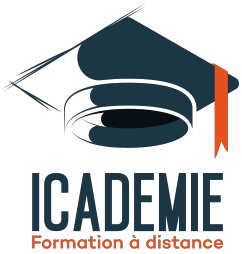

MongoDB

Samih Habbani

Table des matières

Introduction

Qu’est-ce que MongoDB?

MongoDB est un Système de Gestion de Base de Données (**SGBD**) de type **NoSQL** créé en 2007 par la société MongoDB Inc. Il est utilisé publiquement pour la première fois en 2009 pour stocker des données comme n’importe quel autre **SGBD**, de manière plus flexible et surtout plus évolutive.

En revanche, contrairement aux bases de données relationnelles traditionnelles comme **MySQL**, MongoDB stocke les données sous forme de « **documents** » au format « **BSON** » dans ce qu'on appelle des « **collections** ».

Chaque document ressemble alors à un objet **JSON**.

Qu’est-ce qu’un Document?

Un document en MongoDB est l’équivalent en MySQL d’un **enregistrement** dans une base de données relationnelle, aussi appelé « **jeu de données** ».

```json
{
  "_id": "248c7d550c7d747m4b42fefg",
  "name": "Samih Habbani",
  "age": 32,
  "email": "samihhabbani@gmail.com",
  "address": {
    "street": "40 rue Guynemer",
    "city": "Saint-Maur-des-Fossés",
    "country": "France"
  },
  "hobbies": ["Football", "Escalade", "Peinture"],
  "isStudent": true
}
```

Un document est constitué de champs.

D’ailleurs, chaque champ est en fait l’association entre un nom et une valeur, ce que l’on appellerait plus communément dans une base de données relationnelle, une « **colonne** ».

Qu’est-ce qu’une Collection?

Le terme de « **collection** » en MongoDB désigne quand à lui un « **groupe de documents** ». C’est un sorte de tableau créé comme l’on pourrait en retrouver en MySQL par exemple, à la différence qu’elle ne dispose pas de structure pré-définie à proprement parlé.

Voici un exemple dans lequel on insère des documents dans une collection :

javascript Copy code

```javascript
db.student.insertMany([
  {
    name: "Sam",
    age: "22",
    marks: 88,
    addmission: "2023-09-15",
    fees: "4000",
    projectCompleted: "true"
  },
  {
    name: "John",
    age: "28",
    marks: 91,
    addmission: "2023-02-01",
    fees: "2000",
    projectCompleted: "true"
  }
]);
```

Qu’est-ce qu’une base de données?

Une base de données MongoDB est quand à elle ce que l’on appel un « **conteneur de collections** ». On pourrait le comparer à un **Système de Gestion de Base de Données** (SGBD ou RDBMS en anglais) qui est un conteneur de tableaux pour les bases de données relationnelles comme MySQL.

Voici comment créer une base données en MongoDB en utilisant **Mongo Shell** via un terminal :

```javascript
use ma_base_de_donnees
```

Pourquoi utiliser MongoDB : Les features intéressantes.

Read-Only Views

MongoDB met à disposition des « **vues** » qui constituent des collections en lecture seule et peuvent utiliser les mêmes index que celles-ci.

En revanche, vous ne pourrez pas :

- modifier les index d'une collection à partir d'une vue basée sur celle-ci
- renommer une vue (vous devrez pour cela la détruire avec un DROP)

Le framework d'agrégation

MongoDB met également à disposition un puissant outil d'analyse et de traitement de l'information nommé « **pipeline d'agrégation** » également appelé « ***framework d'agrégation*** ».

Les vues en MongoDB sont alors ce que l'on appel un « pipline d'agrégation », en SQL on pourrait les comparer au « **tables virtuelles** ».

Les vues ne sont pas stockées sur le disque et sont calculées lorsque l'utilisateur effectue une requête.
Elles sont donc utilisées dans un objectif de démonstration.

Considérons une collection « **orders** » contenant les données suivantes :

```javascript
db.orders.insertMany([
  { _id: 1, customer_id: 1, total_amount: 45 },
  { _id: 2, customer_id: 2, total_amount: 20 },
  { _id: 3, customer_id: 3, total_amount: 150 }
])
```

Nous pouvons ensuite créer une « **vue agrégée** » pour obtenir le montant total par client :

```javascript
db.createView(
  "montant_total_par_client",  // Nom de la vue
  "orders",                    // Nom de la collection source
  [
    {
      $group: {
        _id: "$customer_id",    // Regroupement par client
        total_amount: { $sum: "$total_amount" } // Somme du montant total
      }
    }
  ]
)
```

On peut ensuite utiliser la vue comme n'importe quelle collection :

```javascript
db.total_amount_per_customer.find()
```

On Demand Materialized View

MongoDB propose en fait à ses utilisateurs deux types de vues différents à savoir « **les vues standard** » expliquées précédemment et « **les vues matérialisées à la demande** ».

Dans les deux cas, ces vues renvoient les résultats d'un « **pipeline d'agrégation** » mais revenons sur leur différences :

- Les vues standards **sont calculées lorsque l'utilisateur lie la vue** et ne sont pas stockées sur le disque.
- Les vues matérialisées à la demande **sont quand à elles stockées et lues sur le disque** et utilisent les étapes « **$merge** » ou « **$out** » pour mettre à jour les données enregistrées.

Une vue matérialisée à la demande est un résultat du pipeline d'agrégation pré-calculé qui est stocké et lu à partir du disque.

Les vues matérialisées à la demande sont généralement le résultat d'un « **$merge** » ou « **$out** » d'une étape (ou stage en anglais).

