Bases de données NoSQL avec MongoDB

Dr. Hmida Hmida

ISET Bizerte - 2020

Table of contents

1. Bases de Données NoSQL avec MongoDB 3
1.1 Présentation de MongoDB 4
2. Installation de MongoDB 6
2.1 Procédure d'installation 6
2.2 Configuration 6
2.3 Vérification de l'installation 6
2.4 Connexion à MongoDB 7
3. Opérations de base 8
3.1 Format des commandes MongoDB 8
3.2 Manipulation des bases de données 8
3.3 Insertion de documents 9
3.4 Interrogation 10
3.5 Mise à jour 15
3.6 Suppression 17
4. Opérations avancées 18
4.1 Agrégations 18
4.2 Exercice 21
4.3 Travail Optionnel 21
5. Réplication Dans MongoDB 22
5.1 Définition 22
5.2 Principe du Replica Set 22
5.3 Mise en place d'un Replica Set 22
5.4 Test du Replica Set 23
5.5 Haute disponibilité 24
6. MongDB Sharding 25
6.1 Définition 25
6.2 Principe de fonctionnement 25
6.3 Mise en place du cluster 25

1. Bases de Données NoSQL avec MongoDB

1.0.1 Objectifs

- Installer MongoDB et créer une base de données
- Interroger une base de données orientée document
- Protéger la base de données contre les pannes avec le **ReplicaSet**
- Passer à l'échelle avec le **Sharding**

1.0.2 Ressources

Outils

- MongoDB Community Server V4.4.1 (Windows 10) ou V4.0.2 (Windows 7 ou 8)
- MongoDB Shell V0.4.2 (Exécuter des commandes intéractives)
- MongoCompass V1.22.1 (GUI pour MongoDB)

Fichiers de données (datasets)

- restaurants.json: 25359 restaurants de New York
- cities.csv: 15000 villes du monde

Sources et référence

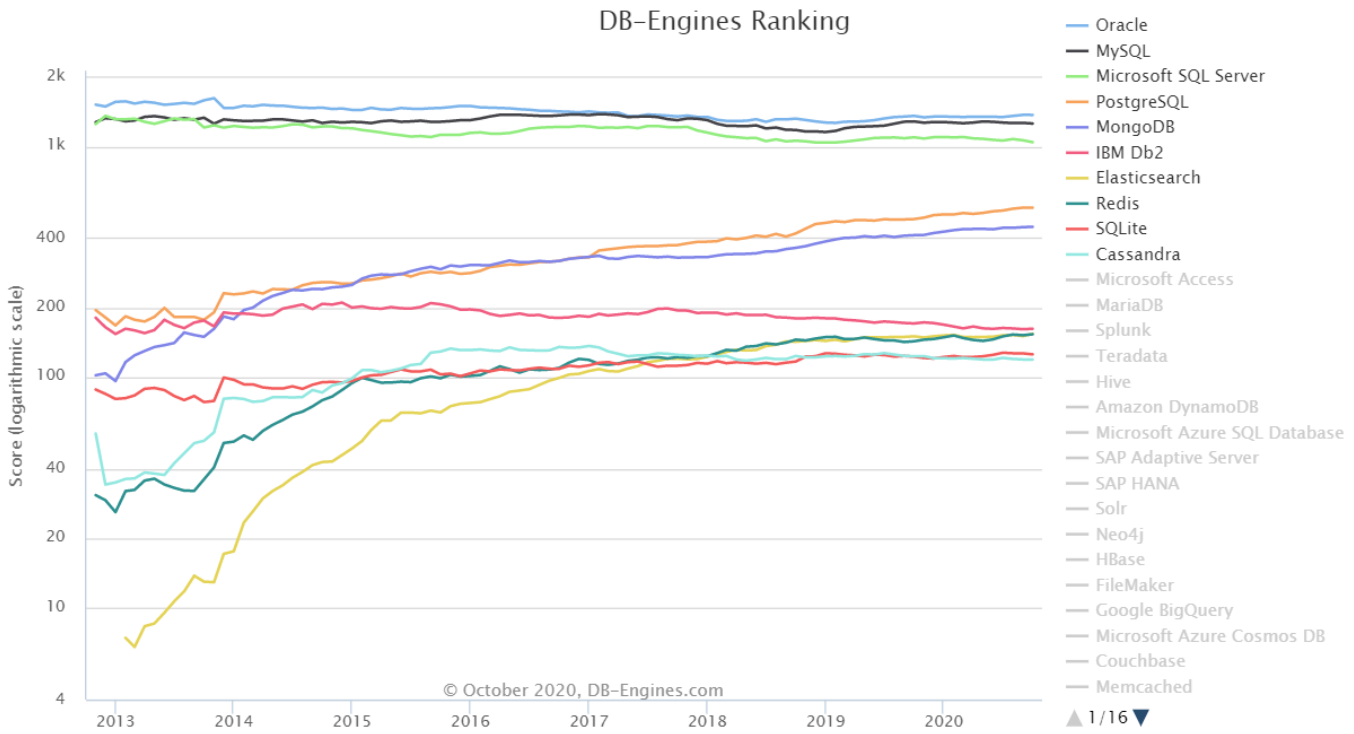
- Documentation MongoDB
- Tutorial MongoDB Sharding sous Windows
- Maîtrisez les bases de données NoSQL Openclassrooms

