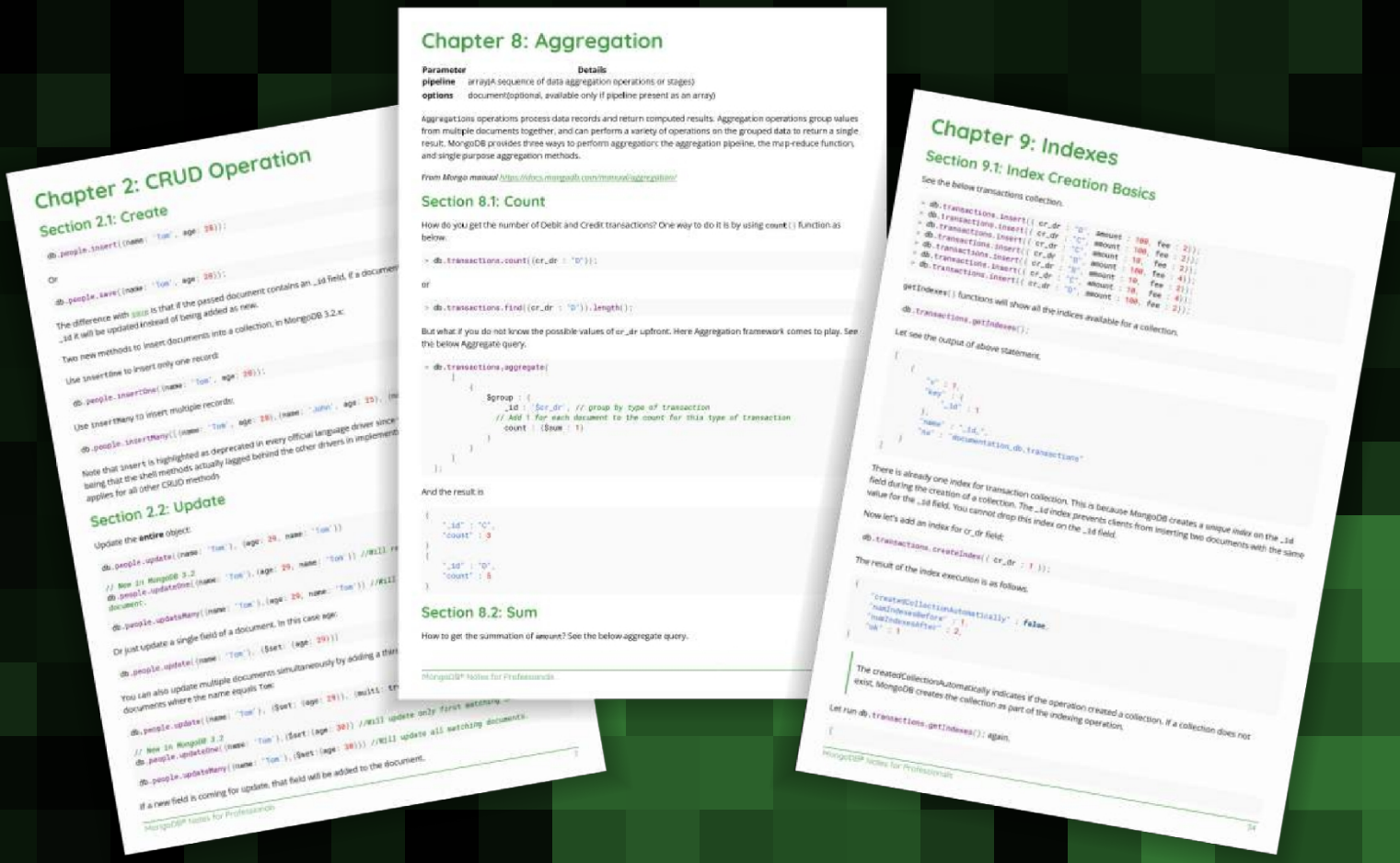
MongoDB®

Notas para los profesionales



Más de 60 páginas

de consejos y trucos profesionales

gratuitos

GoalKicker.com
Libros de programación

Contenido

Acerca de

Por favor, siéntase libre de compartir este PDF con cualquier persona de forma gratuita, la última versión de este libro se puede descargar desde: https://goalkicker.com/MongoDBBook

Este libro *MongoDB® Notes for Professionals* ha sido compilado a partir de la Documentación de Stack Overflow, el contenido ha sido escrito por la hermosa gente de Stack Overflow. El contenido del texto está liberado bajo Creative Commons BY-SA, ver los créditos al final de este libro de quién contribuyó a los distintos capítulos. Las imágenes pueden ser copyright de sus respectivos propietarios a menos que se especifique lo contrario.