Autres différences :

Les vues standard **utilisent les index de la collection** sur laquelle elles se basent, il est donc impossible de créer, supprimer ou reconstruire directement des index sur votre vue, tout comme il sera impossible d'obtenir une liste d'index sur la vue.

Contrairement aux vues standard, **vous pouvez créer des index directement sur des vues matérialisées à la demande** car elles sont stockées sur disque.

Côté performance, l'avantage va aux vues matérialisées à la demande qui offrent de meilleures performances car lues à partir du disque et non calculées lors d'une requête comme avec une vue standard.

Les vues matérialisées ne sont pas disponibles nativement mais vous pouvez créer une pseudo-vue matérialisée en utilisant une collection classique et en mettant à jour cette collection à l'aide de scripts lorsqu'il y a des modifications dans les données source.

Cette collection fera office de vue matérialisée :

```javascript
db.createCollection(vue_materialisee)
```

Ce code MongoDB exécute une agrégation sur une collection source pour créer ou mettre à jour une vue matérialisée simulée à l'aide de l'opérateur « **$merge** ».

```javascript
// Exemple de script pour mettre à jour la vue matérialisée
db.source_collection.aggregate([
  { $group: { _id: "$group_field", count: { $sum: 1 } } },
  { $merge: { into: vue_materialisee, whenMatched: "merge", whenNotMatched: "insert" } }
])
```

1. **aggregate()** permet d'exécuter des opérations d'agrégation sur la collection source.
2. **{ $group: { _id: "$group_field", count: { $sum: 1 } } }:** L'opérateur **$group** pour regrouper les documents de la collection source selon la valeur du champ **group_field**. **$sum: 1** permet de compter les occurrences pour chaque groupe identifié par **_id**.
3. **{ $merge: { into: vue_materialisee, whenMatched: "merge", whenNotMatched: "insert" } }:** L'opérateur **$merge** met à jour les résultats de l'agrégation dans une pseudo-vue matérialisée nommée « **vue_materialisee** ».

L'option **whenMatched**: "**merge**" signifie que si un document correspondant existe déjà dans la vue, il sera mis à jour. **whenNotMatched**: "**insert**" signifie que si le document n'existe pas dans la vue, il sera inséré.

Peut-on réaliser un CRUD avec MongoDB?

La réponse est oui ! En MongoDB, les opérations CRUD vous permettront de **créer**, **lire**, **mettre à jour** et **supprimer des documents**.

CREATE

Les opérations **CREATE** ou **INSERT** vous permettront d'ajouter des documents à une collection. Si la collection n'existe pas, les opérations d'insert créerons la collection.

Voici les méthodes d'insertion :

- **db.collection.insertOne()**
- **db.collection.insertMany()**

READ

Les opérations de lecture récupère les documents d'une collection.

Voici la méthode de lecture :

- db.collection.find()

Vous pouvez d'ailleurs spécifier des filtres et critères pour identifier les documents à retourner. Nous verrons cela un peu plus tard dans la section agrégation de ce support.

UPDATE

Les opérations de mise à jour modifient les documents existants dans une collection

Voici les méthodes de mise à jour des données :

- db.collection.updateOne()

- db.collection.updateMany()

- db.collection.replaceOne()

Vous pouvez évidemment spécifier des critères ou des filtres pour identifier les documents à mettre à jour. Ces filtres utilisent d'ailleurs la même syntaxe que les opérations de lecture.

DELETE

Les opérations de suppression permettent de supprimer un ou plusieurs documents d'une collection.

Voici les méthodes de suppression des données :

- db.collection.deleteOne()

- db.collection.deleteMany()

Vous pouvez également spécifier des critères ou des filtres pour identifier les documents à supprimer. Ces filtres utilisent d'ailleurs la même syntaxe que les opérations de lecture ou de mise à jour.

Que sont les BSON Types et à quoi servent-ils?

Le **BSON** est un format d'échange de données utilisé pour stocker, formater et transférer des données par le réseau dans la base de données MongoDB.

Le format **BSON** est un format « **binaire** » qui permet de représenter des structures de données simples et des documents dans MongoDB.

Il est basé sur le nom du fameux format **JSON** et signifie « **Binary JSON** ».

Exemple de BSON Types :

```json
{
  "_id": "248c7d550c7d747m4b42fefg",
  "name": "Samih Habbani",
  "age": 32,
  "email": "samihhabbani@gmail.com",
  "address": {
    "street": "40 rue Guynemer",
    "city": "Saint-Maur-des-Fossés",
    "country": "France"
  },
  "hobbies": ["Football", "Escalade", "Peinture"],
  "isStudent": true
}
```

Les avantages :

1. Stockage binaire efficace
2. Types de données supplémentaires
3. Prise en charge des valeurs nulles
4. Compatibilité avec les langages de programmation
5. Traitement des champs complexes
6. Optimisation pour MongoDB

Installation et configuration

Installation de MongoDB Community Edition

Rdv sur la documentation officielle pour installer MongoDB Community Edition et choisir l'installation adaptée à votre système d'exploitation :

Install MongoDB Community Edition

NOTE

MongoDB Atlas

MongoDB Atlas is a hosted MongoDB service option in the cloud which requires no installation overhead and offers a free tier to get started.

These documents provide instructions to install MongoDB Community Edition.

Install on Linux

Install MongoDB Community Edition and required dependencies on Linux.

Install on macOS

Install MongoDB Community Edition on macOS systems from MongoDB archives.

Install on Windows

Install MongoDB Community Edition on Windows systems and optionally start MongoDB as a Windows service.

Install on Docker

Install a MongoDB Community Docker container.

Une fois installée, nous allons utiliser « **mongosh** » (démonstration sur un terminal UNIX/ LINUX sur Mac) pour interagir avec notre base de données MongoDB.

