EBook Gratis

APRENDIZAJE MongoDB

Free unaffiliated eBook created from
Stack Overflow contributors.

#mongodb

Tabla de contenido

Acerca de

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

Capítulo 1: Empezando con MongoDB

Observaciones

- Los datos en el mundo comenzaron a crecer enormemente después de que las aplicaciones móviles ingresaran al mercado. Esta enorme cantidad de datos se volvió casi imposible de manejar con la base de datos relacional tradicional: SQL. Las bases de datos NoSQL se introducen para manejar aquellos datos en los que se obtiene mucha más flexibilidad, como el número variable de columnas para cada información.
- MongoDB es una de las principales bases de datos NoSQL. Cada colección contiene una serie de documentos JSON. Cualquier modelo de datos que se pueda expresar en un documento JSON se puede almacenar fácilmente en MongoDB.
- MongoDB es una base de datos cliente-servidor. El servidor normalmente se ejecuta con el archivo binario `mongod` y el cliente se ejecuta con `mongo` .
- No hay una operación de unión en MongoDB antes de v.3.2, por varias razones filosóficas y pragmáticas . Pero el shell Mongo admite javascript, por lo que si $ búsqueda no está disponible, uno puede simular operaciones de unión en documentos en javascript antes de insertar.
- Para ejecutar una instancia en un entorno de producción, se recomienda encarecidamente seguir la Lista de verificación de operaciones .

Versiones

Versión	Fecha de lanzamiento
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

Instalación

Para instalar MongoDB, siga los pasos a continuación:

- **Para Mac OS:**
  - Hay dos opciones para Mac OS: instalación manual o homebrew .
  - **Instalación con homebrew :**
    - Escriba el siguiente comando en el terminal:

```
$ brew install mongodb
```

  - **Instalación manual:**
    - Descarga la última versión aquí . Asegúrese de que está descargando el archivo apropiado, compruebe especialmente si su tipo de sistema operativo es de 32 bits o de 64 bits. El archivo descargado está en formato `tgz` .
    - Ir al directorio donde se descarga este archivo. Luego escribe el siguiente comando:

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

      En lugar de `xyz` , habría alguna información sobre la versión y el tipo de sistema. La carpeta extraída tendría el mismo nombre que el archivo `tgz` . Dentro de la carpeta, habría una subcarpeta llamada `bin` que contendría varios archivos binarios junto con `mongod` y `mongo` .
    - Por defecto el servidor mantiene los datos en la carpeta `/data/db` . Entonces, tenemos que crear ese directorio y luego ejecutar el servidor con los siguientes comandos:

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

    - Para iniciar el servidor, se debe dar el siguiente comando desde la ubicación actual:

```
$ ./mongod
```

      Se iniciaría el servidor en el puerto 27017 por defecto.
    - Para iniciar el cliente, se debe abrir una nueva terminal con el mismo directorio

que antes. Luego, el siguiente comando iniciaría el cliente y se conectaría al servidor.

```
$ ./mongo
```

Por defecto se conecta a la base de datos de `test` . Si ves la línea como `connecting to: test` . Entonces usted ha instalado con éxito MongoDB. Felicidades Ahora, puedes probar Hello World para tener más confianza.

- **Para ventanas:**
  - Descarga la última versión aquí . Asegúrese de que está descargando el archivo apropiado, compruebe especialmente si su tipo de sistema operativo es de 32 bits o de 64 bits.
  - El archivo binario descargado tiene extensión `exe` . Ejecutarlo. Se le pedirá un asistente de instalación.
  - Haga clic en **siguiente** .
  - **Acepte** el acuerdo de licencia y haga clic en **Siguiente** .
  - Seleccione Instalación **completa** .
  - Haga clic en **Instalar** . Puede que aparezca una ventana para pedir permiso al administrador. Haga clic en **Sí** .
  - Después de la instalación, haga clic en **Finalizar** .
  - Ahora, el mongodb está instalado en la ruta `C:/Program Files/MongoDB/Server/3.2/bin` . En lugar de la versión 3.2, podría haber alguna otra versión para su caso. El nombre de la ruta se cambiaría en consecuencia.
  - `bin` directorio `bin` contiene varios archivos binarios junto con `mongod` y `mongo` . Para ejecutarlo desde otra carpeta, puede agregar la ruta en la ruta del sistema. Para hacerlo:
    - Haga clic derecho en **Mi PC** y seleccione **Propiedades** .
    - Haga clic en **Configuración avanzada del sistema** en el panel izquierdo.
    - Haga clic en **Variables de entorno ...** en la pestaña Opciones **avanzadas** .
    - Seleccione **Ruta** en la sección de **variables** del **sistema** y haga clic en **Editar ....**
    - Antes de Windows 10, agregue un punto y coma y pegue la ruta indicada anteriormente. Desde Windows 10, hay un botón **Nuevo** para agregar una nueva ruta.
    - Haga **clic en** s para guardar los cambios.
  - Ahora, cree una carpeta llamada `data` con una subcarpeta llamada `db` en la que desea ejecutar el servidor.

- Iniciar el símbolo del sistema desde su. Cambiando la ruta en cmd o haciendo clic en **Abrir ventana de comando aquí,** que sería visible después de hacer clic derecho en el espacio vacío de la carpeta GUI presionando las teclas Mayús y Ctrl juntas.
- Escriba el comando para iniciar el servidor:

```
> mongod
```

  Se iniciaría el servidor en el puerto 27017 por defecto.

- Abra otro símbolo del sistema y escriba lo siguiente para iniciar el cliente:

```
> mongo
```

- Por defecto se conecta a la base de datos de `test` . Si ves la línea como `connecting to: test` . Entonces usted ha instalado con éxito MongoDB. Felicidades Ahora, puedes probar Hello World para tener más confianza.

- **Para Linux:** casi lo mismo que Mac OS, excepto que se necesita un comando equivalente.
  - Para las distribuciones basadas en Debian (usando `apt-get`):
    - Importar clave de repositorio MongoDB.

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

    - Agregue repositorio a la lista de paquetes en **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
```

    - en **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
```

    - Actualizar la lista de paquetes.

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

    - Instalar MongoDB.

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

  - Para distribuciones basadas en Red Hat (usando `yum`):
    - Usa un editor de texto que prefieras.

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

- Pega el siguiente texto.

```
[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
```

- Actualizar la lista de paquetes.

```
$ sudo yum update
```

- Instalar MongoDB

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

Hola Mundo

Después del proceso de instalación, las siguientes líneas deben ingresarse en mongo shell (terminal de cliente).

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

Hola Mundo!

Explicación:

- En la primera línea, hemos insertado un documento emparejado `{ key : value }` en la `test` base de datos predeterminada y en la colección denominada `world` .
- En la segunda línea recuperamos los datos que acabamos de insertar. Los datos recuperados se mantienen en una variable de javascript llamada `cur` . Luego, mediante la función `next()` , recuperamos el primer y único documento y lo guardamos en otra variable js llamada `x` . Luego imprimió el valor del documento proporcionando la clave.

Terminos Complementarios

Términos SQL	Términos de MongoDB
Base de datos	Base de datos
Mesa	Colección
Entidad / Fila	Documento

Términos SQL	Términos de MongoDB
Columna	Clave / Campo
Unirse a la mesa	Documentos incrustados
Clave primaria	Clave principal (Clave predeterminada `_id` proporcionada por mongodb en sí)

Ejecución de un archivo JavaScript en MongoDB

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

Explicación: esta operación ejecuta el script `myjsfile.js` en un shell `mongo` que se conecta a la base de datos `mydb` en la instancia `mongod` accesible a través de la interfaz `localhost` en el puerto `27017` . `localhost:27017` no es obligatorio ya que este es el puerto predeterminado que usa `mongodb` .

Además, puede ejecutar un archivo `.js` desde la consola de `mongo` .

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

Haciendo la salida de encontrar legible en shell

Agregamos tres registros a nuestra prueba de colección como:

```
> 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 los vemos a través de encontrar, se verán muy feos.

