GEEKFLARE

48 Commandes et requêtes MongoDB à connaître en tant que développeur et DBA

By Asad Ali
sur Juillet 4, 2020

> Cet article décrit les requêtes et commandes fréquemment utilisées de MongoDB utilisé par les développeurs et les administrateurs de bases de données dans leur vie quotidienne de développement et d'exploitation.

Quick Intro

Dans cette nouvelle ère de technologie intelligente, les données sont générées en grand volume et chaque élément de données est tout aussi important pour les industries en croissance. Les utilisateurs génèrent des données structurées, semi-structurées et non structurées en quantité illimitée. Les données structurées comprennent le stockage des données dans des tableaux et des lignes, tandis que les données non structurées se composent d'images, de vidéos et de clips vocaux. En raison du volume croissant de données structurées et non structurées, la nécessité de Base de données NoSQL vient dans l'image.

Il offre aux développeurs la facilité de concevoir un schéma et une plate-forme évolutive aux administrateurs de base de données, il fournit une plate-forme sécurisée et rapide.

What is MongoDB?

MongoDB est une base de données NoSQL orientée document, multiplateforme et open source utilisée pour stocker des données semi-structurées écrites en C ++. Au

utilisées.

Commençons.

Basic Commands

1. Vérification de la version

La commande principale consiste à vérifier la version installée du serveur MongoDB et Mongo Shell. Exécutez cette commande sur le terminal à l'invite Linux ou CMD sous Windows.

```
mongod --version
```

```
C:\Windows\System32>mongod --version
db version v4.2.7
git version: 51d9fe12b5d19720e72dcd7db0f2f17dd9a19212
allocator: tcmalloc
modules: none
build environment:
    distmod: 2012plus
    distarch: x86_64
    target_arch: x86_64
```

```
C:\Windows\System32>mongo --version
MongoDB shell version v4.2.7
git version: 51d9fe12b5d19720e72dcd7db0f2f17dd9a19212
allocator: tcmalloc
modules: none
build environment:
    distmod: 2012plus
    distarch: x86_64
    target_arch: x86_64
```

2. Liste des commandes MongoDB

Cette commande aidera les utilisateurs à découvrir toutes les commandes qui peuvent être utilisées dans MongoDB. Exécutez la commande sur Mongo Shell.

```
help()
```

```
mongo> db.help()
DB methods:
        db.adminCommand(nameOrDocument) - switches to 'admin' db, and runs co
        db.aggregate([pipeline], {options}) - performs a collectionless aggre
        db.auth(username, password)
        db.cloneDatabase(fromhost) - will only function with MongoDB 4.0 and
        db.commandHelp(name) returns the help for the command
```

```
db.createView(name, viewOn, [{$operator: {...}}, ...], {viewOptions})
db.currentOp() displays currently executing operations in the db
db.dropDatabase(writeConcern)
db.dropUser(username)
db.eval() - deprecated
db.fsyncLock() flush data to disk and lock server for backups
db.fsyncUnlock() unlocks server following a db.fsyncLock()
db.getCollection(cname) same as db['cname'] or db.cname
db.getCollectionInfos([filter]) - returns a list that contains the na
db.getCollectionNames()
db.getLastError() - just returns the err msg string
db.getLastErrorObj() - return full status object
db.getLogComponents()
db.getMongo() get the server connection object
db.getMongo().setSlaveOk() allow queries on a replication slave serve
db.getName()
db.getProfilingLevel() - deprecated
db.getProfilingStatus() - returns if profiling is on and slow thresho
db.getReplicationInfo()
db.getSiblingDB(name) get the db at the same server as this one
db.getWriteConcern() - returns the write concern used for any operati
db.hostInfo() get details about the server's host
db.isMaster() check replica primary status
db.killOp(opid) kills the current operation in the db
db.listCommands() lists all the db commands
db.loadServerScripts() loads all the scripts in db.system.js
```

```
db.printShardingStatus()
db.printSlaveReplicationInfo()
db.resetError()
db.runCommand(cmdObj) run a database command.  if cmdObj is a string,
db.serverStatus()
db.setLogLevel(level,<component>)
db.setProfilingLevel(level,slowms) 0=off 1=slow 2=all
db.setVerboseShell(flag) display extra information in shell output
db.setWriteConcern(<write concern doc>) - sets the write concern for
db.shutdownServer()
db.stats()
db.unsetWriteConcern(<write concern doc>) - unsets the write concern
db.version() current version of the server
db.watch() - opens a change stream cursor for a database to report on
```

3. Statistiques DB

La commande ci-dessous donnera des détails sur les bases de données ainsi que plusieurs collections et paramètres associés de cette base de données.

```
db.stats()
```

```
> db.stats()
{
        "db" : "test",
```

```
        "avgObjSize" : 0,
        "dataSize" : 0,
        "storageSize" : 0,
        "numExtents" : 0,
        "indexes" : 0,
        "indexSize" : 0,
        "scaleFactor" : 1,
        "fileSize" : 0,
        "fsUsedSize" : 0,
        "fsTotalSize" : 0,
        "ok" : 1
}
```

4. Créer un nouveau DB ou basculer vers un DB existant

Cette simple commande permet de créer une nouvelle base de données si elle n'existe pas ou de basculer vers la base de données existante. Dans MongoDB, "test" est la base de données par défaut et les utilisateurs utilisent "**tester**"DB une fois que Mongo Shell est connecté.

```
use DB_Name
```

```
mongos> use geekFlareDB
switched to db geekFlareDB
```

```
show dbs
```

```
mongo> show dbs
admin         0.000GB
config        0.002GB
geekFlareDB   0.000GB
test          0.000GB
```

6. Vérifiez la base de données actuellement utilisée

Exécutez la commande ci-dessous sur Mongo Shell pour voir la base de données actuellement utilisée.

```
db
```

```
> db
GeekFlare
```

7. Supprimer la base de données

La commande donnée aide l'utilisateur à supprimer la base de données requise. Exécutez la commande sur le client MongoDB. Veuillez vous assurer de sélectionner

```
db.dropDatabase()
```

Commençons par lister toutes les bases de données, basculons vers l'une d'entre elles, puis déposons-la

```
> show dbs
admin     0.000GB
config    0.001GB
local     0.000GB
test      0.000GB
training  0.000GB
>
> use training
switched to db training
>
> db.dropDatabase()
{ "dropped" : "training", "ok" : 1 }
```

8. Créer une collection

Les collections sont similaires aux tables du SGBDR.

La commande Créer une collection se compose de deux paramètres. La collection se compose de zéro ou plusieurs documents. Par conséquent, pour créer une

- Créer une collection simple.