Télécharger ce document en PDF

1.1 Présentation de MongoDB

1.1.1 Vocabulaire

MongoDB est un système de gestion de bases de données NoSQL orienté documents. Il est parmi les SGBD les plus utilisés.



Source : https://db-engines.com/en/ranking_trend

Le document est l'unité de référence dans ce système. Elle est l'équivalent de tuple ou enregistrement dans les bases de données relationnelles.
MongoDB gère plusieurs bases de données. Chaque base est composée de Collections (équivalent de table). La struture d'un document MongoDB ressembée à celle de JSON. Le modèle est très simple :

- Tout est clé/valeur : "clé" : "valeur".
- Un document est encapsulé dans des accolades {...}, pouvant contenir des listes de clés/valeurs.
- Une valeur peut être un type scalaire (entier, nombre, texte, booléen, null), des listes de valeurs [...], ou des documents imbriqués.

Schéma de la base MongoDB

MongoDB ne gère aucun schéma pour les collections. Donc pas de structure fixe ni de typage. Du coup, il est difficile lors de l'interrogation de connaître le contenu de la base de données. Le logiciel Compass permet de faire une analyse du schéma et fournit des statistiques sur les types des clés et les fréquences des valeurs.

Exemple de document MongoDB


```
{
  "_id":{
    "$oid":"5f7cefe98c3d2e2bd862b3ff"
  },
  "address":{
    "building":"469",
    "coord":{
      "type":"Point",
      "coordinates":[-73.961704,40.662942]
    },
    "street":"Flatbush Avenue",
    "zipcode":"11225"
  },
  "borough":"Brooklyn",
  "cuisine":"Hamburgers",
  "grades":[
    {"date":{"$date":"2014-12-30T00:00:00.000Z"},"grade":"A","score":8},
    {"date":{"$date":"2014-07-01T00:00:00.000Z"},"grade":"B","score":23},
    {"date":{"$date":"2013-04-30T00:00:00.000Z"},"grade":"A","score":12},
    {"date":{"$date":"2012-05-08T00:00:00.000Z"},"grade":"A","score":12}
  ],
  "name":"Wendy'S",
  "restaurant_id":"30112340"
}
```


Ce 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 (tableaux de documents).

Note

La clé `"_id:"` est l'identifiant du document. Si sa valeur n'est pas spécifiée implicitement lors de l'insertion, elle est automatiquement initialisée par MongoDB.

2. Installation de MongoDB

2.1 Procédure d'installation

1. Lancer l'installation du package ***MSI***.
2. Choisir l'installation en tant que service.
3. Il est possible d'installer aussi MongoDB Compass.

2.2 Configuration

MongoDB est installé dans le dossier `C:\Program Files\MongoDB\Server\4.0\` contenant trois dossiers :

- **bin** : Contenant
  - Les fichiers binaires : comme mongod (serveur), mongos (Shard server) et mongo (Shell mongoDB par défaut).
  - Des utilitaires comme celui d'importation de données (mongoimport) ou de monitoring (mongotop).
  - Fichier de configuration mongod.cfg
- **data** : L'espace de stockage.
- **logs** : Emplacement du journal.

La configuration par défaut est :

mongod.cfg

```
# Where and how to store data.
storage:
  dbPath: C:\Program Files\MongoDB\Server\4.0\data
  journal:
    enabled: true

# where to write logging data.
systemLog:
  destination: file
  logAppend: true
  path:  C:\Program Files\MongoDB\Server\4.0\log\mongod.log

# network interfaces
net:
  port: 27017
  bindIp: 127.0.0.1
```

2.3 Vérification de l'installation

Le serveur peut être contrôlé par la console Microsoft de gestion des services Windows (`services.msc`).

Pour vérifier l'installation taper la commade : `mongo -version`. Le résultat sera :

> **Astuce**
>
> Pour simplifier l'accès auxx différents binaires MongoDB, ajouter dans la variable PATH le chemin au dossier d'installation `C:\Program Files\MongoDB\Server\4.0\bin`.

2.4 Connexion à MongoDB

2.4.1 Avec le shell par défaut

```
mongo
>
```

2.4.2 Avec MongoDB Shell *mongosh*

C'est une version améliorée du Shell par défaut :

Il permet d'ajouter au shell :

- La coloration syntaxique
- La complétion automatique
- Des messages d'erreur plus détaillés

> **Note**
>
> MongoDB Shell est l'outil recommandé pour ce TP. Il est basé sur javascript.