Quelle est la différence entre Service et Client ?

Le service MongoDB

Le service MongoDB fait référence au processus qui s'exécute sur un serveur pour gérer une base de données MongoDB.

Le service MongoDB est le moteur de la base de données qui stocke, récupère et manipule les données à l'intérieur.

Evidemment ce service peut être démarré, configuré ou encore arrêté en ligne de commandes via un terminal avec la commande « **mongod** ».

Attention !

- **mongod** est le processus principal du serveur MongoDB
- **mongo** est le shell du client pour interagir avec une instance MongoDB
- **mongosh** est également un shell pour interagir avec MongoDB, mais il s'agit d'une version améliorée par rapport au shell mongo classique. C'est celui que nous utiliserons pour les futures démonstrations

- Pour démarrer le service MongoDB :

```bash
mongod
```

- Pour ouvrir le Shell MongoDB :

```bash
mongosh
```

Le client MongoDB

Le client MongoDB est quand à lui l'application qui interagit avec le service MongoDB pour effectuer les différentes opérations énumérées plus tôt à savoir les opérations CRUD (Create -Read-Update-Delete) qui permettent de créer, lire, mettre à jour et encore supprimer un document.

Pour se connecter au service MongoDB, le client utilisent un driver ou une bibliothèque liée au langage de programmation du client.

Les différentes façons d'exécuter des commandes MongoDB

1. **Mongo Shell**
2. **Scripting avec le Shell :** Vous pouvez écrire des scripts JavaScript contenant des commandes MongoDB et les exécuter dans le shell avec la commande « **load()** ».
3. **Driver officiel :** Vous pouvez utiliser les pilotes officiels MongoDB avec votre langage de programmation pour écrire des applications qui interagissent avec une BDD MongoDB.
4. **Outils de gestion tiers :** Vous pouvez utiliser des outils tiers comme Compass, Robo 3T, Studio 3T, etc., offrent des interfaces utilisateur pour interagir avec les bases de données MongoDB.
5. **REST API :** Vous pouvez utiliser des services **RESTful** pour communiquer avec MongoDB.

Qui peut être un client MongoDB?

- Une applications web
- Une application mobile
- Des scripts qui utilisent des bibliothèques comme **pymongo** pour Python ou le driver **MongoDB** pour Node.js pour interagir avec la base de données.

Conclusion

En résumé, le service MongoDB est le moteur de base de données qui stocke les données, alors que le client MongoDB est l'application ou le code qui communique avec ce service pour persister des données en base de données.

Configurations

Exécuter MongoDB avec un utilisateur dédié

- **Créer un utilisateur MongoDB**

```bash
sudo adduser nom_utilisateur
```

- **Donner les autorisations nécessaires**

```bash
sudo usermod -aG mongodb nom_utilisateur
```

- **Modifier les autorisations des fichiers de MongoDB**

Assurez-vous que les fichiers de MongoDB sont accessibles en écriture pour cet utilisateur.

```bash
sudo chown -R nom_utilisateur:nom_utilisateur /var/lib/mongodb
sudo chown -R nom_utilisateur:nom_utilisateur /var/log/mongodb
```

Par ailleurs, assurez-vous de remplacer **/var/lib/mongodb** et **/var/log/mongodb** par les répertoires réels où MongoDB stock ses données et journaux.

Lancer MongoDB avec l'utilisateur dédié

Modifiez le fichier de configuration de MongoDB pour démarrer le service en utilisant l'utilisateur dédié :

Le fichier de configuration se trouve généralement dans **/etc/mongod.conf**

```yaml
security:
  authorization: enabled # assurez-vous que l'authentification est activé

processManagement:
  runAsUser: nom_utilisateur # Spécifiez l'utilisateur pour exécuter le s
```

Redémarrez ensuite le service MongoDB

```bash
sudo systemctl restart mongod
```

Désactiver les status HTTP

Il faut s'avoir qu'en MongoDB, les statuts HTTP sont liés à l'interface REST, qui permet d'interagir avec MongoDB via des requêtes HTTP. La désactivation des statuts HTTP dans MongoDB peut être recommandée pour des raisons de sécurité.

Par défaut, cette fonctionnalité est désactivée dans les versions récentes de MongoDB.

Si en revanche vous utilisez une version plus ancienne de MongoDB, voici la configuration à effectuer pour désactiver explicitement l'interface REST :

- Ouvrez le fichier de configuration de MongoDB (habituellement **/etc/mongod.conf**).
- Recherchez une ligne ressemblant à celle-ci :

```yaml
net:
  http:
    enabled: true
```

- Puis procéder à sa désactivation en effectuant la modification suivante :

```yaml
net:
  http:
    enabled: false
```

- Il ne vous restera plus qu'à redémarrer le service de MongoDB en procédant à la commande suivante :

```bash
sudo systemctl restart mongod
```

Désactiver le SSS (Serveur Side Scripting)

Le scripting côté serveur est une technique utilisée dans le développement web qui consiste à utiliser des scripts sur un serveur web pour produire une réponse personnalisée à la requête de chaque utilisateur (client) sur le site web.

Les opérations MongoDB suivantes vous permettent par exemple d'exécuter des expressions JavaScript directement sur le serveur :

- **$where**
- **mapReduce**
- **group**

Ces méthodes peuvent être pratiques, mais elles présentent un risque de sécurité pour l'intégrité de votre base de données si votre application n'échappe pas correctement les valeurs fournies par l'utilisateur.