Syntaxe: `db.createCollection(Name,Options)`

Exemple:

```
> use geekFlare
switched to db geekFlare
>
> db.createCollection("geekFlareCollection")
{ "ok" : 1 }
>
> show collections
geekFlareCollection
```

- Créer une collection limitée

Dans ce cas, limitez la taille et le nombre de documents à insérer dans la collection. La collection plafonnée a une propriété pour supprimer les documents les plus anciens pour faire de la place pour les nouveaux documents.

Syntaxe:

```
 db.createCollection(Name,{capped : true, size : sizeLimit , max :
documentLimit })
```

```
> db.createCollection("Login",{capped:true,max:1,size:200})
{ "ok" : 1 }
>
> db.Login.insertMany([{"id":1,status:"Active"},{"id":2,status:"Hold"},{"id":
{
        "acknowledged" : true,
        "insertedIds" : [
                ObjectId("5edc5f4f411247725e75e52e"),
                ObjectId("5edc5f4f411247725e75e52f"),
                ObjectId("5edc5f4f411247725e75e530")
        ]
}
>
> db.Login.find()
{ "_id" : ObjectId("5edc5f4f411247725e75e530"), "id" : 3, "status" : "Pending
```

9. Collecte des gouttes

La commande Drop Collection est similaire à DDL dans le SGBDR. Il acquiert des verrous sur la collection requise jusqu'à l'exécution de la commande. Drop collection supprime la collection de la base de données avec tous les index associés à cette collection. Pour supprimer la collection, la méthode drop () est requise.

Syntaxe: `collectionName.drop()`

Exemple:

```
> use geekFlare
switched to db geekFlare
>
> show collections
geekFlareCollection
>
> db.geekFlareCollection.drop()
true
>
> db.geekFlareCollection.drop()
false
```

CRUD Operations related

10. Insérer le document dans la collection

Dans MongoDB, le document est similaire à un tuple dans le SGBDR.

Pour créer un document, le `insert()` méthode est utilisée. La méthode insert () crée un ou plusieurs documents dans la collection existante. Il crée également une collection si elle n'est pas présente dans DB. Dans MongoDB, le document est sans

- **Insérer un seul enregistrement**

Pour insérer un enregistrement `insert()` or `insertOne()` méthode peut être utilisée.

Syntaxe: `collectionName.insertOne({document})`

Exemple:

```
> db.geekFlareCollection.insertOne( {
 code: "P123", Qty: 200, status: "Active"
});
{
        "acknowledged" : true,
        "insertedId" : ObjectId("5ed309725429283aee2e134d")
}
```

- **Insertion de plusieurs enregistrements**

Pour insérer de nombreux enregistrements, une liste d'enregistrements sera transmise à `insert()` or `insertMany()` méthode.

Syntaxe:

```
 collectionName.insertMany([{document1},{document2},{ document3}….{
documentn}])
```

```
... { code: "P1", Qty: 100, status: "Active"},
... { code: "P2", Qty: 200, status: "Active"},
... { code: "P3", Qty: 0, status: "Dective"}
... ]);
{
        "acknowledged" : true,
        "insertedIds" : [
                ObjectId("5edf7b4e18b2c26b9dfe8cac"),
                ObjectId("5edf7b4e18b2c26b9dfe8cad"),
                ObjectId("5edf7b4e18b2c26b9dfe8cae")
        ]
}
> db.geekFlareCollection.find()
{ "_id" : ObjectId("5edf546fdfa12b33b7cb75b8"), "product" : "bottles", "Qty"
{ "_id" : ObjectId("5edf546fdfa12b33b7cb75b9"), "product" : "bread", "Qty" :
{ "_id" : ObjectId("5edf546fdfa12b33b7cb75ba"), "product" : "yogurt", "Qty" :
{ "_id" : ObjectId("5edf7b4e18b2c26b9dfe8cac"), "code" : "P1", "Qty" : 100, "
{ "_id" : ObjectId("5edf7b4e18b2c26b9dfe8cad"), "code" : "P2", "Qty" : 200, "
{ "_id" : ObjectId("5edf7b4e18b2c26b9dfe8cae"), "code" : "P3", "Qty" : 0, "st
>
```

- **Insérer un enregistrement en masse**

Un grand nombre de documents peuvent également être insérés de manière ordonnée et non ordonnée en exécutant `initializeOrderedBulkOp()` et

```
var bulk = db.collectionName.initializeUnorderedBulkOp();

bulk.insert({document1} );

bulk.insert({document2} );

bulk.insert({documentn} );

bulk.execute();
```

Mise en situation :

```
> var bulk = db.geekFlareCollection.initializeUnorderedBulkOp();
> bulk.insert({ code: "P1", Qty: 100, status: "Active"});
> bulk.insert({ code: "P2", Qty: 200, status: "Active"});
> bulk.insert({ code: "P3", Qty: 0, status: "Dective"});
> bulk.execute();
BulkWriteResult({
        "writeErrors" : [ ],
        "writeConcernErrors" : [ ],
        "nInserted" : 3,
        "nUpserted" : 0,
        "nMatched" : 0,
        "nModified" : 0,
```

```
> db.geekFlareCollection.find()
{ "_id" : ObjectId("5edf7be318b2c26b9dfe8caf"), "code" : "P1", "Qty" : 100, "
{ "_id" : ObjectId("5edf7be318b2c26b9dfe8cb0"), "code" : "P2", "Qty" : 200, "
{ "_id" : ObjectId("5edf7be318b2c26b9dfe8cb1"), "code" : "P3", "Qty" : 0, "st
>
```

11. Récupérer le document d'une collection

Pour rechercher le document stocké dans une collection, la méthode find () peut être utilisée. La commande ci-dessous sera utilisée pour récupérer tous les documents de la collection.

- **`find()`** peut être utilisée pour récupérer tous les documents stockés dans une collection.

Syntaxe: `collectionName.find()`

Exemple:

```
> db.geekFlareCollection.find()
{ "_id" : ObjectId("5ed31186b6f2c2bb1edb86ce"), "code" : "P1", "Qty" : 200, "
{ "_id" : ObjectId("5ed31186b6f2c2bb1edb86cf"), "code" : "P2", "Qty" : 200, "
{ "_id" : ObjectId("5ed31186b6f2c2bb1edb86d0"), "code" : "P3", "Qty" : 200, "
{ "_id" : ObjectId("5ed3159eb6f2c2bb1edb86d1"), "code" : "P4", "Qty" : 100, "
```

récupérer la valeur de type BSON.