2.4.3 Avec MongoDB Compass

Spécifier la chaîne de connexion (mongodb://localhost:27017) et cliquer sur .
Si la configuration standard n'est pas modifiée, vous pouvez laisser la chaîne de connexion vide.

Sélectionner la base *local* et puis la collection *startup_log* qui contient les informations de connexions.

3. Opérations de base

3.1 Format des commandes MongoDB

Format des appels

```
//Appel d'opération sur la base en cours
> db.operation()

//Appel d'opération sur la collection
> db.collection.operation()
```

3.2 Manipulation des bases de données

3.2.1 Lister les bases

show

Voir les bases existantes

```
> show dbs
```

Pour voir la base sélectionnée

```
> db
```

3.2.2 Sélectionner ou créer une BD

C'est la base qui sera active et représente par le préfixe db. Toutes les opérations exécutées ensuite sont exécutées sur cette base.

use

```
> use nom_base
```

Example

Créer une base bibliotheque

```
> use bibliotheque
```

3.2.3 Supprimer la base de données en cours

dropDatabse

```
> db.dropDatabse()
```

Afficher la liste des opérations sur une BD

```
> db.help()
```

3.2.4 Les collections de la base en cours

Création de collection

```
> db.createCollection("nom")
```

Exemple

Créer une collection livres

```
> db.cretateCollection("livres")
```

Lister les collections

```
> show collections
```

Supprimer une collection

```
> db.nom_collection.drop()
```

3.2.5 Importation de données

Pour alimenter la base avec des données existantes dans un format supporté (JSON, CSV, TSV), c'est l'utilitaire mongoimport qui sera utilisé.

mongoimport

Terminal Windows

```
mongoimport [-h host] -d database -c collection [--jsonarray] fichier
```

Exemple

Importer le dataset des restaurants (voir Ressources).

Terminal Windows

```
mongoimport -d emplacements -c restaurants restaurants.json
```

3.3 Insertion de documents

Un document est inséré dans une collection. Il est possible d'utiliser les méthodes insert(), insertOne() ou insertMany().

Exemple

```
> db.livres.insert({
  "Titre":"Mongodb: the definitive guide, 2nd edition",
  "Catégorie":"Informatique",
  "Pages":434,
  "Année":2013,
  "Prix":42.73,
  "Résumé":"This updated second edition provides guidance for database developers,
  advanced configuration for system administrators, and an overview of the concepts
  and use cases for other people on your project.",
  "Langue":"EN",
  "ISBN":9781449344689,
  "Editeur":{"Nom":"","Pays":""},
  "Auteurs":[{"Nom":"Chodorow","Prénom":"Kristina"}],
  "Mots clés":["MongoDB","NoSQL","Database"]
})
```

Exercice
Ajouter le livre ayant cette couverture :

Note

- MongoDB associe un identifiant unique à chaque document appelé _id, et lui attribue une valeur si elle n'est pas indiquée explicitement.
- MongoDB ne fait aucun contrôle de structure ou autres contraintes sur les données. Ces contrôles seront réalisées.

3.4 Interrogation

C'est la méthode find() qui réalise l'interrogation dans MongoDB.
Les requêtes seront effectuées sur la base de données des restaurants importée précédemment : `use emplacements`

3.4.1 Format de la méthpde find()

find possède 2 paramètres optionnels représentés également en format JSON. Le premier représente la requête de filtrage (clause WHERE en SQL). Le second paramètre exprime l'opération la projection.

find

```
> db.collection.find(query, projection)
```

3.4.2 Projection

```
{clé:<0,1>}
```

0 : exclusion, 1 : inclusion

- L'identifiant _id est implicitement inclus dans le résultat. Pour l'exclure, ajouter **_id:0**.
- Il n'est pas possible de mélanger inclusions et exclusions sauf pour le champ _id.
- Pour accéder à un champ d'un document imbriqué, utiliser . : **champ1.champ2**.

Exemples

Afficher les noms des restaurants et leurs spécialités (cuisine).

MongoDB

```
> db.restaurants.find({},{name:1, cuisine:1})
```

SQL

```
SELECT name, cuisine FROM restaurants
```

Afficher tous les noms des restaurants et la rue (street) et pas le _id.

MongoDB

```
> db.restaurants.find({},{name:1, "address.street":1, _id:0})
```

SQL

```
SELECT name, street FROM restaurants
```

3.4.3 Contenu de collection

find() affiche le contenu de la collection tandis que count() retourne le nombre de documents de la collection.

Exemple

MongoDB

```
> db.restaurants.find()
> db.restaurants.count()
```

SQL

```
SELECT * FROM restaurants
SELECT count(*) FROM restaurants
```

3.4.4 Sélection

Exemples

Trouver les restaurants du quartier Manhattan.

