eBook Gratuit

APPRENEZ MongoDB

eBook gratuit non affilié créé à partir des **contributeurs de Stack Overflow.**

#mongodb

Table des matières

À propos

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: mongodb

It is an unofficial and free MongoDB ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official MongoDB.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapitre 1: Démarrer avec MongoDB

Remarques

- Les données dans le monde ont commencé à croître énormément après la mise sur le marché des applications mobiles. Cette énorme quantité de données est devenue presque impossible à gérer avec la base de données relationnelle traditionnelle - SQL. Des bases de données NoSQL sont introduites pour gérer les données pour lesquelles beaucoup plus de flexibilité est apparue comme un nombre variable de colonnes pour chaque donnée.
- MongoDB est l'une des principales bases de données NoSQL. Chaque collection contient un certain nombre de documents JSON. Tout modèle de données pouvant être exprimé dans un document JSON peut être facilement stocké dans MongoDB.
- MongoDB est une base de données serveur-client. Le serveur s'exécute généralement avec le fichier binaire `mongod` et le client s'exécute avec `mongo` .
- Il n'y a pas d'opération de jointure dans MongoDB avant la v.3.2, pour diverses raisons philosophiques et pragmatiques . Mais le shell Mongo supporte le javascript, donc si $ lookup n'est pas disponible, on peut simuler des opérations de jointure sur des documents en javascript avant de les insérer.
- Pour exécuter une instance dans un environnement de production, il est fortement recommandé de suivre la liste de contrôle des opérations .

Versions

Version	Date de sortie
3.4	2016-11-29
3.2	2015-12-08
3.0	2015-03-03
2.6	2014-04-08
2.4	2013-03-19
2.2	2012-08-29
2.0	2011-09-12
1.8	2011-03-16
1.6	2010-08-31
1.4	2010-03-25
1.2	2009-12-10

Examples

Installation

Pour installer MongoDB, suivez les étapes ci-dessous:

- **Pour Mac OS:**
  - Il existe deux options pour Mac OS: installation manuelle ou homebrew .
  - **Installation avec homebrew :**
    - Tapez la commande suivante dans le terminal:

```
$ brew install mongodb
```

  - **Installation manuelle:**
    - Téléchargez la dernière version ici . Assurez-vous que vous téléchargez le fichier approprié, vérifiez particulièrement si votre type de système d'exploitation est 32 bits ou 64 bits. Le fichier téléchargé est au format `tgz` .
    - Accédez au répertoire dans lequel ce fichier est téléchargé. Ensuite, tapez la commande suivante:

```
$ tar xvf mongodb-osx-xyz.tgz
```

      Au lieu de `xyz` , il y aurait des informations sur la version et le type de système. Le dossier extrait porterait le même nom que le fichier `tgz` . Dans le dossier, il y aurait un sous-dossier nommé `bin` qui contiendrait plusieurs fichiers binaires avec `mongod` et `mongo` .
    - Par défaut, le serveur conserve les données dans le dossier `/data/db` . Donc, nous devons créer ce répertoire, puis exécuter le serveur avec les commandes suivantes:

```
$ sudo bash
# mkdir -p /data/db
# chmod 777 /data
# chmod 777 /data/db
# exit
```

    - Pour démarrer le serveur, la commande suivante doit être donnée à partir de l'emplacement actuel:

```
$ ./mongod
```

      Il lancerait le serveur sur le port 27017 par défaut.
    - Pour démarrer le client, un nouveau terminal doit être ouvert avec le même

répertoire que précédemment. Ensuite, la commande suivante démarre le client et se connecte au serveur.

```
$ ./mongo
```

Par défaut, il se connecte à la base de données de `test` . Si vous voyez la ligne comme se `connecting to: test` . Ensuite, vous avez installé MongoDB avec succès. Félicitations! Maintenant, vous pouvez tester Hello World pour être plus confiant.

- **Pour les fenêtres:**
  - Téléchargez la dernière version ici . Assurez-vous que vous téléchargez le fichier approprié, vérifiez particulièrement si votre type de système d'exploitation est 32 bits ou 64 bits.
  - Le fichier binaire téléchargé a l'extension `exe` . Exécuter. Il vous demandera un assistant d'installation.
  - Cliquez sur **Suivant** .
  - **Acceptez** le contrat de licence et cliquez sur **Suivant** .
  - Sélectionnez Installation **complète** .
  - Cliquez sur **Installer** . Il peut demander une fenêtre pour demander l'autorisation de l'administrateur. Cliquez sur **Oui** .
  - Après l'installation, cliquez sur **Terminer** .
  - Maintenant, le mongodb est installé sur le chemin `C:/Program Files/MongoDB/Server/3.2/bin` . Au lieu de la version 3.2, il pourrait y avoir une autre version pour votre cas. Le nom du chemin serait modifié en conséquence.
  - `bin` répertoire `bin` contient plusieurs fichiers binaires avec `mongod` et `mongo` . Pour l'exécuter à partir d'un autre dossier, vous pouvez ajouter le chemin dans le chemin du système. Pour le faire:
    - Cliquez avec le bouton droit sur **Poste de travail** et sélectionnez **Propriétés** .
    - Cliquez sur **Paramètres système avancés** dans le volet gauche.
    - Cliquez sur **Variables d'environnement ...** sous l'onglet **Avancé** .
    - Sélectionnez **Path** from **System variables** section et cliquez sur **Edit ....**
    - Avant Windows 10, ajoutez un point-virgule et collez le chemin indiqué ci-dessus. À partir de Windows 10, il existe un bouton **Nouveau** pour ajouter un nouveau chemin.
    - Cliquez sur **OK** pour enregistrer les modifications.
  - Maintenant, créez un dossier nommé `data` contenant un sous-dossier nommé `db` où vous souhaitez exécuter le serveur.

- Démarrer l'invite de commande depuis leur. Soit changer le chemin dans cmd ou en cliquant sur **Ouvrir la fenêtre de commande ici** qui serait visible après un clic droit sur l'espace vide de l'interface graphique du dossier en appuyant sur la touche Maj et Ctrl ensemble.
- Écrivez la commande pour démarrer le serveur:

```
> mongod
```

  Il lancerait le serveur sur le port 27017 par défaut.

- Ouvrez une autre invite de commande et tapez ce qui suit pour démarrer le client:

```
> mongo
```

- Par défaut, il se connecte à la base de données de `test` . Si vous voyez la ligne comme se `connecting to: test` . Ensuite, vous avez installé MongoDB avec succès. Félicitations! Maintenant, vous pouvez tester Hello World pour être plus confiant.

- **Pour Linux:** Presque identique à Mac OS sauf qu'une commande équivalente est nécessaire.
  - Pour les distributions basées sur Debian (en utilisant `apt-get`):
    - Importer la clé du référentiel MongoDB.

```
$ sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv EA312927
gpg: Total number processed: 1\
gpg:               imported: 1  (RSA: 1)
```

    - Ajouter un référentiel à la liste de paquets sur **Ubuntu 16.04** .

```
$ echo "deb http://repo.mongodb.org/apt/ubuntu xenial/mongodb-org/3.2
multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-3.2.list
```

    - sur **Ubuntu 14.04** .

```
$ echo "deb http://repo.mongodb.org/apt/ubuntu trusty/mongodb-org/3.2
multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-3.2.list
```

    - Mettre à jour la liste des paquets.

```
$ sudo apt-get update
```

    - Installez MongoDB.

```
$ sudo apt-get install mongodb-org
```

  - Pour les distributions basées sur Red Hat (en utilisant `yum`):

- utilisez un éditeur de texte que vous préférez.

  $ vi /etc/yum.repos.d/mongodb-org-3.4.repo

- Coller le texte suivant.

```
[mongodb-org-3.4]
name=MongoDB Repository
baseurl=https://repo.mongodb.org/yum/redhat/$releasever/mongodb-
org/3.4/x86_64/
gpgcheck=1
enabled=1
gpgkey=https://www.mongodb.org/static/pgp/server-3.4.asc
```

- Mettre à jour la liste des paquets.

```
$ sudo yum update
```

- Installez MongoDB

```
$ sudo yum install mongodb-org
```

Bonjour le monde

Après l'installation, les lignes suivantes doivent être entrées dans mongo shell (terminal client).

```
> db.world.insert({ "speech" : "Hello World!" });
> cur = db.world.find();x=cur.next();print(x["speech"]);
```

Bonjour le monde!

Explication:

- Dans la première ligne, nous avons inséré un document apparié `{ key : value }` dans le `test` base de données par défaut et dans la collection nommée `world` .
- Dans la deuxième ligne, nous récupérons les données que nous venons d'insérer. Les données extraites sont conservées dans une variable javascript nommée `cur` . Ensuite, par la fonction `next()` , nous avons récupéré le premier et le seul document et l'avons conservé dans une autre variable js nommée `x` . Ensuite, imprimé la valeur du document fournissant la clé.

Conditions complémentaires

Termes SQL	Termes MongoDB
Base de données	Base de données
Table	Collection

Termes SQL	Termes MongoDB
Entité / Ligne	Document
Colonne	Clé / champ
Table Join	Documents intégrés
Clé primaire	Clé primaire (clé par défaut `_id` fournie par mongodb lui-même)

Exécution d'un fichier JavaScript dans MongoDB

```
./mongo localhost:27017/mydb myjsfile.js
```

Explication: Cette opération exécute le script `myjsfile.js` dans un shell `mongo` qui se connecte à la base de données `mydb` sur l'instance `mongod` accessible via l'interface `localhost` sur le port `27017` . `localhost:27017` n'est pas obligatoire car c'est le port par défaut utilisé par `mongodb` .

En outre, vous pouvez exécuter un fichier `.js` depuis la console `mongo` .

```
>load("myjsfile.js")
```

Rendre la sortie de find lisible en shell

Nous ajoutons trois enregistrements à notre test de collecte:

```
> db.test.insert({"key":"value1","key2":"Val2","key3":"val3"})
WriteResult({ "nInserted" : 1 })
> db.test.insert({"key":"value2","key2":"Val21","key3":"val31"})
WriteResult({ "nInserted" : 1 })
> db.test.insert({"key":"value3","key2":"Val22","key3":"val33"})
WriteResult({ "nInserted" : 1 })
```

Si nous les voyons via find, ils seront très laids.

```
> db.test.find()
{ "_id" : ObjectId("5790c5cecae25b3d38c3c7ae"), "key" : "value1", "key2" : "Val2
", "key3" : "val3" }
{ "_id" : ObjectId("5790c5d9cae25b3d38c3c7af"), "key" : "value2", "key2" : "Val2
1", "key3" : "val31" }
{ "_id" : ObjectId("5790c5e9cae25b3d38c3c7b0"), "key" : "value3", "key2" : "Val2
2", "key3" : "val33" }
```

Pour contourner ce problème et les rendre lisibles, utilisez la fonction **pretty** ().

```
> db.test.find().pretty()
{
        "_id" : ObjectId("5790c5cecae25b3d38c3c7ae"),
        "key" : "value1",
        "key2" : "Val2",
        "key3" : "val3"
```

```
}
{
        "_id" : ObjectId("5790c5d9cae25b3d38c3c7af"),
        "key" : "value2",
        "key2" : "Val21",
        "key3" : "val31"
}
{
        "_id" : ObjectId("5790c5e9cae25b3d38c3c7b0"),
        "key" : "value3",
        "key2" : "Val22",
        "key3" : "val33"
}
>
```

Commandes de base sur la coquille mongo

Afficher toutes les bases de données disponibles:

```
show dbs;
```

Sélectionnez une base de données particulière à laquelle accéder, par exemple `mydb` . Cela créera `mydb` s'il n'existe pas déjà:

```
use mydb;
```

Afficher toutes les collections de la base de données (veillez à en sélectionner une, voir ci-dessus):

```
show collections;
```

Affiche toutes les fonctions pouvant être utilisées avec la base de données:

```
db.mydb.help();
```

Pour vérifier votre base de données sélectionnée, utilisez la commande `db`

```
> db
mydb
```

`db.dropDatabase()` commande `db.dropDatabase()` permet de supprimer une base de données existante.

```
db.dropDatabase()
```

Lire Démarrer avec MongoDB en ligne: https://riptutorial.com/fr/mongodb/topic/691/demarrer-avec-mongodb

Chapitre 2: 2dsphere Index

Examples

Créer un index 2dsphere

`db.collection.createIndex()` méthode `db.collection.createIndex()` est utilisée pour créer un index `2dsphere` . Le modèle d'un index `2dsphere` :

```
db.collection.createIndex( { <location field> : "2dsphere" } )
```

Ici, le `location field` est la clé et `2dsphere` est le type de l'index. Dans l'exemple suivant, nous allons créer un index `2dsphre` dans la collection de `places` .

```
db.places.insert(
{
  loc : { type: "Point", coordinates: [ -73.97, 40.77 ] },
  name: "Central Park",
  category : "Parks"
})
```

L'opération suivante créera l'index `2dsphere` sur le champ `loc` de la collection de `places` .

```
db.places.createIndex( { loc : "2dsphere" } )
```

Lire 2dsphere Index en ligne: https://riptutorial.com/fr/mongodb/topic/6632/2dsphere-index

Chapitre 3: Agrégation

Introduction

`Aggregations` opérations d' `Aggregations` traitent les enregistrements de données et renvoient les résultats calculés. Les opérations d'agrégation regroupent les valeurs de plusieurs documents et peuvent effectuer diverses opérations sur les données groupées pour renvoyer un seul résultat. MongoDB propose trois méthodes pour effectuer l'agrégation: le pipeline d'agrégation, la fonction de réduction de carte et les méthodes d'agrégation à un seul objectif.

À partir du manuel Mongo https://docs.mongodb.com/manual/aggregation/

Syntaxe

- **db.collection.aggregate (pipeline, options)**

Paramètres

Paramètre	Détails
pipeline	array (Une séquence d'opérations ou d'étapes d'agrégation de données)
options	document (facultatif, disponible uniquement si le pipeline est présent sous forme de tableau)

Remarques

La structure d'agrégation dans MongoDB est utilisée pour obtenir la fonctionnalité `GROUP BY` commune de SQL.

