
MONGODB
www.postparaprogramadores.com

Contenido

MongoDB - Sharding

MongoDB - Crear copia de seguridad

MongoDB - Implementación

MongoDB - Java

MongoDB - PHP

MongoDB avanzado

MongoDB - Relaciones

MongoDB - Referencias de bases de datos

MongoDB - Consultas cubiertas

MongoDB - Análisis de consultas

MongoDB - Operaciones atómicas

MongoDB - Indexación avanzada

MongoDB - Limitaciones de indexación

MongoDB - ObjectId

MongoDB - Mapa Reducir

MongoDB - Búsqueda de texto

MongoDB - Expresión regular

Trabajando con Rockmongo

MongoDB - GridFS

MongoDB - Colecciones tapadas

Secuencia de incremento automático

MongoDB - Descripción general

MongoDB es una base de datos multiplataforma orientada a documentos que proporciona alto rendimiento, alta disponibilidad y fácil escalabilidad. MongoDB trabaja en concepto de colección y documento.

Base de datos

La base de datos es un contenedor físico para colecciones. Cada base de datos obtiene su propio conjunto de archivos en el sistema de archivos. Un único servidor MongoDB generalmente tiene múltiples bases de datos.

Colección

Collection es un grupo de documentos de MongoDB. Es el equivalente de una tabla RDBMS. Existe una colección dentro de una única base de datos. Las colecciones no imponen un esquema. Los documentos dentro de una colección pueden tener diferentes campos. Por lo general, todos los documentos de una colección tienen un propósito similar o relacionado.

Documento

Un documento es un conjunto de pares clave-valor. Los documentos tienen un esquema dinámico. El esquema dinámico significa que los documentos en la misma colección no necesitan tener el mismo conjunto de campos o estructura, y los campos comunes en los documentos de una colección pueden contener diferentes tipos de datos.

La siguiente tabla muestra la relación de la terminología RDBMS con MongoDB.

RDBMS	MongoDB
Base de datos	Base de datos
Mesa	Colección
Tupla / fila	Documento
columna	Campo
Mesa Unir	Documentos incrustados
Clave primaria	Clave primaria (clave predeterminada _id proporcionada por el propio mongodb)
Servidor de bases de datos y cliente	
Mysqld / Oracle	mongod
mysql / sqlplus	mongo

Documento de muestra

El siguiente ejemplo muestra la estructura de documentos de un sitio de blog, que es simplemente un par de valores clave separados por comas.

```
{
   _id: ObjectId(7df78ad8902c)
   title: 'MongoDB Overview',
   description: 'MongoDB is no sql database',
```

```
   by: 'postparaprogramadores',
   url: 'http://www.postparaprogramadores.com',
   tags: ['mongodb', 'database', 'NoSQL'],
   likes: 100,
   comments: [
      {
         user:'user1',
         message: 'My first comment',
         dateCreated: new Date(2011,1,20,2,15),
         like: 0
      },
      {
         user:'user2',
         message: 'My second comments',
         dateCreated: new Date(2011,1,25,7,45),
         like: 5
      }
   ]
}
```

_id es un número hexadecimal de 12 bytes que garantiza la unicidad de cada documento. Puede proporcionar _id mientras inserta el documento. Si no proporciona, MongoDB proporciona 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 VALOR incremental simple.

MongoDB - Ventajas

Cualquier base de datos relacional tiene un diseño de esquema típico que muestra el número de tablas y la relación entre estas tablas. Mientras que en MongoDB, no existe un concepto de relación.

Ventajas de MongoDB sobre RDBMS

- **Sin esquema** : MongoDB es una base de datos de documentos en la que una colección contiene diferentes documentos. El número de campos, el contenido y el tamaño del documento pueden diferir de un documento a otro.
- La estructura de un solo objeto es clara.
- No se unen complejos.
- Capacidad de consulta profunda. MongoDB admite consultas dinámicas en documentos utilizando un lenguaje de consulta basado en documentos que es casi tan poderoso como SQL.
- Afinación.
- **Facilidad de escalado** : MongoDB es fácil de escalar.
- Conversión / mapeo de objetos de aplicación a objetos de base de datos no necesarios.

- Utiliza memoria interna para almacenar el conjunto de trabajo (en ventana), lo que permite un acceso más rápido a los datos.

¿Por qué usar MongoDB?

- **Almacenamiento orientado a documentos** : los datos se almacenan en forma de documentos de estilo JSON.
- Indice sobre cualquier atributo
- Replicación y alta disponibilidad.
- Fragmentación automática
- Consultas enriquecidas
- Actualizaciones rápidas en el lugar
- Apoyo profesional de MongoDB

¿Dónde usar MongoDB?

- Big Data
- Gestión de contenido y entrega
- Infraestructura móvil y social
- Gestión de datos del usuario
- Centro de datos

MongoDB - Medio ambiente

Veamos ahora cómo instalar MongoDB en Windows.

Instalar MongoDB en Windows

Para instalar MongoDB en Windows, primero descargue la última versión de MongoDB desde https://www.mongodb.org/downloads . Asegúrese de obtener la versión correcta de MongoDB dependiendo de su versión de Windows. Para obtener su versión de Windows, abra el símbolo del sistema y ejecute el siguiente comando.

```
C:\>wmic os get osarchitecture
OSArchitecture
64-bit
C:\>
```

Las versiones de 32 bits de MongoDB solo admiten bases de datos de menos de 2 GB y solo son adecuadas para fines de prueba y evaluación.

Ahora extraiga el archivo descargado en c: \ drive o en cualquier otra ubicación. Asegúrese de que el nombre de la carpeta extraída sea mongodb-win32-i386- [versión] o mongodb-win32-x86_64- [versión]. Aquí [versión] es la versión de descarga de MongoDB.

A continuación, abra el símbolo del sistema y ejecute el siguiente comando.

```
C:\>move mongodb-win64-* mongodb
```

```
   1 dir(s) moved.
C:\>
```

En caso de que haya extraído el MongoDB en una ubicación diferente, vaya a esa ruta utilizando el comando **cd FOLDER / DIR** y ahora ejecute el proceso dado anteriormente.

MongoDB requiere una carpeta de datos para almacenar sus archivos. La ubicación predeterminada para el directorio de datos de MongoDB es c: \ data \ db. Por lo tanto, debe crear esta carpeta con el símbolo del sistema. Ejecute la siguiente secuencia de comandos.

```
C:\>md data
C:\md data\db
```

Si tiene que instalar MongoDB en una ubicación diferente, debe especificar una ruta alternativa para **\ data \ db** configurando la ruta **dbpath** en **mongod.exe** . Por lo mismo, emita los siguientes comandos.

En el símbolo del sistema, navegue hasta el directorio bin presente en la carpeta de instalación de MongoDB. Supongamos que mi carpeta de instalación es **D: \ set up \ mongodb**

```
C:\Users\XYZ>d:
D:\>cd "set up"
D:\set up>cd mongodb
D:\set up\mongodb>cd bin
D:\set up\mongodb\bin>mongod.exe --dbpath "d:\set
up\mongodb\data"
```

Esto mostrará el mensaje **esperando conexiones** en la salida de la consola, lo que indica que el proceso mongod.exe se está ejecutando correctamente.

Ahora para ejecutar MongoDB, debe abrir otro símbolo del sistema y emitir el siguiente comando.

```
D:\set up\mongodb\bin>mongo.exe
MongoDB shell version: 2.4.6
connecting to: test
>db.test.save( { a: 1 } )
>db.test.find()
{ "_id" : ObjectId(5879b0f65a56a454), "a" : 1 }
>
```

Esto mostrará que MongoDB está instalado y se ejecuta correctamente. La próxima vez que ejecute MongoDB, solo debe emitir comandos.

```
D:\set up\mongodb\bin>mongod.exe --dbpath "d:\set
up\mongodb\data"
D:\set up\mongodb\bin>mongo.exe
```

Instalar MongoDB en Ubuntu

Ejecute el siguiente comando para importar la clave GPG pública de MongoDB:

```
sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --
recv 7F0CEB10
```

Cree un archivo /etc/apt/sources.list.d/mongodb.list con el siguiente comando.

```
echo 'deb http://downloads-distro.mongodb.org/repo/ubuntu-
upstart dist 10gen'
   | sudo tee /etc/apt/sources.list.d/mongodb.list
```

Ahora emita el siguiente comando para actualizar el repositorio:

```
sudo apt-get update
```

A continuación, instale MongoDB utilizando el siguiente comando:

```
apt-get install mongodb-10gen = 2.2.3
```

En la instalación anterior, 2.2.3 se publica actualmente la versión MongoDB. Asegúrese de instalar siempre la última versión. Ahora MongoDB está instalado con éxito.

Inicie MongoDB

```
sudo service mongodb start
```

Detener MongoDB

```
sudo service mongodb stop
```

Reiniciar MongoDB

```
sudo service mongodb restart
```

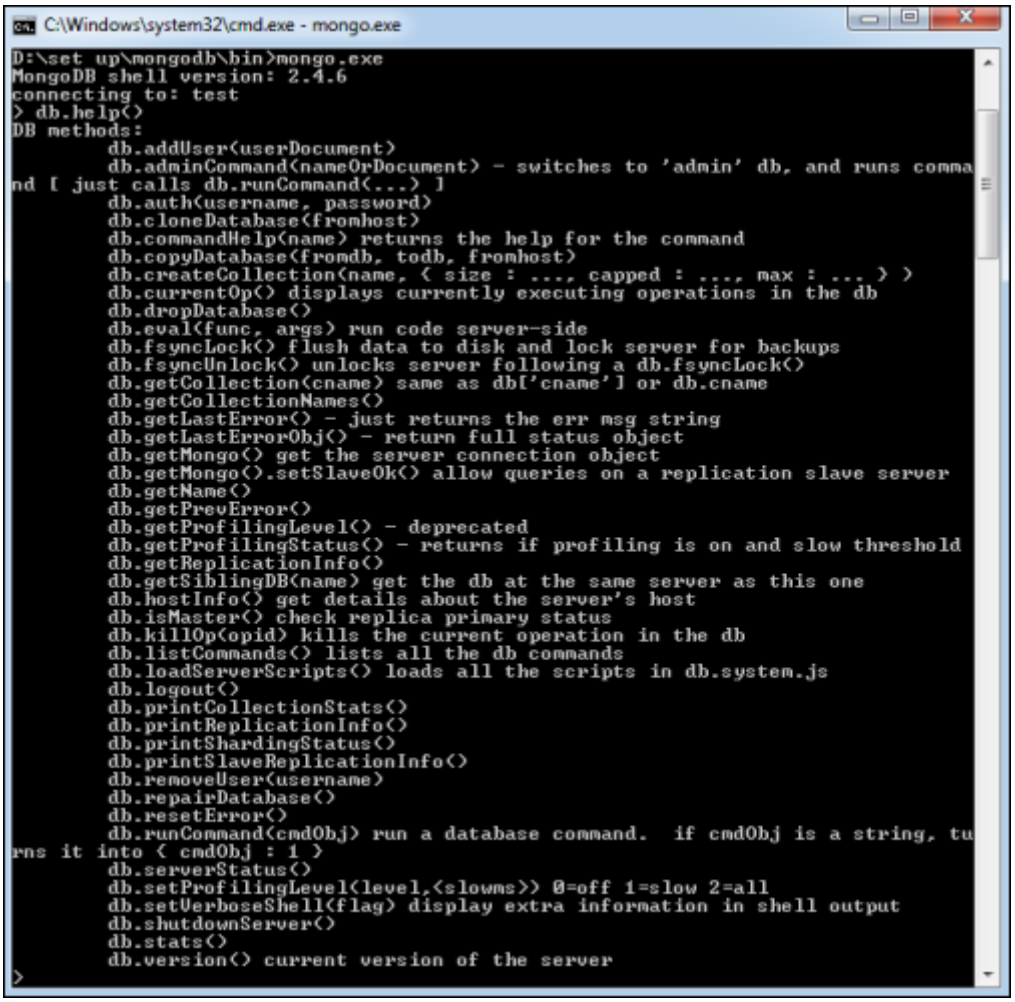
Para usar MongoDB, ejecute el siguiente comando.

```
mongo
```

Esto lo conectará a la ejecución de la instancia de MongoDB.

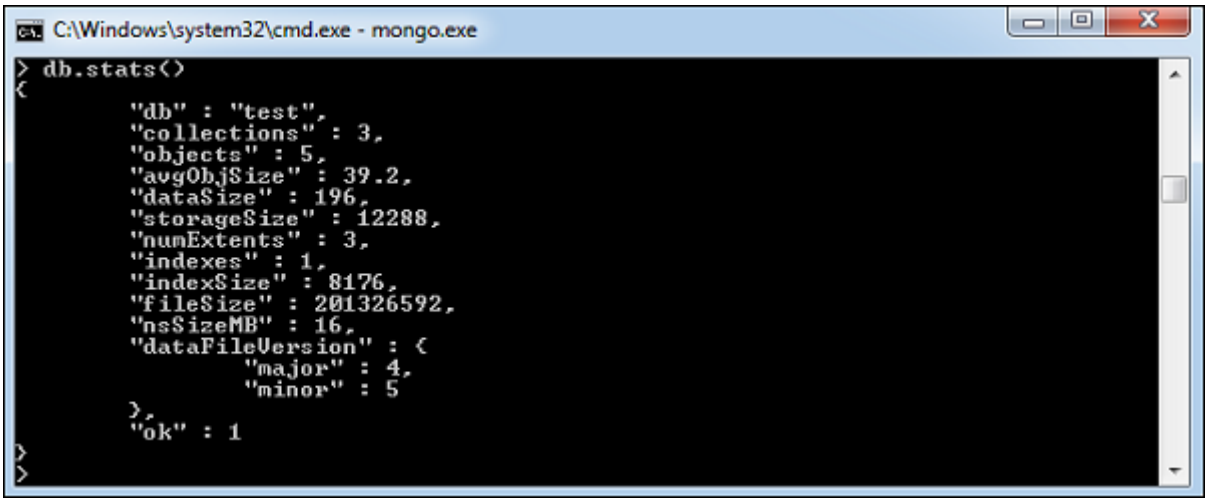
Ayuda MongoDB

Para obtener una lista de comandos, escriba **db.help ()** en el cliente MongoDB. Esto le dará una lista de comandos como se muestra en la siguiente captura de pantalla.



Estadísticas MongoDB

Para obtener estadísticas sobre el servidor MongoDB, escriba el comando **db.stats ()** en el cliente MongoDB. Esto mostrará el nombre de la base de datos, el número de recopilación y los documentos en la base de datos. La salida del comando se muestra en la siguiente captura de pantalla.



Los datos en MongoDB tienen un esquema flexible. Documentos en la misma colección. No necesitan tener el mismo conjunto de campos o estructura, y los campos comunes en los documentos de una colección pueden contener diferentes tipos de datos.

Algunas consideraciones al diseñar Schema en MongoDB

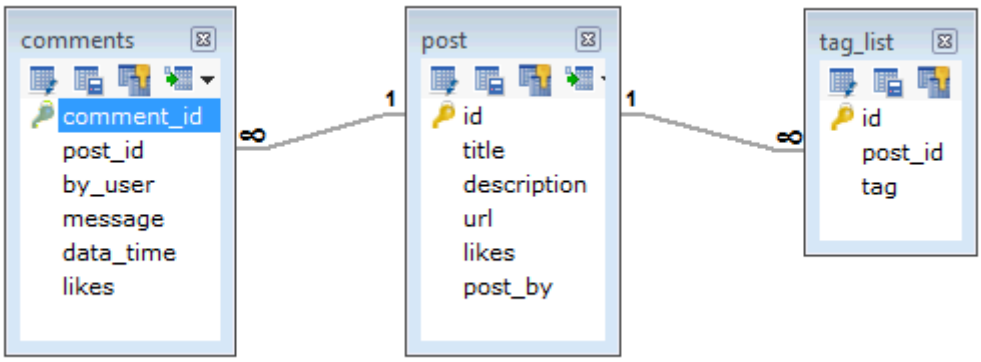
- Diseñe su esquema de acuerdo con los requisitos del usuario.
- Combine objetos en un documento si los usará juntos. De lo contrario, sepárelos (pero asegúrese de que no haya necesidad de unir).
- Duplique los datos (pero limitados) porque el espacio en disco es barato en comparación con el tiempo de cálculo.
- Haz uniones mientras escribes, no al leer.
- Optimice su esquema para los casos de uso más frecuentes.
- Hacer una agregación compleja en el esquema.

Ejemplo

Suponga que un cliente necesita un diseño de base de datos para su blog / sitio web y vea las diferencias entre RDBMS y el diseño de esquema MongoDB. El sitio web tiene los siguientes requisitos.