```
> 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" }
```

Para solucionar esto y hacer que sean legibles, use la función **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"
}
>
```

Comandos básicos en mongo shell

Mostrar todas las bases de datos disponibles:

```
show dbs;
```

Seleccione una base de datos particular para acceder, por ejemplo, `mydb` . Esto creará `mydb` si aún no existe:

```
use mydb;
```

Muestre todas las colecciones en la base de datos (asegúrese de seleccionar una, vea arriba):

```
show collections;
```

Muestra todas las funciones que se pueden utilizar con la base de datos:

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

Para verificar su base de datos seleccionada actualmente, use el comando `db`

```
> db
mydb
```

`db.dropDatabase()` comando `db.dropDatabase()` se usa para eliminar una base de datos existente.

```
db.dropDatabase()
```

Lea Empezando con MongoDB en línea: https://riptutorial.com/es/mongodb/topic/691/empezando-con-mongodb

Capítulo 2: Actualizando la versión de MongoDB

Introducción

Cómo actualizar la versión de MongoDB en su máquina en diferentes plataformas y versiones.

Observaciones

Si tiene una versión anterior de MongoDB, debe actualizar la ruta completa a la versión más reciente. Por ejemplo, si está ejecutando la versión 3.0 y desea obtener la versión 3.4, debe actualizar 3.0-> 3.2-> 3.4.

Examples

Actualización a 3.4 en Ubuntu 16.04 usando apt

Debes tener 3.2 para poder actualizar a 3.4. Este ejemplo asume que estás usando `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`

Asegúrate de que la nueva versión se esté ejecutando con `mongo` . El shell imprimirá la versión del servidor MongoDB que debería ser 3.4 ahora.

Lea Actualizando la versión de MongoDB en línea:
https://riptutorial.com/es/mongodb/topic/9851/actualizando-la-version-de-mongodb

Capítulo 3: Agregación

Introducción

`Aggregations` operaciones de `Aggregations` procesan registros de datos y devuelven resultados calculados. Las operaciones de agregación agrupan los valores de varios documentos y pueden realizar una variedad de operaciones en los datos agrupados para devolver un solo resultado. MongoDB proporciona tres formas de realizar la agregación: el canal de agregación, la función de reducción de mapas y los métodos de agregación de propósito único.

Del manual de Mongo https://docs.mongodb.com/manual/aggregation/

Sintaxis

- **db.collection.aggregate (canalización, opciones)**

Parámetros

Parámetro	Detalles
tubería	matriz (una secuencia de operaciones o etapas de agregación de datos)
opciones	documento (opcional, disponible solo si la canalización está presente como una matriz)

Observaciones

El marco de agregación en MongoDB se utiliza para lograr la funcionalidad común de `GROUP BY` de SQL.

Considere las siguientes inserciones en la colección con nombre de `transactions` para cada ejemplo.

```
> 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

Contar

¿Cómo se obtiene el número de transacciones de débito y crédito? Una forma de hacerlo es utilizando la función `count()` como se muestra a continuación.

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

o

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

Pero qué `cr_dr` si no conoce los posibles valores de `cr_dr` upfront. Aquí el marco de agregación viene a jugar. Consulte la siguiente consulta de agregados.

```
> 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}
              }
          }
      ]
  );
```

Y el resultado es

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

Suma

¿Cómo obtener la suma de `amount` ? Vea la siguiente consulta agregada.

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

Y el resultado es

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

Otra versión que suma `amount` y `fee` .

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

Y el resultado es

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

Promedio

¿Cómo obtener el monto promedio de las transacciones de débito y crédito?

```
> 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
            }
        }
    ]
)
```

El resultado es

```
{
    "_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
}
```

Operaciones con matrices.

Cuando desee trabajar con las entradas de datos en matrices, primero debe desenrollar la matriz. La operación de desenrollado crea un documento para cada entrada en la matriz. Cuando tenga muchos documentos con arreglos grandes verá una explosión en el número de documentos.

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

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

Un aviso importante es que cuando un documento no contiene la matriz, se perderá. Desde Mongo 3.2 y hasta hay una opción de desenrollar "preserveNullAndEmptyArrays" agregada. Esta opción garantiza que el documento se conserva cuando falta la matriz.

```
{ "_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" } }] )
```

Partido

¿Cómo escribir una consulta para obtener todos los departamentos donde la edad promedio de los empleados que ganan menos de $ 70000 es mayor que o igual a 35?

Para eso necesitamos escribir una consulta para igualar a los empleados que tienen un salario menor o igual a $ 70000. Luego agregue la etapa agregada para agrupar a los empleados por departamento. Luego, agregue un acumulador con un campo denominado, por ejemplo, average_age para encontrar la edad promedio por departamento usando el acumulador $ avg y debajo de $ match existente y los agregados de $ group agregan otro agregado de $ match para que solo podamos recuperar los resultados con un valor de average_age. mayor que o igual a 35.

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

```
  {"$match": {"average_age": {"$gte": 35}}}
])
```

El resultado es:

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

Eliminar documentos que tienen un campo duplicado en una colección (dedupe)

Tenga en cuenta que la opción allowDiskUse: true es opcional pero ayudará a mitigar los problemas de falta de memoria, ya que esta agregación puede ser una operación que requiere mucha memoria si el tamaño de su colección es grande, por lo que recomiendo usarla siempre.

```
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}})
```

Lea Agregación en línea: https://riptutorial.com/es/mongodb/topic/3852/agregacion

Capítulo 4: Agregación de MongoDB

Examples

Agregue ejemplos de consultas útiles para trabajar y aprender.

La agregación se utiliza para realizar operaciones complejas de búsqueda de datos en la consulta de mongo que no se pueden realizar en la consulta normal de "búsqueda".

Crear algunos datos ficticios:

```
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});
```

Ejemplos por tema:

1. Coincidencia: se utiliza para emparejar documentos (como SQL donde la cláusula)

```
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. Proyecto: se utiliza para poblar valores de campos específicos

la etapa del proyecto incluirá el campo _id automáticamente a menos que especifique deshabilitar.

```
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. Grupo: $ grupo se usa para agrupar documentos por campo específico, aquí los documentos se agrupan por el valor del campo "dept". Otra característica útil es que puede agrupar por nulo, ya que todos los documentos se agregarán en uno.

```
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. Suma: $ suma se usa para contar o sumar los valores dentro de un grupo.

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

5. Promedio: calcula el promedio del valor del campo específico por grupo.

```
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. Mínimo: encuentra el valor mínimo de un campo en cada grupo.

```
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. Máximo: encuentra el valor máximo de un campo en cada grupo.

```
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. Obtención del valor del campo específico del primer y último documento de cada grupo: Funciona bien cuando se ordena el resultado del documento.

```
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. Mínimo con máximo:

```
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 y addToSet: Push agrega un formulario de valor de campo de cada documento en grupo a una matriz utilizada para proyectar datos en formato de matriz, addToSet es similar a push pero omite valores duplicados.

```
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. Desenrollar: se utiliza para crear varios documentos en memoria para cada valor en el campo de tipo de matriz especificado, luego podemos hacer una agregación adicional basada en esos valores.

```
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. Clasificación:

```
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. Omitir:

```
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. Límite:

```
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. Operador de comparación en proyección:

```
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. Operador de comparación en el partido:

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

Lista de operadores de comparación: $ cmp, $ eq, $ gt, $ gte, $ lt, $ lte y $ ne

17. Opertor de agregación booleana en proyección:

```
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. Opertor de agregación booleana en el partido:

```
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 }
```

Lista de operadores de agregación booleana: $ y, $ o, y $ no.

Referencia completa: https://docs.mongodb.com/v3.2/reference/operator/aggregation/

Ejemplo de Java y Spring

Este es un código de ejemplo para crear y ejecutar la consulta agregada en MongoDB usando Spring Data.

```
    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();
    }
```

Consulte el valor "resultSet" en formato JSON para comprender el formato de salida:

```
[{
    "_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"
    }]
}]
```

El "resultSet" contiene una entrada para cada grupo, "ageSet" contiene la lista de la edad de cada empleado de ese grupo, "_id" contiene el valor del campo que se está utilizando para agrupar y "docs" contiene datos de cada empleado de ese grupo que se puede utilizar en nuestro propio código y UI.

Obtener datos de muestra

Para obtener datos aleatorios de cierta recopilación, consulte `$sample` agregación de `$sample` .

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

donde el `size` representa el número de elementos a seleccionar.

Unión externa izquierda con agregación ($ Búsqueda)

```
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);
```

```
});
```

Esta característica se lanzó recientemente en la **versión 3.2 de** mongodb, que le da al usuario una etapa para unirse a una colección con los atributos coincidentes de otra colección.

Mongodb $ LookUp documentación

Lea Agregación de MongoDB en línea: https://riptutorial.com/es/mongodb/topic/7417/agregacion-de-mongodb

Capítulo 5: Colecciones

Observaciones

Crear base de datos

Examples

Crear una colección

Primero seleccione o cree una base de datos.