Considérez les insertions suivantes dans les `transactions` nommées collection pour chaque exemple.

```
> db.transactions.insert({ cr_dr : "D", amount : 100, fee : 2});
> db.transactions.insert({ cr_dr : "C", amount : 100, fee : 2});
> db.transactions.insert({ cr_dr : "C", amount : 10,  fee : 2});
> db.transactions.insert({ cr_dr : "D", amount : 100, fee : 4});
> db.transactions.insert({ cr_dr : "D", amount : 10,  fee : 2});
> db.transactions.insert({ cr_dr : "C", amount : 10,  fee : 4});
> db.transactions.insert({ cr_dr : "D", amount : 100, fee : 2});
```

Examples

Compter

Comment obtenez-vous le nombre d'opérations de débit et de crédit? Une façon de le faire est d'utiliser la fonction `count()` comme ci-dessous.

```
> db.transactions.count({cr_dr : "D"});
```

ou

```
> db.transactions.find({cr_dr : "D"}).length();
```

Mais que faire si vous ne connaissez pas les valeurs possibles de `cr_dr` upfront. Ici, le cadre d'agrégation vient jouer. Voir la requête d'agrégation ci-dessous.

```
> db.transactions.aggregate(
      [
          {
              $group : {
                  _id : '$cr_dr', // group by type of transaction
                // Add 1 for each document to the count for this type of transaction
                  count : {$sum : 1}
              }
          }
      ]
  );
```

Et le résultat est

```
{
    "_id" : "C",
    "count" : 3
}
{
    "_id" : "D",
    "count" : 5
}
```

Somme

Comment obtenir la somme du `amount` ? Voir la requête globale ci-dessous.

```
> db.transactions.aggregate(
     [
         {
             $group : {
                 _id : '$cr_dr',
                 count : {$sum : 1},    //counts the number
                 totalAmount : {$sum : '$amount'}    //sums the amount
             }
         }
     ]
 );
```

Et le résultat est

```
{
    "_id" : "C",
    "count" : 3.0,
    "totalAmount" : 120.0
}
{
    "_id" : "D",
    "count" : 5.0,
    "totalAmount" : 410.0
}
```

Une autre version qui calcule le `amount` et les `fee` .

```
> db.transactions.aggregate(
      [
          {
              $group : {
                  _id : '$cr_dr',
                  count : {$sum : 1},
                  totalAmount : {$sum : { $sum : ['$amount', '$fee']}}
              }
          }
      ]
  );
```

Et le résultat est

```
{
    "_id" : "C",
    "count" : 3.0,
    "totalAmount" : 128.0
}
{
    "_id" : "D",
    "count" : 5.0,
    "totalAmount" : 422.0
}
```

Moyenne

Comment obtenir le montant moyen des transactions de débit et de crédit?

```
> db.transactions.aggregate(
    [
        {
            $group : {
                _id : '$cr_dr', // group by type of transaction (debit or credit)
                count : {$sum : 1},    // number of transaction for each type
                totalAmount : {$sum : { $sum : ['$amount', '$fee']}},    // sum
                averageAmount : {$avg : { $sum : ['$amount', '$fee']}}   // average
            }
        }
    ]
)
```

Le résultat est

```
{
    "_id" : "C", // Amounts for credit transactions
    "count" : 3.0,
    "totalAmount" : 128.0,
    "averageAmount" : 40.0
}
{
    "_id" : "D", // Amounts for debit transactions
    "count" : 5.0,
    "totalAmount" : 422.0,
    "averageAmount" : 82.0
}
```

Opérations avec des tableaux.

Lorsque vous voulez travailler avec les entrées de données dans les tableaux, vous devez d'abord dérouler le tableau. L'opération de déroulement crée un document pour chaque entrée du tableau. Lorsque vous avez beaucoup de documents avec de grands tableaux, vous verrez une explosion du nombre de documents.

```
{ "_id" : 1, "item" : "myItem1", sizes: [ "S", "M", "L"] }
{ "_id" : 2, "item" : "myItem2", sizes: [ "XS", "M", "XL"] }

db.inventory.aggregate( [ { $unwind : "$sizes" }] )
```

Une remarque importante est que lorsqu'un document ne contient pas le tableau, il sera perdu. À partir de mongo 3.2, il y a une option de déroulage "preserveNullAndEmptyArrays" ajoutée. Cette option garantit que le document est conservé lorsque le tableau est manquant.

```
{ "_id" : 1, "item" : "myItem1", sizes: [ "S", "M", "L"] }
{ "_id" : 2, "item" : "myItem2", sizes: [ "XS", "M", "XL"] }
{ "_id" : 3, "item" : "myItem3" }

db.inventory.aggregate( [ { $unwind : { path: "$sizes", includeArrayIndex: "arrayIndex" } }] )
```

Rencontre

Comment rédiger une requête pour obtenir tous les départements où l'âge moyen des employés gagnant moins de 70000 $ est supérieur ou égal à 35?

Pour cela, nous devons rédiger une requête afin de correspondre aux employés dont le salaire est inférieur ou égal à 70000 dollars. Ajoutez ensuite l'étape de regroupement pour regrouper les employés par le service. Ajoutez ensuite un accumulateur avec un champ nommé par exemple plus grand que ou égal à 35.

```
db.employees.aggregate([
  {"$match": {"salary": {"$lte": 70000}}},
  {"$group": {"_id": "$dept",
              "average_age": {"$avg": "$age"}
             }
  },
  {"$match": {"average_age": {"$gte": 35}}}
```

```
])
```

Le résultat est:

```
{
  "_id": "IT",
  "average_age": 31
}
{
  "_id": "Customer Service",
  "average_age": 34.5
}
{
  "_id": "Finance",
  "average_age": 32.5
}
```

Supprimer les documents qui ont un champ en double dans une collection (dedupe)

Notez que l'option allowDiskUse: true est facultative mais aidera à atténuer les problèmes de mémoire, car cette agrégation peut nécessiter beaucoup de mémoire si la taille de votre collection est importante. Je vous recommande donc de toujours l'utiliser.

```
var duplicates = [];

db.transactions.aggregate([
 { $group: {
   _id: { cr_dr: "$cr_dr"},
   dups: { "$addToSet": "$_id" },
   count: { "$sum": 1 }
 }
},
{ $match: {
  count: { "$gt": 1 }
}}
],allowDiskUse: true}
)
.result
.forEach(function(doc) {
  doc.dups.shift();
  doc.dups.forEach( function(dupId){
    duplicates.push(dupId);
  }
)
})
// printjson(duplicates);

// Remove all duplicates in one go
db.transactions.remove({_id:{$in:duplicates}})
```

Lire Agrégation en ligne: https://riptutorial.com/fr/mongodb/topic/3852/agregation

Chapitre 4: Agrégation MongoDB

Examples

Des exemples de requêtes agrégés utiles pour le travail et l'apprentissage

L'agrégation est utilisée pour effectuer des opérations de recherche de données complexes dans la requête mongo, ce qui ne peut être fait dans une requête "find" normale.

Créez des données factices:

```
db.employees.insert({"name":"Adma","dept":"Admin","languages":["german","french","english","hindi"],"ag
"totalExp":10});
db.employees.insert({"name":"Anna","dept":"Admin","languages":["english","hindi"],"age":35,
"totalExp":11});
db.employees.insert({"name":"Bob","dept":"Facilities","languages":["english","hindi"],"age":36,
"totalExp":14});
db.employees.insert({"name":"Cathy","dept":"Facilities","languages":["hindi"],"age":31,
"totalExp":4});
db.employees.insert({"name":"Mike","dept":"HR","languages":["english", "hindi",
"spanish"],"age":26, "totalExp":3});
db.employees.insert({"name":"Jenny","dept":"HR","languages":["english", "hindi",
"spanish"],"age":25, "totalExp":3});
```

Exemples par sujet:

1. Correspondance: Utilisé pour correspondre à des documents (comme la clause SQL where)

```
db.employees.aggregate([{$match:{dept:"Admin"}}]);
Output:
{ "_id" : ObjectId("54982fac2e9b4b54ec384a0d"), "name" : "Adma", "dept" : "Admin", "languages"
: [ "german", "french", "english", "hindi" ], "age" : 30, "totalExp" : 10 }
{ "_id" : ObjectId("54982fc92e9b4b54ec384a0e"), "name" : "Anna", "dept" : "Admin", "languages"
: [ "english", "hindi" ], "age" : 35, "totalExp" : 11 }
```

2. Projet: Utilisé pour remplir les valeurs de champ spécifiques

L'étape du projet inclura automatiquement le champ _id à moins que vous ne spécifiiez de le désactiver.

```
db.employees.aggregate([{$match:{dept:"Admin"}}, {$project:{"name":1, "dept":1}}]);
Output:
{ "_id" : ObjectId("54982fac2e9b4b54ec384a0d"), "name" : "Adma", "dept" : "Admin" }
{ "_id" : ObjectId("54982fc92e9b4b54ec384a0e"), "name" : "Anna", "dept" : "Admin" }

db.employees.aggregate({$project: {'_id':0, 'name': 1}})
Output:
{ "name" : "Adma" }
{ "name" : "Anna" }
{ "name" : "Bob" }
{ "name" : "Cathy" }
{ "name" : "Mike" }
```

```
{ "name" : "Jenny" }
```

3. Group: $ group est utilisé pour regrouper les documents par champs spécifiques, ici les documents sont regroupés par valeur de champ "département". Une autre fonctionnalité utile est que vous pouvez regrouper par null, cela signifie que tous les documents seront regroupés en un seul.

```
db.employees.aggregate([{$group:{"_id":"$dept"}}]);


{ "_id" : "HR" }

{ "_id" : "Facilities" }

{ "_id" : "Admin" }

db.employees.aggregate([{$group:{"_id":null, "totalAge":{$sum:"$age"}}}]);
Output:
{ "_id" : null, "noOfEmployee" : 183 }
```

4. Sum: $ sum est utilisé pour compter ou additionner les valeurs dans un groupe.

```
db.employees.aggregate([{$group:{"_id":"$dept", "noOfDept":{$sum:1}}}]);
Output:
{ "_id" : "HR", "noOfDept" : 2 }
{ "_id" : "Facilities", "noOfDept" : 2 }
{ "_id" : "Admin", "noOfDept" : 2 }
```

5. Moyenne: Calcule la moyenne de la valeur d'un champ spécifique par groupe.

```
db.employees.aggregate([{$group:{"_id":"$dept", "noOfEmployee":{$sum:1},
"avgExp":{$avg:"$totalExp"}}}]);
Output:
{ "_id" : "HR", "noOfEmployee" : 2, "totalExp" : 3 }
{ "_id" : "Facilities", "noOfEmployee" : 2, "totalExp" : 9 }
{ "_id" : "Admin", "noOfEmployee" : 2, "totalExp" : 10.5 }
```

6. Minimum: Trouve la valeur minimum d'un champ dans chaque groupe.

```
db.employees.aggregate([{$group:{"_id":"$dept", "noOfEmployee":{$sum:1},
"minExp":{$min:"$totalExp"}}}]);
Output:
{ "_id" : "HR", "noOfEmployee" : 2, "totalExp" : 3 }
{ "_id" : "Facilities", "noOfEmployee" : 2, "totalExp" : 4 }
{ "_id" : "Admin", "noOfEmployee" : 2, "totalExp" : 10 }
```

7. Maximum: Trouve la valeur maximale d'un champ dans chaque groupe.

```
db.employees.aggregate([{$group:{"_id":"$dept", "noOfEmployee":{$sum:1},
"maxExp":{$max:"$totalExp"}}}]);
Output:
{ "_id" : "HR", "noOfEmployee" : 2, "totalExp" : 3 }
{ "_id" : "Facilities", "noOfEmployee" : 2, "totalExp" : 14 }
{ "_id" : "Admin", "noOfEmployee" : 2, "totalExp" : 11 }
```

8. Obtenir la valeur du champ spécifique du premier et du dernier document de chaque groupe: fonctionne bien lorsque le résultat du document est trié.

```
db.employees.aggregate([{$group:{"_id":"$age", "lasts":{$last:"$name"},
"firsts":{$first:"$name"}}}]);
Output:
{ "_id" : 25, "lasts" : "Jenny", "firsts" : "Jenny" }
{ "_id" : 26, "lasts" : "Mike", "firsts" : "Mike" }
{ "_id" : 35, "lasts" : "Cathy", "firsts" : "Anna" }
{ "_id" : 30, "lasts" : "Adma", "firsts" : "Adma" }
```

9. Minumum au maximum:

```
db.employees.aggregate([{$group:{"_id":"$dept", "noOfEmployee":{$sum:1},
"maxExp":{$max:"$totalExp"}, "minExp":{$min: "$totalExp"}}}]);
Output:
{ "_id" : "HR", "noOfEmployee" : 2, "maxExp" : 3, "minExp" : 3 }
{ "_id" : "Facilities", "noOfEmployee" : 2, "maxExp" : 14, "minExp" : 4 }
{ "_id" : "Admin", "noOfEmployee" : 2, "maxExp" : 11, "minExp" : 10 }
```

10. Push et addToSet: Push ajoute la valeur d'un champ de chaque document du groupe à un tableau utilisé pour projeter des données au format tableau, addToSet est simple à utiliser, mais omet les valeurs en double.

```
db.employees.aggregate([{$group:{"_id":"dept", "arrPush":{$push:"$age"}, "arrSet":
{$addToSet:"$age"}}}]);
Output:
{ "_id" : "dept", "arrPush" : [ 30, 35, 35, 35, 26, 25 ], "arrSet" : [ 25, 26, 35, 30 ] }
```

11. Dérouler: Utilisé pour créer plusieurs documents en mémoire pour chaque valeur dans le champ de type de tableau spécifié, nous pouvons procéder à une agrégation plus poussée en fonction de ces valeurs.

```
db.employees.aggregate([{$match:{"name":"Adma"}}, {$unwind:"$languages"}]);
Output:
{ "_id" : ObjectId("54982fac2e9b4b54ec384a0d"), "name" : "Adma", "dept" : "HR", "languages" :
"german", "age" : 30, "totalExp" : 10 }
{ "_id" : ObjectId("54982fac2e9b4b54ec384a0d"), "name" : "Adma", "dept" : "HR", "languages" :
"french", "age" : 30, "totalExp" : 10 }
{ "_id" : ObjectId("54982fac2e9b4b54ec384a0d"), "name" : "Adma", "dept" : "HR", "languages" :
"english", "age" : 30, "totalExp" : 10 }
{ "_id" : ObjectId("54982fac2e9b4b54ec384a0d"), "name" : "Adma", "dept" : "HR", "languages" :
"hindi", "age" : 30, "totalExp" : 10 }
```

12. Tri:

```
db.employees.aggregate([{$match:{dept:"Admin"}}, {$project:{"name":1, "dept":1}}, {$sort:
{name: 1}}]);
Output:
{ "_id" : ObjectId("57ff3e553dedf0228d4862ac"), "name" : "Adma", "dept" : "Admin" }
{ "_id" : ObjectId("57ff3e5e3dedf0228d4862ad"), "name" : "Anna", "dept" : "Admin" }

db.employees.aggregate([{$match:{dept:"Admin"}}, {$project:{"name":1, "dept":1}}, {$sort:
{name: -1}}]);
```

```
Output:
{ "_id" : ObjectId("57ff3e5e3dedf0228d4862ad"), "name" : "Anna", "dept" : "Admin" }
{ "_id" : ObjectId("57ff3e553dedf0228d4862ac"), "name" : "Adma", "dept" : "Admin" }
```

13. Ignorer:

```
db.employees.aggregate([{$match:{dept:"Admin"}}, {$project:{"name":1, "dept":1}}, {$sort:
{name: -1}}, {$skip:1}]);
Output:
{ "_id" : ObjectId("57ff3e553dedf0228d4862ac"), "name" : "Adma", "dept" : "Admin" }
```

14. Limite:

```
db.employees.aggregate([{$match:{dept:"Admin"}}, {$project:{"name":1, "dept":1}}, {$sort:
{name: -1}}, {$limit:1}]);
Output:

{ "_id" : ObjectId("57ff3e5e3dedf0228d4862ad"), "name" : "Anna", "dept" : "Admin" }
```

15. Opérateur de comparaison en projection:

```
db.employees.aggregate([{$match:{dept:"Admin"}}, {$project:{"name":1, "dept":1, age: {$gt:
["$age", 30]}}}]);
Output:
{ "_id" : ObjectId("57ff3e553dedf0228d4862ac"), "name" : "Adma", "dept" : "Admin", "age" :
false }
{ "_id" : ObjectId("57ff3e5e3dedf0228d4862ad"), "name" : "Anna", "dept" : "Admin", "age" :
true }
```

16. Opérateur de comparaison en correspondance:

```
db.employees.aggregate([{$match:{dept:"Admin", age: {$gt:30}}}, {$project:{"name":1,
"dept":1}}]);
Output:
{ "_id" : ObjectId("57ff3e5e3dedf0228d4862ad"), "name" : "Anna", "dept" : "Admin" }
```

Liste des opérateurs de comparaison: $ cmp, $ eq, $ gt, $ gte, $ lt, $ lte et $ ne

17. Opérateur d'agrégation booléenne en projection:

```
db.employees.aggregate([{$match:{dept:"Admin"}}, {$project:{"name":1, "dept":1, age: { $and: [
{ $gt: [ "$age", 30 ] }, { $lt: [ "$age", 36 ] } ] }}}]);

Output:
{ "_id" : ObjectId("57ff3e553dedf0228d4862ac"), "name" : "Adma", "dept" : "Admin", "age" :
false }
{ "_id" : ObjectId("57ff3e5e3dedf0228d4862ad"), "name" : "Anna", "dept" : "Admin", "age" :
true }
```

18. Opérateur d'agrégation booléenne en correspondance:

```
db.employees.aggregate([{$match:{dept:"Admin", $and: [{age: { $gt:  30 }}, {age: {$lt: 36 }} ]
}}, {$project:{"name":1, "dept":1, age: { $and: [ { $gt: [ "$age", 30 ] }, { $lt: [ "$age", 36
```

```
] } ] }}}]);
Output:
{ "_id" : ObjectId("57ff3e5e3dedf0228d4862ad"), "name" : "Anna", "dept" : "Admin", "age" :
true }
```

Liste des opérateurs de regroupement booléens: $ et, $ or et $ not.