- Cada publicación tiene el título, la descripción y la URL únicos.
- Cada publicación puede tener una o más etiquetas.
- Cada publicación tiene el nombre de su editor y el número total de Me gusta.
- Cada publicación tiene comentarios de los usuarios junto con su nombre, mensaje, tiempo de datos y me gusta.
- En cada publicación, puede haber cero o más comentarios.

En el esquema RDBMS, el diseño para los requisitos anteriores tendrá un mínimo de tres tablas.



Mientras esté en el esquema MongoDB, el diseño tendrá una publicación de colección y la siguiente estructura:

```
{
   _id: POST_ID
   title: TITLE_OF_POST,
   description: POST_DESCRIPTION,
   by: POST_BY,
   url: URL_OF_POST,
   tags: [TAG1, TAG2, TAG3],
   likes: TOTAL_LIKES,
   comments: [
      {
         user:'COMMENT_BY',
         message: TEXT,
         dateCreated: DATE_TIME,
         like: LIKES
      },
      {
         user:'COMMENT_BY',
         message: TEXT,
         dateCreated: DATE_TIME,
         like: LIKES
      }
   ]
}
```

Entonces, al mostrar los datos, en RDBMS necesita unir tres tablas y en MongoDB, los datos se mostrarán solo desde una colección.

MongoDB - Crear base de datos

En este capítulo, veremos cómo crear una base de datos en MongoDB.

El comando de uso

MongoDB **usa DATABASE_NAME** se usa para crear una base de datos. El comando creará una nueva base de datos si no existe, de lo contrario devolverá la base de datos existente.

Sintaxis

La sintaxis básica de la instrucción **DATABASE** de **uso** es la siguiente:

```
use DATABASE_NAME
```

Ejemplo

Si desea usar una base de datos con el nombre **<mydb>** , **use la** instrucción **DATABASE** sería la siguiente:

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

Para verificar la base de datos seleccionada actualmente, use el comando **db**

```
>db
mydb
```

Si desea verificar su lista de bases de datos, use el comando **show dbs** .

```
>show dbs
local      0.78125GB
test       0.23012GB
```

Su base de datos creada (mydb) no está presente en la lista. Para mostrar la base de datos, debe insertar al menos un documento en ella.

```
>db.movie.insert({"name":"postparaprogramadores"})
>show dbs
local       0.78125GB
mydb        0.23012GB
test        0.23012GB
```

En MongoDB, la base de datos predeterminada es test. Si no creó ninguna base de datos, las colecciones se almacenarán en la base de datos de prueba.

MongoDB - Descartar base de datos

En este capítulo, veremos cómo descartar una base de datos usando el comando MongoDB.

El método dropDatabase ()

El **comando** MongoDB **db.dropDatabase ()** se usa para descartar una base de datos existente.

Sintaxis

La sintaxis básica del **comando dropDatabase ()** es la siguiente:

```
db.dropDatabase()
```

Esto eliminará la base de datos seleccionada. Si no ha seleccionado ninguna base de datos, eliminará la base de datos 'prueba' predeterminada.

Ejemplo

Primero, verifique la lista de bases de datos disponibles con el comando **show dbs** .

```
>show dbs
local       0.78125GB
mydb        0.23012GB
test        0.23012GB
>
```

Si desea eliminar la nueva base de datos **<mydb>** , el comando **dropDatabase ()** sería el siguiente:

```
>use mydb
switched to db mydb
>db.dropDatabase()
>{ "dropped" : "mydb", "ok" : 1 }
>
```

Ahora revise la lista de bases de datos.

```
>show dbs
local      0.78125GB
test       0.23012GB
>
```

MongoDB - Crear colección

En este capítulo, veremos cómo crear una colección usando MongoDB.

El método createCollection ()

MongoDB **db.createCollection (nombre, opciones)** se utiliza para crear una colección.

Sintaxis

La sintaxis básica del **comando createCollection ()** es la siguiente:

```
db.createCollection(name, options)
```

En el comando, **nombre** es el nombre de la colección que se creará. **Opciones** es un documento y se utiliza para especificar la configuración de la colección.

Parámetro	Tipo	Descripción
Nombre	Cuerda	Nombre de la colección a crear.
Opciones	Documento	(Opcional) Especifique opciones sobre el tamaño de la memoria y la indexación

El parámetro de opciones es opcional, por lo que debe especificar solo el nombre de la colección. La siguiente es la lista de opciones que puede usar:

Campo	Tipo	Descripción
tapado	Booleano	(Opcional) Si es verdadero, habilita una colección limitada. La colección con límite es una colección de tamaño fijo que sobrescribe automáticamente sus entradas más antiguas cuando alcanza su tamaño máximo. **Si especifica true, también debe**

		especificar el parámetro de tamaño.
autoIndexId	Booleano	(Opcional) Si es verdadero, cree automáticamente el índice en _id field.s El valor predeterminado es falso.
Talla	número	(Opcional) Especifica un tamaño máximo en bytes para una colección limitada. **Si capped es verdadero, también debe especificar este campo.**
max	número	(Opcional) Especifica el número máximo de documentos permitidos en la colección limitada.

Al insertar el documento, MongoDB primero verifica el campo de tamaño de la colección limitada, luego verifica el campo máximo.

Ejemplos

La sintaxis básica del método **createCollection ()** sin opciones es la siguiente:

```
>use test
switched to db test
>db.createCollection("mycollection")
{ "ok" : 1 }
>
```

Puede verificar la colección creada usando el comando **show collections** .

```
>show collections
mycollection
system.indexes
```

El siguiente ejemplo muestra la sintaxis del método **createCollection ()** con algunas opciones importantes:

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

En MongoDB, no necesita crear una colección. MongoDB crea una colección automáticamente cuando inserta algún documento.

```
>db.postparaprogramadores.insert({"name" :
"postparaprogramadores"})
>show collections
mycol
mycollection
system.indexes
postparaprogramadores
```

```
>
```

MongoDB - Drop Collection

En este capítulo, veremos cómo descartar una colección usando MongoDB.

El método drop ()

Db.collection.drop () de MongoDB se usa para descartar una colección de la base de datos.

Sintaxis

La sintaxis básica del comando **drop ()** es la siguiente:

```
db.COLLECTION_NAME.drop()
```

Ejemplo

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

```
>use mydb
switched to db mydb
>show collections
mycol
mycollection
system.indexes
postparaprogramadores
>
```

Ahora suelte la colección con el nombre **mycollection** .

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

Nuevamente revise la lista de colecciones en la base de datos.

```
>show collections
mycol
system.indexes
postparaprogramadores
>
```

El método drop () devolverá verdadero, si la colección seleccionada se descarta con éxito, de lo contrario, devolverá falso.

MongoDB - Tipos de datos

MongoDB admite muchos tipos de datos. Algunos de ellos son -

- **Cadena** : este es el tipo de datos más utilizado para almacenar los datos. La cadena en MongoDB debe ser válida para UTF-8.

- **Entero** : este tipo se utiliza para almacenar un valor numérico. El entero puede ser de 32 bits o de 64 bits, dependiendo de su servidor.
- **Booleano** : este tipo se utiliza para almacenar un valor booleano (verdadero / falso).
- **Doble** : este tipo se utiliza para almacenar valores de punto flotante.
- **Teclas mín. / Máx .** : este tipo se utiliza para comparar un valor con los elementos BSON más bajos y más altos.
- **Matrices** : este tipo se utiliza para almacenar matrices o listas o valores múltiples en una clave.
- Marca de **tiempo** : **marca de** tiempo. Esto puede ser útil para grabar cuando un documento ha sido modificado o agregado.
- **Objeto** : este tipo de datos se utiliza para documentos incrustados.
- **Nulo** : este tipo se utiliza para almacenar un valor nulo.
- **Símbolo** : este tipo de datos se usa de forma idéntica a una cadena; sin embargo, generalmente está reservado para idiomas que usan un tipo de símbolo específico.
- **Fecha** : este tipo de datos se utiliza para almacenar la fecha u hora actual en formato de hora UNIX. Puede especificar su propia fecha y hora creando un objeto de fecha y pasando el día, el mes y el año.
- **ID de objeto** : este tipo de datos se utiliza para almacenar la ID del documento.
- **Datos binarios** : este tipo de datos se utiliza para almacenar datos binarios.
- **Código** : este tipo de datos se utiliza para almacenar código JavaScript en el documento.
- **Expresión regular** : este tipo de datos se utiliza para almacenar expresiones regulares.

MongoDB - Insertar documento

En este capítulo, aprenderemos cómo insertar documentos en la colección MongoDB.

El método insert ()

Para insertar datos en la colección MongoDB, debe usar el método **insert ()** o **save () de** MongoDB .

Sintaxis

La sintaxis básica del comando **insert ()** es la siguiente:

```
>db.COLLECTION_NAME.insert(document)
```

Ejemplo

```
>db.mycol.insert({
   _id: ObjectId(7df78ad8902c),
```

```
   title: 'MongoDB Overview',
   description: 'MongoDB is no sql database',
   by: 'postparaprogramadores',
   url: 'http://www.postparaprogramadores.com',
   tags: ['mongodb', 'database', 'NoSQL'],
   likes: 100
})
```

Aquí **mycol** es el nombre de nuestra colección, tal como se creó en el capítulo anterior. Si la colección no existe en la base de datos, MongoDB creará esta colección y luego insertará un documento en ella.

En el documento insertado, si no especificamos el parámetro _id, MongoDB asigna un ObjectId único para este documento.

_id es un número hexadecimal de 12 bytes único para cada documento de una colección. 12 bytes se dividen de la siguiente manera:

```
_id: ObjectId(4 bytes timestamp, 3 bytes machine id, 2 bytes
process id,
   3 bytes incrementer)
```

Para insertar varios documentos en una sola consulta, puede pasar una matriz de documentos en el comando insert ().

Ejemplo

```
>db.post.insert([
   {
      title: 'MongoDB Overview',
      description: 'MongoDB is no sql database',
      by: 'postparaprogramadores',
      url: 'http://www.postparaprogramadores.com',
      tags: ['mongodb', 'database', 'NoSQL'],
      likes: 100
   },

   {
      title: 'NoSQL Database',
      description: "NoSQL database doesn't have tables",
      by: 'postparaprogramadores',
      url: 'http://www.postparaprogramadores.com',
      tags: ['mongodb', 'database', 'NoSQL'],
      likes: 20,
      comments: [
         {
            user:'user1',
            message: 'My first comment',
            dateCreated: new Date(2013,11,10,2,35),
            like: 0
         }
      ]
   }
])
```

Para insertar el documento, también puede usar **db.post.save (documento)** . Si no especifica **_id** en el documento, el método **save ()** funcionará igual que el método **insert ()** . Si especifica _id, reemplazará los datos completos del documento que contiene _id como se especifica en el método save ().

MongoDB - Documento de consulta

En este capítulo, aprenderemos cómo consultar documentos de la colección MongoDB.

El método find ()

Para consultar datos de la colección MongoDB, debe usar el método **find () de** MongoDB .

Sintaxis

La sintaxis básica del método **find ()** es la siguiente:

```
>db.COLLECTION_NAME.find()
```

El método **find ()** mostrará todos los documentos de forma no estructurada.

El método pretty ()

Para mostrar los resultados de forma formateada, puede usar el método **pretty ()** .

Sintaxis

```
>db.mycol.find().pretty()
```

Ejemplo

```
>db.mycol.find().pretty()
{
   "_id": ObjectId(7df78ad8902c),
   "title": "MongoDB Overview",
   "description": "MongoDB is no sql database",
   "by": "postparaprogramadores",
   "url": "http://www.postparaprogramadores.com",
   "tags": ["mongodb", "database", "NoSQL"],
   "likes": "100"
}
>
```

Además del método find (), existe el método **findOne ()** , que solo devuelve un documento.

RDBMS Donde equivalentes de cláusulas en MongoDB

Para consultar el documento en función de alguna condición, puede utilizar las siguientes operaciones.

Operación	Sintaxis	Ejemplo	RDBMS equivalente
Igualdad	{<key>: <value>}	db.mycol.find ({"by": "postparaprogramadores"}). pretty ()	donde by = 'punto de tutoriales'
Menos que	{<key>: {$ lt: <value>}}	db.mycol.find ({"me gusta": {$ lt: 50}}). bonita ()	donde le gusta <50
Menos que iguales	{<key>: {$ lte: <value>}}	db.mycol.find ({"me gusta": {$ lte: 50}}). bonita ()	donde le gusta <= 50
Mas grande que	{<key>: {$ gt: <value>}}	db.mycol.find ({"me gusta": {$ gt: 50}}). bonita ()	donde le gusta> 50
Mayor que igual	{<key>: {$ gte: <value>}}	db.mycol.find ({"me gusta": {$ gte: 50}}). bonita ()	donde le gusta> = 50
No es igual	{<key>: {$ ne: <value>}}	db.mycol.find ({"me gusta": {$ ne: 50}}). bonita ()	donde le gusta! = 50

Y en MongoDB

Sintaxis

En el método **find ()** , si pasa varias claves separándolas por ',' MongoDB lo trata como condición **AND** . La siguiente es la sintaxis básica de **AND** -

```
>db.mycol.find(
   {
      $and: [
         {key1: value1}, {key2:value2}
      ]
```

```
    }
).pretty()
```

Ejemplo

El siguiente ejemplo mostrará todos los tutoriales escritos por 'punto de tutoriales' y cuyo título es 'Descripción general de MongoDB'.

```
>db.mycol.find({$and:[{"by":"postparaprogramadores"},{"title"
: "MongoDB Overview"}]}).pretty() {
   "_id": ObjectId(7df78ad8902c),
   "title": "MongoDB Overview",
   "description": "MongoDB is no sql database",
   "by": "postparaprogramadores",
   "url": "http://www.postparaprogramadores.com",
   "tags": ["mongodb", "database", "NoSQL"],
   "likes": "100"
}
```

Para el ejemplo anterior, la cláusula where equivalente será **'where by =' postparaprogramadores 'AND title =' MongoDB Overview "** . Puede pasar cualquier número de pares clave, valor en la cláusula find.

O en MongoDB

Sintaxis

Para consultar documentos basados en la condición OR, debe usar **$ o** palabra clave. La siguiente es la sintaxis básica de **OR** -

```
>db.mycol.find(
   {
      $or: [
         {key1: value1}, {key2:value2}
      ]
   }
).pretty()
```

Ejemplo

El siguiente ejemplo mostrará todos los tutoriales escritos por 'punto de tutoriales' o cuyo título es 'Descripción general de MongoDB'.

```
>db.mycol.find({$or:[{"by":"postparaprogramadores"},{"title":
"MongoDB Overview"}]}).pretty()
{
   "_id": ObjectId(7df78ad8902c),
   "title": "MongoDB Overview",
   "description": "MongoDB is no sql database",
   "by": "postparaprogramadores",
   "url": "http://www.postparaprogramadores.com",
   "tags": ["mongodb", "database", "NoSQL"],
```

```
   "likes": "100"
}
>
```

Usando AND y OR juntos

Ejemplo

El siguiente ejemplo mostrará los documentos que tienen Me gusta mayor que 10 y cuyo título es 'Visión general de MongoDB' o 'Punto de tutoriales'. La cláusula where equivalente de SQL es **'where likes> 10 AND (by =' postparaprogramadores 'OR title =' MongoDB Overview ')'**

```
>db.mycol.find({"likes": {$gt:10}, $or: [{"by":
"postparaprogramadores"},
   {"title": "MongoDB Overview"}]}).pretty()
{
   "_id": ObjectId(7df78ad8902c),
   "title": "MongoDB Overview",
   "description": "MongoDB is no sql database",
   "by": "postparaprogramadores",
   "url": "http://www.postparaprogramadores.com",
   "tags": ["mongodb", "database", "NoSQL"],
   "likes": "100"
}
>
```

MongoDB - Actualizar documento

Los métodos **update () y save () de** MongoDB se utilizan para actualizar el documento en una colección. El método update () actualiza los valores en el documento existente mientras que el método save () reemplaza el documento existente con el documento pasado en el método save ().

Método MongoDB Update ()

El método update () actualiza los valores en el documento existente.

Sintaxis

La sintaxis básica del método **update ()** es la siguiente:

```
>db.COLLECTION_NAME.update(SELECTION_CRITERIA, UPDATED_DATA)
```

Ejemplo

Considere que la colección mycol tiene los siguientes datos.

```
{ "_id" : ObjectId(5983548781331adf45ec5), "title":"MongoDB
Overview"}
```

```
{ "_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL
Overview"}
{ "_id" : ObjectId(5983548781331adf45ec7),
"title":"Postparaprogramadores Overview"}
```

El siguiente ejemplo establecerá el nuevo título 'Nuevo tutorial de MongoDB' de los documentos cuyo título es 'Descripción general de MongoDB'.

```
>db.mycol.update({'title':'MongoDB
Overview'},{$set:{'title':'New MongoDB Tutorial'}})
>db.mycol.find()
{ "_id" : ObjectId(5983548781331adf45ec5), "title":"New
MongoDB Tutorial"}
{ "_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL
Overview"}
{ "_id" : ObjectId(5983548781331adf45ec7),
"title":"Postparaprogramadores Overview"}
>
```

Por defecto, MongoDB actualizará solo un documento. Para actualizar varios documentos, debe establecer un parámetro 'multi' en verdadero.