Envíe sus comentarios y correcciones web@petercv.com

Capítulo 1: Introducción a MongoDB

Versión	Fecha de publicación
3.6.1	2017-12-26
3.4	2016-11-29
3.2	2015-12-08
3.0	2015-03-03
2.6	2014-04-08
2.4	2013-03-19
2.2	2012-08-29
2.0	2011-09-12
1.8	2011-03-16
1.6	2010-08-31
1.4	2010-03-25
1.2	2009-12-10

Sección 1.1: Ejecución de un archivo JavaScript en MongoDB

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

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

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

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

Sección 1.2: Cómo hacer que la salida de find sea legible en el shell

Añadimos tres registros a nuestra prueba de colección como:

```
> db.test.insert({"clave": "valor1", "clave2": "Val2", "clave3":
"val3"}) WriteResult({"nInserted" : 1 })
> db.test.insert({"clave": "valor2", "clave2": "Val21", "clave3":
"val31"}) WriteResult({"nInserted" : 1 })
> db.test.insert({"clave": "valor3", "clave2": "Val22", "clave3":
"val33"}) WriteResult({"nInserted" : 1 })
```

Si los vemos vía hallazgo, se verán muy feos.

```
> db.test.find()
{ "_id" : ObjectId("5790c5cecae25b3d38c3c7ae")), "key" : "value1", "key2" : "Val2 ",
"key3" : "val3" }
{ "_id" : ObjectId("5790c5d9cae25b3d38c3c7af"), "key" : "valor2", "key2" : "Val2 1",
"key3" : "val31" }
{ "_id" : ObjectId("5790c5e9cae25b3d38c3c7b0"), "key" : "valor3", "key2" : "Val2 2",
"key3" : "val33" }
```

Para evitarlo y hacerlos legibles, utilice la **pretty**().

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

```
{
        "_id" : ObjectId("5790c5cecae25b3d38c3c7ae"),
        "key" : "value1",
        "clave2" : "Val2",
        "clave3" : "val3"
}
{
        "_id" : ObjectId("5790c5d9cae25b3d38c3c7af"),
        "key" : "value2",
        "key2" : "Val21",
        "clave3" : "val31"
}
{
        "_id" : ObjectId("5790c5e9cae25b3d38c3c7b0"),
        "key" : "value3",
        "key2" : "Val22",
        "clave3" : "val33"
}
>
```

Sección 1.3: Términos complementarios

Términos SQL	Términos MongoDB
Base de datos	Tabla Colección
Entidad / Fila	Documento
Columna	Clave / Campo
Tabla Join	Documentos incrustados
Clave primaria	Clave primaria (Clave por defecto `_id` proporcionada por el propio mongodb)

Sección 1.4: Instalación

Para instalar MongoDB, siga los pasos que se indican a continuación:

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

      ```
      $ brew install mongodb
      ```
  - **Instalación manual:**
    - Descargue la última versión aquí. Asegúrese de que está descargando el archivo adecuado, especialmente compruebe si su tipo de sistema operativo es de 32 o 64 bits. El archivo descargado tiene formato `tgz`.
    - Vaya al directorio donde se ha descargado este archivo. A continuación, escriba el siguiente comando:

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

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

ejecute el servidor con los siguientes comandos:

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

- Para iniciar el , debe darse el siguiente comando desde la ubicación actual:

```
$ ./mongod
```

Iniciaría el servidor en el puerto 27017 por defecto.

- Para iniciar el cliente, debe abrirse un nuevo terminal con el mismo directorio que antes. A continuación, el siguiente comando iniciaría el cliente y se conectaría al servidor.

```
$ ./mongo
```

Por defecto se conecta a la base de datos de `prueba`. Si ves la línea como `conectando a: test`. Entonces has instalado MongoDB con éxito. ¡Enhorabuena! Ahora, puedes probar Hello World para estar más seguro.

- **Para Windows:**

  - Descargue la última versión aquí. Asegúrese de que está descargando el archivo adecuado, especialmente compruebe si su sistema operativo es de 32 o 64 bits.
  - El archivo binario descargado tiene extensión `exe`. Ejecútelo. Aparecerá un asistente de
  - instalación. Haga clic en **Siguiente**.
  - **Acepte** el acuerdo de licencia y haga clic en **Siguiente**.
  - Seleccione Instalación **completa**.
  - Haga clic en **Instalar**. Puede que aparezca una ventana pidiendo permiso al administrador.
  - Haga clic en **Sí**. Tras la instalación, haga clic en **Finalizar**.
  - Ahora, el mongodb está instalado en la ruta `C:/Archivos de Programa/MongoDB/Server/3.2/bin`. En lugar de la versión 3.2, podría haber alguna otra versión para tu caso. El nombre de la ruta se cambiaría en consecuencia.
  - El directorio `bin` contiene varios archivos binarios junto con `mongod` y `mongo`. Para ejecutarlo desde otra carpeta, puede añadir la ruta en la ruta del sistema. Para :
    - Haga clic con el botón derecho en **Mi PC** y seleccione **Propiedades**.
    - Haga clic en **Configuración avanzada del sistema** en el panel izquierdo.
    - Haga clic en **Variables de entorno...** en la pestaña **Avanzadas**.
    - Seleccione **Path** en la sección **System variables** y haga clic en Edit....
    - Antes de Windows , añada un punto y coma y pegue la ruta indicada anteriormente. A partir de Windows 10, hay un botón **Nuevo** para añadir una nueva ruta.
    - Haga clic en **Aceptar** para guardar los cambios.
  - Ahora, crea una carpeta llamada `data` con una subcarpeta llamada `db` donde quieras ejecutar el servidor.
  - Inicie el símbolo del sistema desde su. Ya sea cambiando la ruta en cmd o haciendo clic en **Abrir ventana de comandos aquí** que sería visible después de hacer clic derecho en el espacio vacío de la carpeta GUI pulsando

las teclas Shift y Ctrl a la vez.