Réfrence complet: https://docs.mongodb.com/v3.2/reference/operator/aggregation/

Exemple Java et Spring

Ceci est un exemple de code pour créer et exécuter la requête d'agrégation dans MongoDB à l'aide des données Spring.

```
    try {
        MongoClient mongo = new MongoClient();
        DB db = mongo.getDB("so");
        DBCollection coll = db.getCollection("employees");

        //Equivalent to $match
        DBObject matchFields = new BasicDBObject();
        matchFields.put("dept", "Admin");
        DBObject match = new BasicDBObject("$match", matchFields);

        //Equivalent to $project
        DBObject projectFields = new BasicDBObject();
        projectFields.put("_id", 1);
        projectFields.put("name", 1);
        projectFields.put("dept", 1);
        projectFields.put("totalExp", 1);
        projectFields.put("age", 1);
        projectFields.put("languages", 1);
        DBObject project = new BasicDBObject("$project", projectFields);

        //Equivalent to $group
        DBObject groupFields = new BasicDBObject("_id", "$dept");
        groupFields.put("ageSet", new BasicDBObject("$addToSet", "$age"));
        DBObject employeeDocProjection = new BasicDBObject("$addToSet", new
BasicDBObject("totalExp", "$totalExp").append("age", "$age").append("languages",
"$languages").append("dept", "$dept").append("name", "$name"));
        groupFields.put("docs", employeeDocProjection);
        DBObject group = new BasicDBObject("$group", groupFields);

        //Sort results by age
        DBObject sort = new BasicDBObject("$sort", new BasicDBObject("age", 1));

        List<DBObject> aggregationList = new ArrayList<>();
        aggregationList.add(match);
        aggregationList.add(project);
        aggregationList.add(group);
        aggregationList.add(sort);
        AggregationOutput output = coll.aggregate(aggregationList);

        for (DBObject result : output.results()) {
            BasicDBList employeeList = (BasicDBList) result.get("docs");
            BasicDBObject employeeDoc = (BasicDBObject) employeeList.get(0);
            String name = employeeDoc.get("name").toString();
            System.out.println(name);
```

```
        }
    }catch (Exception ex){
        ex.printStackTrace();
    }
```

Voir la valeur "resultSet" au format JSON pour comprendre le format de sortie:

```
[{
    "_id": "Admin",
    "ageSet": [35.0, 30.0],
    "docs": [{
        "totalExp": 11.0,
        "age": 35.0,
        "languages": ["english", "hindi"],
        "dept": "Admin",
        "name": "Anna"
    }, {
        "totalExp": 10.0,
        "age": 30.0,
        "languages": ["german", "french", "english", "hindi"],
        "dept": "Admin",
        "name": "Adma"
    }]
}]
```

Le "resultSet" contient une entrée pour chaque groupe, "ageSet" contient la liste d'âge de chaque employé de ce groupe, "_id" contient la valeur du champ utilisé pour le regroupement et "docs" contient les données de chaque employé de ce groupe pouvant être utilisé dans notre propre code et interface utilisateur.

Obtenir des exemples de données

Pour obtenir des données aléatoires à partir de certaines collections, reportez-vous à l'agrégation `$sample` .

```
db.emplyees.aggregate({ $sample: { size:1 } })
```

où la `size` représente le nombre d'éléments à sélectionner.

Jointure externe gauche avec agrégation (recherche $)

```
let col_1 = db.collection('col_1');
let col_2 = db.collection('col_2');
col_1 .aggregate([
    { $match: { "_id": 1 } },
    {
        $lookup: {
            from: "col_2",
            localField: "id",
            foreignField: "id",
            as: "new_document"
        }
    }
],function (err, result){
```

```
    res.send(result);
});
```

Cette fonctionnalité a été récemment publiée dans la **version 3.2 de** mongodb, qui permet à l'utilisateur de rejoindre une collection avec les attributs correspondants d'une autre collection.

Documentation Mongodb $ LookUp

Lire Agrégation MongoDB en ligne: https://riptutorial.com/fr/mongodb/topic/7417/agregation-mongodb

Chapitre 5: Collections

Remarques

Créer une base de données

Examples

Créer une collection

First Select Ou Créer une base de données.

```
> use mydb
switched to db mydb
```

En utilisant la `db.createCollection("yourCollectionName")` , vous pouvez créer explicitement Collection.

```
> db.createCollection("newCollection1")
{ "ok" : 1 }
```

En utilisant la commande `show collections` voir toutes les collections de la base de données.

```
> show collections
newCollection1
system.indexes
>
```

La méthode `db.createCollection()` a les paramètres suivants:

Paramètre	Type	La description
prénom	chaîne	Le nom de la collection à créer.
options	document	*Optionnel.* Options de configuration pour créer une collection plafonnée ou pour préallouer de l'espace dans une nouvelle collection.

L'exemple ci-dessous montre la syntaxe de la méthode `createCollection()` avec peu d'options importantes

```
>db.createCollection("newCollection4", {capped :true, autoIndexId : true, size : 6142800, max
: 10000})
{ "ok" : 1 }
```

Les opérations `db.collection.insert()` et `db.collection.createIndex()` créent leurs collections

respectives si elles n'existent pas déjà.

```
> db.newCollection2.insert({name : "XXX"})
> db.newCollection3.createIndex({accountNo : 1})
```

Maintenant, Afficher toutes les collections à l'aide `show collections` commande `show collections`

```
> show collections
newCollection1
newCollection2
newCollection3
newCollection4
system.indexes
```

Si vous souhaitez voir le document inséré, utilisez la commande `find()` .

```
> db.newCollection2.find()
{ "_id" : ObjectId("58f26876cabafaeb509e9c1f"), "name" : "XXX" }
```

Drop Collection

`db.collection.drop()` de MongoDB est utilisé pour supprimer une collection de la base de données.

Tout d'abord, vérifiez les collections disponibles dans votre base de données `mydb` .

```
> use mydb
switched to db mydb

> show collections
newCollection1
newCollection2
newCollection3
system.indexes
```

Maintenant, déposez la collection sous le nom `newCollection1` .

```
> db.newCollection1.drop()
true
```

Remarque: Si la collection a été supprimée avec succès, la méthode renverra `true` sinon elle retournera `false` .

Encore une fois, vérifiez la liste des collections dans la base de données.

```
> show collections
newCollection2
newCollection3
system.indexes
```

Référence: Méthode MongoDB drop () .

Lire Collections en ligne: https://riptutorial.com/fr/mongodb/topic/9732/collections

Chapitre 6: Configuration

Paramètres

Paramètre	Défaut
systemLog.verbosité	0
systemLog.quiet	faux
systemLog.traceAllExceptions	faux
systemLog.syslogFacility	utilisateur
systemLog.path	-
systemLog.logAppend	faux
systemLog.logRotate	Renommer
systemLog.destination	stdout
systemLog.timeStampFormat	iso8601-local
systemLog.component.accessControl.verbosity	0
systemLog.component.command.verbosity	0
systemLog.component.control.verbosity	0
systemLog.component.ftdc.verbosity	0
systemLog.component.geo.verbosity	0
systemLog.component.index.verbosity	0
systemLog.component.network.verbo	0
systemLog.component.query.verbosity	0
systemLog.component.replication.verbosity	0
systemLog.component.sharding.verbosity	0
systemLog.component.storage.verbosity	0
systemLog.component.storage.journal.verbosity	0
systemLog.component.write.verbosity	0

Paramètre	Défaut
processManagement.fork	faux
processManagement.pidFilePath	aucun
net.port	27017
net.bindIp	0.0.0.0
net.maxIncomingConnections	65536
net.wireObjectCheck	vrai
net.ipv6	faux
net.unixDomainSocket.enabled	vrai
net.unixDomainSocket.pathPrefix	/ tmp
net.unixDomainSocket.filePermissions	0700
net.http.enabled	faux
net.http.JSONPEnabled	faux
net.http.RESTInterfaceEnabled	faux
net.ssl.sslOnNormalPorts	faux
net.ssl.mode	désactivée
net.ssl.PEMKeyFile	aucun
net.ssl.PEMKeyPassword	aucun
net.ssl.clusterFile	aucun
net.ssl.clusterPassword	aucun
net.ssl.CAFile	aucun
net.ssl.CRLFile	aucun
net.ssl.allowConnectionsWithoutCertificates	faux
net.ssl.allowInvalidCertificates	faux
net.ssl.allowInvalidHostnames	faux
net.ssl.disabledProtocols	aucun

Paramètre	Défaut
net.ssl.FIPSMode	faux

Examples

Démarrer mongo avec un fichier de configuration spécifique

Utiliser l'indicateur `--config` .

```
$ /bin/mongod --config /etc/mongod.conf
$ /bin/mongos --config /etc/mongos.conf
```

Notez que `-f` est le synonyme le plus court pour `--config` .

Lire Configuration en ligne: https://riptutorial.com/fr/mongodb/topic/5985/configuration

Chapitre 7: Gestion de MongoDB

Examples

Liste des requêtes en cours d'exécution

La commande suivante répertorie les requêtes en cours d'exécution sur le serveur

```
db.currentOp()
```

La sortie ressemble à ceci

```
{
    "inprog" : [
        {
            "opid" : "302616759",
            "active" : true,
            "secs_running" : 1,
            "microsecs_running" : NumberLong(1167662),
            "op" : "getmore",
            "ns" : "local.oplog.rs",
            "query" : {

            },
            ...
        },
        {
            "desc" : "conn48",
            "threadId" : "0x114c00700",
            "connectionId" : 48,
            "opid" : "mdss_shard00:302616760",
            "active" : true,
            "secs_running" : 1,
            "microsecs_running" : NumberLong(1169659),
            "op" : "getmore",
            "ns" : "local.oplog.rs"
            ...
        }
    ]
}
```

L'attribut `inprog` indique que les requêtes sont en cours. L' `opid` est Id de la requête ou de l'opération. `secs_running` indique l'heure pour laquelle il a été exécuté. Ceci est parfois utile pour identifier les requêtes longues.

Lire Gestion de MongoDB en ligne: https://riptutorial.com/fr/mongodb/topic/7553/gestion-de-mongodb

Chapitre 8: Index

Syntaxe

- `db.collection.createIndex({ <string field> : <1|-1 order> [, <string field> : <1|-1 order>] });`

Remarques

Impact sur les performances : Notez que les index améliorent les performances en lecture, mais peuvent avoir un impact négatif sur les performances en écriture, car l'insertion d'un document nécessite la mise à jour de tous les index.

Examples

Champ unique

```
db.people.createIndex({name: 1})
```

Cela crée un index de champ unique ascendant sur le *nom* du champ.

Dans ce type d'index, l'ordre de tri n'est pas pertinent, car mongo peut parcourir l'index dans les deux sens.

Composé

```
db.people.createIndex({name: 1, age: -1})
```

Cela crée un index sur plusieurs champs, dans ce cas sur les champs `name` et `age` . Ce sera ascendant en `name` et descendant en `age` .

Dans ce type d'index, l'ordre de tri est pertinent, car il déterminera si l'index peut prendre en charge une opération de tri ou non. Le tri inverse est pris en charge sur tout préfixe d'un index composé, tant que le tri est dans le sens inverse du tri pour **toutes** les clés du tri. Sinon, le tri des index composés doit correspondre à l'ordre de l'index.

L'ordre des champs est également important. Dans ce cas, l'index sera trié d'abord par `name` , et dans chaque valeur de nom, trié en fonction des valeurs du champ d' `age` . Cela permet à l'index d'être utilisé par les requêtes sur le champ de `name` , ou sur le `name` et l' `age` , mais pas uniquement sur l' `age` .

Effacer

Pour supprimer un index, vous pouvez utiliser le nom d'index

```
db.people.dropIndex("nameIndex")
```

Ou le document de spécification d'index

```
db.people.dropIndex({name: 1})
```

liste

```
db.people.getIndexes()
```

Cela retournera un tableau de documents décrivant chacun un index sur la collection de *personnes*

Les bases de la création d'index

Voir la collection de transactions ci-dessous.

```
> db.transactions.insert({ cr_dr : "D", amount : 100, fee : 2});
> db.transactions.insert({ cr_dr : "C", amount : 100, fee : 2});
> db.transactions.insert({ cr_dr : "C", amount : 10,  fee : 2});
> db.transactions.insert({ cr_dr : "D", amount : 100, fee : 4});
> db.transactions.insert({ cr_dr : "D", amount : 10,  fee : 2});
> db.transactions.insert({ cr_dr : "C", amount : 10,  fee : 4});
> db.transactions.insert({ cr_dr : "D", amount : 100, fee : 2});
```

`getIndexes()` fonctions `getIndexes()` afficheront tous les index disponibles pour une collection.

```
db.transactions.getIndexes();
```

Laissez voir le résultat de la déclaration ci-dessus.

```
[
    {
        "v" : 1,
        "key" : {
            "_id" : 1
        },
        "name" : "_id_",
        "ns" : "documentation_db.transactions"
    }
]
```

Il existe déjà un index pour la collecte des transactions. C'est parce que MongoDB crée un *index unique* sur le champ `_id` lors de la création d'une collection. L'index `_id` empêche les clients d'insérer deux documents ayant la même valeur pour le champ `_id` . Vous ne pouvez pas supprimer cet index sur le champ `_id` .

Ajoutons maintenant un index pour le champ cr_dr;

```
db.transactions.createIndex({ cr_dr : 1 });
```

Le résultat de l'exécution de l'index est le suivant.

```
{
    "createdCollectionAutomatically" : false,
    "numIndexesBefore" : 1,
    "numIndexesAfter" : 2,
    "ok" : 1
}
```

> Le createdCollectionAutomatic indique si l'opération a créé une collection. Si une collection n'existe pas, MongoDB crée la collection dans le cadre de l'opération d'indexation.