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

Método Save () de MongoDB

El método **save ()** reemplaza el documento existente con el nuevo documento pasado en el método save ().

Sintaxis

La sintaxis básica del método **save ()** de MongoDB se muestra a continuación:

```
>db.COLLECTION_NAME.save({_id:ObjectId(),NEW_DATA})
```

Ejemplo

El siguiente ejemplo reemplazará el documento con el _id '5983548781331adf45ec5'.

```
>db.mycol.save(
   {
      "_id" : ObjectId(5983548781331adf45ec5),
"title":"Postparaprogramadores New Topic",
      "by":"Postparaprogramadores"
   }
)
>db.mycol.find()
{ "_id" : ObjectId(5983548781331adf45ec5),
"title":"Postparaprogramadores New Topic",
   "by":"Postparaprogramadores"}
```

```
{ "_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL
Overview"}
{ "_id" : ObjectId(5983548781331adf45ec7),
"title":"Postparaprogramadores Overview"}
>
```

MongoDB - Eliminar documento

En este capítulo, aprenderemos cómo eliminar un documento usando MongoDB.

El método remove ()

El método **remove () de** MongoDB se usa para eliminar un documento de la colección. El método remove () acepta dos parámetros. Uno es el criterio de eliminación y el segundo es solo una bandera.

- **criterios de eliminación** : (Opcional) se eliminarán los criterios de eliminación de acuerdo con los documentos.
- **justOne** : (Opcional) si se establece en verdadero o 1, elimine solo un documento.

Sintaxis

La sintaxis básica del método **remove ()** es la siguiente:

```
>db.COLLECTION_NAME.remove(DELLETION_CRITTERIA)
```

Ejemplo

Considere que la colección mycol tiene los siguientes datos.

```
{ "_id" : ObjectId(5983548781331adf45ec5), "title":"MongoDB
Overview"}
{ "_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL
Overview"}
{ "_id" : ObjectId(5983548781331adf45ec7),
"title":"Postparaprogramadores Overview"}
```

El siguiente ejemplo eliminará todos los documentos cuyo título es 'Descripción general de MongoDB'.

```
>db.mycol.remove({'title':'MongoDB Overview'})
>db.mycol.find()
{ "_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL
Overview"}
{ "_id" : ObjectId(5983548781331adf45ec7),
"title":"Postparaprogramadores Overview"}
>
```

Eliminar solo uno

Si hay varios registros y desea eliminar solo el primer registro, configure el parámetro **justOne** en el método **remove ()** .

```
>db.COLLECTION_NAME.remove(DELETION_CRITERIA,1)
```

Eliminar todos los documentos

Si no especifica los criterios de eliminación, MongoDB eliminará documentos completos de la colección. **Esto es equivalente al comando truncado de SQL.**

```
>db.mycol.remove({})
>db.mycol.find()
>
```

MongoDB - Proyección

En MongoDB, la proyección significa seleccionar solo los datos necesarios en lugar de seleccionar todos los datos de un documento. Si un documento tiene 5 campos y necesita mostrar solo 3, seleccione solo 3 campos de ellos.

El método find ()

El método **find ()** de MongoDB , explicado en el Documento de consulta de MongoDB, acepta el segundo parámetro opcional que es la lista de campos que desea recuperar. En MongoDB, cuando ejecuta el método **find ()** , muestra todos los campos de un documento. Para limitar esto, debe establecer una lista de campos con el valor 1 o 0. 1 se usa para mostrar el campo mientras que 0 se usa para ocultar los campos.

Sintaxis

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

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

Ejemplo

Considere la colección mycol tiene los siguientes datos:

```
{ "_id" : ObjectId(5983548781331adf45ec5), "title":"MongoDB
Overview"}
{ "_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL
Overview"}
{ "_id" : ObjectId(5983548781331adf45ec7),
"title":"Postparaprogramadores Overview"}
```

El siguiente ejemplo mostrará el título del documento mientras consulta el documento.

```
>db.mycol.find({},{"title":1,_id:0})
{"title":"MongoDB Overview"}
```

```
{"title":"NoSQL Overview"}
{"title":"Postparaprogramadores Overview"}
>
```

Tenga en cuenta que el campo **_id** siempre se muestra al ejecutar el método **find ()** , si no desea este campo, debe configurarlo como 0.

MongoDB - Limit Records

En este capítulo, aprenderemos cómo limitar registros usando MongoDB.

El método Limit ()

Para limitar los registros en MongoDB, debe usar el método **limit ()** . El método acepta un argumento de tipo de número, que es el número de documentos que desea que se muestren.

Sintaxis

La sintaxis básica del método **limit ()** es la siguiente:

```
>db.COLLECTION_NAME.find().limit(NUMBER)
```

Ejemplo

Considere que la colección myycol tiene los siguientes datos.

```
{ "_id" : ObjectId(5983548781331adf45ec5), "title":"MongoDB
Overview"}
{ "_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL
Overview"}
{ "_id" : ObjectId(5983548781331adf45ec7),
"title":"Postparaprogramadores Overview"}
```

El siguiente ejemplo mostrará solo dos documentos mientras consulta el documento.

```
>db.mycol.find({},{"title":1,_id:0}).limit(2)
{"title":"MongoDB Overview"}
{"title":"NoSQL Overview"}
>
```

Si no especifica el argumento de número en el método **limit ()** , mostrará todos los documentos de la colección.

Método Skip () de MongoDB

Además del método limit (), hay un método más **skip ()** que también acepta el argumento de tipo de número y se utiliza para omitir el número de documentos.

Sintaxis

La sintaxis básica del método **skip ()** es la siguiente:

```
>db.COLLECTION_NAME.find().limit(NUMBER).skip(NUMBER)
```

Ejemplo

El siguiente ejemplo mostrará solo el segundo documento.

```
>db.mycol.find({},{"title":1,_id:0}).limit(1).skip(1)
{"title":"NoSQL Overview"}
>
```

Tenga en cuenta que el valor predeterminado en el método **skip ()** es 0.

MongoDB - Ordenar registros

En este capítulo, aprenderemos cómo ordenar registros en MongoDB.

El método sort ()

Para ordenar documentos en MongoDB, debe usar el método **sort ()** . El método acepta un documento que contiene una lista de campos junto con su orden de clasificación. Para especificar el orden de clasificación se utilizan 1 y -1. 1 se usa para el orden ascendente, mientras que -1 se usa para el orden descendente.

Sintaxis

La sintaxis básica del método **sort ()** es la siguiente:

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

Ejemplo

Considere que la colección myycol tiene los siguientes datos.

```
{ "_id" : ObjectId(5983548781331adf45ec5), "title":"MongoDB
Overview"}
{ "_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL
Overview"}
{ "_id" : ObjectId(5983548781331adf45ec7),
"title":"Postparaprogramadores Overview"}
```

El siguiente ejemplo mostrará los documentos ordenados por título en orden descendente.

```
>db.mycol.find({},{"title":1,_id:0}).sort({"title":-1})
{"title":"Postparaprogramadores Overview"}
{"title":"NoSQL Overview"}
{"title":"MongoDB Overview"}
>
```

Tenga en cuenta que si no especifica la preferencia de clasificación, el método **sort ()** mostrará los documentos en orden ascendente.

MongoDB - Indexación

Los índices admiten la resolución eficiente de consultas. Sin índices, MongoDB debe escanear cada documento de una colección para seleccionar aquellos documentos que coincidan con la declaración de la consulta. Este escaneo es altamente ineficiente y requiere que MongoDB procese un gran volumen de datos.

Los índices son estructuras de datos especiales, que almacenan una pequeña porción del conjunto de datos en una forma fácil de recorrer. El índice almacena el valor de un campo específico o conjunto de campos, ordenado por el valor del campo como se especifica en el índice.

El método sureIndex ()

Para crear un índice, debe utilizar el método sureIndex () de MongoDB.

Sintaxis

La sintaxis básica del método **sureIndex ()** es la siguiente ().

```
>db.COLLECTION_NAME.ensureIndex({KEY:1})
```

Aquí la clave es el nombre del campo en el que desea crear el índice y 1 es para el orden ascendente. Para crear el índice en orden descendente, debe usar -1.

Ejemplo

```
>db.mycol.ensureIndex({"title":1})
>
```

En el método **allowIndex ()** puede pasar múltiples campos, para crear índices en múltiples campos.

```
>db.mycol.ensureIndex({"title":1,"description":-1})
>
```

El método **sureIndex ()** también acepta la lista de opciones (que son opcionales). La siguiente es la lista:

Parámetro	Tipo	Descripción
antecedentes	Booleano	Crea el índice en segundo plano para que la creación de un índice no bloquee otras actividades de la base de datos. Especifique true para construir en segundo plano. El valor predeterminado

		es **falso** .
único	Booleano	Crea un índice único para que la colección no acepte la inserción de documentos donde la clave o claves de índice coinciden con un valor existente en el índice. Especifique verdadero para crear un índice único. El valor predeterminado es **falso** .
nombre	cuerda	El nombre del índice. Si no se especifica, MongoDB genera un nombre de índice concatenando los nombres de los campos indexados y el orden de clasificación.
dropDups	Booleano	Crea un índice único en un campo que puede tener duplicados. MongoDB indexa solo la primera aparición de una clave y elimina todos los documentos de la colección que contienen sucesos posteriores de esa clave. Especifique verdadero para crear un índice único. El valor predeterminado es **falso** .
escaso	Booleano	Si es verdadero, el índice solo hace referencia a documentos con el campo especificado. Estos índices usan menos espacio pero se comportan de manera diferente en algunas situaciones (en particular, los tipos). El valor predeterminado es **falso** .
expireAfterSeconds	entero	Especifica un valor, en segundos, como TTL para controlar cuánto tiempo MongoDB retiene los documentos en esta colección.
v	versión índice	El número de versión del índice. La versión predeterminada del índice depende de la versión de MongoDB que se ejecute al crear el índice.
pesos	documento	El peso es un número que va de 1 a 99,999 y denota la importancia del campo en relación con los otros campos indexados en términos de puntaje.
idioma predeterminado	cuerda	Para un índice de texto, el idioma que determina la lista de palabras de detención y las reglas para el stemmer y el tokenizer. El valor predeterminado es **inglés** .

language_override	cuerda	Para un índice de texto, especifique el nombre del campo en el documento que contiene, el idioma para anular el idioma predeterminado. El valor predeterminado es el idioma.

MongoDB - Agregación

Las operaciones de agregaciones procesan registros de datos y devuelven resultados calculados. Las operaciones de agregación agrupan valores de varios documentos y pueden realizar una variedad de operaciones en los datos agrupados para obtener un único resultado. En SQL count (*) y con group by es un equivalente de la agregación mongodb.

El método agregado ()

Para la agregación en MongoDB, debe usar **el** método **agreggate ()** .

Sintaxis

La sintaxis básica del método **agregado ()** es la siguiente:

```
>db.COLLECTION_NAME.aggregate(AGGREGATE_OPERATION)
```

Ejemplo

En la colección tiene los siguientes datos:

```
{
   _id: ObjectId(7df78ad8902c)
   title: 'MongoDB Overview',
   description: 'MongoDB is no sql database',
   by_user: 'postparaprogramadores',
   url: 'http://www.postparaprogramadores.com',
   tags: ['mongodb', 'database', 'NoSQL'],
   likes: 100
},
{
   _id: ObjectId(7df78ad8902d)
   title: 'NoSQL Overview',
   description: 'No sql database is very fast',
   by_user: 'postparaprogramadores',
   url: 'http://www.postparaprogramadores.com',
   tags: ['mongodb', 'database', 'NoSQL'],
   likes: 10
},
{
   _id: ObjectId(7df78ad8902e)
   title: 'Neo4j Overview',
   description: 'Neo4j is no sql database',
   by_user: 'Neo4j',
   url: 'http://www.neo4j.com',
```

```
   tags: ['neo4j', 'database', 'NoSQL'],
   likes: 750
},
```

Ahora, de la colección anterior, si desea mostrar una lista que indique cuántos tutoriales escribe cada usuario, entonces usará el siguiente método **agregar ()** :

```
> db.mycol.aggregate([{$group : {_id : "$by_user",
num_tutorial : {$sum : 1}}}])
{
   "result" : [
      {
         "_id" : "postparaprogramadores",
         "num_tutorial" : 2
      },
      {
         "_id" : "Neo4j",
         "num_tutorial" : 1
      }
   ],
   "ok" : 1
}
>
```

La consulta equivalente de SQL para el caso de uso anterior se **seleccionará by_user, count (*) from mycol group by by_user** .

En el ejemplo anterior, hemos agrupado los documentos por campo **by_user** y en cada aparición de by_user se incrementa el valor anterior de la suma. A continuación se incluye una lista de expresiones de agregación disponibles.

Expresión	Descripción	Ejemplo
$ suma	Resume el valor definido de todos los documentos en la colección.	db.mycol.aggregate ([{$ group: {_id: "$ by_user", num_tutorial: {$ sum: "$ likes"}}}])
$ promedio	Calcula el promedio de todos los valores dados de todos los documentos de la colección.	db.mycol.aggregate ([{$ group: {_id: "$ by_user", num_tutorial: {$ avg: "$ likes"}}}])
$ min	Obtiene el mínimo de los valores correspondientes de todos los documentos de la colección.	db.mycol.aggregate ([{$ group: {_id: "$ by_user", num_tutorial: {$

		min: "$ likes"}}}])
$ max	Obtiene el máximo de los valores correspondientes de todos los documentos de la colección.	db.mycol.aggregate ([{$ group: {_id: "$ by_user", num_tutorial: {$ max: "$ likes"}}}])
$ push	Inserta el valor en una matriz en el documento resultante.	db.mycol.aggregate ([{$ group: {_id: "$ by_user", url: {$ push: "$ url"}}}])
$ addToSet	Inserta el valor en una matriz en el documento resultante pero no crea duplicados.	db.mycol.aggregate ([{$ group: {_id: "$ by_user", url: {$ addToSet: "$ url"}}}])
$ primero	Obtiene el primer documento de los documentos de origen según la agrupación. Por lo general, esto solo tiene sentido junto con algunas etapas "$ sort" aplicadas previamente.	db.mycol.aggregate ([{$ group: {_id: "$ by_user", first_url: {$ first: "$ url"}}}])
$ último	Obtiene el último documento de los documentos de origen según la agrupación. Por lo general, esto solo tiene sentido junto con algunas etapas "$ sort" aplicadas previamente.	db.mycol.aggregate ([{$ group: {_id: "$ by_user", last_url: {$ last: "$ url"}}}])

Concepto de tubería

En el comando UNIX, la canalización de shell significa la posibilidad de ejecutar una operación en alguna entrada y usar la salida como entrada para el siguiente comando y así sucesivamente. MongoDB también admite el mismo concepto en el marco de agregación. Hay un conjunto de etapas posibles y cada una de ellas se toma como un conjunto de documentos como entrada y produce un conjunto resultante de documentos (o el documento JSON final resultante al final de la tubería). Esto a su vez se puede utilizar para la siguiente etapa y así sucesivamente.

Las siguientes son las posibles etapas en el marco de agregación:

- **$ project** : se utiliza para seleccionar algunos campos específicos de una colección.
- **$ match** : esta es una operación de filtrado y, por lo tanto, puede reducir la cantidad de documentos que se proporcionan como entrada para la siguiente etapa.

- **$ group** - Esto hace la agregación real como se discutió anteriormente.
- **$ sort** - Ordena los documentos.
- **$ skip** : con esto, es posible saltar hacia adelante en la lista de documentos para una cantidad dada de documentos.
- **Límite de $** : esto limita la cantidad de documentos a mirar, por el número dado a partir de las posiciones actuales.
- **$ unwind** : se utiliza para desenrollar documentos que utilizan matrices. Cuando se usa una matriz, los datos se unen previamente y esta operación se deshacerá para tener documentos individuales nuevamente. Por lo tanto, con esta etapa aumentaremos la cantidad de documentos para la siguiente etapa.

MongoDB - Replicación

La replicación es el proceso de sincronización de datos en varios servidores. La replicación proporciona redundancia y aumenta la disponibilidad de datos con múltiples copias de datos en diferentes servidores de bases de datos. La replicación protege una base de datos de la pérdida de un solo servidor. La replicación también le permite recuperarse de fallas de hardware e interrupciones del servicio. Con copias adicionales de los datos, puede dedicar uno a la recuperación de desastres, informes o copias de seguridad.

¿Por qué replicación?