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

Usando el `db.createCollection("yourCollectionName")` puede crear una Colección explícitamente.

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

Usando el comando `show collections` vea todas las colecciones en la base de datos.

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

El método `db.createCollection()` tiene los siguientes parámetros:

Parámetro	Tipo	Descripción
nombre	cuerda	El nombre de la colección a crear.
opciones	documento	*Opcional.* Opciones de configuración para crear una colección con límite o para preasignar espacio en una nueva colección.

El ejemplo flotante muestra la sintaxis del método `createCollection()` con algunas opciones importantes

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

Tanto las `db.collection.insert()` como `db.collection.createIndex()` crean su colección respectiva si aún no existen.

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

Ahora, Mostrar todas las colecciones usando el comando `show collections`

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

Si desea ver el documento insertado, use el comando `find()` .

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

Coleccion Drop

El `db.collection.drop()` MongoDB se usa para eliminar una colección de la base de datos.

Primero, verifique las colecciones disponibles en su base de datos `mydb` .

```
> use mydb
switched to db mydb

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

Ahora suelte la colección con el nombre `newCollection1` .

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

Nota: Si la colección se eliminó correctamente, el método devolverá `true` contrario, devolverá `false` .

Verifique nuevamente la lista de colecciones en la base de datos.

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

Referencia: MongoDB drop () Método.

Lea Colecciones en línea: https://riptutorial.com/es/mongodb/topic/9732/colecciones

Capítulo 6: Conductor de pitón

Sintaxis

- mongodb: // [username: password @] host1 [: port1] [, host2 [: port2], ... [, hostN [: portN]]] [/ [database] [? options]]

Parámetros

Parámetro	Detalle
hostX	Opcional. Puede especificar tantos hosts como sea necesario. Especificaría varios hosts, por ejemplo, para conexiones a conjuntos de réplicas.
: portX	Opcional. El valor predeterminado es: 27017 si no se especifica.
base de datos	Opcional. El nombre de la base de datos para autenticarse si la cadena de conexión incluye credenciales de autenticación. Si no se especifica la base de datos / y la cadena de conexión incluye credenciales, el controlador se autenticará en la base de datos de administración.
opciones	Opciones específicas de conexión

Examples

Conéctate a MongoDB usando 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()
```

Consultas de PyMongo

Una vez que tienes un objeto de `collection` , las consultas usan la misma sintaxis que en el shell mongo. Algunas diferencias leves son:

- Cada llave debe estar entre corchetes. Por ejemplo:

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

  se convierte en `pymongo` (note el `True` en mayúsculas):

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

- los objetos como los identificadores de objeto o `ISODate` se manipulan utilizando clases de python. PyMongo usa su propia clase `ObjectId` para tratar con identificadores de objetos, mientras que las fechas usan el paquete estándar de `datetime` y `datetime` . Por ejemplo, si desea consultar todos los eventos entre 2010 y 2011, puede hacer:

  ```
  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 } }):
  ```

Actualizar todos los documentos en una colección usando PyMongo

Digamos que necesita agregar un campo a cada documento en una colección.

```
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)
```

El método de `find` devuelve un `Cursor` , en el que puede iterar fácilmente usando la sintaxis `for in` . Luego, llamamos al método de `update` , especificando el `_id` y que agregamos un campo (`$set`). Los parámetros `upsert` y `multi` vienen de mongodb (ver aquí para más información).

Lea Conductor de pitón en línea: https://riptutorial.com/es/mongodb/topic/7843/conductor-de-piton

Capítulo 7: Configuración

Parámetros

Parámetro	Defecto
systemLog.verbosity	0
systemLog.quiet	falso
systemLog.traceAllExceptions	falso
systemLog.syslogFacility	usuario
systemLog.path	-
systemLog.logAppend	falso
systemLog.logRotate	rebautizar
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

Parámetro	Defecto
processManagement.fork	falso
processManagement.pidFilePath	ninguna
net.port	27017
net.bindIp	0.0.0.0
net.maxIncomingConnections	65536
net.wireObjectCheck	cierto
net.ipv6	falso
net.unixDomainSocket.enabled	cierto
net.unixDomainSocket.pathPrefix	/ tmp
net.unixDomainSocket.filePermissions	0700
net.http.enabled	falso
net.http.JSONPEnabled	falso
net.http.RESTInterfaceEnabled	falso
net.ssl.sslOnNormalPorts	falso
net.ssl.mode	discapacitado
net.ssl.PEMKeyFile	ninguna
net.ssl.PEMKeyPassword	ninguna
net.ssl.clusterFile	ninguna
net.ssl.clusterPassword	ninguna
net.ssl.CAFile	ninguna
net.ssl.CRLFile	ninguna
net.ssl.allowConnectionsWithoutCertificates	falso
net.ssl.allowInvalidCertificates	falso
net.ssl.allowInvalidHostnames	falso
net.ssl.disabledProtocols	ninguna

Parámetro	Defecto
net.ssl.FIPSMode	falso

Examples

Iniciar mongo con un archivo de configuración específico

Usando la bandera `--config` .

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

Tenga en cuenta que `-f` es el sinónimo más corto de `--config` .

Lea Configuración en línea: https://riptutorial.com/es/mongodb/topic/5985/configuracion

Capítulo 8: Consulta de datos (Primeros pasos)

Introducción

Ejemplos básicos de consulta

Examples

Encontrar()

recuperar todos los documentos en una colección

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

recuperar documentos en una colección usando una condición (similar a DONDE en MYSQL)

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

recuperar documentos en una colección usando condiciones booleanas (operadores de consultas)

```
//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 } } )
```

Más operaciones y ejemplos booleanos se pueden encontrar aquí

NOTA: *find ()* continuará buscando en la colección incluso si se ha encontrado una coincidencia de documento, por lo tanto, es ineficiente cuando se usa en una colección grande, sin embargo, al modelar cuidadosamente sus datos y / o usar índices puede aumentar la eficiencia de *búsqueda ( )*

Encuentra uno()

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

la funcionalidad de consulta es similar a la de find () pero esto terminará la ejecución en el momento en que encuentre que un documento coincide con su condición, si se usa con un objeto vacío, buscará el primer documento y lo devolverá. findOne () mongodb api documentación

Documento de consulta: uso de las condiciones AND, OR e IN

Todos los documentos de la colección de los `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
}
```

Consulta `mySql` similar del comando anterior.

```
SELECT * FROM students;
```

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

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

Consulta `mySql` similar del comando anterior.

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

Y consultas

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

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

Consulta `mySql` similar del comando anterior.

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

O consultas

```
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 }
```

Consulta `mySql` similar del comando anterior.

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

Y OR consultas

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

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

Consulta mySql similar del comando anterior.

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

Consultas IN Estas consultas pueden mejorar el uso múltiple de consultas 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 }
```

Consulta mySql similar al comando anterior

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

método find () con proyección

La sintaxis básica del método `find()` con proyección es la siguiente

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

Si desea mostrar todos los documentos sin el campo de antigüedad, el comando es el siguiente

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

Si desea mostrar a todos los documentos el campo de antigüedad, el comando es el siguiente

Método encontrar () con proyección

En MongoDB, proyección significa seleccionar solo los datos necesarios en lugar de seleccionar la totalidad de los datos de un documento.

La sintaxis básica del método `find()` con proyección es la siguiente

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

Si desea mostrar todo el documento sin el campo de antigüedad, el comando es el siguiente

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

Si desea mostrar solo el campo de edad, el comando es el siguiente

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

Nota: el campo `_id` siempre se muestra mientras se ejecuta el método `find()` , si no desea este campo, entonces debe configurarlo como `0` .