Laissez exécuter `db.transactions.getIndexes();` encore.

```
[
    {
        "v" : 1,
        "key" : {
            "_id" : 1
        },
        "name" : "_id_",
        "ns" : "documentation_db.transactions"
    },
    {
        "v" : 1,
        "key" : {
            "cr_dr" : 1
        },
        "name" : "cr_dr_1",
        "ns" : "documentation_db.transactions"
    }
]
```

Vous voyez maintenant que la collecte des transactions a deux indices. Par défaut `_id` index et `cr_dr_1` que nous avons créé. Le nom est attribué par MongoDB. Vous pouvez définir votre propre nom comme ci-dessous.

```
db.transactions.createIndex({ cr_dr : -1 },{name : "index on cr_dr desc"})
```

Maintenant, `db.transactions.getIndexes();` vous donnera trois indices.

```
[
    {
        "v" : 1,
        "key" : {
            "_id" : 1
        },
        "name" : "_id_",
        "ns" : "documentation_db.transactions"
    },
    {
        "v" : 1,
        "key" : {
            "cr_dr" : 1
```

```
        },
        "name" : "cr_dr_1",
        "ns" : "documentation_db.transactions"
    },
    {
        "v" : 1,
        "key" : {
            "cr_dr" : -1
        },
        "name" : "index on cr_dr desc",
        "ns" : "documentation_db.transactions"
    }
]
```

Lors de la création de l'index `{ cr_dr : -1 }` 1 signifie que l'index sera dans l'ordre `ascending` et -1 dans `descending` ordre `descending` .

2.4

Index hachés

Les index peuvent également être définis comme *hachés* . Ceci est plus performant sur *les requêtes d'égalité* , mais n'est pas efficace pour les *requêtes de plage* ; Cependant, vous pouvez définir des index hachés et ascendants / descendants sur le même champ.

```
> db.transactions.createIndex({ cr_dr : "hashed" });

> db.transactions.getIndexes(
[
    {
        "v" : 1,
        "key" : {
            "_id" : 1
        },
        "name" : "_id_",
        "ns" : "documentation_db.transactions"
    },
    {
        "v" : 1,
        "key" : {
            "cr_dr" : "hashed"
        },
        "name" : "cr_dr_hashed",
        "ns" : "documentation_db.transactions"
    }
]
```

Suppression / Suppression d'un index

Si le nom de l'index est connu,

```
db.collection.dropIndex('name_of_index');
```

Si le nom de l'index n'est pas connu,

```
db.collection.dropIndex( { 'name_of_field' : -1 } );
```

Obtenir les indices d'une collection

```
 db.collection.getIndexes();
```

Sortie

```
[
    {
        "v" : 1,
        "key" : {
            "_id" : 1
        },
        "name" : "_id_",
        "ns" : "documentation_db.transactions"
    },
    {
        "v" : 1,
        "key" : {
            "cr_dr" : 1
        },
        "name" : "cr_dr_1",
        "ns" : "documentation_db.transactions"
    },
    {
        "v" : 1,
        "key" : {
            "cr_dr" : -1
        },
        "name" : "index on cr_dr desc",
        "ns" : "documentation_db.transactions"
    }
]
```

Index unique

```
db.collection.createIndex( { "user_id": 1 }, { unique: true } )
```

appliquer l'unicité à l'index défini (unique ou composé). La construction de l'index échouera si la collection contient déjà des valeurs en double; l'indexation échouera également avec plusieurs entrées manquant le champ (puisqu'elles seront toutes indexées avec la valeur `null`) sauf si `sparse: true` est spécifié.

Index clairsemés et index partiels

Index clairsemés:

Celles-ci peuvent être particulièrement utiles pour les champs facultatifs mais qui doivent également être uniques.

```
{ "_id" : "john@example.com", "nickname" : "Johnnie" }
```

```
{ "_id" : "jane@example.com" }
{ "_id" : "julia@example.com", "nickname" : "Jules"}
{ "_id" : "jack@example.com" }
```

Comme deux entrées n'ont pas de "pseudonyme" spécifié et que l'indexation traitera les champs non spécifiés comme null, la création de l'index échouerait avec 2 documents ayant la valeur "null", donc:

```
db.scores.createIndex( { nickname: 1 } , { unique: true, sparse: true } )
```

vous laissera encore des surnoms "null".

Les index fragmentés sont plus compacts car ils ignorent / ignorent les documents qui ne spécifient pas ce champ. Donc, si vous avez une collection dans laquelle seulement moins de 10% des documents spécifient ce champ, vous pouvez créer des index beaucoup plus petits - en utilisant mieux la mémoire limitée si vous souhaitez effectuer des requêtes telles que:

```
db.scores.find({'nickname': 'Johnnie'})
```

Index partiels:

> Les index partiels représentent un sur-ensemble des fonctionnalités offertes par les index clairsemés et doivent être préférés aux index clairsemés. (*Nouveau dans la version 3.2*)

Les index partiels déterminent les entrées d'index en fonction du filtre spécifié.

```
db.restaurants.createIndex(
  { cuisine: 1 },
  { partialFilterExpression: { rating: { $gt: 5 } } }
)
```

Si la `rating` est supérieure à 5, la `cuisine` sera indexée. Oui, nous pouvons spécifier une propriété à indexer en fonction de la valeur des autres propriétés également.

Différence entre les indices clairsemés et partiels:

Les index fragmentés sélectionnent les documents à indexer uniquement en fonction de l'existence du champ indexé ou, pour les index composés, de l'existence des champs indexés.

Les index partiels déterminent les entrées d'index en fonction du filtre spécifié. Le filtre peut inclure des champs autres que les clés d'index et peut spécifier des conditions autres qu'un contrôle d'existence.

Cependant, un index partiel peut implémenter le même comportement qu'un index fragmenté

Par exemple:

```
db.contacts.createIndex(
   { name: 1 },
```

```
    { partialFilterExpression: { name: { $exists: true } } }
)
```

Remarque: Les deux l'option *partialFilterExpression* et l'option *clairsemés* ne peuvent pas être spécifiées en même temps.

Lire Index en ligne: https://riptutorial.com/fr/mongodb/topic/3934/index

Chapitre 9: Mécanismes d'authentification dans MongoDB

Introduction

L'authentification est le processus de vérification de l'identité d'un client. Lorsque le contrôle d'accès, à savoir l'autorisation, est activé, MongoDB exige que tous les clients s'authentifient pour déterminer leur accès.

MongoDB prend en charge un certain nombre de mécanismes d'authentification que les clients peuvent utiliser pour vérifier leur identité. Ces mécanismes permettent à MongoDB de s'intégrer dans votre système d'authentification existant.

Examples

Mécanismes d'authentification

MongoDB prend en charge plusieurs mécanismes d'authentification.

Mécanismes d'authentification client et utilisateur

- SCRAM-SHA-1
- Authentification par certificat X.509
- MongoDB Challenge and Response (MONGODB-CR)
- Authentification proxy LDAP et
- Authentification Kerberos

Mécanismes d'authentification interne

- Fichier clé
- X.509

Lire Mécanismes d'authentification dans MongoDB en ligne:
https://riptutorial.com/fr/mongodb/topic/8113/mecanismes-d-authentification-dans-mongodb

Chapitre 10: Mettre à jour les opérateurs

Syntaxe

- {$ set: {<field1>: <valeur1>, <champ2>: <valeur2>, ...}}

Paramètres

paramètres	Sens
nom de domaine	Le champ sera mis à jour: { ***name*** : 'Tom'}
targetvaule	La valeur sera attribuée au champ: {name: ***'Tom'*** }

Remarques

Référence pour l'opérateur $ set: $ défini sur le site officiel

Examples

Opérateur $ set pour mettre à jour les champs spécifiés dans les documents

I. Aperçu

Une différence significative entre MongoDB et SGBDR est que MongoDB a plusieurs types d'opérateurs. L'un d'eux est l'opérateur de mise à jour, utilisé dans les instructions de mise à jour.

II. Que se passe-t-il si nous n'utilisons pas les opérateurs de mise à jour?

Supposons que nous ayons une collection d' **étudiants** pour stocker les informations sur les étudiants (vue Table):

age	name	sex
20	Tom	M
25	Billy	M
18	Mary	F
40	Ken	M

Un jour, vous obtenez un travail qui doit changer le sexe de Tom de "M" à "F". C'est facile, non? Donc, vous écrivez ci-dessous la déclaration très rapidement en fonction de votre expérience de SGBDR:

```
db.student.update(
    {name: 'Tom'}, // query criteria
    {sex: 'F'} // update action
);
```

Voyons quel est le résultat:

Nous avons perdu l'âge et le nom de Tom! À partir de cet exemple, nous pouvons savoir que **l'ensemble du document sera surchargé** sans opérateur de mise à jour dans la déclaration de mise à jour. C'est le comportement par défaut de MongoDB.

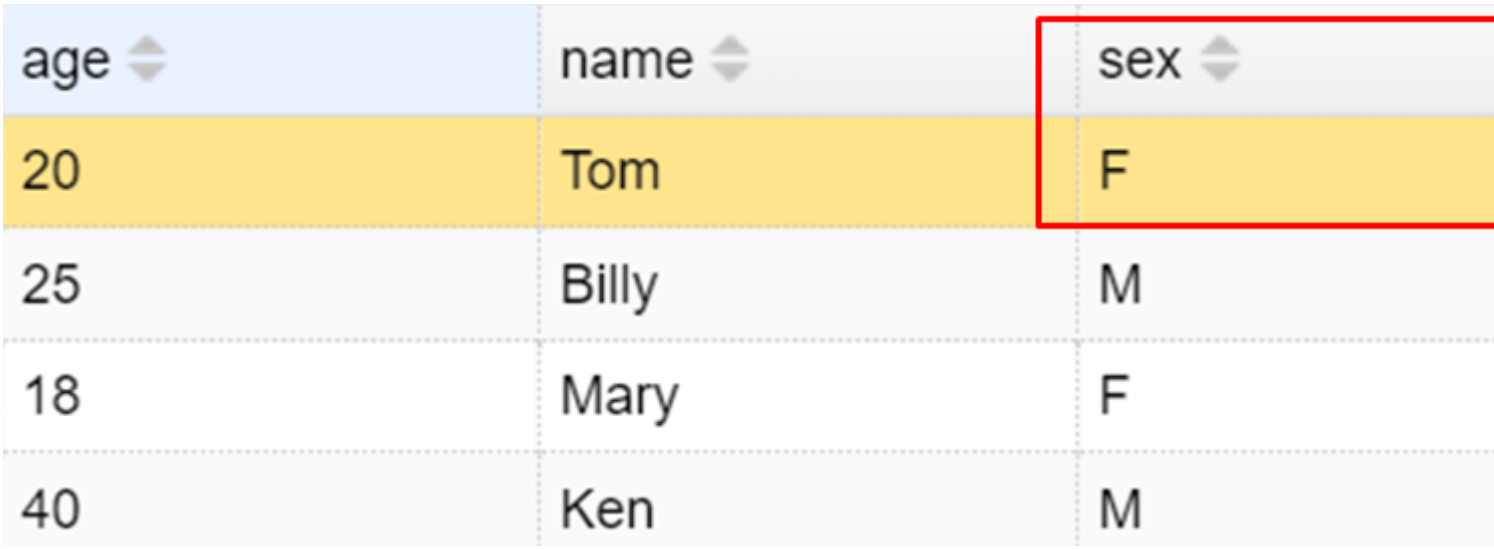
III. Opérateur set $

Si nous voulons changer uniquement le champ "sexe" dans le document de Tom, nous pouvons utiliser `$set` pour spécifier le ou les champs que nous voulons mettre à jour:

```
db.student.update(
    {name: 'Tom'}, // query criteria
    {$set: {sex: 'F'}} // update action
);
```

La valeur de `$set` est un objet, ses champs correspondent aux champs que vous souhaitez mettre à jour dans les documents et les valeurs de ces champs sont les valeurs cibles.

Donc, le résultat est correct maintenant:



age	name	sex
20	Tom	F
25	Billy	M
18	Mary	F
40	Ken	M

De même, si vous souhaitez modifier à la fois le sexe et l'âge, vous pouvez les ajouter à `$set` :

```
db.student.update(
    {name: 'Tom'}, // query criteria
    {$set: {sex: 'F', age: 40}} // update action
);
```

Lire Mettre à jour les opérateurs en ligne: https://riptutorial.com/fr/mongodb/topic/5880/mettre-a-jour-les-operateurs

Chapitre 11: Mise à niveau de la version MongoDB

Introduction

Comment mettre à jour la version de MongoDB sur votre machine sur différentes plates-formes et versions.

Remarques

Si vous avez une ancienne version de MongoDB, vous devez mettre à niveau tout le chemin vers la version la plus récente. Par exemple, si vous utilisez la version 3.0 et que vous souhaitez obtenir la version 3.4, vous devez mettre à niveau 3.0-> 3.2-> 3.4.

Examples

Mise à niveau vers 3.4 sur Ubuntu 16.04 en utilisant apt

Vous devez avoir 3.2 pour pouvoir passer à la version 3.4. Cet exemple suppose que vous utilisez `apt` .

0. `sudo service mongod stop`
1. `sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv 0C49F3730359A14518585931BC711F9BA15703C6`
2. `echo "deb [ arch=amd64,arm64 ] http://repo.mongodb.org/apt/ubuntu xenial/mongodb-org/3.4 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-3.4.list`
3. `sudo apt-get update`
4. `sudo apt-get upgrade`
5. `sudo service mongod start`

Assurez-vous que la nouvelle version fonctionne avec `mongo` . Le shell imprimera la version du serveur MongoDB qui devrait maintenant être 3.4.

Lire Mise à niveau de la version MongoDB en ligne:
https://riptutorial.com/fr/mongodb/topic/9851/mise-a-niveau-de-la-version-mongodb

Chapitre 12: Modèle d'autorisation MongoDB

Introduction

L'autorisation est fondamentalement la vérification des privilèges de l'utilisateur. MongoDB supporte différents types de modèles d'autorisation. 1. **Contrôle d'accès à la base de rôles**
 Rôle: groupe de privilèges, actions sur les ressources. Ce sont des avantages pour les utilisateurs sur un espace de noms donné (base de données). Les actions sont performantes sur les ressources. Les ressources sont tout objet contenant un état dans la base de données.

Examples

Rôles intégrés

Les rôles de rôles d'utilisateur de base de données et de rôles d'administration de base de données intégrés existent dans chaque base de données.

Rôles d'utilisateur de base de données

1. `read`
2. `readwrite`

Lire Modèle d'autorisation MongoDB en ligne:
https://riptutorial.com/fr/mongodb/topic/8114/modele-d-autorisation-mongodb

Chapitre 13: Mongo comme jeu de répliques

Examples

Mongodb comme un jeu de répliques

Nous créerions mongodb en tant que jeu de répliques comportant 3 instances. Une instance serait primaire et les deux autres seraient secondaires.

Pour simplifier, je vais avoir une réplique avec 3 instances de mongodb s'exécutant sur le même serveur et pour cela, les trois instances mongodb s'exécuteront sur des numéros de port différents.

Dans un environnement de production où une instance mongodb dédiée est exécutée sur un seul serveur, vous pouvez réutiliser les mêmes numéros de port.

1. Créer des répertoires de données (chemin où les données mongodb seraient stockées dans un fichier)

```
- mkdir c:\data\server1 (datafile path for instance 1)
- mkdir c:\data\server2 (datafile path for instance 2)
- mkdir c:\data\server3 (datafile path for instance 3)
```

2. une. Commencez la première instance de mongod

- Ouvrez l'invite de commande et tapez la touche presse suivante.

```
mongod --replSet s0 --dbpath c:\data\server1 --port 37017  --smallfiles --oplogSize 100
```

La commande ci-dessus associe l'instance de mongodb à un nom replicaSet "s0" et lance la première instance de mongodb sur le port 37017 avec oplogSize 100 Mo

2. b. De même commence la deuxième instance de Mongodb

```
mongod --replSet s0 --dbpath c:\data\server2 --port 37018  --smallfiles --oplogSize 100
```

La commande ci-dessus associe l'instance de mongodb à un nom replicaSet "s0" et lance la première instance de mongodb sur le port 37018 avec oplogSize 100 Mo

2. c. Commencez maintenant la troisième instance de Mongodb

```
mongod --replSet s0 --dbpath c:\data\server3 --port 37019  --smallfiles --oplogSize 100
```

La commande ci-dessus associe l'instance de mongodb à un nom replicaSet "s0" et lance la première instance de mongodb sur le port 37019 avec oplogSize 100MB

Avec les 3 instances démarrées, ces 3 instances sont indépendantes les unes des autres

actuellement. Nous devrions maintenant regrouper ces instances en tant que jeu de réplicas. Nous faisons cela avec l'aide d'un objet de configuration.