- Para mantener sus datos seguros
- Alta disponibilidad de datos (24 * 7)
- Recuperación de desastres
- Sin tiempo de inactividad por mantenimiento (como copias de seguridad, reconstrucciones de índice, compactación)
- Escala de lectura (copias adicionales para leer)
- El conjunto de réplicas es transparente para la aplicación

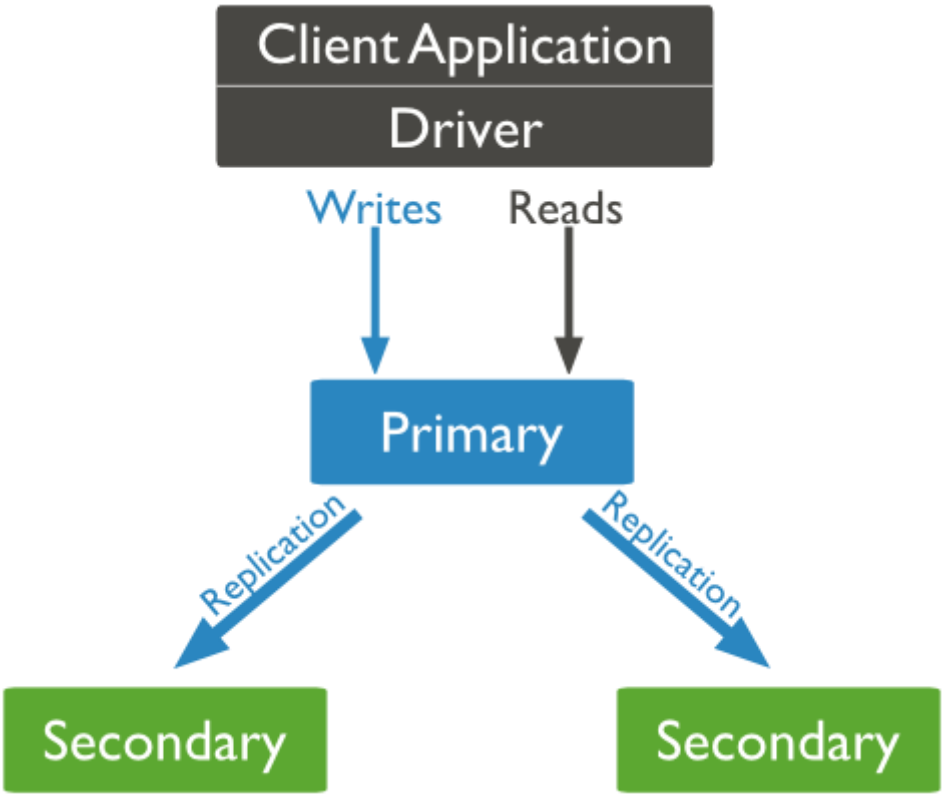
Cómo funciona la replicación en MongoDB

MongoDB logra la replicación mediante el uso del conjunto de réplicas. Un conjunto de réplicas es un grupo de instancias **mongod** que alojan el mismo conjunto de datos. En una réplica, un nodo es el nodo primario que recibe todas las operaciones de escritura. Todas las demás instancias, como las secundarias, aplican operaciones desde la primaria para que tengan el mismo conjunto de datos. El conjunto de réplicas solo puede tener un nodo primario.

- El conjunto de réplicas es un grupo de dos o más nodos (generalmente se requieren un mínimo de 3 nodos).
- En un conjunto de réplica, un nodo es el nodo primario y los nodos restantes son secundarios.
- Todos los datos se replican del nodo primario al secundario.
- En el momento de la conmutación por error o el mantenimiento automáticos, se establece la elección para primaria y se elige un nuevo nodo primario.

- Después de la recuperación del nodo fallido, vuelve a unirse al conjunto de réplicas y funciona como un nodo secundario.

Se muestra un diagrama típico de la replicación MongoDB en el que la aplicación cliente siempre interactúa con el nodo primario y el nodo primario luego replica los datos a los nodos secundarios.



Características del conjunto de réplicas

- Un grupo de N nodos
- Cualquier nodo puede ser primario
- Todas las operaciones de escritura van a primaria
- Conmutación por error automática
- Recuperación automática
- Elección consensuada de primaria

Configurar un conjunto de réplicas

En este tutorial, convertiremos la instancia independiente de MongoDB en un conjunto de réplicas. Para convertir al conjunto de réplica, los siguientes son los pasos:

- El apagado ya está ejecutando el servidor MongoDB.
-
- Inicie el servidor MongoDB especificando la opción replSet. La siguiente es la sintaxis básica de --replSet -

```
mongod --port "PORT" --dbpath "YOUR_DB_DATA_PATH" --replSet
"REPLICA_SET_INSTANCE_NAME"
```

Ejemplo

```
mongod --port 27017 --dbpath "D:\set up\mongodb\data" --
replSet rs0
```

- Comenzará una instancia mongod con el nombre rs0, en el puerto 27017.
- Ahora inicie el símbolo del sistema y conéctese a esta instancia mongod.
- En el cliente Mongo, emita el comando **rs.initiate ()** para iniciar un nuevo conjunto de réplicas.
- Para verificar la configuración del conjunto de réplicas, emita el comando **rs.conf ()** . Para verificar el estado del conjunto de réplicas, emita el comando **rs.status ()** .

Agregar miembros al conjunto de réplicas

Para agregar miembros al conjunto de réplicas, inicie instancias mongod en varias máquinas. Ahora inicie un cliente mongo y emita un comando **rs.add ()** .

Sintaxis

La sintaxis básica del **comando rs.add ()** es la siguiente:

```
>rs.add(HOST_NAME:PORT)
```

Ejemplo

Supongamos que el nombre de su instancia de mongod es **mongod1.net** y se ejecuta en el puerto **27017** . Para agregar esta instancia al conjunto de réplicas, emita el comando **rs.add ()** en el cliente Mongo.

```
>rs.add("mongod1.net:27017")
>
```

Puede agregar una instancia de mongod al conjunto de réplicas solo cuando esté conectado al nodo primario. Para verificar si está conectado al primario o no, emita el comando **db.isMaster ()** en el cliente mongo.

MongoDB - Sharding

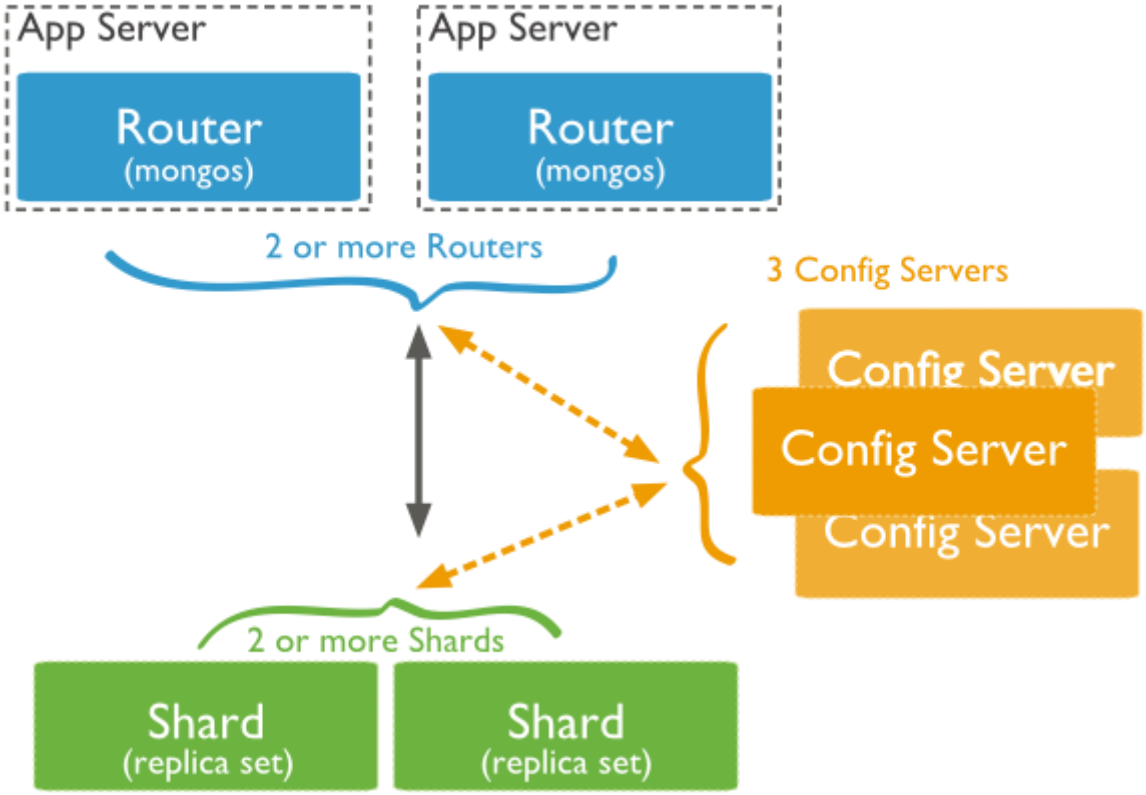
Sharding es el proceso de almacenar registros de datos en varias máquinas y es el enfoque de MongoDB para satisfacer las demandas del crecimiento de datos. A medida que aumenta el tamaño de los datos, una sola máquina puede no ser suficiente para almacenar los datos ni proporcionar un rendimiento aceptable de lectura y escritura. Sharding resuelve el problema con la escala horizontal. Con el fragmentación, agrega más máquinas para admitir el crecimiento de datos y las demandas de las operaciones de lectura y escritura.

¿Por qué fragmentar?

- En la replicación, todas las escrituras van al nodo maestro
- Las consultas sensibles a la latencia aún se dirigen al maestro
- El conjunto de réplica única tiene una limitación de 12 nodos
- La memoria no puede ser lo suficientemente grande cuando el conjunto de datos activo es grande
- El disco local no es lo suficientemente grande
- El escalado vertical es demasiado caro.

Fragmento en MongoDB

El siguiente diagrama muestra el fragmentación en MongoDB utilizando el clúster fragmentado.



En el siguiente diagrama, hay tres componentes principales:

- **Fragmentos** : los fragmentos se utilizan para almacenar datos. Proporcionan alta disponibilidad y consistencia de datos. En el entorno de producción, cada fragmento es un conjunto de réplicas separado.
- **Servidores de** configuración: los servidores de configuración almacenan los metadatos del clúster. Estos datos contienen una asignación del conjunto de datos del clúster a los fragmentos. El enrutador de consultas utiliza estos metadatos para dirigir operaciones a fragmentos específicos. En el entorno de producción, los clústeres fragmentados tienen exactamente 3 servidores de configuración.

- **Enrutadores de** consulta: los enrutadores de consulta son básicamente instancias de mongo, interactúan con las aplicaciones cliente y dirigen las operaciones al fragmento apropiado. El enrutador de consultas procesa y dirige las operaciones a fragmentos y luego devuelve los resultados a los clientes. Un clúster fragmentado puede contener más de un enrutador de consulta para dividir la carga de solicitud del cliente. Un cliente envía solicitudes a un enrutador de consultas. En general, un clúster fragmentado tiene muchos enrutadores de consulta.

MongoDB - Crear copia de seguridad

En este capítulo, veremos cómo crear una copia de seguridad en MongoDB.

Volcado de datos de MongoDB

Para crear una copia de seguridad de la base de datos en MongoDB, debe usar el comando **mongodump** . Este comando volcará todos los datos de su servidor en el directorio de volcado. Hay muchas opciones disponibles por las cuales puede limitar la cantidad de datos o crear copias de seguridad de su servidor remoto.

Sintaxis

La sintaxis básica del comando **mongodump** es la siguiente:

```
>mongodump
```

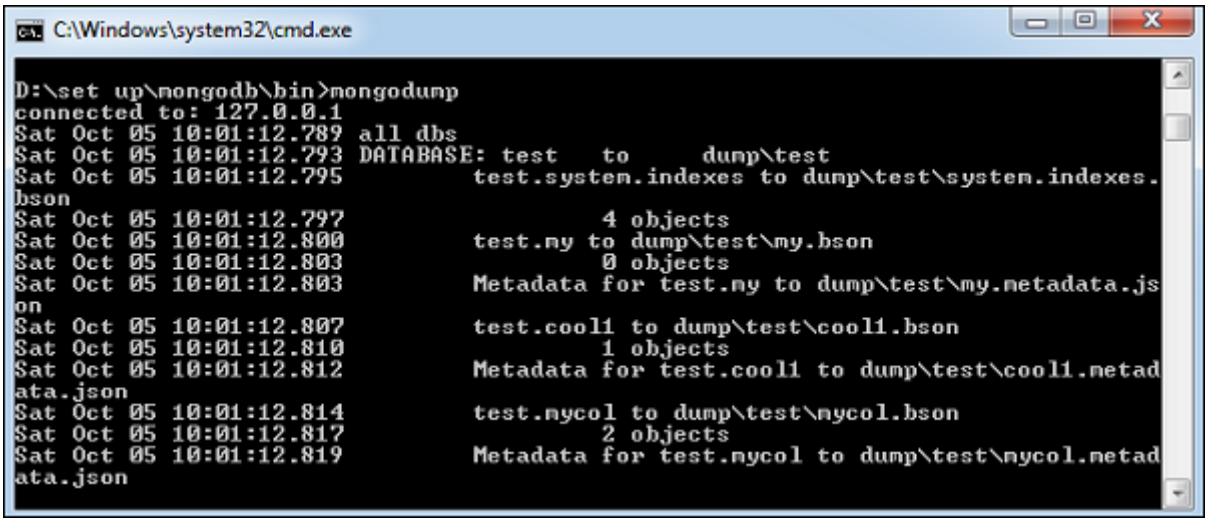
Ejemplo

Inicie su servidor mongod. Suponiendo que su servidor mongod se está ejecutando en el host local y el puerto 27017, abra un símbolo del sistema y vaya al directorio bin de su instancia mongodb y escriba el comando **mongodump**

Considere que la colección mycol tiene los siguientes datos.

```
>mongodump
```

El comando se conectará al servidor que se ejecuta en **127.0.0.1** y el puerto **27017** y respaldará todos los datos del servidor al directorio **/ bin / dump /** . A continuación se muestra el resultado del comando:



A continuación hay una lista de opciones disponibles que se pueden usar con el comando **mongodump** .

Sintaxis	Descripción	Ejemplo
mongodump --host HOST_NAME --port PORT_NUMBER	Este comando respaldará todas las bases de datos de la instancia mongod especificada.	mongodump --host postparaprogramadores.com --port 27017
mongodump --dbpath DB_PATH --out BACKUP_DIRECTORY	Este comando respaldará solo la base de datos especificada en la ruta especificada.	mongodump --dbpath / data / db / --out / data / backup /
mongodump --collection COLLECTION --db DB_NAME	Este comando respaldará solo la colección especificada de la base de datos especificada.	mongodump --collection mycol --db test

Restaurar datos

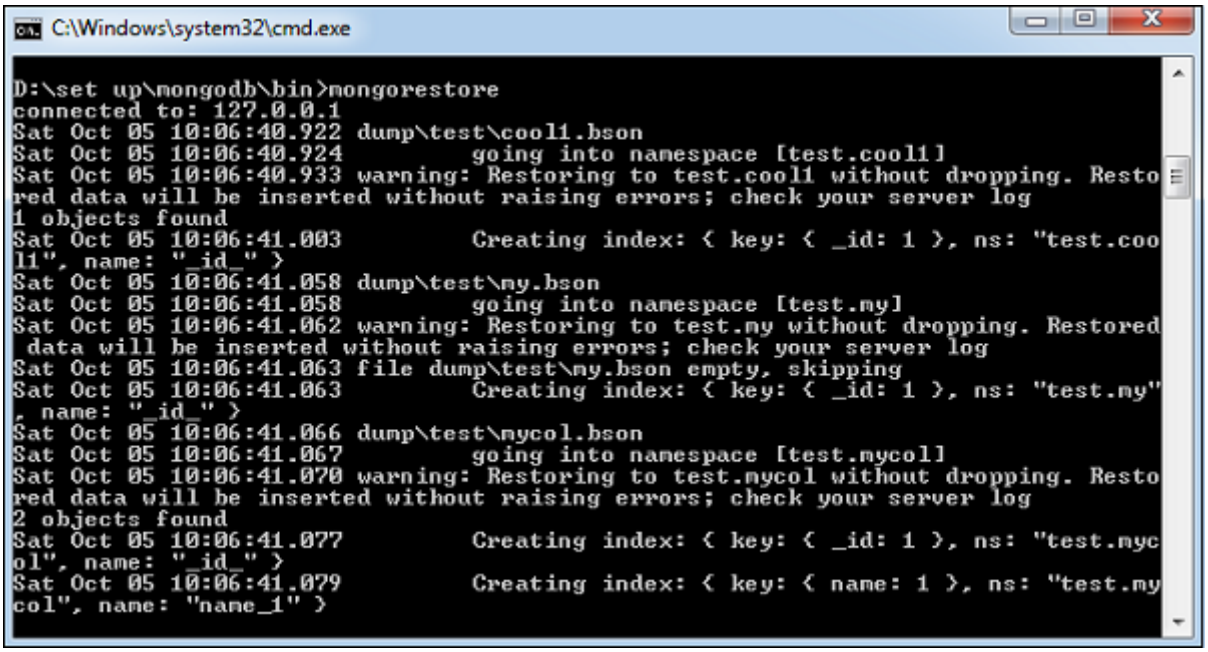
Para restaurar los datos de la copia de seguridad, se **utiliza el** comando **mongorestore** de MongoDB . Este comando restaura todos los datos del directorio de respaldo.