- Escribe el comando para iniciar el servidor:

```
> mongod
```

Iniciaría el servidor en el puerto 27017 por defecto.

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

```
> mongo
```

- Por defecto se conecta a la base de datos de `prueba`. Si ves la línea como `conectando a: test`. Entonces has instalado MongoDB con éxito. ¡Enhorabuena! Ahora, puedes probar Hello World para estar más seguro.

- **Para Linux:** Casi igual que Mac OS excepto que se necesita algún comando equivalente.

  - Para distros basadas en Debian (usando `apt-get`):
    - Importar la clave del repositorio MongoDB.

```
$ sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv EA312927 gpg:
Número total procesado: 1\
gpg:               importado: 1 (RSA: 1)
```

    - Añadir repositorio a la lista de paquetes en **Ubuntu 16.04**.

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

    - en **Ubuntu 14.04**.

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

    - Actualizar la lista de paquetes.

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

    - Instale MongoDB.

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

  - Para distros basadas en Red Hat (usando `yum`):
    - utilice el editor de texto que prefiera.

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

    - el siguiente texto.

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

    - Actualizar la lista de paquetes.

```
$ sudo yum update
```

    - Instalar MongoDB

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

Sección 1.5: Comandos básicos en mongo shell

Mostrar todas las bases de datos disponibles:

```
mostrar dbs;
```

Seleccione una base de datos concreta a la que acceder, por ejemplo, `mydb`. Se creará `mydb` si aún no existe:

```
utilizar mydb;
```

Mostrar todas las colecciones de la base de datos (asegúrese de seleccionar una primero, véase más arriba):

```
mostrar colecciones;
```

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

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

Para comprobar la base de datos seleccionada actualmente, utilice el comando `db`

```
> db
mydb
```

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

```
db.dropBaseDeDatos()
```

Sección 1.6: Hola Mundo

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

```
> db.world.insert({ "discurso" : "¡Hola Mundo!" });
> cur= db.world.find();x= cur.next();print(x["speech"]);
```

> Hola a todos.

Explicación:

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

Capítulo 2: Operación CRUD

Sección 2.1: Crear

```
db.people.insert({nombre: 'Tom', edad: 28});
```

O

```
db.people.save({nombre: 'Tom', edad: 28});
```

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

Dos nuevos métodos para insertar documentos en una colección, en MongoDB

3.2.x: Utilice `insertOne` para insertar un solo registro:

```
db.people.insertOne({nombre: 'Tom', edad: 28});
```

Utilice `insertMany` para insertar varios registros:

```
db.people.insertMany([{nombre: 'Tom', edad: 28},{nombre: 'John', edad: 25}, {nombre: 'Kathy', edad: 23}])
```

Nótese que `insert` está marcado como obsoleto en todos los controladores de lenguaje oficiales desde la versión 3.0. La diferencia en que los métodos shell se han quedado rezagados con respecto a los demás controladores a la hora de implementar el método. Lo mismo se aplica a todos los demás métodos CRUD

Sección 2.2: Actualización

Actualiza **todo el** objeto:

```
db.people.update({nombre: 'Tom'}, {edad: 29, nombre: 'Tom'})

// Nuevo en MongoDB 3.2
db.people.updateOne({nombre: 'Tom'},{edad: 29, nombre: 'Tom'}) //Reemplazará sólo el primer documento coincidente.

db.people.updateMany({nombre: 'Tom'},{edad: 29, nombre: 'Tom'}) //Reemplazará todos los documentos coincidentes.
```

O simplemente actualizar un único campo de un documento. En este caso `la edad`:

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

También puede actualizar varios documentos simultáneamente añadiendo un tercer parámetro. Esta consulta actualizará todos los documentos cuyo nombre sea igual a `Tom`:

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

// Nuevo en MongoDB 3.2
db.people.updateOne({nombre: 'Tom'},{$set:{edad: 30}) //Actualizará sólo el primer documento coincidente.

db.people.updateMany({nombre: 'Tom'},{$set:{edad: 30}}) //Actualizará todos los documentos coincidentes.
```

Si se actualiza un nuevo campo, éste se añadirá al documento.

```
db.people.updateMany({name: 'Tom'},{$set:{age: 30, salary:50000}})// El documento tendrá el campo `salary`.
```

Si es necesario sustituir un documento,

```
db.collection.replaceOne({nombre:'Tom'}, {nombre:'Lakmal',edad:25,dirección:'Sri Lanka'})
```

se puede utilizar.

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

Sección 2.3: Suprimir

Elimina todos los documentos que coincidan con el parámetro de consulta:

```
// Nuevo en MongoDB 3.2
db.people.deleteMany({nombre: 'Tom'})

// Todas las versiones
db.people.remove({nombre: 'Tom'})
```

O sólo uno

```
// Nuevo en MongoDB 3.2
db.people.deleteOne({nombre: 'Tom'})