3.a Connectez-vous à l'un des serveurs mongod via le shell mongo. Pour ce faire, ouvrez l'invite de commande et tapez.

```
mongo --port 37017
```

Une fois connecté au shell mongo, créez un objet de configuration

```
   var config = {"_id":"s0", members[]};
```

cet objet de configuration a 2 attributs

- 1. _id: le nom du jeu de répliques ("s0")
  2. members: [] (members est un tableau d'instances mongod. Gardons ce blanc pour l'instant, nous allons ajouter des membres via la commande push.

3.b Pour pousser (ajouter) des instances mongod au tableau des membres dans l'objet config. Sur le type de coquille mongo

```
config.members.push({"_id":0,"host":"localhost:37017"});
config.members.push({"_id":1,"host":"localhost:37018"});
config.members.push({"_id":2,"host":"localhost:37019"});
```

Nous assignons à chaque instance mongod un _id et un hôte. _id peut être n'importe quel numéro unique et l'hôte doit être le nom d'hôte du serveur sur lequel il s'exécute suivi du numéro de port.

4. Lancez l'objet config par la commande suivante dans le shell mongo.

```
rs.initiate(config)
```

5. Donnez-lui quelques secondes et nous avons un jeu de répliques de 3 instances mongod qui s'exécutent sur le serveur. Tapez la commande suivante pour vérifier l'état du jeu de réplicas et identifier celui qui est principal et le second.

```
rs.status();
```

Lire Mongo comme jeu de répliques en ligne: https://riptutorial.com/fr/mongodb/topic/6603/mongo-comme-jeu-de-repliques

Chapitre 14: Mongo comme jeu de répliques

Examples

Vérifier les états du jeu de répliques MongoDB

Utilisez la commande ci-dessous pour vérifier l'état du jeu de réplicas.

Commande : rs.status ()

Connecter un membre du réplica et déclencher cette commande, il donnera l'état complet du jeu de réplicas

Exemple :

```
{
 "set" : "ReplicaName",
 "date" : ISODate("2016-09-26T07:36:04.935Z"),
 "myState" : 1,
 "term" : NumberLong(-1),
 "heartbeatIntervalMillis" : NumberLong(2000),
 "members" : [
         {
                 "_id" : 0,
                 "name" : "<IP>:<PORT>,
                 "health" : 1,
                 "state" : 1,
                 "stateStr" : "PRIMARY",
                 "uptime" : 5953744,
                 "optime" : Timestamp(1474875364, 36),
                 "optimeDate" : ISODate("2016-09-26T07:36:04Z"),
                 "electionTime" : Timestamp(1468921646, 1),
                 "electionDate" : ISODate("2016-07-19T09:47:26Z"),
                 "configVersion" : 6,
                 "self" : true
         },
         {
                 "_id" : 1,
                 "name" : "<IP>:<PORT>",
                 "health" : 1,
                 "state" : 2,
                 "stateStr" : "SECONDARY",
                 "uptime" : 5953720,
                 "optime" : Timestamp(1474875364, 13),
                 "optimeDate" : ISODate("2016-09-26T07:36:04Z"),
                 "lastHeartbeat" : ISODate("2016-09-26T07:36:04.244Z"),
                 "lastHeartbeatRecv" : ISODate("2016-09-26T07:36:03.871Z"),
                 "pingMs" : NumberLong(0),
                 "syncingTo" : "10.9.52.55:10050",
                 "configVersion" : 6
         },
         {
                 "_id" : 2,
                 "name" : "<IP>:<PORT>",
                 "health" : 1,
```

```
                "state" : 7,
                "stateStr" : "ARBITER",
                "uptime" : 5953696,
                "lastHeartbeat" : ISODate("2016-09-26T07:36:03.183Z"),
                "lastHeartbeatRecv" : ISODate("2016-09-26T07:36:03.715Z"),
                "pingMs" : NumberLong(0),
                "configVersion" : 6
        },
        {
                "_id" : 3,
                "name" : "<IP>:<PORT>",
                "health" : 1,
                "state" : 2,
                "stateStr" : "SECONDARY",
                "uptime" : 1984305,
                "optime" : Timestamp(1474875361, 16),
                "optimeDate" : ISODate("2016-09-26T07:36:01Z"),
                "lastHeartbeat" : ISODate("2016-09-26T07:36:02.921Z"),
                "lastHeartbeatRecv" : ISODate("2016-09-26T07:36:03.793Z"),
                "pingMs" : NumberLong(22),
                "lastHeartbeatMessage" : "syncing from: 10.9.52.56:10050",
                "syncingTo" : "10.9.52.56:10050",
                "configVersion" : 6
        }
    ],
    "ok" : 1
    }
```

De ce qui précède, nous pouvons connaître le statut complet de l'ensemble de répliques

Lire Mongo comme jeu de répliques en ligne: https://riptutorial.com/fr/mongodb/topic/7043/mongo-comme-jeu-de-repliques

Chapitre 15: Mongo comme Shards

Examples

Configuration de l'environnement de fragmentation

Sharding Group Members:

Pour le sharding, il y a trois joueurs.

1. Serveur de configuration
2. Jeux de répliques
3. Mongos

   Pour un fragment mongo, nous devons configurer les trois serveurs ci-dessus.

Configuration du serveur de configuration: ajoutez le fichier suivant au fichier de configuration mongod

```
sharding:
  clusterRole: configsvr
replication:
  replSetName: <setname>
```

exécuter: mongod --config

nous pouvons choisir le serveur de configuration comme jeu de réplicas ou peut être un serveur autonome. Sur la base de nos exigences, nous pouvons choisir le meilleur. Si la configuration doit s'exécuter dans le jeu de réplicas, nous devons suivre la configuration du jeu de réplicas

Configuration du réplica: Créer un jeu de réplicas // Veuillez vous référer à la configuration du réplica

Configuration MongoS: Mongos est la configuration principale de Shard. Son est le routeur de requête pour accéder à tous les jeux de réplicas

Ajoutez ce qui suit dans le fichier de configuration mongos

```
    sharding:
      configDB: <configReplSetName>/cfg1.example.net:27017;
```

Configurer partagé:

Connecter les mongos via le shell (mongo --host --port)

1. sh.addShard ("/s1-mongo1.example.net:27017")

2. sh.enableSharding ("")
3. sh.shardCollection ("<base de données>. <collection>", {<clé>: <direction>})
4. sh.status () // Pour assurer le sharding

Lire Mongo comme Shards en ligne: https://riptutorial.com/fr/mongodb/topic/7044/mongo-comme-shards

Chapitre 16: MongoDB - Configurer un ReplicaSet pour prendre en charge TLS / SSL

Introduction

Comment configurer un ReplicaSet pour prendre en charge TLS / SSL?

Nous déploierons un ReplicaSet à 3 nœuds dans votre environnement local et nous utiliserons un certificat auto-signé. N'utilisez pas de certificat auto-signé dans PRODUCTION.

Comment connecter votre client à ce ReplicaSet?

Nous allons connecter un Mongo Shell.

Une description des certificats TLS / SSL, PKI (infrastructure à clé publique) et de l'autorité de certification dépasse le cadre de cette documentation.

Examples

Comment configurer un ReplicaSet pour prendre en charge TLS / SSL?

Créer le certificat racine

Le certificat racine (fichier CA) sera utilisé pour signer et identifier votre certificat. Pour le générer, exécutez la commande ci-dessous.

```
openssl req -nodes -out ca.pem -new -x509 -keyout ca.key
```

Conservez le certificat racine et sa clé avec soin, les deux seront utilisés pour signer vos certificats. Le certificat racine peut également être utilisé par votre client.

Générer les demandes de certificat et les clés privées

Lors de la génération de la demande de signature de certificat (aka CSR), **saisissez le nom d'hôte (ou IP) exact de votre noeud dans le champ Nom commun (CN). Les autres champs doivent avoir exactement la même valeur.** Vous devrez peut-être modifier votre fichier */ etc / hosts* .

Les commandes ci-dessous génèrent les fichiers CSR et les clés privées RSA (4096 bits).

```
openssl req -nodes -newkey rsa:4096 -sha256 -keyout mongodb_node_1.key -out mongodb_node_1.csr
openssl req -nodes -newkey rsa:4096 -sha256 -keyout mongodb_node_2.key -out mongodb_node_2.csr
openssl req -nodes -newkey rsa:4096 -sha256 -keyout mongodb_node_3.key -out mongodb_node_3.csr
```

Vous devez générer un CSR pour chaque nœud de votre ReplicaSet. N'oubliez pas que le nom commun n'est pas le même d'un nœud à un autre. Ne basez pas plusieurs CSR sur la même clé privée.

Vous devez maintenant avoir 3 CSR et 3 clés privées.

```
mongodb_node_1.key - mongodb_node_2.key - mongodb_node_3.key
mongodb_node_1.csr - mongodb_node_2.csr - mongodb_node_3.csr
```

Signer vos demandes de certificat

Utilisez le fichier CA (ca.pem) et sa clé privée (ca.key) générés précédemment pour signer chaque demande de certificat en exécutant les commandes ci-dessous.

```
openssl x509 -req -in mongodb_node_1.csr -CA ca.pem -CAkey ca.key -set_serial 00 -out
mongodb_node_1.crt
openssl x509 -req -in mongodb_node_2.csr -CA ca.pem -CAkey ca.key -set_serial 00 -out
mongodb_node_2.crt
openssl x509 -req -in mongodb_node_3.csr -CA ca.pem -CAkey ca.key -set_serial 00 -out
mongodb_node_3.crt
```

Vous devez signer chaque CSR.

Vous devez maintenant avoir 3 CSR, 3 clés privées et 3 certificats auto-signés. Seules les clés privées et les certificats seront utilisés par MongoDB.

```
mongodb_node_1.key - mongodb_node_2.key - mongodb_node_3.key
mongodb_node_1.csr - mongodb_node_2.csr - mongodb_node_3.csr
mongodb_node_1.crt - mongodb_node_2.crt - mongodb_node_3.crt
```

Chaque certificat correspond à un nœud. Rappelez-vous soigneusement quel CN / nom d'hôte vous avez donné à chaque CSR.

Concattez chaque certificat de noeud avec sa clé

Exécutez les commandes ci-dessous pour concaténer chaque certificat de nœud avec sa clé dans un fichier (exigence MongoDB).

```
cat mongodb_node_1.key mongodb_node_1.crt > mongodb_node_1.pem
cat mongodb_node_2.key mongodb_node_2.crt > mongodb_node_2.pem
cat mongodb_node_3.key mongodb_node_3.crt > mongodb_node_3.pem
```

Vous devez maintenant avoir 3 fichiers PEM.

```
mongodb_node_1.pem - mongodb_node_2.pem - mongodb_node_3.pem
```

Déployer votre ReplicaSet

Nous supposerons que vos fichiers pem se trouvent dans votre dossier actuel, ainsi que des données / données1, des données / données2 et des données / données3.

Exécutez les commandes ci-dessous pour déployer votre ReplicaSet à 3 nœuds en écoutant les ports 27017, 27018 et 27019.

```
mongod --dbpath data/data_1 --replSet rs0 --port 27017 --sslMode requireSSL --sslPEMKeyFile
mongodb_node_1.pem
mongod --dbpath data/data_2 --replSet rs0 --port 27018 --sslMode requireSSL --sslPEMKeyFile
mongodb_node_2.pem
mongod --dbpath data/data_3 --replSet rs0 --port 27019 --sslMode requireSSL --sslPEMKeyFile
mongodb_node_3.pem
```

Vous avez maintenant un ReplicaSet à 3 nœuds déployé sur votre environnement local et toutes leurs transactions sont chiffrées. Vous ne pouvez pas vous connecter à ce ReplicaSet sans utiliser TLS.

Déployer votre ReplicaSet for Mutual SSL / Mutual Trust

Pour forcer votre client à fournir un certificat client (SSL mutuel), vous devez ajouter le fichier CA lors de l'exécution de vos instances.

```
mongod --dbpath data/data_1 --replSet rs0 --port 27017 --sslMode requireSSL --sslPEMKeyFile
mongodb_node_1.pem --sslCAFile ca.pem
mongod --dbpath data/data_2 --replSet rs0 --port 27018 --sslMode requireSSL --sslPEMKeyFile
mongodb_node_2.pem --sslCAFile ca.pem
mongod --dbpath data/data_3 --replSet rs0 --port 27019 --sslMode requireSSL --sslPEMKeyFile
mongodb_node_3.pem --sslCAFile ca.pem
```

Vous avez maintenant un ReplicaSet à 3 nœuds déployé sur votre environnement local et toutes leurs transactions sont chiffrées. Vous ne pouvez pas vous connecter à ce ReplicaSet sans utiliser TLS ou sans fournir un certificat client approuvé par votre autorité de certification.

Comment connecter votre client (Mongo Shell) à un ReplicaSet?

Aucun SSL mutuel

Dans cet exemple, nous pourrions utiliser le fichier CA (ca.pem) que vous avez généré lors de la section " *Comment configurer un ReplicaSet pour prendre en charge TLS / SSL?* ". Nous supposerons que le fichier CA est situé dans votre dossier actuel.

Nous supposerons que vos 3 nœuds s'exécutent sur mongo1: 27017, mongo2: 27018 et mongo3: 27019. (Vous devrez peut-être modifier votre fichier */ etc / hosts* .)

À partir de MongoDB 3.2.6, si votre fichier CA est enregistré dans le Trust Store de votre système d'exploitation, vous pouvez vous connecter à votre ReplicaSet sans fournir le fichier CA.

```
mongo --ssl --host rs0/mongo1:27017,mongo2:27018,mongo3:27019
```

Sinon, vous devez fournir le fichier CA.

```
mongo --ssl --sslCAFile ca.pem --host rs0/mongo1:27017,mongo2:27018,mongo3:27019
```

Vous êtes maintenant connecté à votre ReplicaSet et toutes les transactions entre votre Mongo Shell et votre ReplicaSet sont cryptées.

Avec SSL mutuel

Si votre ReplicaSet vous demande un certificat client, vous devez en fournir un signé par l'autorité de certification utilisée par le déploiement ReplicaSet. Les étapes pour générer le certificat client sont presque les mêmes que pour générer le certificat de serveur.