Sintaxis

La sintaxis básica del comando **mongorestore** es:

```
>mongorestore
```

A continuación se muestra el resultado del comando:



MongoDB - Implementación

Cuando esté preparando una implementación de MongoDB, debe tratar de comprender cómo su aplicación se mantendrá en producción. Es una buena idea desarrollar un enfoque coherente y repetible para administrar su entorno de implementación para que pueda minimizar cualquier sorpresa una vez que esté en producción.

El mejor enfoque incorpora la creación de prototipos de su configuración, la realización de pruebas de carga, el monitoreo de métricas clave y el uso de esa información para escalar su configuración. La parte clave del enfoque es monitorear proactivamente todo su sistema; esto lo ayudará a comprender cómo se mantendrá su sistema de producción antes de la implementación y determinar dónde necesitará agregar capacidad. Tener una idea de los picos potenciales en el uso de su memoria, por ejemplo, podría ayudar a apagar un incendio de bloqueo de escritura antes de que comience.

Para monitorear su implementación, MongoDB proporciona algunos de los siguientes comandos:

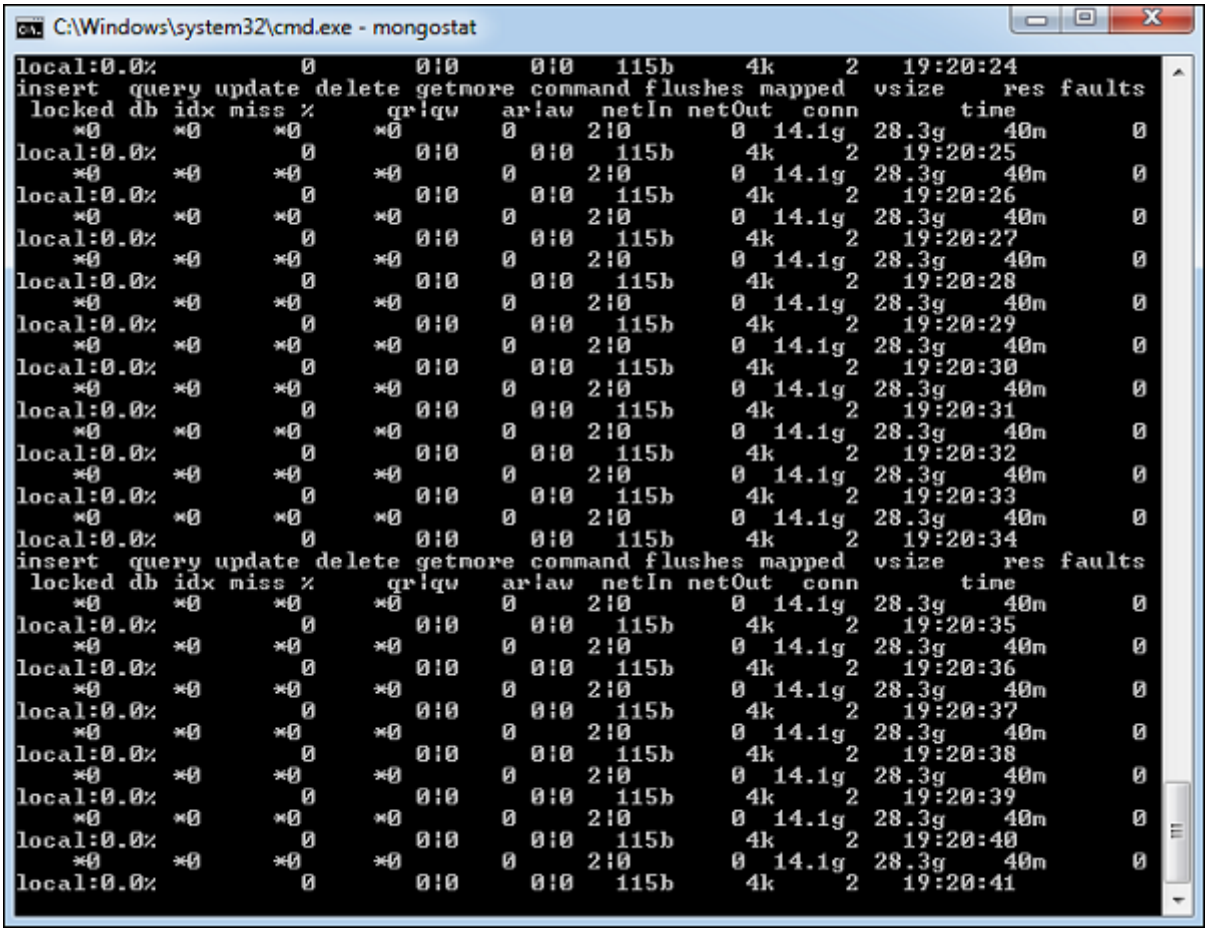
mongostat

Este comando verifica el estado de todas las instancias de mongod en ejecución y muestra los contadores de las operaciones de la base de datos. Estos contadores incluyen inserciones, consultas, actualizaciones, eliminaciones y cursores. El comando también se muestra cuando estás alcanzando fallas de página y muestra tu porcentaje de bloqueo. Esto significa que se está quedando sin memoria, está llegando a la capacidad de escritura o tiene algún problema de rendimiento.

Para ejecutar el comando, inicie su instancia mongod. En otro símbolo del sistema, vaya al directorio **bin** de su instalación de mongodb y escriba **mongostat** .

```
D:\set up\mongodb\bin>mongostat
```

A continuación se muestra el resultado del comando:



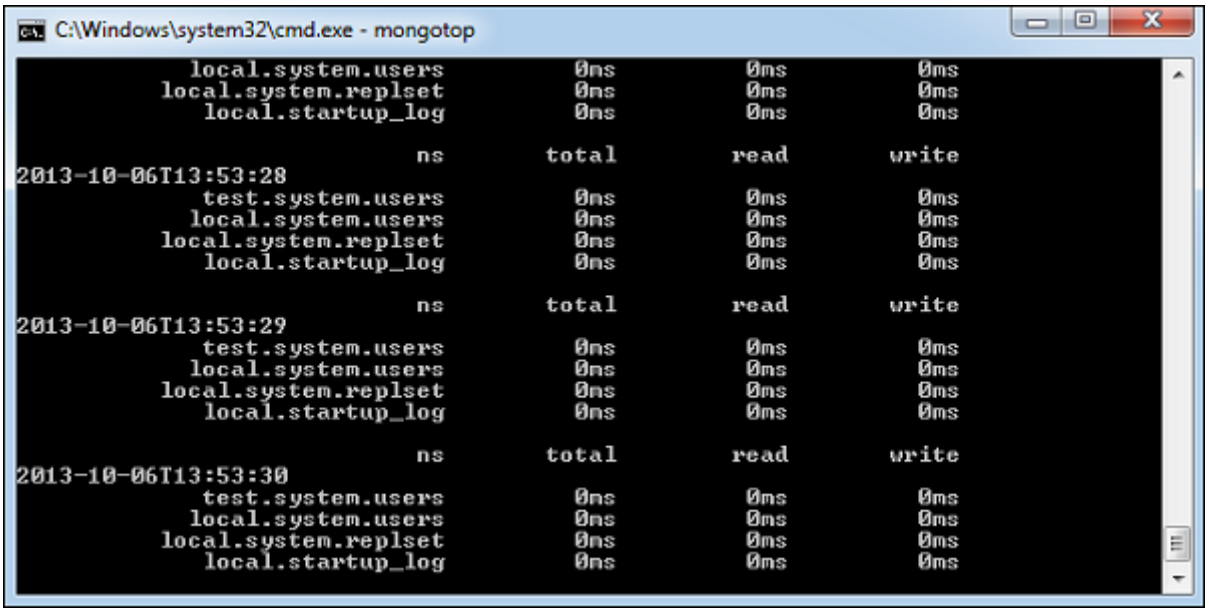
mongotop

Este comando rastrea e informa la actividad de lectura y escritura de la instancia de MongoDB en una colección. De forma predeterminada, **mongotop** devuelve información en cada segundo, que puede cambiar en consecuencia. Debe verificar que esta actividad de lectura y escritura coincida con la intención de su aplicación, y no está disparando demasiadas escrituras a la base de datos a la vez, leyendo con demasiada frecuencia desde un disco, o está excediendo el tamaño de su conjunto de trabajo.

Para ejecutar el comando, inicie su instancia mongod. En otro símbolo del sistema, vaya al directorio **bin** de su instalación de mongodb y escriba **mongotop** .

```
D:\set up\mongodb\bin>mongotop
```

A continuación se muestra el resultado del comando:



Para cambiar el comando **mongotop** para que devuelva información con menos frecuencia, especifique un número específico después del comando mongotop.

```
D:\set up\mongodb\bin>mongotop 30
```

El ejemplo anterior devolverá valores cada 30 segundos.

Además de las herramientas MongoDB, 10gen proporciona un servicio de monitoreo alojado gratuito, MongoDB Management Service (MMS), que proporciona un panel de control y le brinda una vista de las métricas de todo su clúster.

MongoDB - Java

En este capítulo, aprenderemos cómo configurar el controlador JDBC de MongoDB.

Instalación

Antes de comenzar a usar MongoDB en sus programas Java, debe asegurarse de tener el controlador JDBC MongoDB y Java configurados en la máquina. Puede consultar el tutorial de Java para la instalación de Java en su máquina. Ahora, veamos cómo configurar el controlador JDBC de MongoDB.

- Necesita descargar el jar desde la ruta Descargar mongo.jar . Asegúrese de descargar la última versión del mismo.
- Debes incluir el mongo.jar en tu classpath.

Conectarse a la base de datos

Para conectar la base de datos, debe especificar el nombre de la base de datos, si la base de datos no existe, MongoDB la crea automáticamente.

El siguiente es el fragmento de código para conectarse a la base de datos:

```
import com.mongodb.client.MongoDatabase;
import com.mongodb.MongoClient;
import com.mongodb.MongoCredential;

public class ConnectToDB {

   public static void main( String args[] ) {

      // Creating a Mongo client
      MongoClient mongo = new MongoClient( "localhost" ,
27017 );

      // Creating Credentials
      MongoCredential credential;
      credential =
MongoCredential.createCredential("sampleUser", "myDb",
         "password".toCharArray());
      System.out.println("Connected to the database
successfully");

      // Accessing the database
      MongoDatabase database = mongo.getDatabase("myDb");
      System.out.println("Credentials ::"+ credential);
   }
}
```

Ahora, compilemos y ejecutemos el programa anterior para crear nuestra base de datos myDb como se muestra a continuación.

```
$javac ConnectToDB.java
$java ConnectToDB
```

Al ejecutar, el programa anterior le ofrece el siguiente resultado.

```
Connected to the database successfully
Credentials ::MongoCredential{
   mechanism = null,
   userName = 'sampleUser',
   source = 'myDb',
   password = <hidden>,
   mechanismProperties = {}
}
```

Crear una colección

Para crear una colección, se **utiliza el** método **createCollection ()** de la clase **com.mongodb.client.MongoDatabase** .

El siguiente es el fragmento de código para crear una colección:

```
import com.mongodb.client.MongoDatabase;
import com.mongodb.MongoClient;
import com.mongodb.MongoCredential;
```

```
public class CreatingCollection {

   public static void main( String args[] ) {

      // Creating a Mongo client
      MongoClient mongo = new MongoClient( "localhost" ,
27017 );

      // Creating Credentials
      MongoCredential credential;
      credential =
MongoCredential.createCredential("sampleUser", "myDb",
         "password".toCharArray());
      System.out.println("Connected to the database
successfully");

      //Accessing the database
      MongoDatabase database = mongo.getDatabase("myDb");

      //Creating a collection
      database.createCollection("sampleCollection");
      System.out.println("Collection created successfully");
   }
}
```

Al compilar, el programa anterior le da el siguiente resultado:

```
Connected to the database successfully
Collection created successfully
```

Obteniendo / Seleccionando una Colección

Para obtener / seleccionar una colección de la base de datos, se **utiliza el** método **getCollection** **()** de la clase **com.mongodb.client.MongoDatabase** .

El siguiente es el programa para obtener / seleccionar una colección:

```
import com.mongodb.client.MongoCollection;
import com.mongodb.client.MongoDatabase;

import org.bson.Document;
import com.mongodb.MongoClient;
import com.mongodb.MongoCredential;

public class selectingCollection {

   public static void main( String args[] ) {

      // Creating a Mongo client
      MongoClient mongo = new MongoClient( "localhost" ,
27017 );

      // Creating Credentials
```

```
      MongoCredential credential;
      credential =
MongoCredential.createCredential("sampleUser", "myDb",
         "password".toCharArray());
      System.out.println("Connected to the database
successfully");

      // Accessing the database
      MongoDatabase database = mongo.getDatabase("myDb");

      // Creating a collection
      System.out.println("Collection created successfully");

      // Retieving a collection
      MongoCollection<Document> collection =
database.getCollection("myCollection");
      System.out.println("Collection myCollection selected
successfully");
   }
}
```

Al compilar, el programa anterior le da el siguiente resultado:

```
Connected to the database successfully
Collection created successfully
Collection myCollection selected successfully
```

Insertar un documento

Para insertar un documento en MongoDB, se utiliza el método **insert ()** de la clase **com.mongodb.client.MongoCollection** .

El siguiente es el fragmento de código para insertar un documento:

```
import com.mongodb.client.MongoCollection;
import com.mongodb.client.MongoDatabase;

import org.bson.Document;
import com.mongodb.MongoClient;
import com.mongodb.MongoCredential;

public class InsertingDocument {

   public static void main( String args[] ) {

      // Creating a Mongo client
      MongoClient mongo = new MongoClient( "localhost" ,
27017 );

      // Creating Credentials
      MongoCredential credential;
      credential =
MongoCredential.createCredential("sampleUser", "myDb",
         "password".toCharArray());
```

```
      System.out.println("Connected to the database
successfully");

      // Accessing the database
      MongoDatabase database = mongo.getDatabase("myDb");

      // Retrieving a collection
      MongoCollection<Document> collection =
database.getCollection("sampleCollection");
      System.out.println("Collection sampleCollection
selected successfully");

      Document document = new Document("title", "MongoDB")
      .append("id", 1)
      .append("description", "database")
      .append("likes", 100)
      .append("url",
"http://www.postparaprogramadores.com/mongodb/")
      .append("by", "postparaprogramadores");
      collection.insertOne(document);
      System.out.println("Document inserted successfully");
   }
}
```

Al compilar, el programa anterior le da el siguiente resultado:

```
Connected to the database successfully
Collection sampleCollection selected successfully
Document inserted successfully
```

Recuperar todos los documentos

Para seleccionar todos los documentos de la colección, se utiliza el método **find ()** de la clase **com.mongodb.client.MongoCollection** . Este método devuelve un cursor, por lo que debe iterar este cursor.

El siguiente es el programa para seleccionar todos los documentos:

```
import com.mongodb.client.FindIterable;
import com.mongodb.client.MongoCollection;
import com.mongodb.client.MongoDatabase;

import java.util.Iterator;
import org.bson.Document;
import com.mongodb.MongoClient;
import com.mongodb.MongoCredential;

public class RetrievingAllDocuments {

   public static void main( String args[] ) {

      // Creating a Mongo client
      MongoClient mongo = new MongoClient( "localhost" ,
27017 );
```

```
      // Creating Credentials
      MongoCredential credential;
      credential =
MongoCredential.createCredential("sampleUser", "myDb",
         "password".toCharArray());
      System.out.println("Connected to the database
successfully");

      // Accessing the database
      MongoDatabase database = mongo.getDatabase("myDb");

      // Retrieving a collection
      MongoCollection<Document> collection =
database.getCollection("sampleCollection");
      System.out.println("Collection sampleCollection
selected successfully");

      // Getting the iterable object
      FindIterable<Document> iterDoc = collection.find();
      int i = 1;

      // Getting the iterator
      Iterator it = iterDoc.iterator();

      while (it.hasNext()) {
         System.out.println(it.next());
      i++;
      }
   }
}
```

Al compilar, el programa anterior le da el siguiente resultado:

```
Document{{
   _id = 5967745223993a32646baab8,
   title = MongoDB,
   id = 1,
   description = database,
   likes = 100,
   url = http://www.postparaprogramadores.com/mongodb/, by =
postparaprogramadores
}}
Document{{
   _id = 7452239959673a32646baab8,
   title = RethinkDB,
   id = 2,
   description = database,
   likes = 200,
   url = http://www.postparaprogramadores.com/rethinkdb/, by
= postparaprogramadores
}}
```

Actualizar documento

Para actualizar un documento de la colección, se **utiliza el** método **updateOne ()** de la clase **com.mongodb.client.MongoCollection** .