// Todas las versiones
db.people.remove({nombre: 'Tom'}, true)
```

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

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

o

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

Sección 2.4: Leer

Consulta todos los documentos de la colección de `personas` que tienen un campo de `nombre con el valor "Tom"`:

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

O sólo primero:

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

También puede especificar qué campos devolver pasando un parámetro de selección de campos. Lo siguiente excluirá el `_id` e incluir únicamente el campo de `edad`:

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

Nota: por defecto, se devolverá el campo _id, aunque no lo pida. Si no desea que se le devuelva el `_id`, puede seguir el ejemplo anterior y pedir que se excluya `el _id` especificando `_id: 0 (o _id: false): 0` (o `_id: false`). Si desea encontrar un registro secundario como el objeto dirección que contiene el país, la ciudad, etc.

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

& especifique también el campo si es necesario

```
db.people.find({'direccion.pais': 'US'}, {'nombre': true, 'direccion.ciudad': true})Recuerda que el
resultado tiene un método `.pretty()` que imprime el JSON resultante:

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

Sección 2.5: Actualización de documentos incrustados

Para el siguiente esquema:

```
{nombre: 'Tom', edad: 28, notas: [50, 60, 70]}
```

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

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

Para el siguiente esquema:

```
{nombre: "Tom", edad: 28, notas: [{asignatura: "Inglés", notas: 90},{asignatura: "Matemáticas", nota:
100},
{tema: "Computa", marcas: 20}]}
```

Actualiza la nota de inglés de Tom a 85 :

```
db.people.update({nombre: "Tom", "asignatura.notas": "Inglés"},{"$set":{"notas.$.notas": 85}})
```

Explicando el ejemplo anterior:

Usando {name: "Tom", "marks.subject": "English"} obtendrás la posición del objeto en el array de marcas, donde subject es English. En "marcas.$.marcas", $ se utiliza para actualizar en esa posición de la matriz de marcas

Actualizar valores en una matriz

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

Considere una colección de estudiantes con los siguientes documentos:

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

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

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

```
)
```

Sección 2.6: Más operadores de actualización

Puedes usar otros operadores además de `$set` cuando actualices un documento. El operador `$push` permite introducir un valor en un array, en este caso añadiremos un nuevo apodo al array `apodos`.

```
db.people.update({nombre: 'Tom'}, {$push: {nombres: 'Tommy'}})
// Esto añade la cadena 'Tommy' al array de apodos del documento de Tom.
```

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

```
db.people.update({nombre: 'Tom'}, {$pull: {nombres: 'Tommy'}})
// Esto elimina la cadena 'Tommy' de la matriz de apodos en el documento de Tom.
```

El operador `$pop` permite eliminar el primer o el último valor de un array. Digamos que el documento de Tom tiene una propiedad llamada hermanos que tiene el valor `['Marie', 'Bob', 'Kevin', 'Alex']`.

```
db.people.update({nombre: 'Tom'}, {$pop: {hermanos: -1}})
// Esto eliminará el primer valor del array de hermanos, que en este es 'Marie'.


db.people.update({nombre: 'Tom'}, {$pop: {hermanos: 1}})
// Esto eliminará el último valor del array de hermanos, que en este es 'Alex'.
```

Sección 2.7: Parámetro "multi" al actualizar varios documentos

Para actualizar varios documentos de una colección, establezca la opción multi en true.

```
db.collection.update(
   consulta,
   actualizar,
   {
     upsert: booleano,
     multi: booleano,
     writeConcern: documento
   }
)
```

multi es opcional. Si es verdadero, actualiza varios documentos que cumplen los criterios de la consulta. Si es false, actualiza un solo documento. El valor por defecto es false.

> db.mycol.find() { "_id" : ObjectId(598354878df45ec5), "title": "Visión general de MongoDB"} { "_id" : ObjectId(59835487adf45ec6), "title": "Visión general de NoSQL"} { "_id" : ObjectId(59835487adf45ec7), "title": "Visión general de Tutoriales"}

> db.mycol.update({'title':'Visión general de MongoDB'}, {$set:{'title':'Nuevo tutorial de MongoDB'}},{multi:true})

Capítulo 3: Obtener información de la base de datos

Sección 3.1: Lista de todas las colecciones de la base de datos

```
mostrar colecciones
```

o

```
mostrar tablas
```

o

```
db.getNombresDeColección()
```

Sección 3.2: Lista de todas las bases de datos

```
mostrar dbs
```

o

```
db.adminCommand('listBasesDeDatos')
```

o

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

Capítulo 4: Consulta de datos (Primeros pasos)

Ejemplos básicos de consulta

Sección 4.1: Buscar()

recuperar todos los documentos de una colección

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

recuperar documentos en una colección utilizando una condición (similar a WHERE en MYSQL)

```
db.collection.find({clave: valor});
ejemplo
  db.users.find({email: "sample@email.com"});
```

recuperar documentos de una colección mediante condiciones booleanas (operadores de consulta)

```
//Y
db.collection.find( {
    $y: [
     { clave: valor }, { clave: valor }
    ]
})
//OR
db.collection.find( {
    $o: [
     { clave: valor }, { clave: valor }
    ]
})
//NO
db.inventory.find( { clave: { $no: valor } } )
```

más operaciones booleanas y ejemplos aquí

NOTA: *find()* seguirá buscando en la colección incluso si se ha encontrado una coincidencia de documento, por lo tanto es ineficiente cuando se utiliza en una colección grande, sin embargo modelando cuidadosamente sus datos y/o utilizando índices puede aumentar la eficiencia de *find()*