En effet, il vous suffit de modifier le champ Nom commun lors de la création de la CSR. Au lieu de fournir 1 nom d'hôte de noeud dans le champ de nom commun, **vous devez fournir tous les noms d'hôte ReplicaSet séparés par une virgule** .

```
openssl req -nodes -newkey rsa:4096 -sha256 -keyout mongodb_client.key -out mongodb_client.csr
...
Common Name (e.g. server FQDN or YOUR name) []: mongo1,mongo2,mongo3
```

Vous pouvez être confronté à la limitation de la taille du nom commun si le champ Nom commun est trop long (plus de 64 octets de long). Pour contourner cette limitation, vous devez utiliser SubjectAltName lors de la génération de la CSR.

```
openssl req -nodes -newkey rsa:4096 -sha256 -keyout mongodb_client.key -out mongodb_client.csr
-config <(
cat <<-EOF
[req]
default_bits = 4096
prompt = no
default_md = sha256
req_extensions = req_ext
distinguished_name = dn

[ dn ]
CN = .

[ req_ext ]
subjectAltName = @alt_names

[ alt_names ]
DNS.1 = mongo1
DNS.2 = mongo2
DNS.3 = mongo3
EOF
)
```

Ensuite, vous signez le CSR à l'aide du certificat et de la clé de l'autorité de certification.

```
openssl x509 -req -in mongodb_client.csr  -CA ca.pem -CAkey ca.key -set_serial 00 -out
mongodb_client.crt
```

Enfin, vous concatrez la clé et le certificat signé.

```
cat mongodb_client.key mongodb_client.crt > mongodb_client.pem
```

Pour vous connecter à votre ReplicaSet, vous pouvez maintenant fournir le nouveau certificat client généré.

```
mongo --ssl --sslCAFile ca.pem --host rs0/mongo1:27017,mongo2:27018,mongo3:27019 --
sslPEMKeyFile mongodb_client.pem
```

Vous êtes maintenant connecté à votre ReplicaSet et toutes les transactions entre votre Mongo Shell et votre ReplicaSet sont cryptées.

Lire MongoDB - Configurer un ReplicaSet pour prendre en charge TLS / SSL en ligne: https://riptutorial.com/fr/mongodb/topic/9539/mongodb---configurer-un-replicaset-pour-prendre-en-charge-tls---ssl

Chapitre 17: Moteurs de stockage enfichables

Remarques

Dans MongoDB 3.0, MMAP (par défaut) et WiredTiger sont les moteurs de stockage stables. En général, si votre application est trop lourde, utilisez MMAP. Si son écriture est lourde, utilisez WiredTiger.

Votre solution peut également comporter un ensemble de membres de réplicas mixtes dans lequel un nœud peut être configuré avec MMAP et un autre avec WiredTiger. Vous pouvez utiliser l'un pour insérer des données massives et l'autre pour lire avec des outils analytiques.

Après MongoDB 3.2, WiredTiger devient le moteur par défaut.

Examples

Le MMAP

MMAP est un moteur de stockage enfichable nommé d'après la commande `mmap()` Linux. Il mappe les fichiers sur la mémoire virtuelle et optimise les appels en lecture. Si vous avez un fichier volumineux mais que vous ne devez en lire qu'une petite partie, `mmap()` est beaucoup plus rapide qu'un appel `read()` qui ramènerait l'intégralité du fichier en mémoire.

Un inconvénient est que vous ne pouvez pas avoir deux appels en écriture traités en parallèle pour la même collection. Ainsi, MMAP dispose d'un verrouillage au niveau de la collection (et non d'un verrouillage au niveau du document comme le propose WiredTiger). Ce verrouillage de collection est nécessaire car un index MMAP peut référencer des documents multiples et si ces documents pouvaient être mis à jour simultanément, l'index serait incohérent.

WiredTiger

WiredTiger prend en charge les **arborescences LSM pour stocker les index** . Les arborescences LSM sont plus rapides pour les opérations d'écriture lorsque vous avez besoin d'écrire des charges de travail énormes d'inserts aléatoires.

Dans WiredTiger, il n'y a **pas de mises à jour sur place** . Si vous devez mettre à jour un élément d'un document, un nouveau document sera inséré lorsque l'ancien document sera supprimé.

WiredTiger offre également une **concurrence au niveau du document** . Cela suppose que deux opérations d'écriture n'affecteront pas le même document, mais si c'est le cas, une opération sera rembobinée et exécutée plus tard. C'est un bon coup de pouce si les rembobinages sont rares.

WiredTiger prend en **charge les algorithmes Snappy et zLib pour la compression** des données et des index dans le système de fichiers. Snappy est la valeur par défaut. Il consomme moins de ressources processeur mais a un taux de compression inférieur à celui de zLib.

Comment utiliser le moteur WiredTiger

```
mongod --storageEngine wiredTiger --dbpath <newWiredTigerDBPath>
```

Remarque:

1. Après mongodb 3.2, le moteur par défaut est WiredTiger.
2. `newWiredTigerDBPath` ne doit pas contenir de données d'un autre moteur de stockage. Pour migrer vos données, vous devez les sauvegarder et les réimporter dans le nouveau moteur de stockage.

```
    mongodump --out <exportDataDestination>
    mongod --storageEngine wiredTiger --dbpath <newWiredTigerDBPath>
    mongorestore <exportDataDestination>
```

En mémoire

Toutes les données sont stockées en mémoire (RAM) pour une lecture / un accès plus rapide.

mongo-roches

Un moteur de valeur-clé créé pour s'intégrer à RocksDB de Facebook.

Fusion-io

Un moteur de stockage créé par SanDisk qui permet de contourner la couche du système de fichiers du système d'exploitation et d'écrire directement sur le périphérique de stockage.

TokuMX

Un moteur de stockage créé par Percona qui utilise des index d'arborescence fractale.

Lire Moteurs de stockage enfichables en ligne:
https://riptutorial.com/fr/mongodb/topic/694/moteurs-de-stockage-enfichables

Chapitre 18: Obtenir des informations sur la base de données

Examples

Liste toutes les bases de données

```
show dbs
```

ou

```
db.adminCommand('listDatabases')
```

ou

```
db.getMongo().getDBNames()
```

Liste toutes les collections dans la base de données

```
show collections
```

ou

```
show tables
```

ou

```
db.getCollectionNames()
```

Lire Obtenir des informations sur la base de données en ligne:
https://riptutorial.com/fr/mongodb/topic/6397/obtenir-des-informations-sur-la-base-de-donnees

Chapitre 19: Opération CRUD

Syntaxe

- insérer (*document ou tableau de documents*)
- insertOne ('document', {writeConcern: 'document'})
- insertMany ({[document 1, document 2, ...]}, {writeConcern: document, ordonné: booléen})
- trouver (*requête* , *projection*)
- findOne (*requête* , *projection*)
- mise à jour (*requête* , *mise à jour*)
- updateOne (*query* , *update* , {upsert: boolean, writeConcern: document})
- updateMany (*requête* , *update* , {upert: boolean, writeConcern: document})
- replaceOne (*requête* , *remplacement* , {upsert: boolean, writeConcern: document})
- supprimer (*query* , *justOne*)
- findAndModify (*requête* , *tri* , *mise à jour* , *options [facultatif]*)

Remarques

La mise à jour et la suppression d'un document doivent être effectuées avec soin. Étant donné que l'opération peut affecter plusieurs documents.

Examples

Créer

```
db.people.insert({name: 'Tom', age: 28});
```

Ou

```
db.people.save({name: 'Tom', age: 28});
```

La différence avec `save` est que si le document contient un passé `_id` champ, si un document existe déjà avec cette `_id` il sera mis à jour au lieu d'être ajouté comme nouveau.

Deux nouvelles méthodes pour insérer des documents dans une collection, dans MongoDB 3.2.x: -

Utilisez `insertOne` pour insérer un seul enregistrement: -

```
db.people.insertOne({name: 'Tom', age: 28});
```

Utilisez `insertMany` pour insérer plusieurs enregistrements: -

```
db.people.insertMany([{name: 'Tom', age: 28},{name: 'John', age: 25}, {name: 'Kathy', age:
```

```
23}])
```

Notez que l' `insert` est mise en évidence comme obsolète dans chaque pilote de langue officielle depuis la version 3.0. La distinction complète est que les méthodes shell ont en fait pris du retard sur les autres pilotes dans la mise en œuvre de la méthode. La même chose s'applique à toutes les autres méthodes CRUD

Mettre à jour

Mettre à jour l'objet **entier** :

```
db.people.update({name: 'Tom'}, {age: 29, name: 'Tom'})

// New in MongoDB 3.2
db.people.updateOne({name: 'Tom'},{age: 29, name: 'Tom'}) //Will replace only first matching
document.

db.people.updateMany({name: 'Tom'},{age: 29, name: 'Tom'}) //Will replace all matching
documents.
```

Ou simplement mettre à jour un seul champ d'un document. Dans ce cas l' `age` :

```
db.people.update({name: 'Tom'}, {$set: {age: 29}})
```

Vous pouvez également mettre à jour plusieurs documents simultanément en ajoutant un troisième paramètre. Cette requête mettra à jour tous les documents dont le nom est égal à `Tom` :

```
db.people.update({name: 'Tom'}, {$set: {age: 29}}, {multi: true})

// New in MongoDB 3.2
db.people.updateOne({name: 'Tom'},{$set:{age: 30}) //Will update only first matching document.

db.people.updateMany({name: 'Tom'},{$set:{age: 30}}) //Will update all matching documents.
```

Si un nouveau champ est à venir pour la mise à jour, ce champ sera ajouté au document.

```
db.people.updateMany({name: 'Tom'},{$set:{age: 30, salary:50000}})// Document will have
`salary` field as well.
```

Si un document doit être remplacé,

```
db.collection.replaceOne({name:'Tom'}, {name:'Lakmal',age:25,address:'Sri Lanka'})
```

peut être utilisé.

Remarque : Les champs que vous utilisez pour identifier l'objet seront enregistrés dans le document mis à jour. Les champs non définis dans la section de mise à jour seront supprimés du document.

Effacer

Supprime tous les documents correspondant au paramètre de requête:

```
// New in MongoDB 3.2
db.people.deleteMany({name: 'Tom'})

// All versions
db.people.remove({name: 'Tom'})
```

Ou juste un

```
// New in MongoDB 3.2
db.people.deleteOne({name: 'Tom'})

// All versions
db.people.remove({name: 'Tom'}, true)
```

La méthode `remove()` de MongoDB. Si vous exécutez cette commande sans argument ou sans argument vide, tous les documents seront supprimés de la collection.

```
db.people.remove();
```

ou

```
db.people.remove({});
```

Lis

Requête pour tous les documents de la collection de `people` qui ont un champ de `name` avec une valeur de `'Tom'` :

```
db.people.find({name: 'Tom'})
```

Ou juste le premier:

```
db.people.findOne({name: 'Tom'})
```

Vous pouvez également spécifier les champs à renvoyer en transmettant un paramètre de sélection de champ. Les éléments suivants excluront le champ `_id` et incluront uniquement le champ `age` :

```
db.people.find({name: 'Tom'}, {_id: 0, age: 1})
```

Note: par défaut, le champ `_id` sera retourné, même si vous ne le demandez pas. Si vous ne souhaitez pas récupérer l' `_id` , vous pouvez simplement suivre l'exemple précédent et demander que l' `_id` soit exclu en spécifiant `_id: 0` (ou `_id: false`). Si vous souhaitez trouver un sous-enregistrement comme l'adresse, l'objet contient le pays , ville, etc.

```
db.people.find({'address.country': 'US'})
```

& spécifier le champ si nécessaire

```
db.people.find({'address.country': 'US'}, {'name': true, 'address.city': true})Remember that
the result has a `.pretty()` method that pretty-prints resulting JSON:

db.people.find().pretty()
```

Plus d'opérateurs de mise à jour

Vous pouvez utiliser d'autres opérateurs que `$set` lors de la mise à jour d'un document. L'opérateur `$push` vous permet de pousser une valeur dans un tableau, dans ce cas nous allons ajouter un nouveau surnom au tableau des `nicknames` .

```
db.people.update({name: 'Tom'}, {$push: {nicknames: 'Tommy'}})
// This adds the string 'Tommy' into the nicknames array in Tom's document.
```

L'opérateur `$pull` est l'opposé de `$push` , vous pouvez extraire des éléments spécifiques des tableaux.

```
db.people.update({name: 'Tom'}, {$pull: {nicknames: 'Tommy'}})
// This removes the string 'Tommy' from the nicknames array in Tom's document.
```

L'opérateur `$pop` vous permet de supprimer la première ou la dernière valeur d'un tableau. Disons que le document de Tom a une propriété appelée frères et sœurs qui a la valeur `['Marie', 'Bob', 'Kevin', 'Alex']` .

```
db.people.update({name: 'Tom'}, {$pop: {siblings: -1}})
// This will remove the first value from the siblings array, which is 'Marie' in this case.


db.people.update({name: 'Tom'}, {$pop: {siblings: 1}})
// This will remove the last value from the siblings array, which is 'Alex' in this case.
```

Paramètre "multi" lors de la mise à jour de plusieurs documents

Pour mettre à jour plusieurs documents dans une collection, définissez l'option multi sur true.

```
db.collection.update(
   query,
   update,
   {
     upsert: boolean,
     multi: boolean,
     writeConcern: document
   }
)
```

multi est facultatif. Si défini sur true, met à jour plusieurs documents répondant aux critères de la requête. Si la valeur est false, met à jour un document. La valeur par défaut est false.

db.mycol.find () {"_id": ObjectId (598354878df45ec5), "title": "Présentation de

MongoDB"} {"_id": ObjectId (59835487adf45ec6), "title": "Présentation de NoSQL"} {"_id": ObjectId (59835487adf45ec7), "title": "Présentation du point de tutoriels"}

db.mycol.update ({'title': 'MongoDB Overview'}, {$ set: {'title': 'Nouveau tutoriel MongoDB'}}, {multi: true})

Mise à jour des documents incorporés.

Pour le schéma suivant:

```
{name: 'Tom', age: 28, marks: [50, 60, 70]}
```

Mettez à jour les marques de Tom à 55 où les marques sont 50 (utilisez l'opérateur de position $):

```
db.people.update({name: "Tom", marks: 50}, {"$set": {"marks.$": 55}})
```

Pour le schéma suivant:

```
{name: 'Tom', age: 28, marks: [{subject: "English", marks: 90},{subject: "Maths", marks: 100},
{subject: "Computes", marks: 20}]}
```

Mettez à jour les marques anglaises de Tom à 85:

```
db.people.update({name: "Tom", "marks.subject": "English"},{"$set":{"marks.$.marks": 85}})
```

Expliquant l'exemple ci-dessus:

En utilisant {name: "Tom", "marks.subject": "English"}, vous obtiendrez la position de l'objet dans le tableau des marques, où sujet est l'anglais. Dans "marks. $. Marks", $ est utilisé pour mettre à jour dans cette position du tableau de marques

Mettre à jour les valeurs dans un tableau

L'opérateur positionnel $ identifie un élément dans un tableau à mettre à jour sans spécifier explicitement la position de l'élément dans le tableau.

Considérez une collection d'étudiants avec les documents suivants:

```
{ "_id" : 1, "grades" : [ 80, 85, 90 ] }
{ "_id" : 2, "grades" : [ 88, 90, 92 ] }
{ "_id" : 3, "grades" : [ 85, 100, 90 ] }
```

Pour mettre à jour 80 à 82 dans le tableau grades du premier document, utilisez l'opérateur positional $ si vous ne connaissez pas la position de l'élément dans le tableau:

```
db.students.update(
   { _id: 1, grades: 80 },
   { $set: { "grades.$" : 82 } }
)
```

Lire Opération CRUD en ligne: https://riptutorial.com/fr/mongodb/topic/1683/operation-crud

Chapitre 20: Opérations en vrac

Remarques

Construire une liste d'opérations d'écriture à exécuter en bloc pour une collection unique.