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

Nota: `1` se usa para mostrar el campo, mientras que `0` se usa para ocultar los campos.

limitar, omitir, ordenar y contar los resultados del método find ()

Similar a los métodos de agregación también por el método find () tiene la posibilidad de limitar, omitir, ordenar y contar los resultados. Digamos que tenemos la siguiente colección:

```
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"}
])
```

Para listar la colección:

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

Volverá

```
{ "_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" }
```

Para saltar los primeros 3 documentos:

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

Volverá

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

Para ordenar descendente por el nombre del campo:

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

Volverá

```
{ "_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 quieres ordenar ascendente simplemente reemplaza -1 por 1

Para contar los resultados:

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

Volverá

```
5
```

También se permiten combinaciones de estos métodos. Por ejemplo, obtenga 2 documentos de la colección ordenada descendente saltándose el primer 1:

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

Volverá

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

Lea Consulta de datos (Primeros pasos) en línea:
https://riptutorial.com/es/mongodb/topic/9271/consulta-de-datos--primeros-pasos-

Capítulo 9: Controlador de Java

Examples

Crear un cursor tailable.

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

Ese código se aplica a la clase `MongoCollection` .

CursorType es una enumeración y tiene los siguientes valores:

```
Tailable
TailableAwait
```

Correspondientes a los tipos de bytes anteriores a la adición de DBCursor (<3.0):

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

Crear un usuario de base de datos

Para crear un usuario **dev** con contraseña **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);
```

Fetch Datos de la colección con la condición

Para obtener datos de `testcollection` recogida en `testdb` base de datos donde `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();
}
```

Lea Controlador de Java en línea: https://riptutorial.com/es/mongodb/topic/6286/controlador-de-java

Capítulo 10: Copia de seguridad y restauración de datos

Examples

Mongoimport con JSON

Ejemplo de conjunto de datos zipcode en zipcodes.json almacenado en 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" }
```

para importar este conjunto de datos a la base de datos llamada "prueba" y la colección llamada "cremalleras"

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

- --db: nombre de la base de datos donde se importarán los datos
- --collection: nombre de la colección en la base de datos en la que se deben anotar los datos
- --drop: deja caer la colección antes de importar
- --type: tipo de documento que necesita ser importado. JSON predeterminado
- --host: host mongodb y puerto en el que se importarán los datos.
- --archivo: ruta donde está el archivo json

salida:

```
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 con CSV

Archivo CSV de conjunto de datos de prueba de muestra almacenado en la ubicación 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
```

para importar este conjunto de datos a la base de datos llamada "prueba" y la colección llamada "muestra"

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

- --headerline: usa la primera línea del archivo csv como campos para el documento json

salida:

```
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
```

O

```
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: lista separada por comas de los campos que deben importarse en el documento json. Salida:

```
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
```

Lea Copia de seguridad y restauración de datos en línea:
https://riptutorial.com/es/mongodb/topic/6290/copia-de-seguridad-y-restauracion-de-datos

Capítulo 11: Copia de seguridad y restauración de datos

Examples

Mongodump básico de instancia mongod local predeterminada

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

Este comando volcará un archivo bson gzipped de su base de datos mongod 'mydb' local en el directorio 'mydb.dump. {Timestamp}'

Almacén mongor básico del volcado mongod predeterminado local

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

Este comando primero eliminará su base de datos 'mydb' actual y luego restaurará su volcado bson gzipped desde el archivo de volcado 'mydb mydb.dump.2016-08-27_12: 44 / mydb'.

Lea Copia de seguridad y restauración de datos en línea:
https://riptutorial.com/es/mongodb/topic/6494/copia-de-seguridad-y-restauracion-de-datos

Capítulo 12: Gestionando MongoDB

Examples

Listado de consultas actualmente en ejecución

El siguiente comando enumera las consultas que se están ejecutando actualmente en el servidor

```
db.currentOp()
```

La salida se ve algo similar a esto.

```
{
    "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"
            ...
        }
    ]
}
```

El atributo `inprog` indica que las consultas están actualmente en curso. El `opid` es Id de la consulta u operación. `secs_running` indica el tiempo durante el cual se ha estado ejecutando. Esto a veces es útil para identificar consultas de larga ejecución.

Lea Gestionando MongoDB en línea: https://riptutorial.com/es/mongodb/topic/7553/gestionando-mongodb

Capítulo 13: Índice 2dsphere

Examples

Crear un índice de 2dsphere

`db.collection.createIndex()` método `db.collection.createIndex()` se utiliza para crear un índice `2dsphere` . El plano de un índice de `2dsphere` :

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

Aquí, el `location field` es la clave y `2dsphere` es el tipo del índice. En el siguiente ejemplo, vamos a crear un índice `2dsphre` en la colección de `places` .

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

La siguiente operación creará el índice `2dsphere` en el campo `loc` de la colección de `places` .

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

Lea Índice 2dsphere en línea: https://riptutorial.com/es/mongodb/topic/6632/indice-2dsphere

Capítulo 14: Índices

Sintaxis

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

Observaciones

Impacto en el rendimiento : tenga en cuenta que los índices mejoran el rendimiento de lectura, pero pueden tener un impacto negativo en el rendimiento de escritura, ya que la inserción de un documento requiere la actualización de todos los índices.

Examples

Campo único

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

Esto crea un índice de campo único ascendente en el *nombre del* campo.

En este tipo de índices, el orden de clasificación es irrelevante, porque mongo puede atravesar el índice en ambas direcciones.

Compuesto

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

Esto crea un índice en varios campos, en este caso en `age` campos de `name` y `age` . Será ascendente en `name` y descendente en `age` .

En este tipo de índice, el orden de clasificación es relevante, ya que determinará si el índice puede admitir una operación de clasificación o no. La clasificación inversa se admite en cualquier prefijo de un índice compuesto, siempre que la clasificación esté en la dirección de clasificación inversa para **todas** las claves de la clasificación. De lo contrario, la clasificación de los índices compuestos debe coincidir con el orden del índice.

El orden de los campos también es importante, en este caso el índice se ordenará primero por `name` y, dentro de cada valor de nombre, se ordenará por los valores del campo de `age` . Esto permite que el índice sea utilizado por consultas en el campo de `name` , o en `name` y `age` , pero no solo en la `age` .

Borrar

Para descartar un índice puedes usar el nombre del índice.

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

O el documento de especificación del índice

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

Lista

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

Esto devolverá una serie de documentos, cada uno describiendo un índice en la colección de *personas*

Conceptos básicos de creación de índices

Vea la colección de transacciones a continuación.

```
> 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()` funciones `getIndexes()` mostrarán todos los índices disponibles para una colección.

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

Vamos a ver la salida de la declaración anterior.

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

Ya hay un índice para la colección de transacciones. Esto se debe a que MongoDB crea un *índice único* en el campo `_id` durante la creación de una colección. El índice `_id` evita que los clientes inserten dos documentos con el mismo valor para el campo `_id` . No puede colocar este índice en el campo `_id` .

Ahora vamos a agregar un índice para el campo cr_dr;

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

El resultado de la ejecución del índice es el siguiente.

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

> El createdCollectionAutomatically indica si la operación creó una colección. Si no existe una colección, MongoDB crea la colección como parte de la operación de indexación.

Deje ejecutar `db.transactions.getIndexes();` otra vez.

```
[
    {
        "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"
    }
]
```

Ahora ves que las transacciones de cobro tienen dos índices. Índice `_id` predeterminado y `cr_dr_1` que creamos. El nombre es asignado por MongoDB. Puede establecer su propio nombre como abajo.

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

Ahora `db.transactions.getIndexes();` Te daremos tres índices.

```
[
    {
        "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"
    }
]
```

Al crear el índice `{ cr_dr : -1 }` 1 significa que el índice estará en orden `ascending` y -1 para el orden `descending` .

2.4

Índices hash

Los índices se pueden definir también como *hash* . Esto es más eficaz en *consultas de igualdad* , pero no es eficiente para *consultas de rango* ; sin embargo, puede definir los índices con hash y ascendente / descendente en el mismo campo.

```
> 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"
    }
]
```

Eliminando / Eliminando un Índice

Si el nombre del índice es conocido,

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

Si no se conoce el nombre del índice,

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

Obtener índices de una colección

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

Salida

```
[
    {
        "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"
    }
]
```

Índice único

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

imponer la unicidad en el índice definido (ya sea individual o compuesto). La creación del índice fallará si la colección ya contiene valores duplicados; la indexación fallará también con varias entradas que faltan en el campo (ya que todas se indexarán con el valor `null`) a menos que se especifique `sparse: true` .

Índices dispersos e índices parciales

Índices dispersos:

Estos pueden ser particularmente útiles para los campos que son opcionales pero que también deben ser únicos.

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

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

Dado que dos entradas no tienen un "apodo" especificado y la indexación tratará los campos no especificados como nulos, la creación del índice fallará con 2 documentos que tengan 'nulo', por lo que:

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

te dejará tener apodos 'nulos'.