Sección 4.2: FindOne()

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

la funcionalidad de consulta es similar a find() pero esta terminará la ejecución en el momento en que encuentre un documento que coincida con su condición, si se utiliza con un objeto vacío, obtendrá el primer documento y lo devolverá. findOne() mongodb api documentation

Sección 4.3: limitar, omitir, ordenar y contar los resultados del método find()

De forma similar a los métodos de agregación, el método find() permite limitar, omitir, ordenar y contar los resultados. Digamos que tenemos la siguiente colección:

```
db.test.insertMany([
    {nombre: "Cualquiera", edad: "21", estado: "ocupado"},
    {nombre: "Tony", edad: "25", estado: "ocupado"},
    {nombre: "Bobby", edad: "28", estado: "en línea"},
    {nombre: "Sonny", edad: "28", estado: "ausente"},
    {nombre: "Cher", edad: "20", estado: "en línea"}
])
```

Para listar la colección:

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

Volverá:

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

Para saltarse los 3 primeros documentos:

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

Volverá:

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

Para ordenar descendentemente por el nombre del campo:

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

Volverá:

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

Si desea ordenar de forma ascendente sólo tiene que

sustituir -1 por 1 Para contar los resultados:

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

Volverá:

```
5
```

También se permiten combinaciones de estos métodos. Por ejemplo, obtener 2 documentos de una colección ordenada descendentemente omitiendo el primero:

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

Volverá:

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

Sección 4.4: Documento de consulta - Uso de las condiciones AND, OR e IN

Todos los documentos de la colección de `estudiantes`.

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

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

> Consulta similar `mySql` del comando anterior.
>
> ```
> SELECT * FROM estudiantes;
> ```

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

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

> Consulta similar `mySql` del comando anterior.
>
> ```
> SELECT * FROM estudiantes WHERE firstName= "Prosen";
> ```

Consultas AND

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

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

> Consulta similar `mySql` del comando anterior.
>
> ```
> SELECT * FROM estudiantes WHERE nombre= "Prosen" AND edad>= 23
> ```

O Consultas

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

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

> Consulta similar `mySql` del comando anterior.
>
> ```
> SELECT * FROM estudiantes WHERE nombre= "Prosen" o edad>= 23
> ```

Y consultas OR

```
db.students.find({
```

```
        firstName : "Prosen",
        $o : [
            {edad : 23},
            {edad : 25}
        ]
});

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

> Consulta similar mySql del comando anterior.
>
> ```
> SELECT * FROM students WHERE firstName= "Prosen" AND age= 23 OR age= 25;
> ```

Consultas IN Estas consultas pueden mejorar el uso múltiple de las consultas OR

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

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

> Consulta mySql similar al comando anterior
>
> ```
> SELECT * FROM estudiantes WHERE apellido EN ('Ghosh', 'Amin')
> ```

Sección 4.5: método find() con proyección

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

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

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

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

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

Apartado 4.6: Método Find() con proyección

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

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

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

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

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

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

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

Nota: El campo `_id` siempre se muestra al ejecutar el método `find()`, si no desea este campo, entonces debe establecerlo como `0`.

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

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

Capítulo 5: Operadores de actualización

parámetros	Significado
fieldName	El campo se actualizará :{***nombre***: 'Tom'}
targetVaule	El valor se asignará al campo :{name: ***'Tom'***}

Sección 5.1: Operador $set para actualizar los campos especificados en los documentos

I. Visión general

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

II. ¿Qué ocurre si no utilizamos operadores de actualización?

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

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

Un día te llega un trabajo en el que tienes que cambiar el sexo de Tom de "M" a "F". Es fácil, ¿verdad? Así que escribes la siguiente sentencia muy rápidamente basándote en tu experiencia con RDBMS:

```
db.student.update(
    {nombre: 'Tom'}, // criterios de consulta
    {sexo: 'F'} // acción de actualización
);
```

Veamos cuál es el resultado:

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

Hemos perdido la edad y el nombre de Tom. A partir de este ejemplo, podemos saber que **todo el documento será anulado** si no hay ningún operador update en la sentencia update. Este es el comportamiento por defecto de MongoDB.

III. operador $set

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

```
db.student.update(
    {nombre: 'Tom'}, // criterios de consulta
    {$set: {sex: 'F'}} // acción de actualización
);
```

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

Por lo tanto, el resultado es correcto ahora:

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

Además, si quieres cambiar "sexo" y "edad" al mismo , puedes añadirlos a `$set` :

```
db.student.update(
    {nombre: 'Tom'}, // criterios de consulta
    {$set: {sexo: 'F', edad: 40}} // acción de actualización
);
```

Capítulo 6: Upserts e inserciones

Sección 6.1: Insertar un documento

_id es un número hexadecimal de 12 bytes que asegura la unicidad de cada documento. Puede proporcionar _id al insertar el . **Si no lo hace, MongoDB proporcionará un id único para cada documento.** Los primeros 4 bytes de estos 12 bytes son para la marca de tiempo actual, los 3 bytes siguientes para el id de máquina, los 2 bytes siguientes para el id de proceso del servidor mongodb y los 3 bytes restantes son un simple valor incremental.

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

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

Capítulo 7: Cobros

Sección 7.1: Crear una colección

Primero seleccione o cree una base de datos.