Examples

Conversion d'un champ en un autre type et mise à jour de la totalité de la collection en bloc

Généralement, le cas où l'on veut changer un type de champ en un autre, par exemple, la collection d'origine peut avoir des champs "numériques" ou "date" enregistrés en tant que chaînes:

```
{
    "name": "Alice",
    "salary": "57871",
    "dob": "1986-08-21"
},
{
    "name": "Bob",
    "salary": "48974",
    "dob": "1990-11-04"
}
```

L'objectif serait de mettre à jour une collection colossale comme ci-dessus pour

```
{
    "name": "Alice",
    "salary": 57871,
    "dob": ISODate("1986-08-21T00:00:00.000Z")
},
{
    "name": "Bob",
    "salary": 48974,
    "dob": ISODate("1990-11-04T00:00:00.000Z")
}
```

Pour des données relativement petites, on peut obtenir ce qui précède en itérant la collection en utilisant un `snapshot` avec la méthode `forEach()` du curseur et en mettant à jour chaque document comme suit:

```
db.test.find({
    "salary": { "$exists": true, "$type": 2 },
    "dob": { "$exists": true, "$type": 2 }
}).snapshot().forEach(function(doc){
    var newSalary = parseInt(doc.salary),
        newDob = new ISODate(doc.dob);
    db.test.updateOne(
```

```
        { "_id": doc._id },
        { "$set": { "salary": newSalary, "dob": newDob } }
    );
});
```

Bien que cela soit optimal pour les petites collections, les performances avec des collections importantes sont considérablement réduites, car la lecture en boucle d'un ensemble de données volumineux et l'envoi de chaque opération de mise à jour par requête entraînent une pénalité informatique.

L'API `Bulk()` vient à la rescousse et améliore considérablement les performances car les opérations d'écriture ne sont envoyées qu'une seule fois au serveur. L'efficacité est atteinte car la méthode n'envoie pas toutes les demandes d'écriture au serveur (comme avec l'instruction de mise à jour actuelle dans la boucle `forEach()`) mais seulement une fois sur 1000, ce qui rend les mises à jour plus efficaces et plus rapides.

En utilisant le même concept ci-dessus avec la boucle `forEach()` pour créer les lots, nous pouvons mettre à jour la collection en bloc comme suit. Dans cette démonstration, l'API `Bulk()` disponible dans les versions MongoDB `>= 2.6` et `< 3.2` utilise la méthode `initializeUnorderedBulkOp()` pour exécuter en parallèle, ainsi que dans un ordre non déterministe, les opérations d'écriture dans les lots.

Il met à jour tous les documents de la collection clients en changeant les `salary` et `dob` champs à `numerical` et `datetime` valeurs respectivement:

```
var bulk = db.test.initializeUnorderedBulkOp(),
    counter = 0; // counter to keep track of the batch update size

db.test.find({
    "salary": { "$exists": true, "$type": 2 },
    "dob": { "$exists": true, "$type": 2 }
}).snapshot().forEach(function(doc){
    var newSalary = parseInt(doc.salary),
        newDob = new ISODate(doc.dob);
    bulk.find({ "_id": doc._id }).updateOne({
        "$set": { "salary": newSalary, "dob": newDob }
    });

    counter++; // increment counter
    if (counter % 1000 == 0) {
        bulk.execute(); // Execute per 1000 operations and re-initialize every 1000 update
statements
        bulk = db.test.initializeUnorderedBulkOp();
    }
});
```

L'exemple suivant s'applique à la nouvelle version `3.2` MongoDB qui a depuis abandonné l'API `Bulk()` et fourni un nouvel ensemble d' `bulkWrite()` utilisant `bulkWrite()` .

Il utilise les mêmes curseurs que ci-dessus, mais crée les tableaux avec les opérations en bloc en utilisant la même méthode de curseur `forEach()` pour pousser chaque document en bloc dans le

tableau. Les commandes d'écriture ne pouvant pas accepter plus de 1 000 opérations, il est nécessaire de regrouper les opérations pour avoir au maximum 1 000 opérations et de ré-initialiser le tableau lorsque la boucle atteint l'itération 1000:

```
var cursor = db.test.find({
        "salary": { "$exists": true, "$type": 2 },
        "dob": { "$exists": true, "$type": 2 }
    }),
    bulkUpdateOps = [];

cursor.snapshot().forEach(function(doc){
    var newSalary = parseInt(doc.salary),
        newDob = new ISODate(doc.dob);
    bulkUpdateOps.push({
        "updateOne": {
            "filter": { "_id": doc._id },
            "update": { "$set": { "salary": newSalary, "dob": newDob } }
         }
    });

    if (bulkUpdateOps.length === 1000) {
        db.test.bulkWrite(bulkUpdateOps);
        bulkUpdateOps = [];
    }
});

if (bulkUpdateOps.length > 0) { db.test.bulkWrite(bulkUpdateOps); }
```

Lire Opérations en vrac en ligne: https://riptutorial.com/fr/mongodb/topic/6211/operations-en-vrac

Chapitre 21: Pilote Java

Examples

Créer un curseur disponible

```
find(query).projection(fields).cursorType(CursorType.TailableAwait).iterator();
```

Ce code s'applique à la classe `MongoCollection` .

CursorType est un enum et il a les valeurs suivantes:

```
Tailable
TailableAwait
```

Correspondant à l'ancien (<3.0) DBCursor addOption Bytes:

```
Bytes.QUERYOPTION_TAILABLE
Bytes.QUERYOPTION_AWAITDATA
```

Créer un utilisateur de base de données

Pour créer un **dev** utilisateur avec mot de passe **password123**

```
MongoClient mongo = new MongoClient("localhost", 27017);
MongoDatabase db =  mongo.getDatabase("testDB");
Map<String, Object> commandArguments = new BasicDBObject();
commandArguments.put("createUser", "dev");
commandArguments.put("pwd", "password123");
String[] roles = { "readWrite" };
commandArguments.put("roles", roles);
BasicDBObject command = new BasicDBObject(commandArguments);
db.runCommand(command);
```

Récupérer les données de la collection avec la condition

Pour obtenir des données de `testcollection` collection dans `testdb` base de données où `name=dev`

```
import org.bson.Document;
import com.mongodb.BasicDBObject;
import com.mongodb.MongoClient;
import com.mongodb.ServerAddress;
import com.mongodb.client.MongoCollection;
import com.mongodb.client.MongoCursor;
import com.mongodb.client.MongoDatabase;

MongoClient mongoClient = new MongoClient(new ServerAddress("localhost", 27017));
MongoDatabase db = mongoClient.getDatabase("testdb");
MongoCollection<Document> collection = db.getCollection("testcollection");
```

```
BasicDBObject searchQuery = new BasicDBObject();
searchQuery.put("name","dev");

MongoCursor<Document> cursor = collection.find(searchQuery).iterator();
try {
    while (cursor.hasNext()) {
        System.out.println(cursor.next().toJson());
    }
} finally {
    cursor.close();
}
```

Lire Pilote Java en ligne: https://riptutorial.com/fr/mongodb/topic/6286/pilote-java

Chapitre 22: Pilote Python

Syntaxe

- mongodb: // [nom d'utilisateur: mot de passe @] host1 [: port1] [, host2 [: port2], ... [, hostN [: portN]]] [/ [base de données] [? options]]

Paramètres

Paramètre	Détail
hostX	Optionnel. Vous pouvez spécifier autant d'hôtes que nécessaire. Vous devez spécifier plusieurs hôtes, par exemple, pour les connexions aux jeux de réplicas.
: portX	Optionnel. La valeur par défaut est: 27017 si non spécifié.
base de données	Optionnel. Le nom de la base de données à authentifier si la chaîne de connexion inclut des informations d'authentification. Si / database n'est pas spécifié et que la chaîne de connexion inclut des informations d'identification, le pilote s'authentifiera dans la base de données admin.
? options	Options spécifiques à la connexion

Examples

Connectez-vous à MongoDB en utilisant pymongo

```
from pymongo import MongoClient

uri = "mongodb://localhost:27017/"

client = MongoClient(uri)

db = client['test_db']
# or
# db = client.test_db

# collection = db['test_collection']
# or
collection = db.test_collection


collection.save({"hello":"world"})

print collection.find_one()
```

PyMongo interroge

Une fois que vous avez un objet de `collection` , les requêtes utilisent la même syntaxe que dans le shell mongo. Quelques légères différences sont:

- chaque clé doit être entre crochets. Par exemple:

  ```
  db.find({frequencies: {$exists: true}})
  ```

  devient en `pymongo` (notez le `True` en majuscule):

  ```
  db.find({"frequencies": { "$exists": True }})
  ```

- Les objets tels que les identificateurs d'objet ou `ISODate` sont manipulés à l'aide de classes python. PyMongo utilise sa propre classe `ObjectId` pour traiter les identifiants d'objet, tandis que les dates utilisent le package standard `datetime` . Par exemple, si vous souhaitez interroger tous les événements entre 2010 et 2011, vous pouvez effectuer les opérations suivantes:

  ```
  from datetime import datetime

  date_from = datetime(2010, 1, 1)
  date_to = datetime(2011, 1, 1)
  db.find({ "date": { "$gte": date_from, "$lt": date_to } }):
  ```

Mettre à jour tous les documents d'une collection en utilisant PyMongo

Disons que vous devez ajouter un champ à chaque document d'une collection.

```
import pymongo

client = pymongo.MongoClient('localhost', 27017)
db = client.mydb.mycollection

for doc in db.find():
   db.update(
       {'_id': doc['_id']},
       {'$set': {'newField': 10} }, upsert=False, multi=False)
```

La méthode `find` renvoie un `Cursor` sur lequel vous pouvez facilement parcourir le syntaxe de `for in` . Ensuite, nous appelons la méthode `update` , en spécifiant le `_id` et en ajoutant un champ (`$set`). Les paramètres `upsert` et `multi` proviennent de mongodb (voir ici pour plus d'informations).

Lire Pilote Python en ligne: https://riptutorial.com/fr/mongodb/topic/7843/pilote-python

Chapitre 23: Querying for Data (Démarrage)

Introduction

Exemples d'interrogation de base

Examples

Trouver()

récupérer tous les documents d'une collection

```
db.collection.find({});
```

récupérer des documents dans une collection en utilisant une condition (similaire à WHERE dans MYSQL)

```
db.collection.find({key: value});
example
  db.users.find({email:"sample@email.com"});
```

récupérer des documents dans une collection en utilisant des conditions booléennes (opérateurs de requête)

```
//AND
db.collection.find( {
    $and: [
     { key: value }, { key: value }
    ]
})
//OR
db.collection.find( {
    $or: [
     { key: value }, { key: value }
    ]
})
//NOT
db.inventory.find( { key: { $not: value } } )
```

plus d'opérations booléennes et des exemples peuvent être trouvés ici

REMARQUE: *find ()* continuera à rechercher la collection même si une correspondance de document a été trouvée. Par conséquent, elle est inefficace dans une collection volumineuse, mais en modélisant soigneusement vos données et / ou en utilisant des index, vous pouvez augmenter l'efficacité de la *recherche ( )*

FindOne ()

```
db.collection.findOne({});
```

La fonctionnalité d'interrogation est similaire à find (), mais cela mettra fin à l'exécution au moment où elle trouvera un document correspondant à sa condition. Si elle est utilisée avec un objet vide, elle ira chercher le premier document et le renverra. Documentation de api findOne () mongodb

Document de requête - Utilisation des conditions AND, OR et IN

Tous les documents de la collection des `students` .

```
> db.students.find().pretty();

{
    "_id" : ObjectId("58f29a694117d1b7af126dca"),
    "studentNo" : 1,
    "firstName" : "Prosen",
    "lastName" : "Ghosh",
    "age" : 25
}
{
    "_id" : ObjectId("58f29a694117d1b7af126dcb"),
    "studentNo" : 2,
    "firstName" : "Rajib",
    "lastName" : "Ghosh",
    "age" : 25
}
{
    "_id" : ObjectId("58f29a694117d1b7af126dcc"),
    "studentNo" : 3,
    "firstName" : "Rizve",
    "lastName" : "Amin",
    "age" : 23
}
{
    "_id" : ObjectId("58f29a694117d1b7af126dcd"),
    "studentNo" : 4,
    "firstName" : "Jabed",
    "lastName" : "Bangali",
    "age" : 25
}
{
    "_id" : ObjectId("58f29a694117d1b7af126dce"),
    "studentNo" : 5,
    "firstName" : "Gm",
    "lastName" : "Anik",
    "age" : 23
}
```

Similaire `mySql` Requête de la commande ci-dessus.

```
SELECT * FROM students;
```

```
db.students.find({firstName:"Prosen"});

{ "_id" : ObjectId("58f2547804951ad51ad206f5"), "studentNo" : "1", "firstName" : "Prosen",
"lastName" : "Ghosh", "age" : "23" }
```

Similaire `mySql` Requête de la commande ci-dessus.

```
SELECT * FROM students WHERE firstName = "Prosen";
```

ET requêtes

```
db.students.find({
    "firstName": "Prosen",
    "age": {
        "$gte": 23
    }
});

{ "_id" : ObjectId("58f29a694117d1b7af126dca"), "studentNo" : 1, "firstName" : "Prosen",
"lastName" : "Ghosh", "age" : 25 }
```

Similaire `mySql` Requête de la commande ci-dessus.

```
SELECT * FROM students WHERE firstName = "Prosen" AND age >= 23
```

Ou des requêtes

```
db.students.find({
     "$or": [{
         "firstName": "Prosen"
     }, {
         "age": {
             "$gte": 23
         }
     }]
 });

{ "_id" : ObjectId("58f29a694117d1b7af126dca"), "studentNo" : 1, "firstName" : "Prosen",
"lastName" : "Ghosh", "age" : 25 }
{ "_id" : ObjectId("58f29a694117d1b7af126dcb"), "studentNo" : 2, "firstName" : "Rajib",
"lastName" : "Ghosh", "age" : 25 }
{ "_id" : ObjectId("58f29a694117d1b7af126dcc"), "studentNo" : 3, "firstName" : "Rizve",
"lastName" : "Amin", "age" : 23 }
{ "_id" : ObjectId("58f29a694117d1b7af126dcd"), "studentNo" : 4, "firstName" : "Jabed",
"lastName" : "Bangali", "age" : 25 }
{ "_id" : ObjectId("58f29a694117d1b7af126dce"), "studentNo" : 5, "firstName" : "Gm",
"lastName" : "Anik", "age" : 23 }
```

Similaire `mySql` Requête de la commande ci-dessus.

```
SELECT * FROM students WHERE firstName = "Prosen" OR age >= 23
```

Et OU questions

```
db.students.find({
        firstName : "Prosen",
        $or : [
            {age : 23},
            {age : 25}
```

```
        ]
});

{ "_id" : ObjectId("58f29a694117d1b7af126dca"), "studentNo" : 1, "firstName" : "Prosen",
"lastName" : "Ghosh", "age" : 25 }
```

Similaire mySql Requête de la commande ci-dessus.

```
SELECT * FROM students WHERE firstName = "Prosen" AND age = 23 OR age = 25;
```

Requêtes IN Ces requêtes peuvent améliorer l'utilisation multiple des requêtes OR

```
db.students.find(lastName:{$in:["Ghosh", "Amin"]})

{ "_id" : ObjectId("58f29a694117d1b7af126dca"), "studentNo" : 1, "firstName" : "Prosen",
"lastName" : "Ghosh", "age" : 25 }
{ "_id" : ObjectId("58f29a694117d1b7af126dcb"), "studentNo" : 2, "firstName" : "Rajib",
"lastName" : "Ghosh", "age" : 25 }
{ "_id" : ObjectId("58f29a694117d1b7af126dcc"), "studentNo" : 3, "firstName" : "Rizve",
"lastName" : "Amin", "age" : 23 }
```

Requête mySql similaire à la commande ci-dessus

```
select * from students where lastName in ('Ghosh', 'Amin')
```

Méthode find () avec Projection

La syntaxe de base de la méthode `find()` avec projection est la suivante

```
> db.COLLECTION_NAME.find({},{KEY:1});
```

Si vous souhaitez afficher tous les documents sans le champ age, la commande est la suivante

```
db.people.find({},{age : 0});
```

Si vous voulez afficher tous les documents dans le champ d'âge, la commande est la suivante

Méthode Find () avec Projection

Dans MongoDB, la projection consiste à sélectionner uniquement les données nécessaires plutôt que de sélectionner la totalité des données d'un document.