Syntaxe: `collectionName.find({ condition })`

Exemple:

```
> db.geekFlareCollection.find({ Qty: { $eq: 100 }});
{ "_id" : ObjectId("5ed3159eb6f2c2bb1edb86d1"), "code" : "P4", "Qty" : 100, "
```

- Pour récupérer un seul document, MongoDB fournit le `findOne` () méthode. Il donne une sortie formatée.

Syntaxe: `collectionName.findOne()`

Exemple:

```
> db.geekFlareCollection.findOne();
{
    "_id" : ObjectId("5ed31186b6f2c2bb1edb86ce"),
        "code" : "P1",
    "Qty" : 200,
    "status" : "Inactive"
}
```

commandes pour obtenir la sortie formatée.

Syntaxe: `collectionName.find().pretty()`

Exemple:

```
> db.geekFlareCollection.find({ Qty: { $eq: 100 }}).pretty();
{
        "_id" : ObjectId("5ed3159eb6f2c2bb1edb86d1"),
        "code" : "P4",
        "Qty" : 100,
        "status" : "Inactive"
}
```

13. Mettre à jour le document dans une collection

MongoDB fournit `update()` pour définir de nouvelles valeurs pour les clés existantes dans les documents. La commande de mise à jour donne des détails sur les documents modifiés et mis en correspondance. La syntaxe de la commande de mise à jour est:

Syntaxe: `collectionName.update({KeyToUpdate},{Set Command})`

Exemple:

GEEKFLARE

```
{ "_id" : ObjectId("5edf3f67d6bfbd8125f58cfd"), "product" : "yogurt", "Qty" :
>
> db.geekFlareCollection.update({"product" : "bottles"},{$set : {"Qty": 10}}
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
> db.geekFlareCollection.find()
{ "_id" : ObjectId("5edf3f67d6bfbd8125f58cfb"), "product" : "bottles", "Qty"
{ "_id" : ObjectId("5edf3f67d6bfbd8125f58cfc"), "product" : "bread", "Qty" :
{ "_id" : ObjectId("5edf3f67d6bfbd8125f58cfd"), "product" : "yogurt", "Qty" :
>
>  db.geekFlareCollection.find()
{ "_id" : ObjectId("5edf3f67d6bfbd8125f58cfb"), "product" : "bottles", "Qty"
{ "_id" : ObjectId("5edf3f67d6bfbd8125f58cfc"), "product" : "bread", "Qty" :
{ "_id" : ObjectId("5edf3f67d6bfbd8125f58cfd"), "product" : "yogurt", "Qty" :
```

- **`updateOne()`** : Pour mettre à jour un seul document, il y a `updateOne()` méthode. `updateOne()` donnez le nombre de documents appariés et modifiés.

Syntaxe: `collectionName.updateOne({SingleKeyToUpdate},{Set Command})`

Exemple:

```
> db.geekFlareCollection.updateOne({"product" : "bottles"},{$set : {"Qty": 40
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
```

certaines conditions, MongoDB a `updateMany()` méthode.

Syntaxe: `collectionName.updateMany({filter},{Set Command})`

Exemple:

```
> db.geekFlareCollection.updateMany( { "Qty" : { $lt: 30 } },{ $set: { "Qty":
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
```

14. Supprimer le document d'une collection

Pour supprimer le document, MongoDB se compose de `deleteOne()` et `deleteMany()` méthodes. La syntaxe des méthodes de suppression est:

- `<strong>deleteOne({condition}) </strong>` supprime le document unique répondant aux critères de suppression.

Syntaxe: `collectionName.deleteOne({DeletionCondition})`

Exemple:

```
> db.geekFlareCollection.deleteOne({"product" : "bread"})
{ "acknowledged" : true, "deletedCount" : 1 }
```

syntaxe: `collectionName.deleteMany({DeletionCondition})`

Exemple:

```
> db.geekFlareCollection.deleteMany({"product" : "bottles"})
{ "acknowledged" : true, "deletedCount" : 2 }
```

- `<strong>remove() </strong>` Il existe une autre méthode pour supprimer tous les documents correspondant aux critères de suppression. remove () prend deux arguments, l'un est la condition de suppression et l'autre n'est qu'un indicateur.

  Notes: La méthode Remove est obsolète dans les versions à venir.

Syntaxe: `collectionName.remove({DeletionCondition},1)`

Exemple:

```
> db.geekFlareCollection.remove({"product" : "bottles"})
WriteResult({ "nRemoved" : 1 })
```

15. Récupérer Distinct

Le `distinct()` méthode est utilisée pour obtenir des enregistrements uniques.

- Pour obtenir des enregistrements distincts à partir d'un champ.

```
> db.geekFlareCollection.distinct("product")
[ "Cheese", "Snacks2", "Snacks3", "bread", "ketchup" ]
```

- Pour obtenir des enregistrements distincts à partir d'un champ tout en spécifiant la requête.

Syntaxe: `collectionName.distinct(field,query)`

Exemple:

```
> db.geekFlareCollection.distinct('product',{"Qty":20})
[ "Snacks3", "bread" ]
```

16. Renommer la collection

MongoDB fournit `renameCollection ()` méthode pour renommer la collection.

Syntaxe: `collectionName.renameCollection(newCollectionName)`

Exemple:

```
>db.geekFlareCollection.renameCollection('geekFlareCol')
{ "ok" : 1 }
```

Indexing

17. Créer un index sur le document

Les index sont une structure de données spéciale qui stocke une petite partie de l'ensemble de données de la collection sous une forme facile à parcourir. Les index prennent en charge l'ordre croissant et décroissant des valeurs des champs et facilitent ainsi de meilleures performances lors de la récupération.

MongoDB fournit le `default_id` indice. En outre, MongoDB prend en charge la création d'index définis par l'utilisateur. Les index MongoDB sont définis au niveau des collections et il fournit des supports au niveau du champ ou du sous-champ d'un document. La syntaxe de create the index est:

- Créez un index sur un seul champ.

Syntaxe: `collectionName.createIndex({Key:1})`

En cela, la clé indique le champ sur lequel l'index est créé et 1 signifie l'ordre croissant. Pour créer un index dans l'ordre décroissant, -1 peut être utilisé.

Exemple:

```
> db.geekFlareCollection.createIndex({"product" : 1})
{
```

```
        "ok" : 1
}
```

- Créez un index sur plusieurs champs.

Syntaxe: `collectionName.createIndex({Key1:1,key2:1…keyn:1})`

Exemple:

```
> db.geekFlareCollection.createIndex({"product" : 1,"Qty":-1})
{
        "createdCollectionAutomatically" : false,
        "numIndexesBefore" : 1,
        "numIndexesAfter" : 2,
        "ok" : 1
}
```

18. Afficher l'index sur le document

MongoDB fournit `getIndexes()` pour lister tous les index créés sur un document.