El siguiente es el programa para seleccionar el primer documento:

```
import com.mongodb.client.FindIterable;
import com.mongodb.client.MongoCollection;
import com.mongodb.client.MongoDatabase;
import com.mongodb.client.model.Filters;
import com.mongodb.client.model.Updates;

import java.util.Iterator;
import org.bson.Document;
import com.mongodb.MongoClient;
import com.mongodb.MongoCredential;

public class UpdatingDocuments {

   public static void main( String args[] ) {

      // Creating a Mongo client
      MongoClient mongo = new MongoClient( "localhost" ,
27017 );

      // Creating Credentials
      MongoCredential credential;
      credential =
MongoCredential.createCredential("sampleUser", "myDb",
         "password".toCharArray());
      System.out.println("Connected to the database
successfully");

      // Accessing the database
      MongoDatabase database = mongo.getDatabase("myDb");

      // Retrieving a collection
      MongoCollection<Document> collection =
database.getCollection("sampleCollection");
      System.out.println("Collection myCollection selected
successfully");

      collection.updateOne(Filters.eq("id", 1),
Updates.set("likes", 150));
      System.out.println("Document update successfully...");

      // Retrieving the documents after updation
      // Getting the iterable object
      FindIterable<Document> iterDoc = collection.find();
      int i = 1;

      // Getting the iterator
```

```
      Iterator it = iterDoc.iterator();

      while (it.hasNext()) {
         System.out.println(it.next());
         i++;
      }
   }
}
```

Al compilar, el programa anterior le da el siguiente resultado:

```
Document update successfully...
Document {{
   _id = 5967745223993a32646baab8,
   title = MongoDB,
   id = 1,
   description = database,
   likes = 150,
   url = http://www.postparaprogramadores.com/mongodb/, by =
postparaprogramadores
}}
```

Eliminar un documento

Para eliminar un documento de la colección, debe usar el método **deleteOne ()** de la clase **com.mongodb.client.MongoCollection** .

El siguiente es el programa para eliminar un documento:

```
import com.mongodb.client.FindIterable;
import com.mongodb.client.MongoCollection;
import com.mongodb.client.MongoDatabase;
import com.mongodb.client.model.Filters;

import java.util.Iterator;
import org.bson.Document;
import com.mongodb.MongoClient;
import com.mongodb.MongoCredential;

public class DeletingDocuments {

   public static void main( String args[] ) {

      // Creating a Mongo client
      MongoClient mongo = new MongoClient( "localhost" ,
27017 );

      // Creating Credentials
      MongoCredential credential;
      credential =
MongoCredential.createCredential("sampleUser", "myDb",
         "password".toCharArray());
      System.out.println("Connected to the database
successfully");
```

```
      // Accessing the database
      MongoDatabase database = mongo.getDatabase("myDb");

      // Retrieving a collection
      MongoCollection<Document> collection =
database.getCollection("sampleCollection");
      System.out.println("Collection sampleCollection
selected successfully");

      // Deleting the documents
      collection.deleteOne(Filters.eq("id", 1));
      System.out.println("Document deleted successfully...");

      // Retrieving the documents after updation
      // Getting the iterable object
      FindIterable<Document> iterDoc = collection.find();
      int i = 1;

      // Getting the iterator
      Iterator it = iterDoc.iterator();

      while (it.hasNext()) {
         System.out.println("Inserted Document: "+i);
         System.out.println(it.next());
         i++;
      }
   }
}
```

Al compilar, el programa anterior le da el siguiente resultado:

```
Connected to the database successfully
Collection sampleCollection selected successfully
Document deleted successfully...
```

Dejar caer una colección

Para descartar una colección de una base de datos, debe usar el método **drop ()** de la clase **com.mongodb.client.MongoCollection** .

El siguiente es el programa para eliminar una colección:

```
import com.mongodb.client.MongoCollection;
import com.mongodb.client.MongoDatabase;

import org.bson.Document;
import com.mongodb.MongoClient;
import com.mongodb.MongoCredential;

public class DropingCollection {

   public static void main( String args[] ) {
```

```
      // Creating a Mongo client
      MongoClient mongo = new MongoClient( "localhost" ,
27017 );

      // Creating Credentials
      MongoCredential credential;
      credential =
MongoCredential.createCredential("sampleUser", "myDb",
         "password".toCharArray());
      System.out.println("Connected to the database
successfully");

      // Accessing the database
      MongoDatabase database = mongo.getDatabase("myDb");

      // Creating a collection
      System.out.println("Collections created successfully");

      // Retieving a collection
      MongoCollection<Document> collection =
database.getCollection("sampleCollection");

      // Dropping a Collection
      collection.drop();
      System.out.println("Collection dropped successfully");
   }
}
```

Al compilar, el programa anterior le da el siguiente resultado:

```
Connected to the database successfully
Collection sampleCollection selected successfully
Collection dropped successfully
```

Listado de todas las colecciones

Para enumerar todas las colecciones en una base de datos, debe usar el método **listCollectionNames** **()** de la clase **com.mongodb.client.MongoDatabase** .

El siguiente es el programa para enumerar todas las colecciones de una base de datos:

```
import com.mongodb.client.MongoDatabase;
import com.mongodb.MongoClient;
import com.mongodb.MongoCredential;

public class ListOfCollection {

   public static void main( String args[] ) {

      // Creating a Mongo client
```

```
      MongoClient mongo = new MongoClient( "localhost" ,
27017 );

      // Creating Credentials
      MongoCredential credential;
      credential =
MongoCredential.createCredential("sampleUser", "myDb",
         "password".toCharArray());

      System.out.println("Connected to the database
successfully");

      // Accessing the database
      MongoDatabase database = mongo.getDatabase("myDb");
      System.out.println("Collection created successfully");
      for (String name : database.listCollectionNames()) {
         System.out.println(name);
      }
   }
}
```

Al compilar, el programa anterior le da el siguiente resultado:

```
Connected to the database successfully
Collection created successfully
myCollection
myCollection1
myCollection5
```

Los métodos restantes de MongoDB **save (), limit (), skip (), sort ()** etc. funcionan igual que se explica en el tutorial posterior.

MongoDB - PHP

Para usar MongoDB con PHP, debe usar el controlador MongoDB PHP. Descargue el controlador desde la url Descargar PHP Driver . Asegúrese de descargar la última versión del mismo. Ahora descomprima el archivo comprimido y coloque php_mongo.dll en su directorio de extensiones PHP ("ext" por defecto) y agregue la siguiente línea a su archivo php.ini:

```
extension = php_mongo.dll
```

Haga una conexión y seleccione una base de datos

Para realizar una conexión, debe especificar el nombre de la base de datos; si la base de datos no existe, MongoDB la crea automáticamente.

El siguiente es el fragmento de código para conectarse a la base de datos:

```
<?php
   // connect to mongodb
   $m = new MongoClient();

   echo "Connection to database successfully";
```

```
   // select a database
   $db = $m->mydb;

   echo "Database mydb selected";
?>
```

Cuando se ejecuta el programa, producirá el siguiente resultado:

```
Connection to database successfully
Database mydb selected
```

Crear una colección

El siguiente es el fragmento de código para crear una colección:

```
<?php
   // connect to mongodb
   $m = new MongoClient();
   echo "Connection to database successfully";

   // select a database
   $db = $m->mydb;
   echo "Database mydb selected";
   $collection = $db->createCollection("mycol");
   echo "Collection created succsessfully";
?>
```

Cuando se ejecuta el programa, producirá el siguiente resultado:

```
Connection to database successfully
Database mydb selected
Collection created succsessfully
```

Insertar un documento

Para insertar un documento en MongoDB, se utiliza el método **insert ()** .

El siguiente es el fragmento de código para insertar un documento:

```
<?php
   // connect to mongodb
   $m = new MongoClient();
   echo "Connection to database successfully";

   // select a database
   $db = $m->mydb;
   echo "Database mydb selected";
   $collection = $db->mycol;
   echo "Collection selected succsessfully";

   $document = array(
      "title" => "MongoDB",
      "description" => "database",
      "likes" => 100,
```

```
      "url" =>
"http://www.postparaprogramadores.com/mongodb/",
      "by" => "postparaprogramadores"
   );

   $collection->insert($document);
   echo "Document inserted successfully";
?>
```

Cuando se ejecuta el programa, producirá el siguiente resultado:

```
Connection to database successfully
Database mydb selected
Collection selected succsessfully
Document inserted successfully
```

Encuentra todos los documentos

Para seleccionar todos los documentos de la colección, se utiliza el método find ().

El siguiente es el fragmento de código para seleccionar todos los documentos:

```
<?php
   // connect to mongodb
   $m = new MongoClient();
   echo "Connection to database successfully";

   // select a database
   $db = $m->mydb;
   echo "Database mydb selected";
   $collection = $db->mycol;
   echo "Collection selected succsessfully";

   $cursor = $collection->find();
   // iterate cursor to display title of documents

   foreach ($cursor as $document) {
      echo $document["title"] . "\n";
   }
?>
```

Cuando se ejecuta el programa, producirá el siguiente resultado:

```
Connection to database successfully
Database mydb selected
Collection selected succsessfully {
   "title": "MongoDB"
}
```

Actualizar un documento

Para actualizar un documento, debe usar el método update ().

En el siguiente ejemplo, actualizaremos el título del documento insertado a **MongoDB Tutorial** . El siguiente es el fragmento de código para actualizar un documento:

```
<?php
   // connect to mongodb
   $m = new MongoClient();
   echo "Connection to database successfully";

   // select a database
   $db = $m->mydb;
   echo "Database mydb selected";
   $collection = $db->mycol;
   echo "Collection selected succsessfully";

   // now update the document
   $collection->update(array("title"=>"MongoDB"),
      array('$set'=>array("title"=>"MongoDB Tutorial")));
   echo "Document updated successfully";

   // now display the updated document
   $cursor = $collection->find();

   // iterate cursor to display title of documents
   echo "Updated document";

   foreach ($cursor as $document) {
      echo $document["title"] . "\n";
   }
?>
```

Cuando se ejecuta el programa, producirá el siguiente resultado:

```
Connection to database successfully
Database mydb selected
Collection selected succsessfully
Document updated successfully
Updated document {
   "title": "MongoDB Tutorial"
}
```

Eliminar un documento

Para eliminar un documento, debe usar el método remove ().

En el siguiente ejemplo, eliminaremos los documentos que tengan el título **MongoDB Tutorial** . El siguiente es el fragmento de código para eliminar un documento:

```
<?php
   // connect to mongodb
   $m = new MongoClient();
   echo "Connection to database successfully";
```

```
   // select a database
   $db = $m->mydb;
   echo "Database mydb selected";
   $collection = $db->mycol;
   echo "Collection selected succsessfully";

   // now remove the document
   $collection->remove(array("title"=>"MongoDB
Tutorial"),false);
   echo "Documents deleted successfully";

   // now display the available documents
   $cursor = $collection->find();

   // iterate cursor to display title of documents
   echo "Updated document";

   foreach ($cursor as $document) {
      echo $document["title"] . "\n";
   }
?>
```

Cuando se ejecuta el programa, producirá el siguiente resultado:

```
Connection to database successfully
Database mydb selected
Collection selected succsessfully
Documents deleted successfully
```

En el ejemplo anterior, el segundo parámetro es de tipo booleano y se usa para el campo **justOne** del método **remove ()** .

Los métodos restantes de MongoDB **findOne (), save (), limit (), skip (), sort ()** etc. funcionan igual que se explicó anteriormente.

MongoDB - Relaciones

Las relaciones en MongoDB representan cómo varios documentos están relacionados lógicamente entre sí. Las relaciones se pueden modelar mediante enfoques **integrados** y **referenciados** . Dichas relaciones pueden ser 1: 1, 1: N, N: 1 o N: N.

Consideremos el caso del almacenamiento de direcciones para usuarios. Por lo tanto, un usuario puede tener varias direcciones, lo que hace que esta sea una relación 1: N.

A continuación se muestra la estructura del documento de muestra del documento del **usuario** :

```
{
   "_id":ObjectId("52ffc33cd85242f436000001"),
   "name": "Tom Hanks",
   "contact": "987654321",
   "dob": "01-01-1991"
}
```

A continuación se muestra la estructura del documento de muestra del documento de **dirección** :

```
{
   "_id":ObjectId("52ffc4a5d85242602e000000"),
   "building": "22 A, Indiana Apt",
   "pincode": 123456,
   "city": "Los Angeles",
   "state": "California"
}
```

Modelado de relaciones integradas

En el enfoque incrustado, incrustaremos el documento de dirección dentro del documento del usuario.

```
{
   "_id":ObjectId("52ffc33cd85242f436000001"),
   "contact": "987654321",
   "dob": "01-01-1991",
   "name": "Tom Benzamin",
   "address": [
      {
         "building": "22 A, Indiana Apt",
         "pincode": 123456,
         "city": "Los Angeles",
         "state": "California"
      },
      {
         "building": "170 A, Acropolis Apt",
         "pincode": 456789,
         "city": "Chicago",
         "state": "Illinois"
      }
   ]
}
```

Este enfoque mantiene todos los datos relacionados en un solo documento, lo que facilita su recuperación y mantenimiento. Todo el documento se puede recuperar en una sola consulta, como:

```
>db.users.findOne({"name":"Tom Benzamin"},{"address":1})
```

Tenga en cuenta que en la consulta anterior, **db** y los **usuarios** son la base de datos y la colección, respectivamente.

El inconveniente es que si el documento incrustado sigue creciendo demasiado, puede afectar el rendimiento de lectura / escritura.

Modelado de relaciones referenciadas

Este es el enfoque del diseño de una relación normalizada. En este enfoque, tanto el documento de usuario como el de dirección se mantendrán por

separado, pero el documento de usuario contendrá un campo que hará referencia al campo de **identificación** del documento de dirección .

```
{
   "_id":ObjectId("52ffc33cd85242f436000001"),
   "contact": "987654321",
   "dob": "01-01-1991",
   "name": "Tom Benzamin",
   "address_ids": [
      ObjectId("52ffc4a5d85242602e000000"),
      ObjectId("52ffc4a5d85242602e000001")
   ]
}
```

Como se muestra arriba, el documento del usuario contiene el campo de matriz **address_ids** que contiene ObjectIds de las direcciones correspondientes. Usando estos ObjectIds, podemos consultar los documentos de dirección y obtener detalles de la dirección desde allí. Con este enfoque, necesitaremos dos consultas: primero para obtener los campos **address_ids** del documento del **usuario** y segundo para obtener estas direcciones de la colección de **direcciones** .

```
>var result = db.users.findOne({"name":"Tom
Benzamin"},{"address_ids":1})
>var addresses =
db.address.find({"_id":{"$in":result["address_ids"]}})
```

MongoDB - Referencias de bases de datos

Como se vio en el último capítulo de las relaciones de MongoDB, para implementar una estructura de base de datos normalizada en MongoDB, utilizamos el concepto de **Relaciones referenciadas,** también referidas como **Referencias manuales,** en las que almacenamos manualmente la identificación del documento referenciado dentro de otro documento. Sin embargo, en los casos en que un documento contiene referencias de diferentes colecciones, podemos usar **MongoDB DBRefs** .

DBRefs vs Referencias manuales

Como escenario de ejemplo, donde usaríamos DBRefs en lugar de referencias manuales, considere una base de datos donde almacenamos diferentes tipos de direcciones (hogar, oficina, correo, etc.) en diferentes colecciones (address_home, address_office, address_mailing, etc.). Ahora, cuando el documento de una colección de **usuarios** hace referencia a una dirección, también necesita especificar en qué colección buscar según el tipo de dirección. En tales escenarios donde un documento hace referencia a documentos de muchas colecciones, deberíamos usar DBRefs.

Usando DBRefs

Hay tres campos en DBRefs:

- **$ ref** - Este campo especifica la colección del documento referenciado

- **$ id** : este campo especifica el campo _id del documento al que se hace referencia
- **$ db** : este es un campo opcional y contiene el nombre de la base de datos en la que se encuentra el documento referenciado

Considere un documento de usuario de muestra con una **dirección de** campo DBRef como se muestra en el fragmento de código:

```
{
   "_id":ObjectId("53402597d852426020000002"),
   "address": {
   "$ref": "address_home",
   "$id": ObjectId("534009e4d852427820000002"),
   "$db": "postparaprogramadores"},
   "contact": "987654321",
   "dob": "01-01-1991",
   "name": "Tom Benzamin"
}
```

El campo de **dirección** DBRef aquí especifica que el documento de dirección referenciado se encuentra en la colección **address_home** en la base de datos de **postparaprogramadores** y tiene una identificación de 534009e4d852427820000002.