La syntaxe de base de la méthode `find()` avec projection est la suivante

```
> db.COLLECTION_NAME.find({},{KEY:1});
```

Si vous souhaitez afficher tous les documents sans le champ d'âge, la commande est la suivante

```
> db.people.find({},{age:0});
```

Si vous souhaitez afficher uniquement le champ d'âge, la commande est la suivante

```
> db.people.find({},{age:1});
```

Remarque: le champ `_id` est toujours affiché lors de l'exécution de la méthode `find()` , si vous ne souhaitez pas utiliser ce champ, vous devez le définir sur `0` .

```
> db.people.find({},{name:1,_id:0});
```

Remarque: `1` est utilisé pour afficher le champ tandis que `0` est utilisé pour masquer les champs.

limiter, ignorer, trier et compter les résultats de la méthode find ()

Semblable aux méthodes d'agrégation également par la méthode find (), vous avez la possibilité de limiter, ignorer, trier et compter les résultats. Disons que nous avons la collection suivante:

```
db.test.insertMany([
    {name:"Any", age:"21", status:"busy"},
    {name:"Tony", age:"25", status:"busy"},
    {name:"Bobby", age:"28", status:"online"},
    {name:"Sonny", age:"28", status:"away"},
    {name:"Cher", age:"20", status:"online"}
])
```

Pour lister la collection:

```
db.test.find({})
```

Reviendra:

```
{ "_id" : ObjectId("592516d7fbd5b591f53237b0"), "name" : "Any", "age" : "21", "status" :
"busy" }
{ "_id" : ObjectId("592516d7fbd5b591f53237b1"), "name" : "Tony", "age" : "25", "status" :
"busy" }
{ "_id" : ObjectId("592516d7fbd5b591f53237b2"), "name" : "Bobby", "age" : "28", "status" :
"online" }
{ "_id" : ObjectId("592516d7fbd5b591f53237b3"), "name" : "Sonny", "age" : "28", "status" :
"away" }
{ "_id" : ObjectId("592516d7fbd5b591f53237b4"), "name" : "Cher", "age" : "20", "status" :
"online" }
```

Pour sauter les 3 premiers documents:

```
db.test.find({}).skip(3)
```

Reviendra:

```
{ "_id" : ObjectId("592516d7fbd5b591f53237b3"), "name" : "Sonny", "age" : "28", "status" :
"away" }
{ "_id" : ObjectId("592516d7fbd5b591f53237b4"), "name" : "Cher", "age" : "20", "status" :
"online" }
```

Pour trier par ordre décroissant du nom du champ:

```
db.test.find({}).sort({ "name" : -1})
```

Reviendra:

```
{ "_id" : ObjectId("592516d7fbd5b591f53237b1"), "name" : "Tony", "age" : "25", "status" :
"busy" }
{ "_id" : ObjectId("592516d7fbd5b591f53237b3"), "name" : "Sonny", "age" : "28", "status" :
"away" }
{ "_id" : ObjectId("592516d7fbd5b591f53237b4"), "name" : "Cher", "age" : "20", "status" :
"online" }
{ "_id" : ObjectId("592516d7fbd5b591f53237b2"), "name" : "Bobby", "age" : "28", "status" :
"online" }
{ "_id" : ObjectId("592516d7fbd5b591f53237b0"), "name" : "Any", "age" : "21", "status" :
"busy" }
```

Si vous voulez trier par ordre croissant, remplacez -1 par 1

Pour compter les résultats:

```
db.test.find({}).count()
```

Reviendra:

```
5
```

Des combinaisons de ces méthodes sont également autorisées. Par exemple, obtenir 2 documents de la collection triée par ordre décroissant en ignorant le premier 1:

```
db.test.find({}).sort({ "name" : -1}).skip(1).limit(2)
```

Reviendra:

```
{ "_id" : ObjectId("592516d7fbd5b591f53237b3"), "name" : "Sonny", "age" : "28", "status" :
"away" }
{ "_id" : ObjectId("592516d7fbd5b591f53237b4"), "name" : "Cher", "age" : "20", "status" :
"online" }
```

Lire Querying for Data (Démarrage) en ligne:
https://riptutorial.com/fr/mongodb/topic/9271/querying-for-data--demarrage-

Chapitre 24: Réplication

Examples

Configuration de base avec trois nœuds

Le jeu de réplicas est un groupe d'instances mongod qui conservent le même ensemble de données.

Cet exemple montre comment configurer un jeu de réplicas avec trois instances sur le même serveur.

Création de dossiers de données

```
mkdir /srv/mongodb/data/rs0-0
mkdir /srv/mongodb/data/rs0-1
mkdir /srv/mongodb/data/rs0-2
```

Commencer des instances de mongod

```
mongod --port 27017 --dbpath /srv/mongodb/data/rs0-0 --replSet rs0
mongod --port 27018 --dbpath /srv/mongodb/data/rs0-1 --replSet rs0
mongod --port 27019 --dbpath /srv/mongodb/data/rs0-2 --replSet rs0
```

Configuration du jeu de réplicas

```
mongo --port 27017    // connection to the instance 27017

rs.initiate();                // initilization of replica set on the 1st node
rs.add("<hostname>:27018")    // adding a 2nd node
rs.add("<hostname>:27019")    // adding a 3rd node
```

Tester votre configuration

Pour vérifier le type de configuration `rs.status()` , le résultat doit être comme `rs.status()` :

```
{
        "set" : "rs0",
        "date" : ISODate("2016-09-01T12:34:24.968Z"),
        "myState" : 1,
        "term" : NumberLong(4),
        "heartbeatIntervalMillis" : NumberLong(2000),
        "members" : [
                {
                        "_id" : 0,
                        "name" : "<hostname>:27017",
                        "health" : 1,
                        "state" : 1,
                        "stateStr" : "PRIMARY",
                        ........................
                },
```

```
                {
                        "_id" : 1,
                        "name" : "<hostname>:27018",
                        "health" : 1,
                        "state" : 2,
                        "stateStr" : "SECONDARY",
                        ............................
                },
                {
                        "_id" : 2,
                        "name" : "<hostname>:27019",
                        "health" : 1,
                        "state" : 2,
                        "stateStr" : "SECONDARY",
                        ............................
                }
        ],
        "ok" : 1
}
```

Lire Réplication en ligne: https://riptutorial.com/fr/mongodb/topic/6205/replication

Chapitre 25: Sauvegarde et restauration de données

Examples

mongoimport avec JSON

Exemple de jeu de données zipcode dans zipcodes.json stocké dans c: \ Users \ yc03ak1 \ Desktop \ zips.json

```
{ "_id" : "01001", "city" : "AGAWAM", "loc" : [ -72.622739, 42.070206 ], "pop" : 15338,
"state" : "MA" }
{ "_id" : "01002", "city" : "CUSHMAN", "loc" : [ -72.51564999999999, 42.377017 ], "pop" :
36963, "state" : "MA" }
{ "_id" : "01005", "city" : "BARRE", "loc" : [ -72.10835400000001, 42.409698 ], "pop" : 4546,
"state" : "MA" }
{ "_id" : "01007", "city" : "BELCHERTOWN", "loc" : [ -72.41095300000001, 42.275103 ], "pop" :
10579, "state" : "MA" }
{ "_id" : "01008", "city" : "BLANDFORD", "loc" : [ -72.936114, 42.182949 ], "pop" : 1240,
"state" : "MA" }
{ "_id" : "01010", "city" : "BRIMFIELD", "loc" : [ -72.188455, 42.116543 ], "pop" : 3706,
"state" : "MA" }
{ "_id" : "01011", "city" : "CHESTER", "loc" : [ -72.988761, 42.279421 ], "pop" : 1688,
"state" : "MA" }
```

pour importer cet ensemble de données dans la base de données nommée "test" et collection nommée "zips"

```
C:\Users\yc03ak1>mongoimport --db test --collection "zips" --drop --type json --host
"localhost:47019"  --file "c:\Users\yc03ak1\Desktop\zips.json"
```

- --db: nom de la base de données dans laquelle les données doivent être importées
- --collection: nom de la collection dans la base de données où les données doivent être imprimées
- --drop: supprime d'abord la collection avant d'importer
- --type: type de document à importer. JSON par défaut
- --host: hôte et port mongodb sur lesquels les données doivent être importées.
- --file: chemin où se trouve le fichier json

sortie:

```
2016-08-10T20:10:50.159-0700    connected to: localhost:47019
2016-08-10T20:10:50.163-0700    dropping: test.zips
2016-08-10T20:10:53.155-0700    [################........] test.zips    2.1 MB/3.0 MB (68.5%)
2016-08-10T20:10:56.150-0700    [########################] test.zips    3.0 MB/3.0 MB (100.0%)
2016-08-10T20:10:57.819-0700    [########################] test.zips    3.0 MB/3.0 MB (100.0%)
2016-08-10T20:10:57.821-0700    imported 29353 documents
```

mongoimport avec CSV

Exemple de fichier CSV de fichier de test stocké à l'emplacement c: \ Users \ yc03ak1 \ Desktop \ testing.csv

```
_id    city    loc    pop    state
1    A    [10.0, 20.0]    2222    PQE
2    B    [10.1, 20.1]    22122    RW
3    C    [10.2, 20.0]    255222    RWE
4    D    [10.3, 20.3]    226622    SFDS
5    E    [10.4, 20.0]    222122    FDS
```

pour importer cet ensemble de données dans la base de données nommée "test" et collection nommée "sample"

```
C:\Users\yc03ak1>mongoimport --db test --collection "sample" --drop --type csv --headerline --
host "localhost:47019"  --file "c:\Users\yc03ak1\Desktop\testing.csv"
```

- --headerline: utilisez la première ligne du fichier csv comme champs du document json

sortie:

```
2016-08-10T20:25:48.572-0700    connected to: localhost:47019
2016-08-10T20:25:48.576-0700    dropping: test.sample
2016-08-10T20:25:49.109-0700    imported 5 documents
```

OU

```
C:\Users\yc03ak1>mongoimport --db test --collection "sample" --drop --type csv --fields
_id,city,loc,pop,state --host "localhost:47019"  --file "c:\Users\yc03ak1\Desktop\testing.csv"
```

- --fields: liste de champs séparés par des virgules à importer dans le document json. Sortie:

```
2016-08-10T20:26:48.978-0700    connected to: localhost:47019
2016-08-10T20:26:48.982-0700    dropping: test.sample
2016-08-10T20:26:49.611-0700    imported 6 documents
```

Lire Sauvegarde et restauration de données en ligne:
https://riptutorial.com/fr/mongodb/topic/6290/sauvegarde-et-restauration-de-donnees

Chapitre 26: Sauvegarde et restauration de données

Examples

Mump de base de l'instance mongod locale par défaut

```
mongodump --db mydb --gzip --out "mydb.dump.$(date +%F_%R)"
```

Cette commande videra l'archive bson gzippée de votre base de données mongod locale 'mydb' dans le répertoire 'mydb.dump. {Timestamp}'

Mongorestore de base du dong mongod local par défaut

```
mongorestore --db mydb mydb.dump.2016-08-27_12:44/mydb --drop --gzip
```

Cette commande supprime d'abord votre base de données 'mydb' actuelle, puis restaure votre fichier de sauvegarde bson compressé à partir du fichier de vidage d'archive 'mydb mydb.dump.2016-08-27_12: 44 / mydb'.

Lire Sauvegarde et restauration de données en ligne:
https://riptutorial.com/fr/mongodb/topic/6494/sauvegarde-et-restauration-de-donnees

Chapitre 27: Upserts et inserts

Examples

Insérer un document

_id est un nombre hexadécimal de 12 octets qui assure l'unicité de chaque document. Vous pouvez fournir _id lors de l'insertion du document. **Si vous n'avez pas fourni, MongoDB fournira un identifiant unique pour chaque document.** Ces 12 premiers octets de 4 octets pour l'horodatage actuel, les 3 octets suivants pour l'ID de machine, les 2 octets suivants pour l'ID de processus du serveur mongodb et les 3 octets restants sont des valeurs incrémentielles simples.

```
db.mycol.insert({
 _id: ObjectId(7df78ad8902c),
 title: 'MongoDB Overview',
 description: 'MongoDB is no sql database',
 by: 'tutorials point',
 url: 'http://www.tutorialspoint.com',
 tags: ['mongodb', 'database', 'NoSQL'],
 likes: 100
})
```

Ici, *mycol* est un nom de collection, si la collection n'existe pas dans la base de données, alors MongoDB créera cette collection et y insérera le document. Dans le document inséré, si nous ne *spécifions* pas le paramètre *_id* , MongoDB attribue un ObjectId unique à ce document.

Lire Upserts et inserts en ligne: https://riptutorial.com/fr/mongodb/topic/10185/upserts-et-inserts

Crédits

S. No	Chapitres	Contributeurs
1	Démarrer avec MongoDB	Abdul Rehman Sayed, Amir Rahimi Farahani, Ashari, Community, ipip, jengeb, manetsus, Peter Mortensen, Prosen Ghosh, Renukaradhya, Scott Weldon, Sean Reilly, Simulant, titogeo, WAF
2	2dsphere Index	gypsyCoder
3	Agrégation	grape, HoefMeistert, Lakmal Vithanage, LoicM, manetsus, RaR, steveinatorx, titogeo
4	Agrégation MongoDB	Andrii Abramov, Avindu Hewa, Erdenezul, saurav
5	Collections	Prosen Ghosh
6	Configuration	Matt Clark
7	Gestion de MongoDB	Ravi Chandra
8	Index	Adam Comerford, Batsu, Constantin Guay, jerry, Juan Carlos Farah, manetsus, Nic Cottrell, RaR, Rick, Sarthak Adhikari, titogeo, Tomás Cañibano
9	Mécanismes d'authentification dans MongoDB	Luzan Baral, Niroshan Ranapathi
10	Mettre à jour les opérateurs	yellowB
11	Mise à niveau de la version MongoDB	Antti_M
12	Modèle d'autorisation MongoDB	Niroshan Ranapathi
13	Mongo comme jeu de répliques	user641887
14	Mongo comme Shards	Selva Kumar

15	MongoDB - Configurer un ReplicaSet pour prendre en charge TLS / SSL	bappr
16	Moteurs de stockage enfichables	Constantin Guay, Jorge Aranda, tim, Zanon
17	Obtenir des informations sur la base de données	fracz
18	Opération CRUD	fracz, Himavanth, Ishan Soni, Jain, jerry, JohnnyHK, Kelum Senanayake, KrisVos130, Lakmal Vithanage, Marco, Mayank Pandeyz, Prosen Ghosh, Renukaradhya, Rick, Rotem, Shrabanee, sstyvane, Thomas Bormans, Tomás Cañibano
19	Opérations en vrac	chridam
20	Pilote Java	dev ツ, Emil Burzo
21	Pilote Python	Derlin, sergiuz
22	Querying for Data (Démarrage)	Avindu Hewa, oggo, Prosen Ghosh, SommerEngineering
23	Réplication	ADIMO
24	Sauvegarde et restauration de données	user641887
25	Upserts et inserts	Kuhan