Syntaxe: `collectionName.getIndexes()`

Exemple:

```
        {
                "v" : 2,
                "key" : {
                        "_id" : 1
                },
                "name" : "_id_",
                "ns" : "geekFlareCollection.geekFlareCollection"
        }
]
```

19. Supprimer l'index du document

`dropIndex()` est utilisée pour supprimer l'index unique et la méthode dropIndexes () est utilisée pour supprimer plusieurs index.

- Supprimer un index unique

Syntaxe: `collectionName.dropIndex({key})`

Exemple:

```
> db.geekFlareCollection.dropIndex({"product" : 1})
{ "nIndexesWas" : 3, "ok" : 1 }
```

- Supprimer plusieurs index.

```
> db.geekFlareCollection.dropIndexes({"product" : 1,"Qty":-1})
{ "nIndexesWas" : 3, "ok" : 1 }
```

Retrieval related

20. Limiter la récupération des documents

`limit()` permet de limiter le nombre de documents renvoyés. La méthode limit () accepte les arguments numériques.

Syntaxe: `collectionName.find().limit(number)`

Exemple:

```
> db.geekFlareCollection.find().limit(2)
{ "_id" : ObjectId("5ed3c9e7b6f2c2bb1edb8702"), "product" : "bottles", "Qty"
{ "_id" : ObjectId("5ed3c9e7b6f2c2bb1edb8703"), "product" : "bread", "Qty" :
```

21. Ignorer la récupération des documents

MongoDB prend en charge `skip()` méthode. Cette méthode ignore le nombre requis de documents. Il accepte un argument numérique.

```
> db.geekFlareCollection.find().skip(2)
{ "_id" : 3, "product" : "yogurt", "Qty" : 30 }
> db.geekFlareCollection.find().skip(3)
```

22. Trier la récupération des documents

MongoDB `sort()` méthode trie les documents de sortie par ordre croissant ou décroissant. Cette méthode accepte le nom des clés avec le nombre pour spécifier l'ordre de tri 1 est utilisé pour l'ordre croissant tandis que -1 est utilisé pour spécifier l'ordre décroissant.

Syntaxe: `collectionName.find().sort({key:1})`

Exemple:

```
> db.geekFlareCollection.find().sort({"Qty":1})
{ "_id" : 2, "product" : "bread", "Qty" : 20 }
{ "_id" : 3, "product" : "yogurt", "Qty" : 30 }
{ "_id" : 1, "product" : "bottles", "Qty" : 100 }
```

Validation related

documents. Les validateurs sont définis lors de la collecte. La création du validateur est requise pour utiliser le mot-clé **validateur** et facultatif **niveau de validation** et **action de validation** pour spécifier le mode de validation. La validation du document ne limite pas l'insertion du nouveau champ dans le document.

Syntaxe: `createCollection("collectionName",{validator:{ fields condition }})`

Mise en situation :

```
> db.createCollection( "Login",
...  { validator: { $and:
...       [
...          { phone: { $type: "string" } },
...          { email: { $regex: /@flares\.com$/ } },
...          { status: { $in: [ "Registered", "Unknown" ] } }
...       ]
...    }
... } )
{ "ok" : 1 }
>
> db.Login.insert({phone:1234})
WriteResult({
        "nInserted" : 0,
        "writeError" : {
                "code" : 121,
                "errmsg" : "Document failed validation"
```

```
> db.Login.insert({phone:"1234",email:"abc@flares.com",status:"Unknown",mode:
WriteResult({ "nInserted" : 1 })
```

24. Validateurs de schéma sur une nouvelle collection

Mot clé supplémentaire **$ jsonSchema** avec **propriétés supplémentaires** valeur comme **Faux** est nécessaire pour mettre une restriction au niveau du schéma. Cela empêche l'ajout de nouveaux champs dans le document.

Syntaxe: `createCollection("collectionName",{validator: { $jsonSchema { schema condition } }})`

Exemple:

```
> db.createCollection( "Login", {
...    validator: { $jsonSchema: {
...        bsonType: "object",
...        "additionalProperties": false,
...       required: [ "email" ],
...       properties: {
...          email: {
...             bsonType : "string",
...             pattern : "@flares\.com$",
...             description: "string meets the given expression"
...          },
```

```
...           }
...        }
...     } },
... } )
{ "ok" : 1 }
>
> db.Login.insert({email:"abc@flares.com"})
WriteResult({
        "nInserted" : 0,
        "writeError" : {
                "code" : 121,
                "errmsg" : "Document failed validation"
        }
})
```

25. Mettre à jour ou créer des validateurs de schéma sur une collection existante

Un validateur peut être créé sur une collection existante en utilisant **`collMod`**

Syntaxe: `runCommand({collMod:"collectionName",validator:{schema condition}})`

Mise en situation :

```
> db.runCommand(  {
collMod: "Login",
```

```
      required: [ "email","status" ],
      properties: {
         email: {
            bsonType : "string",
            pattern : "@flares\.com$",
            description: "string meets the given expression"
         },
         status: {
            enum: [ "registered", "Invalid" ],
            description: "status must be within enum values"
         }
      }
   } },
      validationAction: "error",
         validationLevel: "strict"
} )
{ "ok" : 1 }
```

26. Supprimer les validateurs de schéma sur une collection existante

Pour supprimer les validateurs de schéma, il faut définir `<strong>validationLevel</strong>` comme off.

Exemple:

```
> db.runCommand({
   collMod:"Login",
   validator:{},
   validationLevel:"off"
})
{ "ok" : 1 }
>
> db.Login.insert({"email":"abc"})
WriteResult({ "nInserted" : 1 })
```

27. Rechercher des validateurs sur une collection existante

Pour vérifier si la collection existante a des validateurs de schéma exécutés ci-dessous la commande. Sans spécifier le nom de la collection `db.getCollectionInfos()` La méthode donne des détails sur les validateurs sur toutes les collections résidant dans une base de données.

Syntaxe: `getCollectionInfos({name : “collectionName”})`

Exemple:

```
> db.getCollectionInfos({name: "Login"})
[
```

```
                "options" : {
                        "validator" : {
                                "email" : {
                                        "$regex" : /@flares\.com$/
                                }
                        }
                },
                "info" : {
                        "readOnly" : false,
                        "uuid" : UUID("646674f6-4b06-466d-93b0-393b5f5cb4ff")
                },
                "idIndex" : {
                        "v" : 2,
                        "key" : {
                                "_id" : 1
                        },
                        "name" : "_id_",
                        "ns" : "geekFlareDB.Login"
                }
        }
]
```

Cursors related

MongoDB hasNext() et forEach() méthode d'itération. Une liste de méthodes de curseur a été fourni.