MongoDB

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

SQL

```
SELECT * FROM restaurants WHERE borough = 'Brooklyn'
```

Trouver les restaurants du quartier Manhattan offrant la cuisine italienne.

MongoDB

```
> db.restaurants.find( { "borough" : "Brooklyn", "cuisine":"Italian" } )
```

SQL

```
SQL SELECT * FROM restaurants WHERE borough = 'Brooklyn' and cuisine = 'Italian'
```

Trouver les restaurants sans nom (null).

MongoDB

```
> db.restaurants.find( { "name" : null } )
```

SQL

```
SELECT * FROM restaurants WHERE name IS NULL
```

Expressions régulières

Avec les champs de type texte, MongoDB accepte les expressions régulières sous le format `"champ":/regex/options`.

Option	Description
i	insensible à la casse.
m	Multiligne.

Exemples

Trouver les restaurants dont le nom contient `Pizza` sans prendre en compte la casse.

MongoDB

```
> db.restaurants.find( { "name" : /pizza/i})
```

SQL

```
SELECT * FROM restaurants WHERE upper(name) like '%PIZZA%'
```

Trouver les restaurants dont le nom commence par 'A'.

MongoDB

```
> db.restaurants.find( { "name" : /^A/})
```

SQL

```
SELECT * FROM restaurants WHERE name like 'A%'
```

Trouver les restaurants dont le nom commence est composé de 7 caractères.

MongoDB

```
> db.restaurants.find( { "name" : /^.{7}$/})
```

SQL

```
SELECT * FROM restaurants WHERE name like '_______'
```

3.4.5 Filtrage avec opérateurs

Pour construire les critères de sélection, plusieurs opérateurs peuvent être utilisés dans la requête selon le format : `{champ1: {<opérateur>:<valeur1>, ...}`

Les opérateurs

Opérateur	Rôle	Exemple
$gt, $gte	>,>=	`"champ":{"$gt":10}`
$lt, $lte	<,<=	`"champ":{"$lt":10}`
$ne	!=	`"champ":{"$ne":10}`
$in, $nin	IN, NOT IN	`"champ":{"$in":[10,20,30]}`
$or	ou logique	`"champ":{"$or":[{"$gt":10},{"$lt":5}]}`
$and	et logique	`"champ":{"$and":[{"$lt":10},{"$gt":5}]}`
$not	négation	`"champ":{"$not":{"$gt":10}}`
$exists	Existence de la clé dans le document	`"champ":{"$exist":1}`
$size	taille d'une champ array	`"champTableau":{"$size":5}`
$elemMatch	vérification conjointe de plusieurs conditions sur un élément du tableau	`"champTableau":{$elemMatch:{"champInterne":{$gt:5}}}`

Exemple 1

Trouver les noms et scores des restaurants de Manhattan ayant reçu au moins un score inférieur à 10.

MongoDB

```
> db.restaurants.find( {
  borough:"Manhattan",
  "grades.score":{"$lt":10}
},
{name:1,"grades.score":1, _id:0})
```

Exemple 2

Trouver les noms et scores des restaurants de Manhattan avec des scores tous inférieur à 10.

MongoDB

```
> db.restaurants.find( {
  borough:"Manhattan",
  "grades.score":{$not:{"$lt":10}}
},
{name:1,"grades.score":1, _id:0})
```

Exemple 3

Trouver les restaurants qui ont une évaluation ayant un grade ‘C’ avec un score inférieur à 40.

MongoDB

```
> db.restaurants.find({
  "grades" :{
    $elemMatch :{
      "grade" : "C",
      "score" : {$lt :40}
    }
  }
},
{"grades.score":1, _id:0})
```

⚠

$elemMatch permet de vérifier les conditions sur le même élément du tableau. Sans cet opérateur on obtient des évaluations avec des scores différents de 30 ou des grades différents de C.

```
> db.restaurants.find({
  "grades.grade" : "C",
  "grades.score" : {$lt : 30}
},
{"Grades.grade":1, "grades.score":1})
```

Exemple 4

Trouver les restaurants qui ont une exactement 4 évaluations.

MongoDB

```
> db.restaurants.find({
  "grades" :{$size:4}
})
```

Exemple 5

Trouver les restaurants qui ont une longitude (la première valeur de coordinates) < -100.

MongoDB

```
> db.restaurants.find({
  "address.coord.coordinates.0" :{$lt:-100}
})
```

Distinct
Pour déterminer les valeurs distinctes d'un champ.

Exemples

Trouver les différents quartiers.

MongoDB

```
> db.restaurants.distinct("borough")
```

SQL

```
SELECT distinct borough FROM restaurants
```

Trouver l'intervalle des scores.