El siguiente código busca dinámicamente en la colección especificada por el parámetro **$ ref** (**address_home** en nuestro caso) para un documento con id como se especifica por el parámetro **$ id** en DBRef.

```
>var user = db.users.findOne({"name":"Tom Benzamin"})
>var dbRef = user.address
>db[dbRef.$ref].findOne({"_id":(dbRef.$id)})
```

El código anterior devuelve el siguiente documento de dirección presente en la colección **address_home** :

```
{
   "_id" : ObjectId("534009e4d852427820000002"),
   "building" : "22 A, Indiana Apt",
   "pincode" : 123456,
   "city" : "Los Angeles",
   "state" : "California"
}
```

MongoDB - Consultas cubiertas

En este capítulo, aprenderemos sobre las consultas cubiertas.

¿Qué es una consulta cubierta?

Según la documentación oficial de MongoDB, una consulta cubierta es una consulta en la que:

- Todos los campos en la consulta son parte de un índice.
- Todos los campos devueltos en la consulta están en el mismo índice.

Dado que todos los campos presentes en la consulta son parte de un índice, MongoDB coincide con las condiciones de la consulta y devuelve el resultado utilizando el mismo índice sin mirar realmente dentro de los documentos. Dado que los índices están presentes en la RAM, la obtención de datos de los índices es mucho más rápida en comparación con la obtención de datos escaneando documentos.

Usar consultas cubiertas

Para probar las consultas cubiertas, considere el siguiente documento en la colección de **usuarios** :

```
{
   "_id": ObjectId("53402597d852426020000002"),
   "contact": "987654321",
   "dob": "01-01-1991",
   "gender": "M",
   "name": "Tom Benzamin",
   "user_name": "tombenzamin"
}
```

Primero crearemos un índice compuesto para la colección de **usuarios** en los campos **género** y **nombre_usuario** utilizando la siguiente consulta:

```
>db.users.ensureIndex({gender:1,user_name:1})
```

Ahora, este índice cubrirá la siguiente consulta:

```
>db.users.find({gender:"M"},{user_name:1,_id:0})
```

Es decir que para la consulta anterior, MongoDB no iría a buscar en los documentos de la base de datos. En cambio, obtendría los datos requeridos de los datos indexados, que es muy rápido.

Como nuestro índice no incluye el campo **_id** , lo hemos excluido explícitamente del conjunto de resultados de nuestra consulta, ya que MongoDB por defecto devuelve el campo _id en cada consulta. Por lo tanto, la siguiente consulta no se habría cubierto dentro del índice creado anteriormente:

```
>db.users.find({gender:"M"},{user_name:1})
```

Por último, recuerde que un índice no puede cubrir una consulta si:

- Cualquiera de los campos indexados es una matriz
- Cualquiera de los campos indexados es un subdocumento

MongoDB - Análisis de consultas

Analizar consultas es un aspecto muy importante para medir la efectividad de la base de datos y el diseño de indexación. Aprenderemos sobre las consultas de **$ explica** y **$ sugerencia de** uso frecuente .

Usando $ explicar

El operador **$ explicar** proporciona información sobre la consulta, los índices utilizados en una consulta y otras estadísticas. Es muy útil al analizar qué tan bien están optimizados sus índices.

En el último capítulo, ya habíamos creado un índice para la colección de **usuarios** en los campos **género** y **nombre_usuario** utilizando la siguiente consulta:

```
>db.users.ensureIndex({gender:1,user_name:1})
```

Ahora usaremos **$ explica** en la siguiente consulta:

```
>db.users.find({gender:"M"},{user_name:1,_id:0}).explain()
```

La consulta explicada () anterior devuelve el siguiente resultado analizado:

```
{
   "cursor" : "BtreeCursor gender_1_user_name_1",
   "isMultiKey" : false,
   "n" : 1,
   "nscannedObjects" : 0,
   "nscanned" : 1,
   "nscannedObjectsAllPlans" : 0,
   "nscannedAllPlans" : 1,
   "scanAndOrder" : false,
   "indexOnly" : true,
   "nYields" : 0,
   "nChunkSkips" : 0,
   "millis" : 0,
   "indexBounds" : {
      "gender" : [
         [
            "M",
            "M"
         ]
      ],
      "user_name" : [
         [
            {
               "$minElement" : 1
            },
            {
               "$maxElement" : 1
            }
         ]
      ]
   }
}
```

Ahora veremos los campos en este conjunto de resultados:

- El verdadero valor de **indexOnly** indica que esta consulta ha utilizado la indexación.

- El campo del **cursor** especifica el tipo de cursor utilizado. El tipo BTreeCursor indica que se utilizó un índice y también proporciona el nombre del índice utilizado. BasicCursor indica que se realizó una exploración completa sin utilizar ningún índice.
- **n** indica el número de documentos coincidentes devueltos.
- **nscannedObjects** indica el número total de documentos escaneados.
- **nscanned** indica el número total de documentos o entradas de índice escaneadas.

Usando $ pista

El operador **$ hint** obliga al optimizador de consultas a usar el índice especificado para ejecutar una consulta. Esto es particularmente útil cuando desea probar el rendimiento de una consulta con diferentes índices. Por ejemplo, la siguiente consulta especifica el índice de los campos **género** y nombre de **usuario** que se utilizarán para esta consulta:

```
>db.users.find({gender:"M"},{user_name:1,_id:0}).hint({gender
:1,user_name:1})
```

Para analizar la consulta anterior usando $ explicar -

```
>db.users.find({gender:"M"},{user_name:1,_id:0}).hint({gender
:1,user_name:1}).explain()
```

MongoDB - Operaciones atómicas

MongoDB no admite **transacciones atómicas de documentos múltiples** . Sin embargo, proporciona operaciones atómicas en un solo documento. Entonces, si un documento tiene cientos de campos, la instrucción de actualización actualizará todos los campos o ninguno, por lo tanto, mantendrá la atomicidad a nivel de documento.

Datos modelo para operaciones atómicas

El enfoque recomendado para mantener la atomicidad sería mantener toda la información relacionada, que con frecuencia se actualiza en un solo documento utilizando **documentos incrustados** . Esto aseguraría que todas las actualizaciones para un solo documento sean atómicas.

Considere el siguiente documento de productos:

```
{
   "_id":1,
   "product_name": "Samsung S3",
   "category": "mobiles",
   "product_total": 5,
   "product_available": 3,
   "product_bought_by": [
      {
         "customer": "john",
         "date": "7-Jan-2014"
```

```
      },
      {
         "customer": "mark",
         "date": "8-Jan-2014"
      }
   ]
}
```

En este documento, hemos incorporado la información del cliente que compra el producto en el campo **product_bought_by** . Ahora, cada vez que un nuevo cliente compre el producto, primero verificaremos si el producto aún está disponible utilizando el campo **product_available** . Si está disponible, reduciremos el valor del campo product_available e insertaremos el documento incrustado del nuevo cliente en el campo product_bought_by. Vamos a utilizar **findAndModify** de comandos para esta funcionalidad, ya que busca y actualiza el documento en el mismo camino.

```
>db.products.findAndModify({
   query:{_id:2,product_available:{$gt:0}},
   update:{
      $inc:{product_available:-1},
      $push:{product_bought_by:{customer:"rob",date:"9-Jan-
2014"}}
   }
})
```

Nuestro enfoque de documento incrustado y el uso de la consulta findAndModify asegura que la información de compra del producto se actualice solo si el producto está disponible. Y toda esta transacción en la misma consulta, es atómica.

En contraste con esto, considere el escenario donde podríamos haber mantenido la disponibilidad del producto y la información sobre quién lo compró, por separado. En este caso, primero verificaremos si el producto está disponible mediante la primera consulta. Luego, en la segunda consulta, actualizaremos la información de compra. Sin embargo, es posible que entre las ejecuciones de estas dos consultas, algún otro usuario haya comprado el producto y ya no esté disponible. Sin saber esto, nuestra segunda consulta actualizará la información de compra basada en el resultado de nuestra primera consulta. Esto hará que la base de datos sea inconsistente porque hemos vendido un producto que no está disponible.

MongoDB - Indexación avanzada

Considere el siguiente documento de la colección de **usuarios** :

```
{
   "address": {
      "city": "Los Angeles",
      "state": "California",
      "pincode": "123"
   },
   "tags": [
```

```
        "music",
        "cricket",
        "blogs"
    ],
    "name": "Tom Benzamin"
}
```

El documento anterior contiene un **subdocumento de dirección** y una **matriz de etiquetas** .

Campos de matriz de indexación

Supongamos que queremos buscar documentos de usuario basados en las etiquetas del usuario. Para esto, crearemos un índice en la matriz de etiquetas en la colección.

Crear un índice en una matriz a su vez crea entradas de índice separadas para cada uno de sus campos. Entonces, en nuestro caso, cuando creamos un índice en la matriz de etiquetas, se crearán índices separados para sus valores de música, cricket y blogs.

Para crear un índice en la matriz de etiquetas, use el siguiente código:

```
>db.users.ensureIndex({"tags":1})
```

Después de crear el índice, podemos buscar en el campo de etiquetas de la colección de esta manera:

```
>db.users.find({tags:"cricket"})
```

Para verificar que se utiliza la indexación adecuada, use el siguiente comando de **explicación** :

```
>db.users.find({tags:"cricket"}).explain()
```

El comando anterior dio como resultado "cursor": "BtreeCursor tags_1" que confirma que se utiliza la indexación adecuada.

Campos de subdocumento de indexación

Supongamos que queremos buscar documentos en función de los campos de ciudad, estado y código PIN. Dado que todos estos campos son parte del campo de subdocumento de dirección, crearemos un índice en todos los campos del subdocumento.

Para crear un índice en los tres campos del subdocumento, use el siguiente código:

```
>db.users.ensureIndex({"address.city":1,"address.state":1,"ad
dress.pincode":1})
```

Una vez que se crea el índice, podemos buscar cualquiera de los campos de subdocumentos utilizando este índice de la siguiente manera:

```
>db.users.find({"address.city":"Los Angeles"})
```

Recuerde que la expresión de consulta debe seguir el orden del índice especificado. Por lo tanto, el índice creado anteriormente admitiría las siguientes consultas:

```
>db.users.find({"address.city":"Los
Angeles","address.state":"California"})
```

También admitirá la siguiente consulta:

```
>db.users.find({"address.city":"LosAngeles","address.state":"
California",
   "address.pincode":"123"})
```

MongoDB - Limitaciones de indexación

En este capítulo, aprenderemos sobre las limitaciones de indexación y sus otros componentes.

Sobrecarga adicional

Cada índice ocupa algo de espacio y genera una sobrecarga en cada inserción, actualización y eliminación. Entonces, si rara vez usa su colección para operaciones de lectura, tiene sentido no usar índices.

Uso de RAM

Dado que los índices se almacenan en la RAM, debe asegurarse de que el tamaño total del índice no exceda el límite de RAM. Si el tamaño total aumenta el tamaño de la RAM, comenzará a eliminar algunos índices, lo que provocará una pérdida de rendimiento.

Limitaciones de consultas

La indexación no se puede usar en consultas que usan:

- Expresiones regulares u operadores de negación como $ nin, $ not, etc.
- Operadores aritméticos como $ mod, etc.
- $ cláusula where

Por lo tanto, siempre es recomendable verificar el uso del índice para sus consultas.

Límites clave del índice

A partir de la versión 2.6, MongoDB no creará un índice si el valor del campo de índice existente excede el límite de clave de índice.

Inserción de documentos que exceden el límite de clave de índice

MongoDB no insertará ningún documento en una colección indexada si el valor del campo indexado de este documento excede el límite de la clave de índice. Lo mismo ocurre con las utilidades mongorestore y mongoimport.

Rangos máximos

- Una colección no puede tener más de 64 índices.
- La longitud del nombre del índice no puede tener más de 125 caracteres.
- Un índice compuesto puede tener un máximo de 31 campos indexados.

MongoDB - ObjectId

Hemos estado usando Id de objeto MongoDB en todos los capítulos anteriores. En este capítulo, entenderemos la estructura de ObjectId.

Un **ObjectId** es un tipo BSON de 12 bytes que tiene la siguiente estructura:

- Los primeros 4 bytes representan los segundos desde la época de Unix
- Los siguientes 3 bytes son el identificador de la máquina.
- Los siguientes 2 bytes consisten en la **identificación** del **proceso**
- Los últimos 3 bytes son un valor de contador aleatorio

MongoDB usa ObjectIds como el valor predeterminado del campo _**id** de cada documento, que se genera durante la creación de cualquier documento. La compleja combinación de ObjectId hace que todos los campos _id sean únicos.

Crear nuevo ObjectId

Para generar un nuevo ObjectId use el siguiente código:

```
>newObjectId = ObjectId()
```

La declaración anterior devolvió la siguiente identificación generada de forma única:

```
ObjectId("5349b4ddd2781d08c09890f3")
```

En lugar de que MongoDB genere el ObjectId, también puede proporcionar una identificación de 12 bytes:

```
>myObjectId = ObjectId("5349b4ddd2781d08c09890f4")
```

Crear marca de tiempo de un documento

Dado que _id ObjectId almacena de manera predeterminada la marca de tiempo de 4 bytes, en la mayoría de los casos no es necesario almacenar el

tiempo de creación de ningún documento. Puede recuperar el tiempo de creación de un documento utilizando el método getTimestamp:

```
>ObjectId("5349b4ddd2781d08c09890f4").getTimestamp()
```

Esto devolverá el tiempo de creación de este documento en formato de fecha ISO:

```
ISODate("2014-04-12T21:49:17Z")
```

Convertir ObjectId a String

En algunos casos, puede necesitar el valor de ObjectId en un formato de cadena. Para convertir el ObjectId en cadena, use el siguiente código:

```
>newObjectId.str
```

El código anterior devolverá el formato de cadena del Guid:

```
5349b4ddd2781d08c09890f3
```

MongoDB - Mapa Reducir

Según la documentación de MongoDB, **Map-reduce** es un paradigma de procesamiento de datos para condensar grandes volúmenes de datos en resultados agregados útiles. MongoDB usa el comando **mapReduce** para las operaciones de reducción de mapas. MapReduce se usa generalmente para procesar grandes conjuntos de datos.

Comando MapReduce

A continuación se muestra la sintaxis del comando mapReduce básico:

```
>db.collection.mapReduce(
   function() {emit(key,value);},  //map function
   function(key,values) {return reduceFunction}, {   //reduce
function
      out: collection,
      query: document,
      sort: document,
      limit: number
   }
)
```

La función map-reduce primero consulta la colección, luego asigna los documentos de resultados para emitir pares clave-valor, que luego se reduce en función de las claves que tienen múltiples valores.

En la sintaxis anterior:

- **map** es una función de JavaScript que asigna un valor con una clave y emite un par clave-valor
- **reducir** es una función de JavaScript que reduce o agrupa todos los documentos que tienen la misma clave

- **out** especifica la ubicación del resultado de la consulta map-reduce
- **consulta** especifica los criterios de selección opcionales para seleccionar documentos
- **sort** especifica los criterios de ordenación opcionales
- **limit** especifica el número máximo opcional de documentos que se devolverán

Usando MapReduce

Considere la siguiente estructura de documento que almacena las publicaciones de los usuarios. El documento almacena el nombre de usuario del usuario y el estado de la publicación.

```
{
   "post_text": "postparaprogramadores is an awesome website
for tutorials",
   "user_name": "mark",
   "status":"active"
}
```

Ahora, utilizaremos una función mapReduce en nuestra colección de **publicaciones** para seleccionar todas las publicaciones activas, agruparlas en función del nombre de usuario y luego contar el número de publicaciones de cada usuario utilizando el siguiente código:

```
>db.posts.mapReduce(
   function() { emit(this.user_id,1); },

   function(key, values) {return Array.sum(values)}, {
      query:{status:"active"},
      out:"post_total"
   }
)
```

La consulta mapReduce anterior genera el siguiente resultado:

```
{
   "result" : "post_total",
   "timeMillis" : 9,
   "counts" : {
      "input" : 4,
      "emit" : 4,
      "reduce" : 2,
      "output" : 2
   },
   "ok" : 1,
}
```

El resultado muestra que un total de 4 documentos coincidieron con la consulta (estado: "activo"), la función de mapa emitió 4 documentos con pares clave-valor y finalmente la función de reducción de documentos mapeados agrupados que tienen las mismas claves en 2.