```
> utilizar mydb
cambiado a db mydb
```

Utilizando el método `db.createCollection("yourCollectionName")` se puede crear explícitamente una colección.

```
> db.createCollection("nuevaColección1")
{ "ok" : 1 }
```

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

```
> mostrar
colecciones
nuevaColección1
sistema.índices
>
```

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

Parámetro	Tipo	Descripción
nombre	cadena	El nombre de la colección a crear.
opciones	document	*Opcional.* Opciones de configuración para crear una colección tapada o para preasignar espacio en una nueva colección.

o

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

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

Tanto las operaciones `db.collection.insert()` como `db.collection.createIndex()` crean sus respectivas colecciones si aún no existen.

```
> db.newCollection2.insert({nombre : "XXX"})
> db.newCollection3.createIndex({número de cuenta : 1})
```

Ahora, muestre todas las colecciones utilizando el comando `show collections`

```
> mostrar
colecciones
nuevaColección1
nuevaColección2
nuevaColección3
nuevaColección4
system.indexes
```

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

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

Sección 7.2: Recogida de gotas

La `función db.collection.drop()` de MongoDB se utiliza para eliminar una colección

de la base de datos. En primer lugar, compruebe las colecciones disponibles en su base

de datos `mydb`.

```
> utilizar mydb
cambiado a db mydb

> mostrar
colecciones
nuevaColección1
nuevaColección2
nuevaColección3
system.indexes
```

Ahora suelta la colección con el nombre `nuevaColección1`.

```
> db.nuevaColección1.drop()
verdadero
```

Nota: Si la colección se ha eliminado correctamente, el método devolverá `true`, de lo contrario devolverá

`false`. Compruebe de nuevo la lista de colecciones en la base de datos.

```
> mostrar
colecciones
nuevaColección2
nuevaColección3
sistema.índices
```

Referencia: Método drop() de MongoDB.

Capítulo 8: Agregación

Parámetro	Detalles
canalización	array(Una secuencia de operaciones o etapas de agregación de datos)
opciones	document(opcional, disponible sólo si pipeline está presente como array)

`Las` operaciones `de agregación` procesan registros de datos y devuelven resultados calculados. Las operaciones de agregación agrupan valores de múltiples documentos, y pueden realizar una variedad de operaciones sobre los datos agrupados para devolver un único resultado. MongoDB proporciona tres formas de realizar la agregación: la canalización de agregación, la función map-reduce y los métodos de agregación de propósito único.

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

Sección 8.1: Recuento

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

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

o

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

Pero ¿qué pasa si usted no sabe los posibles valores de `cr_dr` por adelantado. Aquí entra en juego el marco de agregación. Vea la siguiente consulta agregada.

```
> db.transactions.aggregate( [
          {
               $grupo : {
                    _id : '$cr_dr', // agrupar por tipo de transacción
                   // Añadir 1 por cada documento al recuento de este tipo de transacción
                    count : {$suma : 1}
               }
          }
     ]
  );
```

Y el resultado es

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

Sección 8.2: Suma

¿Cómo obtener la suma de los `importes`? Consulte la siguiente consulta agregada.

```
> db.transactions.aggregate( [
         {
              $grupo : {
                   _id : '$cr_dr',
                   count : {$suma : 1},
                                          //cuenta    el número
                   totalImporte : {$suma : '$importe'}    //suma    el importe
              }
         }
     ]
 );
```

Y el resultado es

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

Otra versión que suma `importe` y `tasa`.

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

Y el resultado es

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

Sección 8.3: Promedio

¿Cómo obtener el importe medio de las operaciones de débito y crédito?

```
> db.transactions.aggregate( [
         {
```

```
            $grupo : {
                _id : '$cr_dr', // agrupar por tipo de transacción (débito o crédito)
                count : {$suma : 1}, // número de transacciones para cada tipo
                totalAmount : {$sum : { $sum : ['$amount', '$fee']}},  // suma
                averageAmount : {$avg : { $sum : ['$amount', '$fee']}}  // media
            }
        }
    ]
)
```

El resultado es

```
{
    "_id" : "C", // Importes de las operaciones de
    crédito
    "count" : 3.0,
    "totalAmount" : 128.0,
    "averageAmount" : 40.0
}
{
    "_id" : "D", // Importes de las operaciones de
    débito
    "count" : 5.0,
    "totalAmount" : 422.0,
    "importePromedio" : 82,0
}
```

Sección 8.4: Operaciones con matrices

Para trabajar con los datos de una matriz, primero hay que . La operación de desenrollado crea un documento para cada entrada del array. Cuando tienes muchos documentos con arrays grandes verás una explosión en el número de documentos.

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

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

Un aviso importante es que cuando un documento no contiene el array se perderá. A partir de mongo 3.2 hay una opción "preserveNullAndEmptyArrays" añadida. Esta opción asegura que el documento se conserva cuando falta el array.

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

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

Sección 8.5: Ejemplos de consultas agregadas útiles para el trabajo y el aprendizaje

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

Cree algunos datos ficticios:

