FSBM

FACULTE DES SCIENCES BEN M'SIK
UNIVERSITE HASSAN II DE CASABLANCA

Data Science & Big Data
Master DSBD

mongoDB.

Manipulation de données depuis le Shell

Mongosh

1. Lancer MongoDB

Assurez-vous que votre serveur MongoDB est en cours d'exécution. Sinon, démarrez-le

Puis ouvrez un autre terminal et connectez-vous au shell MongoDB

2. Utiliser la base de données `mongo_practice`

Dans le shell MongoDB :

use mongo_practice

3. Insertion de Documents

Dans la collection movies :

```
db.movies.insertMany([
  {
    title: "Fight Club",
    writer: "Chuck Palahniuk",
    year: 1999,
    actors: ["Brad Pitt", "Edward Norton"]
  },
  {
    title: "Pulp Fiction",
    writer: "Quentin Tarantino",
    year: 1994,
    actors: ["John Travolta", "Uma Thurman"]
  },
  {
    title: "Inglorious Basterds",
    writer: "Quentin Tarantino",
```

```
    year: 2009,
    actors: ["Brad Pitt", "Diane Kruger", "Eli Roth"]
  },
  {
    title: "The Hobbit: An Unexpected Journey",
    writer: "J.R.R. Tolkien",
    year: 2012,
    franchise: "The Hobbit"
  },
  {
    title: "The Hobbit: The Desolation of Smaug",
    writer: "J.R.R. Tolkien",
    year: 2013,
    franchise: "The Hobbit"
  },
  {
    title: "The Hobbit: The Battle of the Five Armies",
    writer: "J.R.R. Tolkien",
    year: 2012,
    franchise: "The Hobbit",
    synopsis: "Bilbo and Company are forced to engage in a war against
an array of combatants and keep the Lonely Mountain from falling into
the hands of a rising darkness."
  },
  { title: "Pee Wee Herman's Big Adventure" },
  { title: "Avatar" }
])
```

4. Requêtes MongoDB

Rechercher des documents

Rechercher tous les documents :

```
db.movies.find()
```

Films écrits par Quentin Tarantino :

```
db.movies.find({ writer: "Quentin Tarantino" })
```

Films avec Brad Pitt :

```
db.movies.find({ actors: "Brad Pitt" })
```

Films de la franchise The Hobbit :

```
db.movies.find({ franchise: "The Hobbit" })
```

Films des années 90 :

db.movies.find({ year: { $gte: 1990, $lt: 2000 } })

Films avant 2000 ou après 2010 :

db.movies.find({ $or: [{ year: { $lt: 2000 } }, { year: { $gt: 2010 } }] })

5. Mettre à Jour des Documents

Mettre à jour la synopsis du film "The Hobbit: An Unexpected Journey" :

db.movies.updateOne(
  { title: "The Hobbit: An Unexpected Journey" },
  { $set: { synopsis: "A reluctant hobbit, Bilbo Baggins, sets out to the Lonely Mountain with a spirited group of dwarves to reclaim their mountain home - and the gold within it - from the dragon Smaug." } }
)

Mettre à jour la synopsis du film "The Hobbit: The Desolation of Smaug" :

db.movies.updateOne(
  { title: "The Hobbit: The Desolation of Smaug" },

{ $set: { synopsis: "The dwarves, along with Bilbo Baggins and Gandalf the Grey, continue their quest to reclaim Erebor, their homeland, from Smaug. Bilbo Baggins is in possession of a mysterious and magical ring." } }
)

Ajouter un acteur à "Pulp Fiction" :

```
db.movies.updateOne(
  { title: "Pulp Fiction" },
  { $push: { actors: "Samuel L. Jackson" } }
)
```

6. Recherche avec Texte

Créer un index textuel sur la synopsis :

```
db.movies.createIndex({ synopsis: "text" })
```

Rechercher les films avec "Bilbo" :

```
db.movies.find({ $text: { $search: "Bilbo" } })
```

Rechercher les films avec "Gandalf" :

```
db.movies.find({ $text: { $search: "Gandalf" } })
```

Rechercher les films avec "Bilbo" mais sans "Gandalf" :

```
db.movies.find({ $text: { $search: "Bilbo -Gandalf" } })
```

Rechercher les films avec "dwarves" ou "hobbit" :

```
db.movies.find({ $text: { $search: "dwarves hobbit" } })
```