Vous pouvez exprimer la plupart des requêtes dans MongoDB sans JavaScript , il est donc souvent plus judicieux de désactiver Javascript côté serveur :

- Pour cela, ouvrez **/etc/mongod.confavec** dans votre éditeur de code et recherchez la section sécurité:

```
security:
    authorization: "enabled"
```

- Ajouter la ligne suivante dans la section security :

```
    javascriptEnabled: false
```

Manipulation de base de données

Création d'une base de données

- Pour créer une base de données veuillez ouvrir votre terminal et démarrez le service MongoDB:

:

```bash
mongod
```

- Ouvrir ensuite le Shell MongoDB :

```bash
mongosh
```

- Pour sélectionner ou créer un nouvelle base de données :

```javascript
use ma_base_de_donnees
```

- Pour vérifier la création de la base de données :

```javascript
show dbs
```

Switcher de bases de données

- Cette commande vous permettra de changer de base de données :

```javascript
use ma_base_de_donnees
```

- Pour vérifier le changement de base de données :

```javascript
db
```

Renommer une base de données

MongoDB ne propose pas une commande native pour renommer directement une base de données. Vous pouvez cependant :

- **Créer une nouvelle base de données avec le nom souhaité.**
- **Copier les données de l'ancienne base vers la nouvelle.**
- **Supprimer l'ancienne base de données si nécessaire.**

- Sauvegardez l'ancienne base de données

```bash
mongodump --db ancienNom --out /chemin/vers/le/dossier
```

- Créez une nouvelle base de données

```javascript
use nouveauNom;
```

- Restaurez la sauvegarde

```bash
mongorestore --db nouveauNom /chemin/vers/le/dossier/ancienNom
```

- Supprimez l'ancienne base de données

```javascript
use ancienNom;
db.dropDatabase();
```

Supprimer une base de données

- Pour supprimer une base de données, vous pouvez exécuter cette commande dans le Shell de MongoDB :

```javascript
use ancienNom;
db.dropDatabase();
```

Manipulation de collection

Création d'une collection

Pour créer une collection dans MongoDB, vous pouvez utiliser la méthode « **db.createCollection()** » dans l'interface de la ligne de commande ou via un pilote MongoDB dans un langage de programmation comme JavaScript.

- En ligne de commande :

```javascript
use ma_base_de_donnees;
db.createCollection("ma_collection");
```

Supprimer d'une collection

Pour supprimer une collection dans MongoDB, vous pouvez utiliser la méthode « **db.collection.drop()** » dans l'interface de la ligne de commande ou via un pilote MongoDB dans un langage de programmation comme JavaScript.

```javascript
db.ma_collection.drop();
```

Modifier une collection

Lorsque vous souhaitez modifier la structure d'une collection dans MongoDB il s'agit généralement d'ajouter ou de supprimer des champs dans les documents existants ou encore de renommer des champs, ou de modifier le type de données d'un champ.

Voici quelques opérations fréquentes qui permettent de modifier la structure d'une collection en MongoDB :

- Ajouter un champ à tous les documents :

```javascript
db.ma_collection.updateMany({}, { $set: { "nouveau_champ": "valeur_par_defaut" } });
```

- Supprimer un champ d'un ou plusieurs documents :

```javascript
db.ma_collection.updateMany({}, { $unset: { "ancien_champ": "" } });
```

- Renommer un champ dans tous les documents :

```javascript
db.ma_collection.updateMany({}, { $rename: { "ancien_champ": "nouveau_champ" } });
```

Modifier le type de données d'un champ

En MongoDB, l'opérateur « **$projec**t » est utilisé dans les opérations d'agrégation pour remodeler les documents en sortie de la requête en incluant, excluant ou en modifiant les champs existants.

L'opérateur « **$project** » vous permettra de créer de nouveaux champs, de renommer des champs existants ou encore d'exclure des champs et de réaliser des opérations de transformation sur les champs existants.

Il existe différentes façons de modifier le type de champ dans MongoDB.

Vous pouvez entre utiliser les opérateurs d'étape (stage) « **$convert** », « **$toInt** », « **$toDate** » et « **$toString** » dans la méthode **db.collection.aggregation()**.

Les opérateurs d'agrégation **$toInt**, **$toDate** et **$toString** sont les formes abrégées de **$convert**.

- Considérons ce document :

javascript Copy code

```javascript
db.student.insertMany([
  {
    name: "Sam",
    age: "22",
    marks: 88,
    addmission: "2023-09-15",
    fees: "4000",
    projectCompleted: "true"
  },
  {
    name: "John",
    age: "28",
    marks: 91,
    addmission: "2023-02-01",
    fees: "2000",
    projectCompleted: "true"
  }
]);
```

- Vous pouvez utiliser l'opérateur **$toInt** pour modifier le type de champ :

```javascript
db.student.aggregate([
  {
    $addFields: {
      fees: { $toInt: "$fees" }
    }
  }
]);
```

Ici, nous avons utilisé le pipeline d'agrégation avec l'étape **$addFields** qui ajoute un nouveau champ « **fees** » à chaque document (il remplace en fait le champ existant) dans la collection « **student** » en changeant le type de champ string à int à l'aide de l'opérateur d'agrégation **$toInt**.

- Vous pouvez utiliser l'opérateur **$toDate** pour passer au type Date :

```javascript
db.student.aggregate([
  {
    $project: {
      addmissionDate: {
        $toDate: "$addmission"
      }
    }
  }
]);
```

Pour remplacer un champ string contenant une date par un type de date, vous pouvez utiliser l'opérateur **$toDate**.

- Vous pouvez utiliser l'opérateur **$toString** pour passer au type String:

```javascript
db.student.aggregate([
  {
    $project: {
      marksString: {
        $toString: "$marks"
      }
    }
  }
]);
```

Vous pouvez utiliser l'opérateur d'agrégation **$toString** pour convertir un champ non string en string.