```
db.employees.insert({"nombre": "Adma", "departamento": "Administración", "idiomas":["alemán", "francés",
"inglés", "hindi"].
edad":30, "totalExp":10}); db.employees.insert({"nombre": "Anna", "departamento": "Administración",
"idiomas":["inglés", "hindi"], "edad":35,
```

```
"totalExp":11}); db.employees.insert({"nombre": "Bob", "departamento": "Instalaciones",
"idiomas":["inglés", "hindi"], "edad":36, "totalExp":14}); db.employees.insert({"nombre": "Cathy",
"departamento": "Instalaciones", "idiomas":["hindi"], "edad":31, "totalExp":4});
db.employees.insert({"nombre": "Mike", "departamento": "RRHH", "idiomas":["inglés",
"hindi", "español"], "edad":26, "totalExp":3}); db.employees.insert({"nombre":
"Jenny", "departamento": "RRHH", "idiomas":["inglés", hindi", "español"], "edad":25,
"totalExp":3});
```

Ejemplos por temas:

1. Coincidencia: Se utiliza para emparejar documentos (como la cláusula where de SQL)

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

2. Proyecto: Utilizado para rellenar los valores de campos específicos

la etapa de proyecto incluirá el campo _id automáticamente a menos que especifique desactivarlo.

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

db.employees.aggregate({$proyecto: {'_id':0, 'nombre': 1}})
Salida:
{ "nombre" : "Adma" }
{ "nombre" : "Anna" }
{ "nombre" : "Bob" }
{ "name" : "Cathy" }
{ "nombre" : "Mike" }
{ "nombre" : "Jenny" }
```

3. Group: $group se utiliza para agrupar documentos por un campo específico, aquí los documentos se agrupan por el valor del campo "dept". Otra característica útil es que se puede agrupar por null, lo que significa que todos los documentos serán agregados en uno.

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


{ "_id" : "HR" }

{ "_id" : "Instalaciones" }

{ "_id" : "Admin" }

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

4. Suma: $sum se utiliza para contar o sumar los valores dentro de un grupo.

```
db.employees.aggregate([{$group:{"_id":"$dept", "noOfDept":{$sum:1}}}]);
Salida:
```

```
{ "_id" : "HR", "noOfDept" : 2 }
{ "_id" : "Instalaciones", "noOfDept" : 2 }
{ "_id" : "Admin", "noOfDept" : 2 }
```

5. Media: Calcula la media del valor de un campo específico por grupo.

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

6. Mínimo: Busca el valor mínimo de un campo en cada grupo.

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

7. Máximo: Busca el valor máximo de un campo en cada grupo.

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

8. Obtener el valor de un campo específico del primer y último documento de cada grupo: Funciona bien cuando el resultado del documento está ordenado.

```
db.employees.aggregate([{$group:{"_id":"$edad", "lasts":{$last:"$nombre"},
"firsts":{$first:"$nombre"}}}]);
Salida:
{ "_id" : 25, "hormas" : Jenny", "primeras" : "Jenny" }
{ "_id" : 26, "hormas" : "Mike", primeras" : "Mike" }
{ "_id" : 35, "hormas" : "Cathy", primeras" : "Anna" }
{ "_id" : 30, "hormas" : "Adma", "primeras" : "Adma" }
```

9. Mínimo con máximo:

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

10. Push y addToSet: Push añade el valor de un campo de cada documento del grupo a un array utilizado para proyectar datos en formato array, addToSet es similar a push pero omite los valores duplicados.

```
db.employees.aggregate([{$group:{"_id": "dept", "arrPush":{$push:"$age"}, "arrSet":
{$addToSet:"$edad"}}}]);
Salida:
```

```
{ "_id" : "dept", "arrPush" : [ 30, 35, 35, 35, 26, 25 ], "arrSet" : [ 25, 26, 35, 30 ] }
```

11. Desenrollar: Se utiliza para crear múltiples documentos en memoria para cada valor del campo de tipo array especificado, luego podemos hacer una agregación posterior basada en esos valores.

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

12. Clasificación:

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

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

13. Skip:

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

14. Límite:

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

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

15. Operador de comparación en proyección:

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

16. Operador de comparación en match:

```
db.employees.aggregate([{$match:{dept: "Admin", edad: {$gt:30}}}, {$project:{"nombre":1, "dept":1}}]);

Salida:
```

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

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

17. Operador booleano de agregación en proyección:

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

Salida:
{ "_id" : ObjectId("57ff3e553dedf0228d4862ac"), "name" : "Adma", "dept" : "Admin", "age" : false }

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

18. Operador booleano de agregación en la coincidencia:

```
db.employees.aggregate([{$match:{dept: "Admin", $and: [{edad: { $gt: 30 }}, {edad: {$lt: 36 }} ] }},
{$proyecto:{"nombre":1, "departamento":1, edad: { $y: [ { $gt: [ "$edad", 30 ] }, { $lt: [ "$edad", 36 ]
} ]
}}}]);
Salida:
{ "_id" : ObjectId("57ff3e5e3dedf0228d4862ad")), "name" : "Anna", "dept" : "Admin", "age" : true }
```

> Lista de operadores de agregación booleana: $and, $or y $not.