MongoDB

```
> db.restaurants.distinct("grades.score")
```

3.5 Mise à jour

Update

```
> db.collection.update(
  {filtre},
  {
    <opérateurUpdate>:{<champ1>:<valeur1>,....},
    <opérateurUpdate>:{<champ1>:<valeur1>,....},
    ....
  },
  {
    "multi":true|false, //mise à jour de plusieurs documents - par défaut false
    "upsert":true|false //insertion si aucun document ne correspond - par défaut false
  }
)
```

Opérateurs de mise à jour

Opérateur	Description	Exemple
$currentDate	Affecter à un champ la date en cours	`$currentDate:{"champ":true}`
$inc, $mul	Incrémenter resp. multiplier la valeur d'un champ par une valeur	`$inc:{"champ":5}`
$min, $max	Modifie le champ si la nouvelle valeur est inférieure resp. supérieure à la valeur actuelle.	`$min:{"champ":10}`
$rename	Renommer un champ.	`$rename{"champ":"nouveauChamp"}`
$set	Affecte une valeur au champ spécifié et l'ajoute s'il n'existe pas.	`$set:{"champ":2}`
$unset	Supprime le champ du document.	`$unset{"champ":""}`
$pop	Supprime le premier ou dernier élément d'un champ tableau.	`$pop{"champTableau":-1}` -1: premier, 1:dernier
$pull	Supprime les éléments qui vérifient la condition.	`$pull{"champTableau":condition}`
$push	Ajoute un ou plusieurs éléments au tableau à une position.	`$push{"champTableau":{$each:[valeur1, valeur2,...], $position: positionInsertion}}`

Exemple 1

Modifier le champ borough en 'Manhattan' et affecter la date actuelle au champ lastupdate du restaurant avec l'identifiant 5f7cefe98c3d2e2bd862b3fe

MongoDB

```
> db.restaurants.update(
  {_id:ObjectId("5f7cefe98c3d2e2bd862b3fe")},
  {
    $set:{"borough":"Manhattan"},
    $currentDate:{"lastUpdated":true}
  }
)
```

SQL

```
  UPDATE restaurants
SET
  borough = 'Manhattan',
  lastUpdated = sysdate,
WHERE _id = '5f7cefe98c3d2e2bd862b3fe'
```

Exemple 2

Supprimer le champ cuisine et ajouter 2 point au premier score du restaurant avec l'identifiant 5f7cefe98c3d2e2bd862b3fe

MongoDB

```
> db.restaurants.update(
  {_id:ObjectId("5f7cefe98c3d2e2bd862b3fe")},
  {
    $unset:{"cuisine":""},
    $inc:{"grades.0.score":2},
  }
)
```

Exemple 3

Ajouter un commentaire avec la valeur acceptable pour les restaurants qui n'ont pas eu le grade 'C'.

MongoDB

```
> db.restaurants.update (
  {"grades.grade" : {$not : {$eq : "C"}}},
  {$set : {"comment" : "acceptable"}},
  {"multi" : true}
)
```

Exercice
Donner la commande qui permet d'annuler celle de l'exemple 2.

3.6 Suppression

Remove

```
> db.collection.remove(
  {filtre},
  {justOne:true|false} //Optionnel par défaut false
)
```

Exemple

Supprimer tous les restaurants dont le nom commence par M.

MongoDB

```
> db.restaurants.remove({name:/^M.*/})
```

SQL

```
DELETE  FROM restaurants WHERE name like 'M%';
```

4. Opérations avancées

4.1 Agrégations

4.1.1 Fonction aggregate

MongoDB fournit la fonction aggregate qui permet de créer un pipeline (séquence) d'opérations ou stage (étape). Elle possède 2 paramètres : un tableau d'opérations et JSON optionnel pour les options d'exécution.

aggregate

```
> db.collection.aggregate([
  {operation1:{}},
  {operation2:{}}, ...
],
{
  option1:value,
  option2:value, ...
}
)
```

4.1.2 Opérateurs d'agrégation (stages)

Opérateurs simples

`{$match: {<query>}}` : C'est une opération de filtrage exactement comme le premier paramètre de la requête find.

`{$project: {champ1:1|0, ...}}` : 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: {champ1:1|-1,....}}` : Trier le résultat final sur les valeurs d'une clé choisi.

Exemples

Exemple1

Afficher le nom et l'adresse des restaurants dont le nom comporte "Staten Island".

MongoDB

```
//avec find
> db.restaurants.find({name:/Staten Island/},
  {_id:0, name:1, address:1}
)
//avec aggregate
> db.restaurants.aggregate([
  {$match{name:/Staten Island/},
  {$project: {_id:0, name:1, address:1}}
])
```

SQL

```
SELECT name, address
FROM restaurants
WHERE name like '%Staten Island%'
```

Trier le résultat selon l'ordre décroissant du nom.