- Vous pouvez finalement utiliser l'opérateur **$toConvert** pour passer à n'importe quel type:

```javascript
db.student.aggregate([
  {
    $project: {
      _id: 1,
      age: {
        $convert: {
          input: "$age",
          to: "int"
        }
      }
    }
  },
  {
    $out: "student"
  }
]);
```

Mettre en place un Schéma de validation sur un collection

Un schéma de validation en MongoDB est une fonctionnalité introduite à partir de la version **3.2** qui permet de définir des règles de validation pour les documents insérés ou mis à jour dans une collection.

Un schéma de validation permet de garantir l'intégrité des données en imposant des contraintes sur la structure et le contenu des documents. On peut comparer le schéma de validation aux contraintes d'intégrité en SQL.

Le schéma de validation utilise des règles de validation définies lors de la création d'une collection qui peuvent inclure les contraintes suivantes :

- **Les champs obligatoires**
- **Le type de données des champs**
- **Les valeurs autorisées pour ces champs**
- **Les Regex ou expressions régulières**

Pour mettre en place un schéma de validation sur une collection MongoDB, vous pouvez utiliser la méthode **db.createCollection()** avec l'option « **validator** » pour spécifier les règles de validation sous forme de schéma JSON.

Exemple :

javascript Copy code

```javascript
db.createCollection("students", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      title: "Student Object Schema Validation",
      required: ["name", "age"],
      properties: {
        name: {
          bsonType: "string",
          description: "'name' must be a string and is required"
        },
        age: {
          bsonType: "int",
          minimum: 1,
          maximum: 120,
          description: "'age' must be an integer in [1, 120] and is required"
        },
        addmission: {
          bsonType: ["date"],
          description: "'addmission' must be a date if the field exists"
        }
      }
    }
  }
});
```

Explications :

- Les champs « **name** » et « **age** » sont requis dans chaque document.
- Le champ « **name** » doit être une chaîne de caractères.
- Le champ "« **age** » doit être un entier compris entre 1 et 120.
- Le champ « **addmission** », s'il existe, doit être de type date.

Pour plus d'informations concernant les schéma de validation en MongoDB, vous pouvez naturellement consulter la documentation officielle.

BSON Types

Comme vu précédemment, le format **BSON** (**Binary JSON**) est le format de stockage des données utilisé par MongoDB pour représenter les documents de données dans une forme binaire.

Le format BSON est similaire au format **JSON**, mais il ajoute des types de données supplémentaires pour représenter des données plus complexes, en MongoDB.

Voici d'ailleurs les Types BSON principaux que l'on peut retrouver en MongoDB :

- **Object ID (ID d'objet) :** Identifiant unique généré par MongoDB pour chaque document.
- **Double (Nombre à virgule flottante) :** Données numériques à virgule flottante en JavaScript ou en JSON.
- **String (Chaîne de caractères) :** Représente une séquence de caractères Unicode.
- **Object (Objet) :** Correspond à un objet JSON.
- **Array (Tableau) :** Représente un tableau JSON.
- **Binary Data (Données binaires) :** Stocke des données binaires telles que des images ou d'autres types de fichiers.
- **Boolean (Booléen) :** Représente les valeurs booléennes (true ou false).
- **Date (Date) :** Stocke la date et l'heure au format UTC.
- **Null (Nul) :** Représente une valeur nulle.
- **Regular Expression (Expression régulière) :** Représente une expression régulière utilisée pour des opérations de recherche.
- **JavaScript Code (Code JavaScript) :** Stocke du code JavaScript.

Pour plus d'informations concernant les types BSON en MongoDB, vous pouvez naturellement consulter la documentation officielle.

Les documents

Structure de document

En MongoDB, un document est une unité de stockage de données, similaire à un enregistrement (« jeu de données ») ou à une ligne en SQL par exemple.

Il est représenté sous forme de **BSON** (Binary JSON), qui est une extension du format **JSON** avec des types de données supplémentaires pour gérer des informations plus complexes qu'en SQL.

- Voici un exemple :

```json
{
  "_id": "248c7d550c7d747m4b42fefg",
  "name": "Samih Habbani",
  "age": 32,
  "email": "samihhabbani@gmail.com",
  "address": {
    "street": "40 rue Guynemer",
    "city": "Saint-Maur-des-Fossés",
    "country": "France"
  },
  "hobbies": ["Football", "Escalade", "Peinture"],
  "isStudent": true
}
```

- Décomposition de ce document :

  - **"_id": "248c7d550c7d747m4b42fefg" :** C'est l'identifiant unique du document (caractères hexadécimale) généré par MongoDB automatiquement pour identifier de manière unique ce document dans une collection.
  - **"name": "Samih Habbani" :** Ce champ représente le nom de la personne associée à ce document. (String)
  - **"age": 32 :** C'est le champ pour l'âge de la personne. (Integer)
  - **"email": « samihhabbani@gmail.com" :** Champ pour l'adresse e-mail de la personne. (String)

- **"address": { "street": "40 rue Guynemer", "city": "Saint-Maur-des-Fossés", "country": "France" } :** Champ pour l'adresse, représenté comme un objet JSON imbriqué avec des sous-champs pour la rue, la ville et le pays.
- **"hobbies": ["Football", "Escalade", "Peinture"] :** Un tableau représentant les hobbies de la personne.
- **"isStudent": true :** Champ booléen pour indiquer si la personne est étudiante.