Los índices dispersos son más compactos, ya que omiten / ignoran los documentos que no especifican ese campo. Entonces, si tiene una colección donde solo menos del 10% de los documentos especifican este campo, puede crear índices mucho más pequeños, haciendo un mejor uso de la memoria limitada si desea hacer consultas como:

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

Índices parciales:

> Los índices parciales representan un superconjunto de la funcionalidad ofrecida por los índices dispersos y deberían preferirse a los índices dispersos. (*Nuevo en la versión 3.2*)

Los índices parciales determinan las entradas de índice según el filtro especificado.

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

Si la `rating` es mayor a 5, entonces la `cuisine` será indexada. Sí, podemos especificar una propiedad para ser indexada en función del valor de otras propiedades también.

Diferencia entre índices dispersos y parciales:

Los índices dispersos seleccionan documentos para indexar únicamente en función de la existencia del campo indexado, o para los índices compuestos, la existencia de los campos indexados.

Los índices parciales determinan las entradas de índice según el filtro especificado. El filtro puede incluir campos que no sean las claves de índice y puede especificar condiciones distintas a solo una verificación de existencia.

Aún así, un índice parcial puede implementar el mismo comportamiento que un índice disperso

P.ej:

```
db.contacts.createIndex(
   { name: 1 },
   { partialFilterExpression: { name: { $exists: true } } }
)
```

> **Nota:** *La* opción *partialFilterExpression* y la opción *sparse* no se pueden especificar al mismo tiempo.

Lea Índices en línea: https://riptutorial.com/es/mongodb/topic/3934/indices

Capítulo 15: Inserciones e inserciones

Examples

Inserte un documento

_id es un número hexadecimal de 12 bytes que asegura la unicidad de cada documento. Puede proporcionar _id mientras inserta el documento. **Si no proporcionó, MongoDB proporcionará una identificación única para cada documento.** Estos 12 bytes, los primeros 4 bytes para la marca de tiempo actual, los siguientes 3 bytes para la identificación de la máquina, los siguientes 2 bytes para la identificación del proceso del servidor mongodb y los 3 bytes restantes son valores incrementales 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
})
```

Aquí *mycol* es un nombre de colección, si la colección no existe en la base de datos, MongoDB creará esta colección y luego insertará el documento en ella. En el documento insertado, si no especificamos el parámetro *_id* , MongoDB asigna un ObjectId único para este documento.

Lea Inserciones e inserciones en línea:
https://riptutorial.com/es/mongodb/topic/10185/inserciones-e-inserciones

Capítulo 16: Mecanismos de autenticación en MongoDB

Introducción

La autenticación es el proceso de verificación de la identidad de un cliente. Cuando el control de acceso, es decir, la autorización, está habilitado, MongoDB requiere que todos los clientes se autentiquen para determinar su acceso.

MongoDB admite varios mecanismos de autenticación que los clientes pueden usar para verificar su identidad. Estos mecanismos permiten que MongoDB se integre en su sistema de autenticación existente.

Examples

Mecanismos de autenticación

MongoDB soporta múltiples mecanismos de autenticación.

Mecanismos de autenticación de cliente y usuario

- SCRAM-SHA-1
- Autentificación del certificado X.509
- MongoDB Challenge and Response (MONGODB-CR)
- Autenticación proxy LDAP, y
- Autenticación Kerberos

Mecanismos de autenticación interna

- Archivo de clave
- X.509

Lea Mecanismos de autenticación en MongoDB en línea:
https://riptutorial.com/es/mongodb/topic/8113/mecanismos-de-autenticacion-en-mongodb

Capítulo 17: Modelo de Autorización MongoDB

Introducción

La autorización es la que básicamente verifica los privilegios de los usuarios. MongoDB soporta diferentes tipos de modelos de autorización. 1. **Control de acceso de base de rol. El** rol es un grupo de privilegios, acciones sobre recursos. Eso es ganancia para los usuarios sobre un espacio de nombres dado (Base de datos). Las acciones se realizan sobre los recursos. Los recursos son cualquier objeto que mantenga el estado en la base de datos.

Examples

Funciones incorporadas

Los roles de usuario de la base de datos incorporados y los roles de administración de base de datos existen en cada base de datos.

Roles de usuario de base de datos

1. `read`
2. `readwrite`

Lea Modelo de Autorización MongoDB en línea:
https://riptutorial.com/es/mongodb/topic/8114/modelo-de-autorizacion-mongodb

Capítulo 18: Mongo como fragmentos

Examples

Configuración de entorno de fragmentación

Miembros del grupo de protección:

Para sharding hay tres jugadores.

1. Config Server
2. Conjuntos de réplicas
3. Mongos

   Para un fragmento de Mongo necesitamos configurar los tres servidores anteriores.

Configuración del servidor de configuración: agregue lo siguiente al archivo mongod conf

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

ejecutar: mongod --config

Podemos elegir el servidor de configuración como conjunto de réplicas o puede ser un servidor independiente. Basados en nuestro requerimiento podemos elegir lo mejor. Si la configuración necesita ejecutarse en el conjunto de réplicas, debemos seguir la configuración del conjunto de réplicas

Configuración de la réplica: Crear conjunto de réplicas // Consulte la configuración de la réplica

Configuración de MongoS: Mongos es la configuración principal en fragmentos. Es un router de consulta para acceder a todos los conjuntos de réplicas.

Agregue lo siguiente en el archivo conf de mongos

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

Configurar Compartido:

Conecte los mongos a través de shell (mongo --host --port)

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

3. sh.shardCollection ("<base de datos>. <colección>", {<clave>: <dirección>})
4. sh.status () // Para asegurar la fragmentación

Lea Mongo como fragmentos en línea: https://riptutorial.com/es/mongodb/topic/7044/mongo-como-fragmentos

Capítulo 19: Mongo como un conjunto de réplicas

Examples

Mongodb como un conjunto de réplicas

Estaríamos creando mongodb como un conjunto de réplicas con 3 instancias. Una instancia sería primaria y las otras 2 serían secundarias.

Para simplificar, voy a tener un conjunto de réplicas con 3 instancias de mongodb ejecutándose en el mismo servidor y, por lo tanto, para lograr esto, las tres instancias de mongodb se ejecutarán en diferentes números de puerto.

En el entorno de producción en el que tiene una instancia de mongodb dedicada que se ejecuta en un solo servidor, puede reutilizar los mismos números de puerto.

1. Crear directorios de datos (ruta donde se almacenarán los datos mongodb en un archivo)

```
- 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. a. Inicia la primera instancia mongod.

- Abra el símbolo del sistema y escriba lo siguiente presione enter.

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

El comando anterior asocia la instancia de mongodb a un nombre de replicaSet "s0" y comienza la primera instancia de mongodb en el puerto 37017 con oplogSize 100MB

2. segundo. Del mismo modo iniciar la segunda instancia de mongodb.

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

El comando anterior asocia la instancia de mongodb a un nombre de replicaSet "s0" y comienza la primera instancia de mongodb en el puerto 37018 con oplogSize 100MB

2. do. Ahora comienza la tercera instancia de Mongodb

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

El comando anterior asocia la instancia de mongodb a un nombre de replicaSet "s0" y comienza la primera instancia de mongodb en el puerto 37019 con oplogSize 100MB

Con las 3 instancias iniciadas, estas 3 instancias son independientes entre sí actualmente. Ahora tendríamos que agrupar estas instancias como un conjunto de réplicas. Lo hacemos con la ayuda de un objeto de configuración.

3.a Conéctate a cualquiera de los servidores mongod a través del shell mongo. Para ello abre el símbolo del sistema y escribe.

```
mongo --port 37017
```

Una vez conectado al shell mongo, crea un objeto de configuración

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

Este objeto de configuración tiene 2 atributos.

- 1. _id: el nombre del conjunto de réplicas ("s0")
  2. miembros: [] (los miembros son una serie de instancias mongod. dejemos esto en blanco por ahora, agregaremos miembros mediante el comando push).

3.b Para empujar (agregar) instancias mongod a la matriz de miembros en el objeto de configuración. En el tipo de concha 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"});
```

Asignamos a cada instancia mongod un _id y un host. _id puede ser cualquier número único y el host debe ser el nombre de host del servidor en el que se ejecuta, seguido del número de puerto.

4. Inicie el objeto de configuración con el siguiente comando en el shell mongo.

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

5. Dale unos segundos y tenemos un conjunto de réplicas de 3 instancias mongod que se ejecutan en el servidor. escriba el siguiente comando para verificar el estado del conjunto de réplicas e identificar cuál es primario y cuál secundario.

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

Lea Mongo como un conjunto de réplicas en línea:
https://riptutorial.com/es/mongodb/topic/6603/mongo-como-un-conjunto-de-replicas

Capítulo 20: Mongo como un conjunto de réplicas

Examples

Compruebe los estados de MongoDB Replica Set

Utilice el siguiente comando para verificar el estado del conjunto de réplicas.

Comando : rs.status ()

Conecte cualquiera de los miembros de la réplica y active este comando para obtener el estado completo del conjunto de réplicas.

Ejemplo:

```
{
 "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 lo anterior podemos conocer el estado completo del conjunto de réplicas.