MongoDB

```
> db.restaurants.aggregate([
  {$match{name:/Staten Island/},
  {$project: {_id:0, name:1, address:1}},
  {$sort:{name:-1}}
])
```

SQL

```
SELECT name, address
FROM restaurants
WHERE name like '%Staten Island%'
ORDER BY name DESC
```

Trier le résultat selon l'ordre décroissant du nom.

Groupage et agrégats

`{$group: {_id: <expression>, <champAgrégé1>: { <accumulator1> : <expression1> },...}}` : 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. _id est le champ du groupement (précéder le nom du champ valeur par $). accumulateur1 est une fonction d'agrégation : $min, $max, $sum, $avg

`{$unwind : <$champTableau>}` : Crée n documents à partir d'un tableau. Il peut être considéré comme une jointure entre le document en cours de collection et le champ tableau. Le nom du champ est précédé par $.

Exemple2

Calculer le nombre de restaurants total.

MongoDB

```
> db.restaurants.aggregate([
  {
    $group:{_id:null, Total:{$sum:1}}
  }
])
```

SQL

```
SELECT count *
FROM restaurants
```

Calculer le nombre de restaurants total par quartier.

MongoDB

```
> db.restaurants.aggregate([
  {
    $group:{_id:"$borough", Total:{$sum:1}}
  }
])
```

SQL

```
SELECT count *
FROM restaurants
```

Calculer le score moyen des restaurants dont le nom contient "Staten Island". Puisque le score est dans le tableau grades, il faut transformer ce tableau avec $unwind. Chaque stage du pipeline peut être sauvegardé dans une variable pour être utilisé dans plusieurs requêtes.

MongoDB

```
> var filtrage = {$match:{name:/Staten Island/}};
> var decomposerGrades = {$unwind : "$grades"};
> var groupage = {$group:{_id:"$_id","Score moyen": {$avg:"$grades.score"}}};
> db.restaurants.aggregate([ filtrage, decomposerGrades, groupage]);
```

SQL

On suppose que grades est un table présente dans la base de données.

```
SELECT _id, AVG(grades.score) "Score moyen"
FROM restaurants JOIN grades
WHERE name like '%Staten Island%'
GROUP BY (_id)
```

Jointure

Nous présentons ici la syntaxe la plus simple pour une équi-jointure.

`{$lookup: {from: <collection2>, localField: <champInterne>,foreignField: <chamCollection2>, as: <nouveauChampTableau>}}` : Cet opérateur effectue une jointure (left outer join) avec une autre collection de la base.

Exemple3

MongoDB

```
> db.managers.aggregate([
  {
    $lookup:{
      from:"restaurants",
      localField:"res_id",
      foreignField:"restaurant_id",
      as:"Restaurant"
    }
  }
])
```

SQL

```
SELECT *, Restaurant
FROM managers
WHERE Restaurant in (SELECT *
                     FROM restaurants
                     WHERE restaurant_id = res_id)
```

Sortie

`{$out: <collection> | {db:<base>, coll:<collection>}}` : Cet opérateur spécifie la base et la collection où le résultat sera inséré.

Exemple 4

Pour enregistrer le résultat de la première requête de l'exemple 1.

MongoDB

```
> db.restaurants.aggregate([
  {$match{name:/Staten Island/},
  {$project: {_id:0, name:1, address:1}},
  {$out:"staten"}
])
```

SQL

```
CREATE TABLE staten AS(
  SELECT name, address
  FROM restaurants
  WHERE name like '%Staten Island%'
)
```

4.2 Exercice

Répondre aux questions suivantes :

1. Quels restaurants contiennent des chiffres dans leurs noms ?
2. Quels sont les noms et identifiants (restaurant_id) des restaurants qui n'ont pas reçu de score > 6?
3. Chercher pour chaque restaurant son score minimal et maximal triés par le score minimal croissant puis le score maximal décroissant.
4. Comment enregistrer la réponse de la question 2 dans une collection appelée "q2".
5. Ajouter l'adresse à chaque restaurant de cette collection.

4.3 Travail Optionnel

Choisir l'un des thèmes et créer des exemples de requêtes sur la base restaurants ou une autre de base de ton choix.

- Indexation
- Objet cursor
- Requêtes géospatiales
- Jointures avec `$lookup` et des conditions multiples.

5. Réplication Dans MongoDB

5.1 Définition

La réplication est une technique commune aux systèmes NoSQL pour assurer la sécurité et la reprise après les pannes. Elle consiste à créer des copies des données sur des serveurs différents pouvant remplacer le serveur en cas de panne.