Rechercher les films avec "gold" et "dragon" :

```
db.movies.find({ $text: { $search: "gold dragon" } })
```

7. Supprimer des Documents

Supprimer le film "Pee Wee Herman's Big Adventure" :

```
db.movies.deleteOne({ title: "Pee Wee Herman's Big Adventure" })
```

Supprimer le film "Avatar" :

```
db.movies.deleteOne({ title: "Avatar" })
```

8. Gestion des Relations

Insertion des Utilisateurs

```
db.users.insertMany([
  { username: "Cinephile123", first_name: "Alice", last_name: "Dupont" },
  { username: "MovieFanatic", first_name: "Bob", last_name: "Martin" }
])
```

Insertion des Critiques

```
let movie1 = db.movies.findOne({ title: "Fight Club" })._id;
let movie2 = db.movies.findOne({ title: "Pulp Fiction" })._id;
let movie3 = db.movies.findOne({ title: "Inglorious Basterds" })._id;
db.reviews.insertMany([
  { username: "Cinephile123", movie: movie1, rating: 5, review: "Un chef-d'œuvre de science-fiction !" },
  { username: "Cinephile123", movie: movie2, rating: 5, review: "Un des meilleurs films de super-héros de tous les temps !" },
```

```
  { username: "MovieFanatic", movie: movie3, rating: 4, review:
"Visuellement impressionnant, mais un peu compliqué." }
])
```

Insertion des Commentaires sur les critiques

```
let review1 = db.reviews.findOne({ review: "Un chef-d'œuvre de
science-fiction !" })._id;
let review2 = db.reviews.findOne({ review: "Un des meilleurs films de
super-héros de tous les temps !" })._id;


db.comments.insertMany([
  { username: "MovieFanatic", comment: "Je suis totalement
d'accord !", review: review1 },
  { username: "Cinephile123", comment: "Merci pour ton avis !",
review: review2 }
])
```

9. Requêtes sur les Relations

Rechercher tous les utilisateurs :

```
db.users.find()
```

Rechercher toutes les critiques :

```
db.reviews.find()
```

Rechercher les critiques de Cinephile123 :

```
db.reviews.find({ username: "Cinephile123" })
```

Rechercher les critiques pour `Fight Club` **:**

```
db.reviews.find({ movie: movie1 })
```

Rechercher tous les commentaires :

```
db.comments.find()
```

Rechercher les commentaires de MovieFanatic :

```
db.comments.find({ username: "MovieFanatic" })
```

Rechercher les commentaires sur la critique de `Fight Club` **:**

```
db.comments.find({ review: review1 })
```

1. Importer les données

Tout d'abord, assurez-vous d'avoir zips.json et utilisez la commande suivante pour importer les données :

mongoimport --db population --collection zipcodes --file ‘path to zips.json ‘ **(le paquet mongo tools doit être installer et ajouter au path : https://www.mongodb.com/try/download/database-tools)**

Ou bien l’ajouter depuis l’interface de MongoCompass

Puis, démarrez le shell MongoDB et sélectionnez la base de données :

mongosh
use population

2. Filtrer les codes postaux d'Atlanta

1. Avec `find()`

db.zipcodes.find({ city: "ATLANTA", state: "GA" })

2. Avec $match dans aggregate

```
db.zipcodes.aggregate([
  { $match: { city: "ATLANTA", state: "GA" } }
])
```

3. Nombre total de codes postaux à Atlanta

```
db.zipcodes.aggregate([
  { $match: { city: "ATLANTA", state: "GA" } },
  { $group: { _id: null, count: { $sum: 1 } } }
```

```
])
```

4. Population totale d'Atlanta

```
db.zipcodes.aggregate([
  { $match: { city: "ATLANTA", state: "GA" } },
  { $group: { _id: null, totalPopulation: { $sum: "$pop" } } }
])
```

5. Population totale par État

```
db.zipcodes.aggregate([
  { $group: { _id: "$state", totalPopulation: { $sum: "$pop" } } }
])
```

6. Trier par population décroissante

```
db.zipcodes.aggregate([
  { $group: { _id: "$state", totalPopulation: { $sum: "$pop" } } },
  { $sort: { totalPopulation: -1 } }
])
```

7. Les 3 États les plus peuplés

```
db.zipcodes.aggregate([
  { $group: { _id: "$state", totalPopulation: { $sum: "$pop" } } },
  { $sort: { totalPopulation: -1 } },
  { $limit: 3 }
```

```
])
```