Manipulation de Document

Création

Pour créer un document en MongoDB, vous devez procéder aux étapes suivantes :

- Ouvrez un terminal ou une console et ouvrez le shell MongoDB :

```bash
mongosh
```

- Connectez-vous à votre base de données MongoDB :
- Sélectionnez votre base de données :

```bash
use mydatabase
```

- Créez une collection :

```javascript
db.createCollection("mycollection");
```

- Insérer votre document dans la collection :

```json
db.mycollection.insertOne({
  "name": "Samih Habbani",
  "age": 32,
  "email": "samihhabbani@gmail.com",
  "address": {
    "street": "40 rue Guynemer",
    "city": "Saint-maur-des-Fossés",
    "country": "France"
  },
  "hobbies": ["Escalade", "Boxe"],
  "isStudent": true
})
```

Lecture

Pour lire un document dans MongoDB, vous pouvez utiliser des commandes spécifiques dans le shell MongoDB ou des requêtes dans votre application MongoDB en utilisant le langage ou le pilote de programmation de votre choix, comme JavaScript que nous avons mentionné précédemment.

- Dans le cas ou vous souhaitez lire tous les documents :

```javascript
db.mycollection.find()
```

Cette commande renverra tous les documents de la collection « **mycollection** ».

- Dans le cas ou vous souhaitez lire un document :

```javascript
db.mycollection.findOne({ "name": "Samih Habbani" })
```

Cette commande retourne le premier document dans la collection « **mycollection** » où le champ « name » est égal à « Samih Habbani ». Si aucun document ne correspond à ce critère, la commande renverra « **null** ».

Agrégation

L'agrégation dans MongoDB est un processus permettant de traiter et d'analyser les données en effectuant des opérations sur plusieurs documents et en retournant un résultat calculé. Cela permet d'effectuer des opérations de calcul, de transformation et de regroupement des données de manière simple et efficace en MongoDB.

Voici une liste des opérations les plus fréquemment utilisées en MongoDB :

- **$match** : Filtre les documents en fonction de critères spécifiques.

```javascript
db.students.aggregate([
  {
    $match: { age: { $gt: 20 } }
  }
])
```

Cette requête « **aggregate** » avec l'opérateur **$match** retournera tous les documents de la collection « **students** » où l'âge des étudiants est supérieur à 20 ans.

$gt est un opérateur de comparaison en MongoDB qui signifie « **greater than** » (supérieur à) et permet de comparer si une valeur est strictement supérieure à une autre lors d'une requête.

Il existe évidemment d'autres opérateurs de comparaison, voici une liste non exhaustive :

- **$eq** : Égal à.
- **$ne** : Non égal à.
- **$gt** : Supérieur à.
- **$lt** : Inférieur à.
- **$gte** : Supérieur ou égal à.
- **$lte** : Inférieur ou égal à.
- **$in** : Dans. Sélectionne les documents où la valeur du champ est égale à l'une des valeurs spécifiées dans un tableau.
- **$nin** : Pas dans. Sélectionne les documents où la valeur du champ n'est pas égale à l'une des valeurs spécifiées dans un tableau.

Pour plus d'informations concernant les opérateurs de comparaison en MongoDB, vous pouvez naturellement consulter *la documentation officielle*.

- **$group** : Regroupe les documents en fonction de certaines clés et effectue des opérations d'agrégation sur les données regroupées.

EXEMPLE :

Imaginez que nous souhaitions trouver le total de stock disponible pour chaque catégorie de produits en regroupant les documents par catégorie à partir de la collection « **products** ».

Nous pourrions pour cela effectuer une opération d'agrégation pour calculer la somme du champ « **stock** » :

javascript Copy code

```javascript
db.products.aggregate([
  {
    $group: {
      _id: "$categorie",
      totalStock: { $sum: "$stock" }
    }
  }
])
```

- **$project** : Modifie la structure des documents en spécifiant les champs à inclure ou exclure, en ajoutant de nouveaux champs calculés, etc :

```javascript
db.student.aggregate([
  {
    $project: {
      addmissionDate: {
        $toDate: "$addmission"
      }
    }
  }
]);
```

Ici, nous avons utilisé l'opérateur **$project** dans une opération d'agrégation modifiant le type d'un champs existant.

Nous avons remplacé un champ string contenant une date par un type de date, en utilisant l'opérateur **$toDate**.

- **$sort** : Trie les documents en fonction de critères spécifiques.

```javascript
db.students.aggregate([
  {
    $sort: { age: -1 }
  }
])
```

Ici nous avons toujours notre collection « **students** » et nous voulons récupérer les étudiants triés par ordre décroissant d'âge.

- **$limit** et **$skip** : Limite ou saute un certain nombre de documents dans les résultats de l'agrégation.

```javascript
db.students.aggregate([
  {
    $skip: 2
  }
])
```

Ici nous avons toujours notre collection « **students** » et nous voulons récupérer les documents après les deux premiers documents c'est à dire en ignorant les deux premiers documents.

```javascript
db.students.aggregate([
  {
    $limit: 3
  }
])
```

Ici nous avons toujours notre collection « **students** » et nous voulons récupérer seulement les 3 premiers documents.

- **$unwind** : Décompose un champ avec une valeur de type tableau (array) dans plusieurs documents pour faciliter l'agrégation.

Imaginons que nous avons une collection « **orders** » avec des documents représentant des commandes, et un champ « **items** » de type tableau contenant des éléments à l'intérieur :

Pour déconstruire le champ « **items** » pour obtenir un document par élément de chaque tableau, nous pouvons donc utiliser « **$unwind** » :

javascript Copy code

javascript Copy code

```
db.orders.aggregate([
  {
    $unwind: "$items"
  }
])
```

```
    ]
  },
  {
    "_id": 2,
    "order_id": "02",
    "items": [
      { "product": "Produit 1", "quantity": 7 },
      { "product": "Produit 4", "quantity": 6 }
    ]
  }
])
```

Pour plus d'informations concernant les opérations d'agrégation en MongoDB, vous pouvez naturellement consulter la documentation officielle.