Exemples:

```
> var newCursor=db.geekFlareCollection.find()
> newCursor.forEach(printjson)
{ "_id" : 1, "product" : "bottles", "Qty" : 100 }
{ "_id" : 2, "product" : "bread", "Qty" : 20 }
{ "_id" : 3, "product" : "yogurt", "Qty" : 30 }
>
> var newCursor1=db.geekFlareCollection.find()
> while(newCursor1.hasNext()){ printjson(newCursor1.next())}
{ "_id" : 1, "product" : "bottles", "Qty" : 100 }
{ "_id" : 2, "product" : "bread", "Qty" : 20 }
{ "_id" : 3, "product" : "yogurt", "Qty" : 30
```

Utility

29. Sauvegarde de la base de données

`mongodump` L'utilitaire est utilisé pour exporter le contenu de la base de données MongoDB en tant que sauvegarde. Cette commande s'exécute depuis la console

Syntaxe:

```
 mongodump --db dbName --out outFile --host "IP:PORT"  --username <user>  --
password <pass>
```

Mise en situation :

```
C:\mongodb\dump>mongodump  --db geekFlareDB --out "C:\mongodb\dump" --host "1
2020-06-02T12:26:34.428+0530    writing geekFlareDB.myTable to
2020-06-02T12:26:34.430+0530    writing geekFlareDB.geekFlareCollection to
2020-06-02T12:26:34.430+0530    writing geekFlareDB.mCollection to
2020-06-02T12:26:34.433+0530    writing geekFlareDB.users to
2020-06-02T12:26:34.434+0530    done dumping geekFlareDB.myTable (2 documents
2020-06-02T12:26:34.434+0530    done dumping geekFlareDB.geekFlareCollection
2020-06-02T12:26:34.435+0530    writing geekFlareDB.contacts2 to
2020-06-02T12:26:34.435+0530    writing geekFlareDB.Login to
2020-06-02T12:26:34.436+0530    done dumping geekFlareDB.mCollection (2 docum
2020-06-02T12:26:34.437+0530    done dumping geekFlareDB.users (1 document)
2020-06-02T12:26:34.437+0530    done dumping geekFlareDB.Login (0 documents)
2020-06-02T12:26:34.438+0530    done dumping geekFlareDB.contacts2 (0 documen
```

30. Restauration de la base de données à partir de la sauvegarde

Syntaxe: `mongorestore --db newDB "pathOfOldBackup"`

Exemple:

```
C:\Users\asad.ali>mongorestore  --db geekFlareNew  "C:\mongodb\dump\geekFlare
2020-06-09T15:49:35.147+0530    the --db and --collection args should only be
2020-06-09T15:49:35.148+0530    building a list of collections to restore fro
2020-06-09T15:49:35.152+0530    reading metadata for geekFlareNew.geekFlareCo
2020-06-09T15:49:35.321+0530    restoring geekFlareNew.geekFlareCollection fr
2020-06-09T15:49:35.461+0530    no indexes to restore
2020-06-09T15:49:35.462+0530    finished restoring geekFlareNew.geekFlareColl
2020-06-09T15:49:35.467+0530    3 document(s) restored successfully. 0 docume
```

31. Exportation de collections

Pour exporter le contenu de la collection vers un fichier (JSON ou CSV) `mongoexport` l'utilité a été fournie. Pour exécuter cette commande, utilisez le terminal système.

- Exportez une seule collection vers un fichier.

Syntaxe: `mongoexport --db dbName --collection collectionName --out outputFile`

Exemple:

```
                                                    exported 6 records
```

- Exportez un champ spécifique de la collection vers un fichier.

Syntaxe: `mongoexport --db dbName --collection collectionName --out outputFile --fields fieldname`

Exemple:

```
C:\mongodb\New folder>mongoexport --db geekFlareDB --collection geekFlareCol
2020-06-06T19:05:22.994+0530    connected to: mongodb://localhost/
2020-06-06T19:05:23.004+0530    exported 6 records
```

32. Importation de collections

Pour importer des données depuis un fichier (CSV ou JSON) `mongoimport` l'outil de ligne de commande peut être utilisé.

Syntaxe: `mongoimport --db dbName --collection collectionName --file inputFile`

```
C:\Users\asad.ali>mongoimport --db geekFlareDB --collection geekFlareNew --fi
2020-06-09T14:52:53.655+0530    connected to: mongodb://localhost/
2020-06-09T14:52:53.924+0530    6 document(s) imported successfully. 0 docume
```

> partage.

33. Réplication MongoDB

La réplication est le processus de synchronisation des données sur plusieurs serveurs. Il évite la perte de données due à un dysfonctionnement matériel ou logiciel. MongoDB réalise la réplication à l'aide d'ensembles de répliques. L'ensemble de réplicas se compose d'ensembles de données Mongo principaux et secondaires dans le cluster.

L'ensemble de données principal accepte toutes les opérations d'écriture et les lectures de l'ensemble de données secondaire à partir du primaire. Au moins 3 ensembles de données sont requis dans l'ensemble de répliques Mongo. Le processus ci-dessous est requis pour configurer un jeu de réplicas:

- Démarrer `mongod` serveur avec `replset` option sur un minimum de 3 nœuds.

```
mongod --port 27017 --dbpath C:\data\data1 --replSet rs0 --oplogSize 128

mongod --port 27018 --dbpath C:\data\data1 --replSet rs0 --oplogSize 128

mongod --port 27019 --dbpath C:\data\data1 --replSet rs0 --oplogSize 128
```

- Initialisez le jeu de réplicas.

```
> rs.initiate( {
...    _id : "rs0",
...    members: [
...   { _id: 0, host: "localhost:27017" },
...   { _id: 1, host: "localhost:27018" },
...   { _id: 2, host: "localhost:27019" }
...    ]
... })
{
        "ok" : 1,
        "$clusterTime" : {
                "clusterTime" : Timestamp(1591089166, 1),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        },
        "operationTime" : Timestamp(1591089166, 1)
}
```