8. Population totale par ville

```
db.zipcodes.aggregate([
  { $group: { _id: { city: "$city", state: "$state" },
totalPopulation: { $sum: "$pop" } } }
])
```

9. Trier les villes par population décroissante

```
db.zipcodes.aggregate([
  { $group: { _id: { city: "$city", state: "$state" },
totalPopulation: { $sum: "$pop" } } },
  { $sort: { totalPopulation: -1 } }
])
```

10. Les 3 villes les plus peuplées

```
db.zipcodes.aggregate([
  { $group: { _id: { city: "$city", state: "$state" },
totalPopulation: { $sum: "$pop" } } },
  { $sort: { totalPopulation: -1 } },
  { $limit: 3 }
])
```

11. Les 3 villes les plus peuplées du Texas

```
db.zipcodes.aggregate([
  { $match: { state: "TX" } },
  { $group: { _id: { city: "$city", state: "$state" },
totalPopulation: { $sum: "$pop" } } },
  { $sort: { totalPopulation: -1 } },
  { $limit: 3 }
])
```

12. Moyenne de la population des villes par État

```
db.zipcodes.aggregate([
  { $group: { _id: { city: "$city", state: "$state" }, cityPopulation:
{ $sum: "$pop" } } },
  { $group: { _id: "$_id.state", avgCityPopulation: { $avg:
"$cityPopulation" } } }
])
```

13. Les 3 États avec la plus grande population moyenne par ville

```
db.zipcodes.aggregate([
  { $group: { _id: { city: "$city", state: "$state" }, cityPopulation:
{ $sum: "$pop" } } },
  { $group: { _id: "$_id.state", avgCityPopulation: { $avg:
"$cityPopulation" } } },
  { $sort: { avgCityPopulation: -1 } },
  { $limit: 3 }
])
```

14. Utilisation de `$lookup` pour combiner les données des collections `users`, `posts`, et `comments`

1. Récupérer les informations d'un utilisateur avec ses posts

Supposons que vous souhaitiez récupérer les informations de chaque utilisateur avec les détails de leurs posts. Vous pouvez utiliser $lookup pour joindre la collection posts avec la collection users :

use mongo_practice

```
db.users.aggregate([
  { $lookup: {
    from: "posts",
    localField: "_id",
    foreignField: "userId",
    as: "user_posts"
  } }
])
```

2. Récupérer les posts avec leurs commentaires

Pour chaque post, vous pouvez récupérer les commentaires associés en utilisant $lookup avec la collection comments :

```
db.posts.aggregate([
  { $lookup: {
    from: "comments",
    localField: "_id",
    foreignField: "postId",
    as: "post_comments"
  } }
```

```
])
```

3. Récupérer les utilisateurs avec leurs posts et commentaires

Pour chaque utilisateur, vous pouvez récupérer tous ses posts et les commentaires associés à ces posts, en enchaînant deux $lookup :

```
db.users.aggregate([
  { $lookup: {
    from: "posts",
    localField: "_id",
    foreignField: "userId",
    as: "user_posts"
  } },
  { $unwind: "$user_posts" },
  { $lookup: {
    from: "comments",
    localField: "user_posts._id",
    foreignField: "postId",
    as: "user_posts_comments"
  } }
])
```

4. Récupérer les commentaires d'un post avec les informations de l'utilisateur

Vous pouvez récupérer les commentaires d'un post, puis ajouter des informations sur l'utilisateur qui a écrit chaque commentaire :

```
db.posts.aggregate([
  { $lookup: {
    from: "comments",
    localField: "_id",
    foreignField: "postId",
    as: "post_comments"
  } },
  { $unwind: "$post_comments" },
  { $lookup: {
    from: "users",
```

```
    localField: "post_comments.userId",
    foreignField: "_id",
    as: "comment_user_info"
  } }
])
```

5. Récupérer les posts avec les utilisateurs qui ont commenté

Pour chaque post, vous pouvez récupérer les utilisateurs qui ont commenté ce post :

```
db.posts.aggregate([
  { $lookup: {
    from: "comments",
    localField: "_id",
    foreignField: "postId",
    as: "post_comments"
  } },
  { $unwind: "$post_comments" },
  { $lookup: {
    from: "users",
    localField: "post_comments.userId",
    foreignField: "_id",
    as: "comment_users"
  } }
])
```