5.2 Principe du Replica Set

- Un Replica Set est un groupe de serveurs mongod qui gèrent les mêmes données.
- Il est composé d'un unique serveur primaire (maître) et de plusieurs serveurs secondaires (esclaves).
- Le serveur primaire répond aux demandes du (lecture/écriture) client (driver). Le client peut également lire à partir des serveurs secondaires.
- Les serveurs secondaires reproduisent les mêmes opérations (écritures) réalisées au niveau du serveur primaire.
- Si le serveur primaire est indisponible, un serveur secondaire est promu en primaire après une procédure d'élection.
- Si le nombre de serveurs est pair, un serveur **arbitre** est ajouté.
- Le nombre minimal de serveurs est de 3 et ils peuvent atteindre 50.
- 7 serveurs au maximum peuvent participer à une élection.

5.3 Mise en place d'un Replica Set

Dans ce qui suit, nous allons créer un Replica Set appelé *RS1* composé de 3 serveurs sur une même machine. Alors, nous allons utiliser des numéros de ports et des dossiers différents.

1️⃣ Créer 3 dossiers différents dans le lecteur de votre choix. Par exemple : `C:/data/db1`, `C:/data/db2` et `C:/data/db3`.

2️⃣ Créer 3 service mongoDB :

- Créer 3 fichiers de configurations : `mongod1.cfg`, `mongod2.cfg`, `mongod3.cfg` pour y mettre les paramètres comme le montre l'aperçu ci-dessous en remplçant les chemins dbpath et logpath :

mongod1.cfg

```
# data directory
dbpath=C:/data/db1

# log file
logpath=C:/logs/log1

logappend=true

#port number
port=30001

#replica set name
replSet=rs1
```

- Créer les services

Terminal Windows

```
mongod --config C:/cfg/mongod1.cfg --serviceName MongoDB1 --serviceDisplayName MongoDB1 --install
mongod --config C:/cfg/mongod2.cfg --serviceName MongoDB2 --serviceDisplayName MongoDB2 --install
mongod --config C:/cfg/mongod3.cfg --serviceName MongoDB3 --serviceDisplayName MongoDB3 --install
```

- Démarrer les 3 services depuis la console Microsoft `services.msc` ou avec la commande `net start MongoDB1`.

3 Se connecter à un serveur (ici le premier)

Terminal Windows

```
mongo --port 30001
```

4 Initier le Replica Set et ajouter les autres serveurs

```
> rs.initiate()
> rs.add ("localhost:30002")
> rs.add ("localhost:30003")
```

Le Replica Set est maintenant démarré. Pour le vérifier sa configuration et déterminer les serveurs secondaires et le serveur primaire, tapez la commande :

```
> rs.status()
```

Ajout d'un arbitre Suivre les mêmes étapes mais utiliser `rs.addArb("localhost:30004")`.

5.4 Test du Replica Set

5.4.1 Test des opérations de lecture/écriture

Essayer les opérations suivantes :

1 Se connecter au serveur primaire et créer une base de données 'Courrier' en suivant les étapes suivantes :

```
> use courrier
> db.createCollection('mail')
> db.mail.insert({from:'support@mongodb.com',subject:'Replica Set Testing',body:'OK'})
```

2 Se connecter aux serveurs secondaires et vérifier le contenu de la base :

```
> use courrier
> db.mail.find()
```

Pour éviter cette erreur, exécuter `> rs.slaveOk()` et réessayer.

3 Vérifier si l'insertion est possible depuis un serveur secondaire.

5.4.2 Test de la reprise sur panne

Pour simuler une panne :

- Se connecter au serveur primaire.
- Arrêter le serveur et examiner le comportement du Replica Set (réponse aux requête, élection d'un nouveau serveur primaire).

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

5.4.3 Le fichier oplog

Le serveur primaire crée automatiquement une collection oplog.rs dans la base de données local pour y enregistrer toutes les opérations de mise à jour. Les serveurs secondaires lisent régulièrement le fichier oplog pour répliquer les données.

Afficher le contenu de cette collection :

```
> use local
> db.oplog.rs.find().pretty()
```

Vous pouvez lire les informations suivantes :

- "op" : code opération
- "ts" : date et heure
- "ns" : collection concernée

Pour fixer la taille de ce fichier modifier le paramètre `oplogSizeMB`

```
replication:
  oplogSizeMB: <taille en MB>
```

5.5 Haute disponibilité

Pour améliorer le temps de réponse des opérations de lecture (l'écriture est exclusivement exécutée sur le serveur primaire). Plusieurs modes d'exécution des lectures sont possibles à travers le pramètre `localThresholdMS` de la section *replication* à savoir :

- Primary : valeur par défaut, lecture sur le serveur d'écriture.
- PrimaryPreferred : si jamais le PRIMARY n'est plus disponible, les requêtes sont routées vers le SECONDARY.
- Secondary : Routé uniquement sur les SECONDARY.
- Nearest : Le serveur physique le plus proche sur le réseau (latence la plus faible) est interrogé directement par le client.

D'autres paramètres permettent aussi d'influencer la haute disponibilité comme la priorité d'élection de chaque serveurs et le paramètre readConcern.