34. Vérifier l'état de la réplication

Exécutez la commande ci-dessous à partir du nœud de réplica principal pour obtenir des informations complètes sur l'ensemble de répliques.

```
rs0:PRIMARY> rs.conf()
{
        "_id" : "rs0",
        "version" : 2,
        "protocolVersion" : NumberLong(1),
        "writeConcernMajorityJournalDefault" : true,
        "members" : [
                {
                        "_id" : 0,
                        "host" : "localhost:27017",
                        "arbiterOnly" : false,
                        "buildIndexes" : true,
                        "hidden" : false,
                        "priority" : 1,
                        "tags" : {

                        },
                        "slaveDelay" : NumberLong(0),
                        "votes" : 1
                },
                {
                        "_id" : 1,
                        "host" : "localhost:27018",
                        "arbiterOnly" : false,
                        "buildIndexes" : true,
```

```
        },
        "slaveDelay" : NumberLong(0),
        "votes" : 1
},
{
        "_id" : 2,
        "host" : "localhost:27019",
        "arbiterOnly" : false,
        "buildIndexes" : true,
        "hidden" : false,
        "priority" : 1,
        "tags" : {

        },
        "slaveDelay" : NumberLong(0),
        "votes" : 1
},
{
        "_id" : 3,
        "host" : "localhost:27016",
        "arbiterOnly" : false,
        "buildIndexes" : true,
        "hidden" : false,
        "priority" : 1,
        "tags" : {
```

```
                        "votes" : 1
                }
        ],
        "settings" : {
                "chainingAllowed" : true,
                "heartbeatIntervalMillis" : 2000,
                "heartbeatTimeoutSecs" : 10,
                "electionTimeoutMillis" : 10000,
                "catchUpTimeoutMillis" : -1,
                "catchUpTakeoverDelayMillis" : 30000,
                "getLastErrorModes" : {

                },
                "getLastErrorDefaults" : {
                        "w" : 1,
                        "wtimeout" : 0
                },
                "replicaSetId" : ObjectId("5ed6180d01a39f2162335de5")
        }
}
```

35. Ajouter une nouvelle instance MongoDB à un jeu de réplicas

Démarrez le client MongoDB principal et exécutez la commande ci-dessous

```
rs0:PRIMARY> rs.add("localhost:27016")
{
        "ok" : 1,
        "$clusterTime" : {
                "clusterTime" : Timestamp(1591094195, 1),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        },
        "operationTime" : Timestamp(1591094195, 1)
}
```

36. Supprimer l'instance MongoDB existante du jeu de réplicas

La commande ci-dessous supprimera l'hôte secondaire requis du jeu de réplicas.

Syntaxe: `rs.remove("localhost:27017")`

Exemple:

```
rs0:PRIMARY> rs.remove("localhost:27016")
{
```

```
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        },
        "operationTime" : Timestamp(1591095681, 1)
}
rs0:PRIMARY>
```

37. Faire du primaire comme jeu de réplicas secondaire

MongoDB fournit une commande pour demander au réplica principal de devenir un jeu de réplicas secondaire.

Syntaxe: `rs.stepDown( stepDownSecs , secondaryCatchupSecs )`

Exemple:

```
rs0:PRIMARY> rs.stepDown(12)
{
        "ok" : 1,
        "$clusterTime" : {
                "clusterTime" : Timestamp(1591096055, 1),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
```

```
        "operationTime" : Timestamp(1591096055, 1)
}
rs0:SECONDARY>
```

38. Vérifiez le décalage de réplique entre le primaire et le secondaire

La commande ci-dessous sera utilisée pour vérifier le décalage de réplication entre tous les jeux de réplicas du primaire.

Syntaxe: `rs.printSlaveReplicationInfo()`

Exemple:

```
rs0:PRIMARY> rs.printSlaveReplicationInfo()
source: localhost:27018
        syncedTo: Tue Jun 02 2020 16:14:04 GMT+0530 (India Standard Time)
        0 secs (0 hrs) behind the primary
source: localhost:27019
        syncedTo: Thu Jan 01 1970 05:30:00 GMT+0530 (India Standard Time)
        1591094644 secs (441970.73 hrs) behind the primary
source: localhost:27016
        syncedTo: Tue Jun 02 2020 16:14:04 GMT+0530 (India Standard Time)
        0 secs (0 hrs) behind the primary
rs0:PRIMARY>
```

39. Transactions dans MongoDB

MongoDB prend en charge les propriétés ACID pour les transactions sur les documents.

Pour démarrer une transaction, une session est requise pour démarrer et une validation est requise pour enregistrer les modifications dans la base de données. Les transactions sont prises en charge sur les jeux de répliques ou les mangues. Une fois la session validée, les opérations effectuées à l'intérieur de la session seront visibles à l'extérieur.

- Démarrer la session

Syntaxe: `session =db.getMongo().startSession()`

- Commencer la transaction,

Syntaxe: `session.startTransaction()`

- Valider la transaction

Syntaxe: `session.commitTransaction()`

Exemple:

```
rs0:PRIMARY> session = db.getMongo().startSession()
session { "id" : UUID("f255a40d-81bd-49e7-b96c-9f1083cb4a29") }
rs0:PRIMARY> session.startTransaction()
rs0:PRIMARY> session.getDatabase("geekFlareDB").geekFlareCollection.insert([
... {_id: 4 , product: "ketchup"},
... {_id: 5, product: "Cheese"}
... ]);
BulkWriteResult({
        "writeErrors" : [ ],
        "writeConcernErrors" : [ ],
        "nInserted" : 2,
        "nUpserted" : 0,
        "nMatched" : 0,
        "nModified" : 0,
        "nRemoved" : 0,
        "upserted" : [ ]
})
rs0:PRIMARY> session.getDatabase("geekFlareDB").geekFlareCollection.find()
{ "_id" : 1, "product" : "bread", "Qty" : 20 }
{ "_id" : 2, "product" : "bottles", "Qty" : 100 }
{ "_id" : 3, "product" : "bread", "Qty" : 20 }
{ "_id" : 4, "product" : "ketchup" }
{ "_id" : 5, "product" : "Cheese" }
rs0:PRIMARY> db.geekFlareCollection.find()
{ "_id" : 1, "product" : "bread", "Qty" : 20 }
{ "_id" : 2, "product" : "bottles", "Qty" : 100 }
```

```
{ "_id" : 1, "product" : "bread", "Qty" : 20 }
{ "_id" : 2, "product" : "bottles", "Qty" : 100 }
{ "_id" : 3, "product" : "bread", "Qty" : 20 }
{ "_id" : 4, "product" : "ketchup" }
{ "_id" : 5, "product" : "Cheese" }
rs0:PRIMARY> session.getDatabase("geekFlareDB").geekFlareCollection.find()
{ "_id" : 1, "product" : "bread", "Qty" : 20 }
{ "_id" : 2, "product" : "bottles", "Qty" : 100 }
{ "_id" : 3, "product" : "bread", "Qty" : 20 }
{ "_id" : 4, "product" : "ketchup" }
{ "_id" : 5, "product" : "Cheese" }
```

40. Conflit de transactions sur un seul document

Si deux transactions tentent de mettre à jour le même document, MongoDB génère une erreur de conflit d'écriture.