Para ver el resultado de esta consulta mapReduce, use el operador de búsqueda:

```
>db.posts.mapReduce(
   function() { emit(this.user_id,1); },
   function(key, values) {return Array.sum(values)}, {
      query:{status:"active"},
      out:"post_total"
   }

).find()
```

La consulta anterior da el siguiente resultado que indica que tanto los usuarios **tom** como **mark** tienen dos publicaciones en estados activos:

```
{ "_id" : "tom", "value" : 2 }
{ "_id" : "mark", "value" : 2 }
```

De manera similar, las consultas MapReduce se pueden utilizar para construir consultas de agregación complejas grandes. El uso de funciones de Javascript personalizadas hace uso de MapReduce, que es muy flexible y potente.

MongoDB - Búsqueda de texto

A partir de la versión 2.4, MongoDB comenzó a admitir índices de texto para buscar dentro del contenido de la cadena. La **búsqueda de texto** utiliza técnicas de derivación para buscar palabras específicas en los campos de cadena al soltar palabras de detención derivadas como **a, an, the,** etc. En la actualidad, MongoDB admite alrededor de 15 idiomas.

Habilitar la búsqueda de texto

Inicialmente, la búsqueda de texto era una función experimental, pero a partir de la versión 2.6, la configuración está habilitada de forma predeterminada. Pero si está utilizando la versión anterior de MongoDB, debe habilitar la búsqueda de texto con el siguiente código:

```
>db.adminCommand({setParameter:true,textSearchEnabled:true})
```

Crear índice de texto

Considere el siguiente documento en la colección de **publicaciones que** contiene el texto de la publicación y sus etiquetas:

```
{
   "post_text": "enjoy the mongodb articles on
postparaprogramadores",
   "tags": [
      "mongodb",
      "postparaprogramadores"
   ]
}
```

Crearemos un índice de texto en el campo post_text para que podamos buscar dentro del texto de nuestras publicaciones:

```
>db.posts.ensureIndex({post_text:"text"})
```

Usar índice de texto

Ahora que hemos creado el índice de texto en el campo post_text, buscaremos todas las publicaciones que tengan la palabra **postparaprogramadores** en su texto.

```
>db.posts.find({$text:{$search:"postparaprogramadores"}})
```

El comando anterior devolvió los siguientes documentos de resultados que tienen la palabra **postparaprogramadores** en su texto de publicación:

```
{
   "_id" : ObjectId("53493d14d852429c10000002"),
   "post_text" : "enjoy the mongodb articles on
postparaprogramadores",
   "tags" : [ "mongodb", "postparaprogramadores" ]
}
{
   "_id" : ObjectId("53493d1fd852429c10000003"),
   "post_text" : "writing tutorials on mongodb",
   "tags" : [ "mongodb", "tutorial" ]
}
```

Si está utilizando versiones antiguas de MongoDB, debe usar el siguiente comando:

```
>db.posts.runCommand("text",{search:" postparaprogramadores
"})
```

El uso de la búsqueda de texto mejora enormemente la eficiencia de la búsqueda en comparación con la búsqueda normal.

Eliminar índice de texto

Para eliminar un índice de texto existente, primero encuentre el nombre del índice utilizando la siguiente consulta:

```
>db.posts.getIndexes()
```

Después de obtener el nombre de su índice de la consulta anterior, ejecute el siguiente comando. Aquí, **post_text_text** es el nombre del índice.

```
>db.posts.dropIndex("post_text_text")
```

MongoDB - Expresión regular

Las expresiones regulares se usan con frecuencia en todos los idiomas para buscar un patrón o palabra en cualquier cadena. MongoDB también proporciona funcionalidad de expresión regular para la coincidencia de

patrones de cadena utilizando el operador **$ regex** . MongoDB utiliza PCRE (expresión regular compatible con Perl) como lenguaje de expresión regular.

A diferencia de la búsqueda de texto, no necesitamos hacer ninguna configuración o comando para usar expresiones regulares.

Considere la siguiente estructura de documento en la colección de **publicaciones que** contiene el texto de la publicación y sus etiquetas:

```
{
   "post_text": "enjoy the mongodb articles on
postparaprogramadores",
   "tags": [
      "mongodb",
      "postparaprogramadores"
   ]
}
```

Usando la expresión regex

La siguiente consulta de expresiones regulares busca todas las publicaciones que contienen puntos de **tutoriales de** cadena en ella:

```
>db.posts.find({post_text:{$regex:"postparaprogramadores"}})
```

La misma consulta también se puede escribir como -

```
>db.posts.find({post_text:/postparaprogramadores/})
```

Uso de expresiones regulares con mayúsculas y minúsculas

Para que el caso de búsqueda no sea sensible, usamos el parámetro **$ options** con el valor **$ i** . El siguiente comando buscará cadenas que tengan la palabra **postparaprogramadores** , independientemente de mayúsculas o minúsculas:

```
>db.posts.find({post_text:{$regex:"postparaprogramadores",$op
tions:"$i"}})
```

Uno de los resultados devueltos por esta consulta es el siguiente documento que contiene la palabra **postparaprogramadores** en diferentes casos:

```
{
   "_id" : ObjectId("53493d37d852429c10000004"),
   "post_text" : "hey! this is my post on
Postparaprogramadores",
   "tags" : [ "postparaprogramadores" ]
}
```

Uso de expresiones regulares para elementos de matriz

También podemos usar el concepto de expresiones regulares en el campo de matriz. Esto es particularmente muy importante cuando implementamos la funcionalidad de las etiquetas. Entonces, si desea buscar todas las publicaciones que tienen etiquetas que comienzan con la palabra tutorial (tutorial o tutoriales o tutorialpoint o tutorialphp), puede usar el siguiente código:

```
>db.posts.find({tags:{$regex:"tutorial"}})
```

Optimización de consultas de expresiones regulares

- Si los campos del documento están **indexados** , la consulta utilizará los valores indexados para que coincidan con la expresión regular. Esto hace que la búsqueda sea muy rápida en comparación con la expresión regular que escanea toda la colección.
- Si la expresión regular es una **expresión de prefijo** , todas las coincidencias deben comenzar con ciertos caracteres de cadena. Por ejemplo, si la expresión regex es **^ tut** , entonces la consulta debe buscar solo aquellas cadenas que comienzan con **tut** .

Trabajando con RockMongo

RockMongo es una herramienta de administración de MongoDB que le permite administrar su servidor, bases de datos, colecciones, documentos, índices y mucho más. Proporciona una forma muy fácil de usar para leer, escribir y crear documentos. Es similar a la herramienta PHPMyAdmin para PHP y MySQL.

Descargando RockMongo

Puede descargar la última versión de RockMongo desde aquí: https://github.com/iwind/rockmongo

Instalando RockMongo

Una vez descargado, puede descomprimir el paquete en la carpeta raíz del servidor y cambiar el nombre de la carpeta extraída a **rockmongo** . Abra cualquier navegador web y acceda a la página **index.php** desde la carpeta rockmongo. Ingrese admin / admin como nombre de usuario / contraseña respectivamente.

Trabajando con RockMongo

Ahora analizaremos algunas operaciones básicas que puede realizar con RockMongo.

Crear nueva base de datos

Para crear una nueva base de datos, haga clic en la pestaña **Bases de datos** . Haga clic en **Crear nueva base de datos** . En la siguiente pantalla, proporcione el nombre de la nueva base de datos y haga clic en **Crear** . Verá que se agrega una nueva base de datos en el panel izquierdo.

Creando nueva colección

Para crear una nueva colección dentro de una base de datos, haga clic en esa base de datos desde el panel izquierdo. Haga clic en el enlace **Nueva colección** en la parte superior. Proporcione el nombre requerido de la colección. No se preocupe por los otros campos de Is Capped, Size y Max. Haz clic en **Crear** . Se creará una nueva colección y podrá verla en el panel izquierdo.

Crear nuevo documento

Para crear un nuevo documento, haga clic en la colección en la que desea agregar documentos. Cuando haga clic en una colección, podrá ver todos los documentos dentro de esa colección enumerados allí. Para crear un nuevo documento, haga clic en el enlace **Insertar** en la parte superior. Puede ingresar los datos del documento en formato JSON o en matriz y hacer clic en **Guardar** .

Exportar / Importar datos

Para importar / exportar datos de cualquier colección, haga clic en esa colección y luego haga clic en **el** enlace **Exportar / Importar** en el panel superior. Siga las siguientes instrucciones para exportar sus datos en formato zip y luego importe el mismo archivo zip para volver a importar los datos.

MongoDB - GridFS

GridFS es la especificación de MongoDB para almacenar y recuperar archivos grandes, como imágenes, archivos de audio, archivos de video, etc. Es una especie de sistema de archivos para almacenar archivos, pero sus datos se almacenan en colecciones MongoDB. GridFS tiene la capacidad de almacenar archivos incluso mayores que su límite de tamaño de documento de 16 MB.

GridFS divide un archivo en fragmentos y almacena cada fragmento de datos en un documento separado, cada uno con un tamaño máximo de 255k.

GridFS por defecto utiliza dos colecciones **fs.files** y **fs.chunks** para almacenar los metadatos y los fragmentos del archivo. Cada fragmento se identifica por su único campo _id ObjectId. El fs.files sirve como documento padre. El campo **files_id** en el documento fs.chunks vincula el fragmento a su padre.

El siguiente es un documento de muestra de la colección fs.files:

```
{
    "filename": "test.txt",
    "chunkSize": NumberInt(261120),
    "uploadDate": ISODate("2014-04-13T11:32:33.557Z"),
    "md5": "7b762939321e146569b07f72c62cca4f",
    "length": NumberInt(646)
}
```

El documento especifica el nombre del archivo, el tamaño del fragmento, la fecha de carga y la longitud.

El siguiente es un documento de muestra del documento fs.chunks:

```
{
    "files_id": ObjectId("534a75d19f54bfec8a2fe44b"),
    "n": NumberInt(0),
    "data": "Mongo Binary Data"
}
```

Agregar archivos a GridFS

Ahora, almacenaremos un archivo mp3 usando GridFS usando el comando **put** . Para esto, utilizaremos la utilidad **mongofiles.exe** presente en la carpeta bin de la carpeta de instalación de MongoDB.

Abra su símbolo del sistema, navegue hasta mongofiles.exe en la carpeta bin de la carpeta de instalación de MongoDB y escriba el siguiente código:

```
>mongofiles.exe -d gridfs put song.mp3
```

Aquí, **gridfs** es el nombre de la base de datos en la que se almacenará el archivo. Si la base de datos no está presente, MongoDB creará automáticamente un nuevo documento sobre la marcha. Song.mp3 es el nombre del archivo cargado. Para ver el documento del archivo en la base de datos, puede usar la consulta de búsqueda:

```
>db.fs.files.find()
```

El comando anterior devolvió el siguiente documento:

```
{
    _id: ObjectId('534a811bf8b4aa4d33fdf94d'),
    filename: "song.mp3",
    chunkSize: 261120,
    uploadDate: new Date(1397391643474), md5:
"e4f53379c909f7bed2e9d631e15c1c41",
    length: 10401959
}
```

También podemos ver todos los fragmentos presentes en la colección fs.chunks relacionados con el archivo almacenado con el siguiente código, utilizando la identificación del documento devuelto en la consulta anterior:

```
>db.fs.chunks.find({files_id:ObjectId('534a811bf8b4aa4d33fdf94d')})
```

En mi caso, la consulta devolvió 40 documentos, lo que significa que todo el documento mp3 se dividió en 40 fragmentos de datos.

MongoDB - Colecciones tapadas

Las colecciones con límite son **colecciones** circulares de tamaño fijo que siguen el orden de inserción para admitir un alto rendimiento en las operaciones de creación, lectura y eliminación. Por circular, significa que cuando se agota el tamaño fijo asignado a la colección, comenzará a eliminar el documento más antiguo de la colección sin proporcionar ningún comando explícito.

Las colecciones con límite restringen las actualizaciones de los documentos si la actualización resulta en un mayor tamaño del documento. Dado que las colecciones con tapa almacenan los documentos en el orden del almacenamiento en disco, garantiza que el tamaño del documento no aumente el tamaño asignado en el disco. Las colecciones con límite son las mejores para almacenar información de registro, datos de caché o cualquier otro dato de alto volumen.

Crear colección con límite

Para crear una colección limitada, usamos el comando createCollection normal pero con la opción **limitada** como **verdadera** y especificando el tamaño máximo de la colección en bytes.

```
>db.createCollection("cappedLogCollection",{capped:true,size:10000})
```

Además del tamaño de la colección, también podemos limitar el número de documentos en la colección usando el parámetro **max** :

```
>db.createCollection("cappedLogCollection",{capped:true,size:10000,max:1000})
```

Si desea verificar si una colección está limitada o no, use el siguiente comando **isCapped** :

```
>db.cappedLogCollection.isCapped()
```

Si hay una colección existente que planea convertir a limitada, puede hacerlo con el siguiente código:

```
>db.runCommand({"convertToCapped":"posts",size:10000})
```

Este código convertiría nuestras **publicaciones de** colección existentes en una colección con límite.

Consulta de la colección con tapa

De forma predeterminada, una consulta de búsqueda en una colección limitada mostrará los resultados en orden de inserción. Pero si desea que los documentos se recuperen en orden inverso, use el comando de **clasificación** como se muestra en el siguiente código:

```
>db.cappedLogCollection.find().sort({$natural:-1})
```

Hay algunos otros puntos importantes con respecto a las colecciones limitadas que vale la pena conocer:

- No podemos eliminar documentos de una colección limitada.
- No hay índices predeterminados presentes en una colección limitada, ni siquiera en el campo _id.
- Al insertar un nuevo documento, MongoDB no tiene que buscar realmente un lugar para acomodar un nuevo documento en el disco. Puede insertar ciegamente el nuevo documento en la cola de la colección. Esto hace que las operaciones de inserción en colecciones con límite sean muy rápidas.
- Del mismo modo, mientras lee documentos, MongoDB devuelve los documentos en el mismo orden que los presentes en el disco. Esto hace que la operación de lectura sea muy rápida.

MongoDB - Secuencia de incremento automático

MongoDB no tiene una funcionalidad de incremento automático lista para usar, como las bases de datos SQL. De manera predeterminada, utiliza el ObjectId de 12 bytes para el campo **_id** como clave principal para identificar de forma exclusiva los documentos. Sin embargo, puede haber escenarios en los que deseemos que el campo _id tenga algún valor de incremento automático distinto del ObjectId.

Dado que esta no es una característica predeterminada en MongoDB, lograremos esta funcionalidad mediante programación mediante el uso de una colección de **contadores** como lo sugiere la documentación de MongoDB.

Usando colección de contador

Considere el siguiente documento de **productos** . Queremos que el campo _id sea una **secuencia entera de incremento automático que** comience desde 1,2,3,4 hasta n.

```
{
  "_id":1,
  "product_name": "Apple iPhone",
  "category": "mobiles"
}
```

Para esto, cree una colección de **contadores** , que hará un seguimiento del último valor de secuencia para todos los campos de secuencia.

```
>db.createCollection("counters")
```

Ahora, insertaremos el siguiente documento en la colección de contadores con **productid** como clave:

```
{
  "_id":"productid",
  "sequence_value": 0
}
```

El campo **secuencia_valor** realiza un seguimiento del último valor de la secuencia.

Use el siguiente código para insertar este documento de secuencia en la colección de contadores:

```
>db.counters.insert({_id:"productid",sequence_value:0})
```

Creando la función Javascript

Ahora, crearemos una función **getNextSequenceValue** que tomará el nombre de secuencia como su entrada, incrementará el número de secuencia en 1 y devolverá el número de secuencia actualizado. En nuestro caso, el nombre de la secuencia es **productid** .

```
>function getNextSequenceValue(sequenceName){

   var sequenceDocument = db.counters.findAndModify({
      query:{_id: sequenceName },
      update: {$inc:{sequence_value:1}},
      new:true
   });

   return sequenceDocument.sequence_value;
}
```

Usando la función Javascript

Ahora usaremos la función getNextSequenceValue al crear un nuevo documento y asignar el valor de secuencia devuelto como el campo _id del documento.

Inserte dos documentos de muestra con el siguiente código:

```
>db.products.insert({
   "_id":getNextSequenceValue("productid"),
   "product_name":"Apple iPhone",
   "category":"mobiles"
})

>db.products.insert({
   "_id":getNextSequenceValue("productid"),
   "product_name":"Samsung S3",
   "category":"mobiles"
})
```

Como puede ver, hemos utilizado la función getNextSequenceValue para establecer el valor del campo _id.

Para verificar la funcionalidad, busquemos los documentos usando el comando find -

```
>db.products.find()
```

La consulta anterior devolvió los siguientes documentos con el campo _id auto-incrementado:

```
{ "_id" : 1, "product_name" : "Apple iPhone", "category" :
"mobiles"}

{ "_id" : 2, "product_name" : "Samsung S3", "category" :
"mobiles" }
```