Lea Mongo como un conjunto de réplicas en línea:
https://riptutorial.com/es/mongodb/topic/7043/mongo-como-un-conjunto-de-replicas

Capítulo 21: MongoDB - Configure un ReplicaSet para soportar TLS / SSL

Introducción

¿Cómo configurar un ReplicaSet para soportar TLS / SSL?

Implementaremos un ReplicaSet de 3 nodos en su entorno local y utilizaremos un certificado autofirmado. No utilice un certificado autofirmado en PRODUCCIÓN.

¿Cómo conectar su cliente a este ReplicaSet?

Conectaremos una Mongo Shell.

Una descripción de los certificados TLS / SSL, PKI (Infraestructura de clave pública) y Autoridad de certificación está más allá del alcance de esta documentación.

Examples

¿Cómo configurar un ReplicaSet para soportar TLS / SSL?

Crear el certificado raíz

El certificado raíz (también conocido como archivo CA) se usará para firmar e identificar su certificado. Para generarlo, ejecute el siguiente comando.

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

Mantenga el certificado raíz y su clave con cuidado, ambos se utilizarán para firmar sus certificados. El certificado raíz también puede ser utilizado por su cliente.

Generar las solicitudes de certificado y las claves privadas

Al generar la Solicitud de firma de certificado (también conocida como CSR), **ingrese el nombre de host (o IP) exacto de su nodo en el campo Nombre común (también conocido como CN). Los otros campos deben tener exactamente el mismo valor.** Es posible que necesite modificar su *archivo / etc / hosts* .

Los siguientes comandos generarán los archivos CSR y las claves privadas 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
```

Debe generar un CSR para cada nodo de su ReplicaSet. Recuerde que el nombre común no es el mismo de un nodo a otro. No base múltiples CSR en la misma clave privada.

Ahora debe tener 3 CSR y 3 claves privadas.

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

Firme sus solicitudes de certificado

Utilice el archivo CA (ca.pem) y su clave privada (ca.key) generada anteriormente para firmar cada solicitud de certificado ejecutando los siguientes comandos.

```
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
```

Debes firmar cada CSR.

Ahora debe tener 3 CSR, 3 claves privadas y 3 certificados autofirmados. MongoDB solo utilizará las Claves privadas y los Certificados.

```
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
```

Cada certificado corresponde a un nodo. Recuerde cuidadosamente qué CN / nombre de host le dio a cada CSR.

Concat cada certificado de nodo con su clave.

Ejecute los comandos a continuación para calcular cada Certificado de nodo con su clave en un archivo (requisito de 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
```

Ahora debe tener 3 archivos PEM.

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

Implementar su ReplicaSet

Asumiremos que sus archivos pem se encuentran en su carpeta actual, así como también data / data1, data / data2 y data / data3.

Ejecute los comandos a continuación para implementar su escucha de 3 Nodos ReplicaSet en los puertos 27017, 27018 y 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
```

Ahora tiene un ReplicaSet de 3 nodos implementado en su entorno local y todas sus transacciones están encriptadas. No puede conectarse a este ReplicaSet sin usar TLS.

Implemente su ReplicaSet para SSL mutuo / confianza mutua

Para forzar a su cliente a proporcionar un certificado de cliente (SSL mutuo), debe agregar el archivo de CA al ejecutar sus instancias.

```
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
```

Ahora tiene un ReplicaSet de 3 nodos implementado en su entorno local y todas sus transacciones están encriptadas. No puede conectarse a este ReplicaSet sin usar TLS o sin proporcionar un Certificado de Cliente en el que confíe su CA.

¿Cómo conectar su cliente (Mongo Shell) a un ReplicaSet?

Sin SSL mutuo

En este ejemplo, podríamos usar el archivo CA (ca.pem) que generó durante la sección " *¿Cómo configurar un ReplicaSet para que admita TLS / SSL?* ". Asumiremos que el archivo CA se encuentra en su carpeta actual.

Asumiremos que sus 3 nodos se ejecutan en mongo1: 27017, mongo2: 27018 y mongo3: 27019. (Es posible que necesite modificar su *archivo* / *etc* / *hosts*).

Desde MongoDB 3.2.6, si su archivo de CA está registrado en su almacén de confianza del sistema operativo, puede conectarse a su ReplicaSet sin proporcionar el archivo de CA.

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

De lo contrario, debe proporcionar el archivo CA.

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

Ahora está conectado a su ReplicaSet y todas las transacciones entre su Shell Mongo y su ReplicaSet están cifradas.

Con SSL mutuo

Si su ReplicaSet solicita un certificado de cliente, debe proporcionar uno firmado por la CA que usa ReploySet Deployment. Los pasos para generar el Certificado de Cliente son casi los mismos que para generar el Certificado de Servidor.

De hecho, solo necesita modificar el campo de nombre común durante la creación de CSR. En lugar de proporcionar 1 nombre de host de nodo en el campo de nombre común, **debe proporcionar todos los nombres de host de ReplicaSet separados por una coma** .

```
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
```

Es posible que enfrente la limitación de tamaño del nombre común si el campo Nombre común es demasiado largo (más de 64 bytes de longitud). Para omitir esta limitación, debe usar el SubjectAltName al generar el 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
)
```

Luego firma el CSR utilizando el certificado y la clave de CA.

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

Finalmente, concat la clave y el certificado firmado.

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

Para conectarse a su ReplicaSet, ahora puede proporcionar el Certificado de Cliente recién generado.

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

Ahora está conectado a su ReplicaSet y todas las transacciones entre su Shell Mongo y su ReplicaSet están cifradas.

Lea MongoDB - Configure un ReplicaSet para soportar TLS / SSL en línea: https://riptutorial.com/es/mongodb/topic/9539/mongodb---configure-un-replicaset-para-soportar-tls--ssl

Capítulo 22: Motores de almacenamiento enchufables

Observaciones

En MongoDB 3.0, MMAP (predeterminado) y WiredTiger son los motores de almacenamiento estable. Por lo general, si su aplicación es de lectura pesada, use MMAP. Si su escritura es pesada, use WiredTiger.

Su solución también puede tener miembros de un conjunto de réplicas mixtas donde puede tener un nodo configurado con MMAP y otro con WiredTiger. Puede usar uno para insertar datos masivos y el otro para leer con herramientas analíticas.

Después de MongoDB 3.2, WiredTiger se convierte en el motor predeterminado.

Examples

MMAP

MMAP es un motor de almacenamiento conectable que lleva el nombre del comando `mmap()` Linux. Asigna archivos a la memoria virtual y optimiza las llamadas de lectura. Si tiene un archivo grande pero necesita leer solo una pequeña parte de él, `mmap()` es mucho más rápido que una llamada `read()` que llevaría todo el archivo a la memoria.

Una desventaja es que no se pueden procesar dos llamadas de escritura en paralelo para la misma colección. Por lo tanto, MMAP tiene un bloqueo a nivel de colección (y no un bloqueo a nivel de documento como lo ofrece WiredTiger). Este bloqueo de recopilación es necesario porque un índice MMAP puede hacer referencia a múltiples documentos y, si esos documentos pudieran actualizarse simultáneamente, el índice sería inconsistente.

WiredTiger

WiredTiger admite **árboles LSM para almacenar índices** . Los árboles LSM son más rápidos para las operaciones de escritura cuando necesita escribir grandes cargas de trabajo de inserciones aleatorias.

En WiredTiger, no **hay actualizaciones en el lugar** . Si necesita actualizar un elemento de un documento, se insertará un nuevo documento mientras que se eliminará el documento anterior.

WiredTiger también ofrece **concurrencia a nivel de documentos** . Se supone que dos operaciones de escritura no afectarán al mismo documento, pero si lo hace, una operación se rebobinará y se ejecutará más adelante. Eso es un gran aumento de rendimiento si los rebobinados son raros.

WiredTiger admite los **algoritmos Snappy y zLib para la compresión** de datos e índices en el

sistema de archivos. Snappy es el predeterminado. Requiere menos CPU, pero tiene una tasa de compresión más baja que zLib.