6. MongDB Sharding

6.1 Définition

C'est une méthode de distribution des données sur plusieurs machines (cluster). Elle permet à MongoDB de passer à l'échelle pour accueillir des bases de données massives avec un débit très important.
Elle fait partie des techniques de mise à l'échelle horizontale : **Horizontal scaling**.

> **shard** : éclat, tesson (fragment)

6.2 Principe de fonctionnement

Le cluster est composé de 3 types de serveurs :

1. *mongos* : routeur pour l'acheminement des requêtes.
2. *Config Server* : enregistre les métadonnées sur et la configuration pour le cluster.
3. *Shard* : contient un sous ensemble des données (chunk ou partition) qui peut être déployé comme Replica Set.

La distribution des données sur le cluster est basée sur une clé shard key. La clé peut être un ou plusieurs champs. Elle est utilisée dans l'une des deux stratégies de distribution :

- Hashed Sharding : un hash de clé détermine quelle dans quelle chunk mettre les données.
- Ranged Sharding : la clé est affecté à un chunk selon valeur.

Hashed Sharding	Ranged Sharding

6.3 Mise en place du cluster

Le cluster est composé de 2 Config Server, 2 Shard et 1 Config Server

6.3.1 1️⃣ Lancer les Config Server en Replica Set

Configuration des Config Server

cs1.cfg

```
storage:
  dbPath: C:\data\cs1
  journal:
    enabled: true
systemLog:
  destination: file
  logAppend: true
  path:  C:\logs\logcs1
net:
  port: 40001
  bindIp: 127.0.0.1
sharding:
  clusterRole: configsvr
replication:
  replSetName: csrs
```

cs2.cfg

```
storage:
  dbPath: C:\data\cs2
  journal:
    enabled: true
systemLog:
  destination: file
  logAppend: true
  path:  C:\logs\logcs2
net:
  port: 40002
  bindIp: 127.0.0.1
sharding:
  clusterRole: configsvr
replication:
  replSetName: csrs
```

- Démarrer les 2 services et initier le Replica Set.

Terminal Windows

```
start mongod --config C:\cfg\cs1.conf
start mongod --config C:\cfg\cs2.conf
mongo --port 40001
```

```
> rs.initiate()
> rs.add ("localhost:40002")
```

6.3.2 2️⃣ Lancer chaque shard en Replica Set

- Créer les dossiers : C:\data\sh1, C:\data\sh2
- Préparer les fichiers de configuration

Configuration des Shard Server

sh1.cfg

```
storage:
  dbPath: C:\data\sh1
  journal:
    enabled: true
systemLog:
  destination: file
  logAppend: true
  path:  C:\logs\logsh1
net:
  port: 50001
  bindIp: 127.0.0.1
sharding:
  clusterRole: shardsvr
replication:
  replSetName: shrs1
```

sh2.cfg

```
storage:
  dbPath: C:\data\sh2
  journal:
    enabled: true
systemLog:
  destination: file
  logAppend: true
  path:  C:\logs\logsh2
net:
  port: 50002
  bindIp: 127.0.0.1
sharding:
  clusterRole: shardsvr
replication:
  replSetName: shrs
```

- Démarrer les 2 services et initier le Replica Set.

Terminal Windows

```
start mongod --config C:\cfg\sh1.conf
start mongod --config C:\cfg\sh2.conf
mongo --port 50001
```

Puis dans le shell MongoDB :

```
> rs.initiate()
```

Puis le deuxième serveur

Terminal Windows

```
mongo --port 50002
```

Et dans le shell MongoDB :

```
> rs.initiate()
```

6.3.3 3 Lancer le Routeur

- Le fichier de configuration

mongos.cfg

```
sharding:
  configDB: csrs/localhost:40001,localhost:40002
net:
  bindIpAll: true
  port: 26000
systemLog:
  destination: file
  path: D:\logs\logMongos
  logAppend: true
```

- Démarrer et enregistrer les Shard

Terminal Windows

```
start mongos --config C:\cfg\mongos.cfg
mongo --port 26000
```

```
> sh.addShard("shrs1/127.0.0.1:50001")
> sh.addShard("shrs2/127.0.0.1:50002")
```

6.3.4 4 Distribuer une base de données

- Créer une nouvelle base et nouvelle collection. Puis créer un index sur _id.

```
> use testDB;
> sh.enableSharding("testDB");
> db.createCollection("test");
> db.test.createIndex({"_id":1});
> sh.shardCollection("testDB.test",{"_id":1});
```

- Importer un dataset et vérifier l'état du Sharding.

Terminal Windows

```
mongoimport -d testDB -c test --port 26000 restaurants.json
mongo --port 26000 --eval "sh.status()"
```

> ⚠
> Les requêtes sont destinées au routeur.

Inspecter les propriétés et déterminer :

- La clé de sharding
- Le nombre de shunks par shard