MapReduce

En MongoDB, le **Map-Reduce** est un paradigme de traitement de données permettant de condenser de grands volumes de données en résultats agrégés utiles.

Pour effectuer des opérations de **Map-Reduce**, MongoDB propose la commande de base de données « **mapReduce** ».

Pour comprendre l'utilité de cette commande, considérons l'exemple ci-dessous :

- Nous avons créé ci-dessous une collection nommée « **orders** » avec 10 documents à l'intérieur, chaque document représentant évidemment une commande passée.

```javascript
db.orders.insertMany([
    { _id: 1, customer_id: "Samih Habbani", price: 35, items: [{ item: "oranges", qty: 5, price: 3.5 }, { item: "apples", qty: 5, price: 3.5 }], status: "INP" },
    { _id: 2, customer_id: "Samih Habbani", price: 40, items: [{ item: "oranges", qty: 8, price: 3.5 }, { item: "chocolates", qty: 5, price: 7 }], status: "INP" },
    { _id: 3, customer_id: "Mathieu Lambert", price: 30, items: [{ item: "oranges", qty: 7, price: 3.5 }, { item: "pears", qty: 7, price: 3.5 }], status: "INP" },
    { _id: 4, customer_id: "Mathieu Lambert", price: 15, items: [{ item: "oranges", qty: 7, price: 3.5 }], status: "INP" },
    { _id: 5, customer_id: "Mathieu Lambert", price: 20, items: [{ item: "chocolates", qty: 5, price: 7 }], status: "INP" },
    { _id: 6, customer_id: "Tim Burd", price: 65, items: [{ item: "carrots", qty: 7, price: 2.0 }, { item: "apples", qty: 7, price: 3.5 }], status: "INP" },
    { _id: 7, customer_id: "Tim Burd", price: 85, items: [{ item: "oranges", qty: 7, price: 3.5 }], status: "INP" },
    { _id: 8, customer_id: "Don Louis", price: 15, items: [{ item: "chocolates", qty: 5, price: 7 }, { item: "apples", qty: 7, price: 3.5 }], status: "INP" },
    { _id: 9, customer_id: "Don Louis", price: 25, items: [{ item: "carrots", qty: 5, price: 2.0 }, { item: "apples", qty: 7, price: 3.5 }], status: "INP" },
    { _id: 10, customer_id: "Don Louis", order_date: new Date("2023-03-23"), price: 15, items: [{ item: "oranges", qty: 7, price: 3.5 }], status: "INP" }
])
```

Nous souhaiterions désormais retourner le prix total par client (« **customer** ») :

Pour cela, nous allons exécuter une opération **map-reduce** dans la collection « **orders** » pour regrouper par « **customer_id** » et calculer la somme des « **price** » pour chaque « **customer_id** ».

- Pour cela nous allons définir une « **fonction map** », pour traiter chaque document d'entrée:

```javascript
var mapFunction = function() {
    emit(this.customer_id, this.price);
};
```

- Dans cette fonction, « **this** » fait référence au document que le map-reduce traite
- La fonction ici map « **price** » à « **customer_id** » pour chaque document et émet un tuple avec une relation clé-valeur avec « **customer_id** » et « **price** ».

Nous allons ensuite définir la fonction « **reduce** » avec deux arguments, « **keyCustId** » et « **valuesPrices** »

```javascript
var reduceFunction = function(keyCustId, valuesPrices) {
    return Array.sum(valuesPrices);
};
```

- « **valuesPrices** » est un tableau dont les éléments sont les valeurs des « **price** » **émisent** par la fonction map « **mapFunction** » regroupées par « **keyCustId** ».
- La fonction « **reduceFunction** » réduit l'array « **valuesPrices** » à la somme de ses éléments

• Pour exécuter le **map-reduce** à tous les documents dans la collection « **orders** » en utilisant « **mapFunction** » et la fonction « **reduceFunction** », voici le code a exécuter :

```javascript
db.orders.mapReduce(
   mapFunction,
   reduceFunction,
   { out: "exemple_map_reduce" }
)
```

• Vous pouvez maintenant récupérer la collection « **exemple_map_reduce** » pour vérifier les résultats :

```javascript
db.exemple_map_reduce.find().sort({ _id: 1 })
```

• Voici les résultats récupérés :

```
MA_BDD> db.exemple_map_reduce.find().sort( { _id: 1 } )
[
  { _id: 'Don Louis', value: 55 },
  { _id: 'Mathieu Lambert', value: 65 },
  { _id: 'Samih Habbani', value: 75 },
  { _id: 'Tim Burd', value: 150 }
]
MA_BDD>
```

Modification

Pour modifier un document dans MongoDB, vous pouvez utiliser plusieurs méthodes :

- « **update()** » ou « **updateOne()** » pour mettre à jour un document spécifique
- « **updateMany()** » pour mettre à jour plusieurs documents.

• **« update() »**

Considérons le document suivant dans une collection nommée « **users** »

```json
{
  "_id": 1,
  "name": "Samih HABBANI",
  "age": 32,
  "email": "samihhabbani@gmail.com"
}
```

Pour **modifier l'âge de Samih à 33 ans**, voici le code à exécuter :

```javascript
db.users.updateOne(
  { "_id": 1 },
  { $set: { "age": 33 } }
)
```

Ce code mettra à jour le document où le champ « **_id** » est égal à 1 en remplaçant la valeur de l'âge par 33 dans la collection « **users** ».