Cómo usar el motor WiredTiger

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

Nota:

1. Después de mongodb 3.2, el motor predeterminado es WiredTiger.
2. `newWiredTigerDBPath` no debe contener datos de otro motor de almacenamiento. Para migrar sus datos, debe volcarlos y volver a importarlos en el nuevo motor de almacenamiento.

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

En memoria

Todos los datos se almacenan en la memoria (RAM) para una lectura / acceso más rápidos.

rocas mongo

Un motor de valor-clave creado para integrarse con RocksDB de Facebook.

Fusión-io

Un motor de almacenamiento creado por SanDisk que hace posible omitir la capa del sistema de archivos del sistema operativo y escribir directamente en el dispositivo de almacenamiento.

TokuMX

Un motor de almacenamiento creado por Percona que utiliza índices de árboles fractales.

Lea Motores de almacenamiento enchufables en línea:
https://riptutorial.com/es/mongodb/topic/694/motores-de-almacenamiento-enchufables

Capítulo 23: Obteniendo información de la base de datos

Examples

Listar todas las bases de datos

```
show dbs
```

o

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

o

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

Listar todas las colecciones en la base de datos

```
show collections
```

o

```
show tables
```

o

```
db.getCollectionNames()
```

Lea Obteniendo información de la base de datos en línea:
https://riptutorial.com/es/mongodb/topic/6397/obteniendo-informacion-de-la-base-de-datos

Capítulo 24: Operación CRUD

Sintaxis

- Insertar (*documento o matriz de documentos*)
- insertOne ('document', {writeConcern: 'document'})
- insertMany ({[documento 1, documento 2, ...]}, {writeConcern: documento, ordenado: booleano})
- encontrar (*consulta* , *proyección*)
- findOne (*consulta* , *proyección*)
- actualizar (*consultar* , *actualizar*)
- updateOne (*consulta* , *actualización* , {upsert: boolean, writeConcern: document})
- updateMany (*consulta* , *actualización* , {upsert: boolean, writeConcern: document})
- replaceOne (*consulta* , *reemplazo* , {upsert: boolean, writeConcern: document})
- eliminar (*consulta* , *justOne*)
- findAndModify (*consultar* , *ordenar* , *actualizar* , *opciones [opcional]*)

Observaciones

La actualización y eliminación de un documento debe hacerse con cuidado. Dado que la operación puede afectar para varios documentos.

Examples

Crear

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

O

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

La diferencia con `save` es que si el documento aprobado contiene un campo `_id` , si ya existe un documento con ese `_id` , se actualizará en lugar de agregarse como nuevo.

Dos nuevos métodos para insertar documentos en una colección, en MongoDB 3.2.x:

Utilice `insertOne` para insertar un solo registro: -

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

Use `insertMany` para insertar múltiples registros: -

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

```
23}])
```

Tenga en cuenta que la `insert` se resalta como obsoleta en todos los controladores de idioma oficiales desde la versión 3.0. La distinción completa es que los métodos de shell en realidad se quedaron atrás de los otros controladores en la implementación del método. Lo mismo se aplica para todos los otros métodos de CRUD

Actualizar

Actualizar **todo el** objeto:

```
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.
```

O simplemente actualizar un solo campo de un documento. En este caso la `age` :

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

También puede actualizar varios documentos simultáneamente agregando un tercer parámetro. Esta consulta actualizará todos los documentos donde el nombre sea igual a `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 se va a actualizar un nuevo campo, ese campo se agregará al documento.

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

Si se necesita reemplazar un documento,

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

puede ser usado.

Nota : Los campos que utiliza para identificar el objeto se guardarán en el documento actualizado. Los campos que no estén definidos en la sección de actualización se eliminarán del documento.

Borrar

Borra todos los documentos que coinciden con el parámetro de consulta:

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

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

O solo uno

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

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

El método `remove()` MongoDB. Si ejecuta este comando sin ningún argumento o sin un argumento vacío, eliminará todos los documentos de la colección.

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

o

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

Leer

Consulte todos los documentos en la colección de `people` que tienen un campo de `name` con un valor de `'Tom'` :

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

O solo el primero:

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

También puede especificar qué campos devolver al pasar un parámetro de selección de campo. Lo siguiente excluirá el campo `_id` y solo incluirá el campo de `age` :

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

Nota: de forma predeterminada, se `_id` campo `_id` , incluso si no lo solicita. Si desea no obtener el `_id` vuelta, sólo puede seguir el ejemplo anterior y pedir la `_id` ser excluido especificando `_id: 0` (o `_id: false`) Si usted quiere encontrar el registro secundario como objeto de dirección contiene país , ciudad, etc.

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

& especifique el campo también si es necesario

```
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()
```

Más operadores de actualización

Puede utilizar otros operadores además de `$set` al actualizar un documento. El operador `$push` permite insertar un valor en una matriz, en este caso agregaremos un nuevo apodo a la matriz de `nicknames` .

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

El operador `$pull` es lo opuesto a `$push` , puede extraer elementos específicos de las matrices.

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

El operador `$pop` permite eliminar el primer o el último valor de una matriz. Digamos que el documento de Tom tiene una propiedad llamada hermanos que tiene el valor `['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.
```

Parámetro "multi" al actualizar varios documentos

Para actualizar varios documentos en una colección, establezca la opción múltiple en verdadero.

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

multi es opcional. Si se establece en verdadero, actualiza varios documentos que cumplen con los criterios de consulta. Si se establece en falso, actualiza un documento. El valor predeterminado es falso.

db.mycol.find () {"_id": ObjectId (598354878df45ec5), "title": "MongoDB Overview"}

{"_id": ObjectId (59835487adf45ec6), "title": "NoSQL Overview"} {"_id": Id. De objeto (59835487adf45ec7), "título": "Descripción general de los puntos de tutoría"}

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

Actualización de documentos incrustados.

Para el siguiente esquema:

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

Actualice las marcas de Tom a 55 donde las marcas son 50 (use el operador posicional $):

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

Para el siguiente esquema:

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

Actualiza las marcas de inglés de Tom a 85:

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

Explicando el ejemplo anterior:

Al usar {name: "Tom", "marks.subject": "English"} obtendrá la posición del objeto en la matriz de marcas, donde el asunto es inglés. En "marcas. $. Marcas", $ se usa para actualizar en esa posición de la matriz de marcas

Actualizar valores en una matriz

El operador $ posicional identifica un elemento en una matriz para actualizar sin especificar explícitamente la posición del elemento en la matriz.

Considere una colección de estudiantes con los siguientes documentos:

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

Para actualizar 80 a 82 en la matriz de calificaciones en el primer documento, use el operador posicional $ si no conoce la posición del elemento en la matriz:

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

Lea Operación CRUD en línea: https://riptutorial.com/es/mongodb/topic/1683/operacion-crud

Capítulo 25: Operaciones masivas

Observaciones

Construyendo una lista de operaciones de escritura para realizar en forma masiva para una sola colección.

Examples

Convertir un campo a otro tipo y actualizar toda la colección en forma masiva

Generalmente, el caso cuando uno quiere cambiar un tipo de campo a otro, por ejemplo, la colección original puede tener campos "numéricos" o "fecha" guardados como cadenas:

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

El objetivo sería actualizar una colección enorme como la anterior para

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

Para datos relativamente pequeños, se puede lograr lo anterior mediante la iteración de la colección utilizando una `snapshot` con el método `forEach()` del cursor y actualizando cada documento de la siguiente manera:

```
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 } }
    );
});
```

Si bien esto es óptimo para colecciones pequeñas, el rendimiento con colecciones grandes se reduce considerablemente, ya que recorrer un gran conjunto de datos y enviar cada operación de actualización por solicitud al servidor conlleva una penalización computacional.

La API `Bulk()` viene al rescate y mejora considerablemente el rendimiento, ya que las operaciones de escritura se envían al servidor solo una vez en forma masiva. La eficiencia se logra ya que el método no envía todas las solicitudes de escritura al servidor (como `forEach()` con la instrucción de actualización actual dentro del bucle `forEach()`) sino solo una vez cada 1000 solicitudes, lo que hace que las actualizaciones sean más eficientes y más rápidas de lo que es actualmente.

Usando el mismo concepto anterior con el bucle `forEach()` para crear los lotes, podemos actualizar la colección de forma masiva de la siguiente manera. En esta demostración, la API `Bulk()` disponible en las versiones MongoDB `>= 2.6` y `< 3.2` utiliza el método `initializeUnorderedBulkOp()` para ejecutar en paralelo, así como en un orden no determinístico, las operaciones de escritura en los lotes.

Actualiza todos los documentos en la colección de clientes cambiando los campos de `salary` y `dob` a valores `numerical` y de `datetime` y `datetime` respectivamente:

