs à 10 et qui n'ont pas de score supérieur à 10, il faut alors combiner la première opération avec une négation de la condition " ≥10 ". La condition est alors vérifiée sur chaque élément de la liste.

6. On pourrait même corser la chose en cherchant les restaurants qui ont un grade 'C' avec un score inférieur à 30.

Le résultat est pour le moins étrange concernant le premier document car il y a bien un grade C, mais avec un score de 56 ! Si l'on regarde la requête de plus près, ce n'est pas étonnant. Y a-t-il un grade 'C' ? oui. Y a-t-il un score inférieur à 40 ? oui… Nous avons oublié de préciser qu'il fallait que ce soit vérifié sur chaque élément !

La vérification se fait sur l'intérieur de la liste, pas sur chaque élément de la liste. Pour cela, un opérateur permet de faire une vérification instance par instance : **`$elemMatch`**.

Si nous voulons chercher les noms et quartiers des restaurants dont la dernière inspection (la plus récente, donc la première de la liste) a donné un grade 'C'. Il faut donc chercher dans le premier élément de la liste. Pour cela il est possible de rajouter l'indice recherché (indice 0) dans la clé.

3. DISTINCT

Quels sont les différentes spécialités des restaurants de la ville de New York ?
Quels sont les différents grades de notation des restaurants ?
…
Pour répondre à ces requêtes, on peut utiliser la fonction "distinct()" avec la clé recherchée pour avoir une liste de valeur :

4. CREER UNE SEQUENCE D'OPERATIONS AVEC « AGGREGATE »

La fonction **aggregate** prend une liste d'opérateurs en paramètre. Il existe plusieurs types d'opérateurs de manipulation de données. Les opérateurs essentiels que nous allons voir par la suite dans un premier temps sont :

- **{$match : {} }** : C'est le plus simple, il correspond au premier paramètre de la requête **find** que nous avons fait jusqu'ici. Il permet donc de filtrer le contenu d'une collection.
- **{$project : {} }** : C'est le second paramètre du **find**. Il donne le format de sortie des documents (projection). Il peut par ailleurs être utilisé pour changer le format d'origine.
- **{$sort : {} }** : Trier le résultat final sur les valeurs d'une clé choisi.
- **{$group : {} }** : C'est l'opération d'agrégation. Il va permettre de grouper les documents par valeur, et appliquer des fonctions d'agrégat. La sortie est une nouvelle collection avec les résultats de l'agrégation.
- **{$unwind : {} }** : Cet opérateur prend une liste de valeur et produit pour chaque élément de la liste un nouveau document en sortie avec cet élément. Il pourrait correspondre à une jointure, à ceci près que celle-ci ne filtre pas les données d'origine, juste un complément qui est imbriqué dans le document. On pourrait le comparer à une jointure externe avec imbrication et listes

La liste complète de ces operateurs vous la retrouverez dans le lien de la documentation officielle https://docs.mongodb.com/manual/reference/operator/aggregation/

1. Equivalent à find avec aggregate

Vous aurez remarqué que chaque opérateur est imbriqué dans un document, et la valeur est elle-même un autre document. C'est la structure imposée pour la définition de l'opérateur. Pour illustrer un **aggregate**, nous allons reprendre notre dernière requête **find** :

Cette requête peut être formulé avec la commande **aggregate** en utilisant les deux operateurs **$match** et **$project** de la manière suivante :

Le résultat est identique, toutefois, c'est un peu plus lourd à écrire.

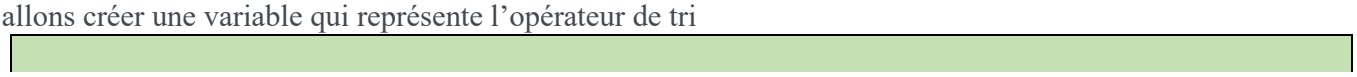
2. Utilisation des variables

On identifie bien chaque opérateur, avec leur définition.
Rappelez-vous que l'interpréteur mongodb utilise le langage Javascript. Chaque opérateur peut alors être défini par une variable que nous pouvons utiliser directement :

Nous utiliserons ces variables par la suite pour alléger le code en ré-utilisant des variables.

3. Le trie

A la suite de cette dernière requête Trions maintenant le résultat par nom de restaurant par ordre croissant alphabétiquement (croissant : 1, décroissant : -1) :
Nous allons créer une variable qui représente l'opérateur de tri



Et en suite nous l'injectons dans la liste des variables dans aggregate :

4. Groupement simple

Comptons maintenant le nombre de ces restaurants (premier rang ayant pour valeur C). Pour cela, il faut définir un opérateur $group. Celui-ci doit contenir obligatoirement une clé de groupement (_id), puis une clé (total) à laquelle on associe la fonction d'agrégation ($sum) :

Ici, pas de valeur de groupement demandé (on compte simplement). Nous avons choisi de mettre la valeur null, on aurait pu mettre "toto", cela fonctionne également. La clé "total" est créée dans le résultat et va effectuer la "somme des 1" pour chaque document source. Faire une somme de 1 est équivalent à compter le nombre d'éléments.

En effet cette requête est équivalente à :

Ou

5. Groupement par valeur

Comptons maintenant par quartier le nombre de ces restaurants (qui toujours vérifient le filtre varMatch). Il faut dans ce cas changer la clé de groupement en y mettant la valeur de la clé "borough". Mais si l'on essaye ceci :

Nous constatons que "borough" est interprété comme étant un texte statique et pas comme un champ de document. Pour prendre la valeur du champ, il faut préfixer la clé par un dollar "$borough". Lorsque la clé d'agrégation "_id" est associé a "$borough" c'est ce champ qui va être utilisé pour le regroupement, voir résultats

Nous pourrons utilisées le même principe pour le regroupement par rapport à la spécialité de cuisine.

Si nous omettons le filtrage par rapport à varMatch

6. Unwind

Maintenant, trouvons le score moyen des restaurants par quartiers, et trions par score décroissant. Pour cela, nous groupons les valeurs par quartier ($borough), nous utilisons la fonction d'agrégat "$avg" pour la moyenne (average), puis nous trions sur la nouvelle clé produite "$moyenne".

Un problème se pose comment calculer la moyenne sur une "grades.score" qui est une liste ? Il faudrait au préalable retirer chaque élément de la liste, puis faire la moyenne sur l'ensemble des valeurs. C'est l'opérateur $unwind qui s'en charge, il suffit de lui donner la clé contenant la liste à désimbriquer.