6. Récupérer tous les posts d'un utilisateur avec les commentaires et leurs auteurs

Vous pouvez récupérer les posts d'un utilisateur, avec tous les commentaires et les utilisateurs qui ont écrit ces commentaires :

```
db.users.aggregate([
  { $lookup: {
    from: "posts",
    localField: "_id",
    foreignField: "userId",
    as: "user_posts"
```

```
  } },
  { $unwind: "$user_posts" },
  { $lookup: {
    from: "comments",
    localField: "user_posts._id",
    foreignField: "postId",
    as: "user_posts_comments"
  } },
  { $unwind: "$user_posts_comments" },
  { $lookup: {
    from: "users",
    localField: "user_posts_comments.userId",
    foreignField: "_id",
    as: "comment_authors"
  } }
])
```

7. Compter le nombre de commentaires par post

Vous pouvez compter le nombre de commentaires pour chaque post, en utilisant $lookup et ensuite un agrégat avec $count :

```
db.posts.aggregate([
  { $lookup: {
    from: "comments",
    localField: "_id",
    foreignField: "postId",
    as: "post_comments"
  } },
  { $project: {
    postTitle: 1,
    commentCount: { $size: "$post_comments" }
  } }
])
```

8. Trouver les utilisateurs qui ont écrit des commentaires, avec les posts associés

En combinant les données des utilisateurs, posts, et commentaires, vous pouvez obtenir les utilisateurs qui ont commenté, ainsi que les posts auxquels ils ont répondu :

```
db.comments.aggregate([
  { $lookup: {
    from: "posts",
    localField: "postId",
    foreignField: "_id",
    as: "post_info"
  } },
  { $unwind: "$post_info" },
  { $lookup: {
    from: "users",
    localField: "userId",
    foreignField: "_id",
    as: "commenter_info"
  } }
])
```

3. Utilisation de JavaScript dans Mongosh

1. Variables JavaScript dans Mongosh

Dans Mongosh, vous pouvez utiliser des variables JavaScript pour stocker et manipuler des données.

Exemple :

```
let myCity = "ATLANTA";
let myState = "GA";
```

2. Fonction JavaScript pour récupérer des documents

Vous pouvez définir une fonction JavaScript dans Mongosh pour effectuer des actions répétitives.

Exemple :

```
function getPopulation(city, state) {
  return db.zipcodes.find({ city: city, state: state });
}
```

Ensuite, vous pouvez appeler cette fonction pour obtenir des résultats spécifiques :

```
getPopulation("ATLANTA", "GA");
```

3. Utilisation de JavaScript dans l'agrégation MongoDB

Vous pouvez aussi utiliser JavaScript dans les pipelines d'agrégation. Par exemple, créer une fonction JavaScript pour calculer une valeur spécifique pendant l'agrégation.

Exemple :

```
db.zipcodes.aggregate([
  {
    $project: {
```

```
      city: 1,
      state: 1,
      adjustedPopulation: {
        $multiply: ["$pop", 1.1]  // Augmente la population de 10%
      }
    }
  }
]);
```

4. Utilisation des `map` et `reduce` dans MongoDB

MongoDB permet d'utiliser les méthodes map et reduce de JavaScript pour effectuer des opérations de transformation de données.

Exemple d'utilisation de map et reduce :

```
let mapFunction = function() {
  emit(this.state, this.pop);
};

let reduceFunction = function(key, values) {
  return Array.sum(values);
};

db.zipcodes.mapReduce(mapFunction, reduceFunction);
```

Cela permet de réduire les populations par état.

5. Manipulation d'objets JavaScript dans Mongosh

Mongosh permet également de manipuler des objets JavaScript directement. Par exemple, vous pouvez créer des objets pour structurer des documents avant de les insérer dans la base de données.

Exemple :

```
let cityData = {
  city: "ATLANTA",
  state: "GA",
  pop: 498044
```

```
};
db.zipcodes.insertOne(cityData);
```

6. Calcul avec JavaScript dans un pipeline d'agrégation

Il est aussi possible d’utiliser des expressions JavaScript pour des calculs complexes dans un pipeline d'agrégation.

Exemple :

```
db.zipcodes.aggregate([
  {
    $project: {
      city: 1,
      state: 1,
      adjustedPopulation: {
        $function: {
          body: function(pop) {
            return pop * 1.1;
          },
          args: ["$pop"],
          lang: "js"
        }
      }
    }
  }
]);
```