```
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();
    }
});
```

El siguiente ejemplo se aplica a la nueva versión `3.2` MongoDB, que desde entonces ha desaprobado la API `Bulk()` y ha proporcionado un conjunto más reciente de apis con `bulkWrite()` .

Utiliza los mismos cursores que los anteriores, pero crea las matrices con las operaciones masivas utilizando el mismo método de cursor `forEach()` para empujar cada documento de escritura masiva a la matriz. Debido a que los comandos de escritura no pueden aceptar más de 1000 operaciones, es necesario agrupar las operaciones para tener como máximo 1000

operaciones y volver a inicializar la matriz cuando el bucle llegue a la iteración 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); }
```

Lea Operaciones masivas en línea: https://riptutorial.com/es/mongodb/topic/6211/operaciones-masivas

Capítulo 26: Operadores de actualización

Sintaxis

- {$ set: {<field1>: <value1>, <field2>: <value2>, ...}}

Parámetros

parámetros	Sentido
nombre del campo	El campo será actualizado: { ***nombre*** : 'Tom'}
targetVaule	El valor será asignado al campo: {nombre: ***'Tom'*** }

Observaciones

Referencia para el operador $ set : $ set en el sitio web oficial

Examples

$ establecer el operador para actualizar los campos especificados en el documento (s)

I.Vista general

Una diferencia significativa entre MongoDB y RDBMS es que MongoDB tiene muchos tipos de operadores. Uno de ellos es el operador de actualización, que se utiliza en las declaraciones de actualización.

II.¿Qué sucede si no usamos operadores de actualización?

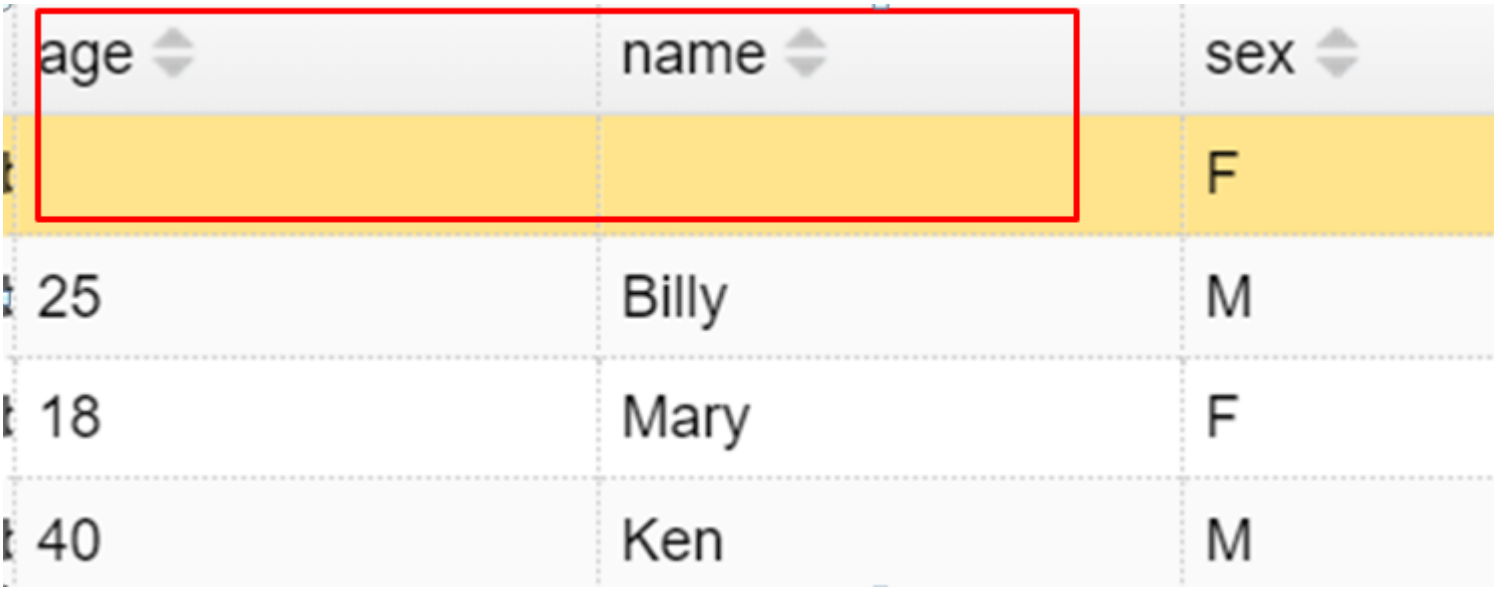
Supongamos que tenemos una colección de **estudiantes** para almacenar información de estudiantes (vista de tabla):

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

Un día obtienes un trabajo que necesita cambiar el género de Tom de "M" a "F". Eso es fácil, ¿verdad? Así que escribe la siguiente declaración muy rápidamente en función de tu experiencia RDBMS:

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

Veamos cual es el resultado:



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

Perdimos la edad y el nombre de Tom! De este ejemplo, podemos saber que **se anulará todo el documento** si no hay un operador de actualización en la declaración de actualización. Este es el comportamiento predeterminado de MongoDB.

III. $ Set operador

Si queremos cambiar solo el campo 'sexo' en el documento de Tom, podemos usar `$set` para especificar qué campo (s) queremos actualizar:

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

El valor de `$set` es un objeto, sus campos representan aquellos campos que desea actualizar en los documentos, y los valores de estos campos son los valores objetivo.

Entonces, el resultado es correcto ahora:

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

Además, si desea cambiar 'sexo' y 'edad' al mismo tiempo, puede agregarlos a `$set` :

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

Lea Operadores de actualización en línea:
https://riptutorial.com/es/mongodb/topic/5880/operadores-de-actualizacion

Capítulo 27: Replicación

Examples

Configuración básica con tres nodos.

El conjunto de réplicas es un grupo de instancias mongod que mantienen el mismo conjunto de datos.

Este ejemplo muestra cómo configurar un conjunto de réplicas con tres instancias en el mismo servidor.

Creando carpetas de datos

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

Iniciando instancias 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
```

Configurando el conjunto 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
```

Probando tu configuración

Para verificar el tipo de configuración `rs.status()` , el resultado debería ser como:

```
{
        "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
}
```

Lea Replicación en línea: https://riptutorial.com/es/mongodb/topic/6205/replicacion

Creditos

S. No	Capítulos	Contributors
1	Empezando con 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	Actualizando la versión de MongoDB	Antti_M
3	Agregación	grape, HoefMeistert, Lakmal Vithanage, LoicM, manetsus, RaR, steveinatorx, titogeo
4	Agregación de MongoDB	Andrii Abramov, Avindu Hewa, Erdenezul, saurav
5	Colecciones	Prosen Ghosh
6	Conductor de pitón	Derlin, sergiuz
7	Configuración	Matt Clark
8	Consulta de datos (Primeros pasos)	Avindu Hewa, oggo, Prosen Ghosh, SommerEngineering
9	Controlador de Java	dev ツ, Emil Burzo
10	Copia de seguridad y restauración de datos	user641887
11	Gestionando MongoDB	Ravi Chandra
12	Índice 2dsphere	gypsyCoder
13	Índices	Adam Comerford, Batsu, Constantin Guay, jerry, Juan Carlos Farah, manetsus, Nic Cottrell, RaR, Rick, Sarthak Adhikari, titogeo, Tomás Cañibano
14	Inserciones e inserciones	Kuhan
15	Mecanismos de autenticación en	Luzan Baral, Niroshan Ranapathi

	MongoDB	
16	Modelo de Autorización MongoDB	Niroshan Ranapathi
17	Mongo como fragmentos	Selva Kumar
18	Mongo como un conjunto de réplicas	user641887
19	MongoDB - Configure un ReplicaSet para soportar TLS / SSL	bappr
20	Motores de almacenamiento enchufables	Constantin Guay, Jorge Aranda, tim, Zanon
21	Obteniendo información de la base de datos	fracz
22	Operación 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
23	Operaciones masivas	chridam
24	Operadores de actualización	yellowB
25	Replicación	ADIMO