- `session1.startTransaction()`
- `session2.startTransaction()`

Effectuez une insertion / mise à jour sur Session1 puis sur Session2. Observez maintenant l'erreur dans l'exemple ci-dessous

Exemple:

```
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
rs0:PRIMARY> session2.getDatabase("geekFlareDB").geekFlareCollection.update({
WriteCommandError({
        "errorLabels" : [
                "TransientTransactionError"
        ],
        "operationTime" : Timestamp(1591174593, 1),
        "ok" : 0,
        "errmsg" : "WriteConflict",
        "code" : 112,
        "codeName" : "WriteConflict",
        "$clusterTime" : {
                "clusterTime" : Timestamp(1591174593, 1),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        }
})
rs0:PRIMARY>
```

41. Transactions multi-documents

MongoDB prend en charge les transactions multi-documents en une seule session.

- `session.commitTransaction ()`

Exemple:

```
rs0:PRIMARY> var session = db.getMongo().startSession()
rs0:PRIMARY> session.startTransaction()
rs0:PRIMARY> session.getDatabase("geekFlareDB").geekFlareCollection.update({_
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 0 })
rs0:PRIMARY> session.getDatabase("geekFlareDB").geekFlareCollection.update({_
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 0 })
rs0:PRIMARY> session.commitTransaction()
rs0:PRIMARY> db.geekFlareCollection.find()
{ "_id" : 1, "product" : "bread", "Qty" : 20 }
{ "_id" : 2, "product" : "Snacks2", "Qty" : 100 }
{ "_id" : 3, "product" : "Snacks3", "Qty" : 20 }
{ "_id" : 4, "product" : "ketchup" }
{ "_id" : 5, "product" : "Cheese" }
rs0:PRIMARY>
```

42. Profilage dans MongoDB

Le profilage permet de consigner les requêtes lentes dans le `system.profile` collecte. Niveau du profileur et le taux d'échantillonnage définissent le pourcentage de requêtes à connecter `system.profile` collecte.

```
 db.setProfilingLevel(profilingLevel,
{"slowms":time,"sampleRate":LoggingPercentage})
```

```
> db.setProfilingLevel(2,{"slowms":1,"sampleRate":1})
{ "was" : 1, "slowms" : 1, "sampleRate" : 0.5, "ok" : 1 }
>
> db.getProfilingLevel()
2
```

- Obtenir l'état du profilage

Syntaxe: db.getProfilingStatus ()

```
> db.getProfilingStatus()
{ "was" : 2, "slowms" : 1, "sampleRate" : 1 }
```

- Pour activer le profilage au niveau de l'instance MongoDB, démarrez l'instance avec les informations du profileur ou ajoutez les détails du profileur dans le fichier de configuration.

Syntaxe:

```
mongod --profile <Level> --slowms <time> --slowOpSampleRate <%Logging>
```

Exemple:

```
2020-06-09T02:34:41.113-0700 I  CONTROL  [initandlisten] MongoDB starting : p
2020-06-09T02:34:41.114-0700 I  CONTROL  [initandlisten] targetMinOS: Windows
2020-06-09T02:34:41.116-0700 I  CONTROL  [initandlisten] db version v4.2.7
2020-06-09T02:34:41.116-0700 I  CONTROL  [initandlisten] git version: 51d9fe1
```

43. MongoDB Explain ()

MongoDB `explains()` La méthode renvoie des statistiques et fournit des informations pour sélectionner un plan gagnant et l'exécuter jusqu'à son terme. Il renvoie les résultats selon le plan de verbosité.

Syntaxe: `collectionName.explain("verbosityName")`

Pour exécuter la méthode / commande describe (), créons un **verbosité** puis exécutez la méthode explic (), jetez un œil à l'exemple ci-dessous, où ces étapes ont été exécutées.

Exemple:

```
> db.runCommand(
...    {
...       explain: { count: "product", query: { Qty: { $gt: 10 } } },
...       verbosity: "executionStats"
...    }
```

```
                "plannerVersion" : 1,
                "namespace" : "test.product",
                "indexFilterSet" : false,
                "winningPlan" : {
                        "stage" : "COUNT",
                        "inputStage" : {
                                "stage" : "EOF"
                        }
                },
                "rejectedPlans" : [ ]
        },
        "executionStats" : {
                "executionSuccess" : true,
                "nReturned" : 0,
                "executionTimeMillis" : 47,
                "totalKeysExamined" : 0,
                "totalDocsExamined" : 0,
                "executionStages" : {
                        "stage" : "COUNT",
                        "nReturned" : 0,
                        "executionTimeMillisEstimate" : 0,
                        "works" : 1,
                        "advanced" : 0,
                        "needTime" : 0,
                        "needYield" : 0,
                        "saveState" : 0,
```

```
                        "nSkipped" : 0,
                        "inputStage" : {
                                "stage" : "EOF",
                                "nReturned" : 0,
                                "executionTimeMillisEstimate" : 0,
                                "works" : 0,
                                "advanced" : 0,
                                "needTime" : 0,
                                "needYield" : 0,
                                "saveState" : 0,
                                "restoreState" : 0,
                                "isEOF" : 1
                        }
                }
        },
        "serverInfo" : {
                "host" : "MCGL-4499",
                "port" : 27017,
                "version" : "4.2.7",
                "gitVersion" : "51d9fe12b5d19720e72dcd7db0f2f17dd9a19212"
        },
        "ok" : 1
}
>
```

```
"queryPlanner" : {
        "plannerVersion" : 1,
        "namespace" : "geekFlareDB.geekFlareCol",
        "indexFilterSet" : false,
        "parsedQuery" : {
                "product" : {
                        "$eq" : "bread"
                }
        },
        "winningPlan" : {
                "stage" : "COLLSCAN",
                "filter" : {
                        "product" : {
                                "$eq" : "bread"
                        }
                },
                "direction" : "forward"
        },
        "rejectedPlans" : [ ]
},
"executionStats" : {
        "executionSuccess" : true,
        "nReturned" : 2,
        "executionTimeMillis" : 0,
        "totalKeysExamined" : 0,
        "totalDocsExamined" : 6,
```

```
                        "product" : {
                                "$eq" : "bread"
                        }
                },
                "nReturned" : 2,
                "executionTimeMillisEstimate" : 0,
                "works" : 8,
                "advanced" : 2,
                "needTime" : 5,
                "needYield" : 0,
                "saveState" : 0,
                "restoreState" : 0,
                "isEOF" : 1,
                "direction" : "forward",
                "docsExamined" : 6
        }
},
"serverInfo" : {
        "host" : "MCGL-4499",
        "port" : 27017,
        "version" : "4.2.7",
        "gitVersion" : "51d9fe12b5d19720e72dcd7db0f2f17dd9a19212"
},
"ok" : 1
}
>
```

44. Contrôle d'accès dans MongoDB

Les fonctionnalités de contrôle d'accès permettent un accès par authentification aux utilisateurs existants. Pour le contrôle d'accès, la base de données activée permet de garantir la création d'un rôle d'administrateur d'utilisateur dans la base de données d'administration.