- « **updateMany()** »

Supposons maintenant que vous souhaitiez mettre à jour l'âge de tous les users ayant moins de 32 ans et passer leur âge à 33 ans.

Voici le code à mettre en place dans ce cas de figure :

```javascript
db.users.updateMany(
  { "age": { $lt: 32 } },
  { $set: { "age": 33 } }
)
```

Suppression

Au même titre que pour modifier un document dans MongoDB, vous pouvez utiliser plusieurs méthodes de suppression pour supprimer un document :

- « **deleteOne()** » pour supprimer un document spécifique
- « **deleteMany()** » pour supprimer plusieurs documents.

- « **deleteOne()** »

```javascript
db.users.deleteOne({ "_id": 1 })
```

La requête ci-dessous supprimera le document de la collection « **users** » ayant le champ « **id** » égal à 1.

- « **deleteMany()** »

```javascript
db.users.deleteMany({ "age": { $lt: 30 } })
```

La requête ci-dessous supprimera quant à elle tous les documents de la collection « **users** » ayant un âge inférieur à 30 ans.

Bulk Writing

Le **Bulk Writing** en MongoDB est une méthode qui permet d'exécuter des opérations massives sur une base de données, comme des insertions, des mises à jour, des suppressions dans une seule opération.

Cette méthode évite d'exécuter chaque opération individuellement, en regroupant ces opérations en un seul lot (que l'on appel **bulk** en anglais) pour les exécuter plus efficacement.

Évidemment, la méthode **bulkWrite()** permettant cette technique supporte les opérations suivantes :

- **insertOne**
- **updateOne**

- **UpdateMany**
- **replaceOne**
- **deleteOne**
- **deleteMany**

Exemple :

Considérons une collection « **pizzas** » :

```javascript
db.pizzas.insertMany([
   { _id: 0, type: "pepperoni", size: "petit", price: 9 },
   { _id: 1, type: "fromage", size: "moyen", price: 12 },
   { _id: 2, type: "vegan", size: "large", price: 14 }
]);
```

La commande **bulkWrite()** ci-dessous effectue les opérations suivantes :

- ajoute deux documents en utilisant « **insertOne** »
- Mets à jour un document en utilisant « **updateOne** »
- Supprime un document en utilisant « **deleteOne** »
- Remplace un document en utilisant « **replaceOne** »

Ce code réalise différentes opérations sur la collection pizzas :

1. **insertOne**: Ajoute deux nouvelles pizzas de types "**boeuf**" et "**poulet**" avec leurs tailles et prix respectifs.
2. **updateOne**: Met à jour le prix des pizzas de type "**fromage**" et le change pour 6€.

```javascript
try {
   db.pizzas.bulkWrite( [
      { insertOne: { document: { _id: 3, type: "boeuf", size: "moyen", price: 16 } } },
      { insertOne: { document: { _id: 4, type: "poulet", size: "large", price: 20 } } },
      { updateOne: {
         filter: { type: "fromage" },
         update: { $set: { price: 6 } }
      } },
      { deleteOne: { filter: { type: "pepperoni"} } },
      { replaceOne: {
         filter: { type: "vegan" },
         replacement: { type: "tofu", size: "petit", price: 14 }
      } }
   ] )
} catch( error ) {
   print( error )
}
```

3. **deleteOne**: Supprime une pizza de type "**pepperoni**".
4. **replaceOne**: Remplace la pizza de type "**vegan**" par une nouvelle pizza de type "**tofu**" de taille "**petit**" avec un prix de **14**.

Pour plus d'informations concernant la méthode de Bulk Writing en MongoDB, vous pouvez naturellement consulter *la documentation officielle*.

Création d'indexes pour optimiser les recherches

Dans MongoDB, les index sont des structures de données qui permettent d'optimiser les requêtes en fournissant un accès rapide aux documents dans une collection.

Ils permettent à MongoDB de localiser plus efficacement les documents correspondants à une requête donnée.

Il existe évidemment plusieurs types d'index :

1. **Index unique :** Garantit que les valeurs d'un champ (ou d'une combinaison de champs) sont uniques pour chaque document dans une collection.
2. **Index multiple :** Permet de créer un index sur plusieurs champs
3. **Index textuel :** Utilisé pour la recherche de texte intégral dans les chaînes de caractères.
4. **Index géospatial :** Optimisé pour la recherche spatiale et la géolocalisation.
5. **Index hashed :** Hache une ou plusieurs clés et stocke les résultats dans l'index.

Ces Index présentent de nombreux avantages :

- **Ils accélèrent les requêtes**
- **Améliore les performances**
- **Réduise les temps de lecture**
- **Optimise les jointures**

Création d'un index en MongoDB

Imaginez une collection.nommée « **users** » et que nous souhaitions créer un index sur le champ « **email** » pour accélérer les requêtes basées sur l'adresse email des utilisateurs.

```javascript
db.users.createIndex({ "email": 1 })
```

Cette commande crée un index sur le champ « **email** » de la collection « **users** ». Le 1 indique un index croissant (tri dans un ordre croissant dans l'index)

Utilisation des index en MongoDB

```javascript
db.users.find({ "email": "samihhabbani@gmail.com" })
```

L'index sur le champ « **email** » accélérera cette recherche en localisant plus rapidement les documents correspondant à cette adresse e-mail.

Vérification des index créés :

```javascript
db.users.getIndexes()
```

Cette commande affichera tous les indexes créés pour la collection « **users** ».

Grâce à cette index, nous avons donc pu améliorer les performances des requêtes qui utilisent le champ « **email** » pour filtrer ou trier les données.