- Connectez DB sans authentification.
- Passer à la base de données

```
use admin
```

- Créer un utilisateur comme ci-dessous

```
db.createUser( {user: “UserAdmin”, pwd: “password” ,role: [adminRole])
```

Mise en situation :

```
db.createUser(
...   {
...     user: "AdminUser",
...     pwd: passwordPrompt(),
...     roles: [ { role: "userAdminAnyDatabase", db: "admin" }, "readWriteAny
...   }
... )
Enter password:
Successfully added user: {
```

```
                    "role" : "userAdminAnyDatabase",
                    "db" : "admin"
                },
                "readWriteAnyDatabase"
        ]
}
```

- recommencer `mongod` instance
- Accédez à nouveau avec l'utilisateur et le mot de passe créés.

```
C:\Users\>mongo  --port 27017  --authenticationDatabase "admin" -u "AdminUser
MongoDB shell version v4.2.7
Enter password:
connecting to: mongodb://127.0.0.1:27017/?authSource=admin&compressors=disabl
```

45. Récupérer et supprimer l'utilisateur du contrôle d'accès

La commande ci-dessous peut être utilisée pour vérifier les informations de l'utilisateur et les supprimer.

- `db.getUser("AdminUser")`
- `db.dropUser("AdminUser")`

```
{
        "_id" : "admin.AdminUser",
        "userId" : UUID("78d2d5bb-0464-405e-b27e-643908a411ce"),
        "user" : "AdminUser",
        "db" : "admin",
        "roles" : [
                {
                        "role" : "userAdminAnyDatabase",
                        "db" : "admin"
                },
                {
                        "role" : "readWriteAnyDatabase",
                        "db" : "admin"
                }
        ],
        "mechanisms" : [
                "SCRAM-SHA-1",
                "SCRAM-SHA-256"
        ]
}
> db.dropUser("AdminUser")
true
> db.getUser("AdminUser")
null
>
```

un tableau de rôles hérités.

- Connectez l'instance MongoDB à l'utilisateur admin.
- Exécutez la commande ci-dessous pour générer un nouveau rôle.

Syntaxe:

```
 db.createRole({role:”roleName”,privileges:[{privilegeName}],roles:
[InheritedArray]})
```

Exemple:

```
use admin
> db.createRole(
...    {
...      role: "abc",
...      privileges: [ { resource: { db: "geekFlareDB", collection: "geekFlar
...      roles: []
...    }
... )
{
        "role" : "abc",
        "privileges" : [
                {
                        "resource" : {
                                "db" : "geekFlareDB",
```

```
                                "killop",
                                "inprog"
                        ]
                }
        ],
        "roles" : [ ]
}
```

47. Révocation des rôles définis par l'utilisateur

Pour modifier les rôles existants, utilisez la commande ci-dessous.

Syntaxe: `db.revokeRolesFromUser( userName, [{ "role" : roleName , db:dbName} ] )`

Exemple:

```
> db.getUser("AdminUser")
{
        "_id" : "admin.AdminUser",
        "userId" : UUID("fe716ed1-6771-459e-be13-0df869c91ab3"),
        "user" : "AdminUser",
        "db" : "admin",
        "roles" : [
                {
```

```
                {
                        "role" : "readWriteAnyDatabase",
                        "db" : "admin"
                }
        ],
        "mechanisms" : [
                "SCRAM-SHA-1",
                "SCRAM-SHA-256"
        ]
}
> db.revokeRolesFromUser( "AdminUser", [{ "role" : "userAdminAnyDatabase" , d
> db.getUser("AdminUser")
{
        "_id" : "admin.AdminUser",
        "userId" : UUID("fe716ed1-6771-459e-be13-0df869c91ab3"),
        "user" : "AdminUser",
        "db" : "admin",
        "roles" : [
                {
                        "role" : "readWriteAnyDatabase",
                        "db" : "admin"
                }
        ],
        "mechanisms" : [
                "SCRAM-SHA-1",
                "SCRAM-SHA-256"
```

GEEKFLARE

48. Connectivité MongoDB avec Python

Le package pymongo est requis pour connecter MongoDB à partir de la console python.

```
>>> from pymongo import MongoClient
>>> mClient=MongoClient("mongodb://127.0.0.1:27017/")
>>> mDB=mClient.geekFlareDB
>>> mRecord={"_id":4,"name":"XYZ"}
>>> mDB.geekFlareCollection.insert_one(mRecord)
<pymongo.results.InsertOneResult object at 0x000002CC44256F48>
>>> for i in mDB.geekFlareCollection.find({"_id":4}):
...     print(i)
...
{'_id': 4, 'name': 'XYZ'}
>>>
```

Quelle est la prochaine?

Consultez cette liste de Clients NoSQL pour gérer MongoDB et d'autres bases de données NoSQL. Si votre travail implique de travailler fréquemment sur MongoDB, vous voudrez peut-être en savoir plus Cours Udemy.

Merci à nos sponsors.

More Great Reading on Geekflare

LIRE →

5 solutions backend pour les applications Web et mobiles [Alternatives Firebase]

LIRE →

LIRE →

Meilleures pratiques de conception de bases de données pour les applications hautes performances

LIRE →

Comment installer PostgreSQL sur Ubuntu, CentOS et Windows ?

LIRE →

11 Meilleure base de données en ligne pour votre prochain produit

vulnérabilités identifiées et générer des résultats exploitables en quelques heures seulement.

ESSAYEZ NETSPARKER →

Semrush

Semrush est une solution de marketing numérique tout-en-un avec plus de 50 outils pour développer l'audience et l'activité.

ESSAYEZ SEMRUSH →

+400 ressources en plus

English Français Español Deutsch

expérimenté, qui souhaite maintenir ses connaissances à jour.

Adresse e-mail Joindre!

EXPLORER

Articles techniques

Outils Geekflare

API Geekflare

Compilateur Geekflare

Collections Geekflare

ENTREPRISE

À Propos

La publicité

Embauchons

Conditions

Confidentialité

Flux RSS