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

Sección 8.6: Partido

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

Para ello tenemos que escribir una consulta que busque empleados con un salario o igual a $70000. A continuación, añada la etapa de agregados para agrupar a los empleados por departamento. A continuación, agregue un acumulador con un campo llamado, por ejemplo, average_age para encontrar la edad media por departamento utilizando el acumulador $avg y por debajo de los agregados $match y $group agregue otro agregado $match para que sólo estemos recuperando resultados con un average_age que sea mayor o igual a 35.

```
db.empleados.agregar([
  {"$match": {"salario": {"$lte": 70000}}},
  {"$grupo": {"_id": "$dept",
              "average_age": {"$avg": "$edad"}
             }
  },
  {"$match": {"average_age": {"$gte": 35}}}
])
```

El resultado es:

```
{
  "_id": "IT",
  "edad_media": 31
}
{
  "_id": "Servicio de Atención
  al Cliente",
```

```
  "edad_media": 34.5
}
{
  "_id": "Finanzas",
  "edad_media": 32.5
}
```

Sección 8.7: Obtener datos de muestra

Para obtener datos aleatorios de una determinada colección consulte la agregación `$muestra`.

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

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

Sección 8.8: Eliminar documentos que tienen un campo duplicado en una colección (dedupe)

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

```
var duplicados= [];

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

// Eliminar todos los duplicados de una sola vez
db.transactions.remove({_id:{$in:duplicados}})
```

Sección 8.9: Left Outer Join con agregación ($Lookup)

```
let   col_1=   db.collection('col_1');
let   col_2=   db.collection('col_2');
col_1 .aggregate([
    { $match: { "_id": 1 } },
    {
        $lookup: {
            de: "col_2",
            localField: "id",
```

```
            foreignField: "id",
            as: "nuevo_documento"
        }
    }
],function (err, result){
    res.send(result);
});
```

Esta función fue lanzada recientemente en la **versión 3.2 de** mongodb, que ofrece al usuario la posibilidad de unir una colección con los atributos correspondientes de otra colección.

Documentación sobre Mongodb $LookUp

Sección 8.10: Agregación de servidores

Solución de Andrew Mao. Consultas de agregación media en Meteor

```
Meteor.publish("algunaAgregación", function (args) {
    var sub= this;
    // Esto funciona para Meteor 0.6.5
    var db= MongoInternals.defaultRemoteCollectionDriver().mongo.db;

    // Tus argumentos para la agregación de Mongo. Hazlos como quieras.
    var pipeline= [
        { $match: doSomethingWith(args) },
        { $group: {
            _id: whatWeAreGroupingWith(args),
            count: { $suma: 1 }
        }}
    ];

    db.collection("nombre_coleccion_servidor").aggregate(
        canalización,
        // Necesitamos envolver el callback para que sea llamado en un Fiber.
        Meteor.bindEnvironment(
            function(err, result) {
                // Añadir cada uno de los resultados a la suscripción.
                _.each(result, function(e) {
                    // Generar un id aleatorio desechable para los documentos agregados
                    sub.added("nombre_coleccion_cliente", Random.id(), {
                        clave: e._id.algoDeInterés,
                        count: e.count
                    });
                });
                sub.ready();
            },
            function(error) {
                Meteor._debug( "Error al agregar: "+ error);
            }
        )
    );
});
```

Sección 8.11: Agregación en un método de servidor

Otra forma de hacer agregaciones es utilizando la función `Mongo.Collection#rawCollection()`

Sólo se puede ejecutar en el Servidor.

Aquí tienes un ejemplo que puedes usar en Meteor 1.3 y superiores:

```
Meteor.methods({ 'aggregateUsers'(someId)
   {
      const colección= MyCollection.rawCollection()
      const agregado= Meteor.wrapAsync(colección.agregado, colección)

      const match= { edad: { $gte: 25 } }
      const grupo= { _id:'$edad', totalUsers: { $suma: 1 } }

      const resultados= agregado([
         { $match: match },
         { $grupo: grupo }
      ])

      devolver resultados
   }
})
```

Sección 8.12: Ejemplo de Java y Spring

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

```
    intentar {
        MongoClient mongo= new MongoClient(); DB
        db=  mongo.getDB("so");
        DBCollection colección= db.getCollection("empleados");

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

        //Equivalente a $proyecto
        DBObject projectFields= new BasicDBObject();
        projectFields.put("_id", 1);
        projectFields.put("nombre", 1);
        projectFields.put("dept", 1);
        projectFields.put("totalExp", 1);
        projectFields.put("edad", 1);
        projectFields.put("idiomas", 1);
        DBObject proyecto= new BasicDBObject("$proyecto", projectFields);

        //Equivalente a $group
        DBObject groupFields=  new BasicDBObject("_id", "$dept");
        groupFields.put("ageSet", new BasicDBObject("$addToSet", "$age")); DBObject
        employeeDocProjection= new BasicDBObject("$addToSet", new
BasicDBObject("totalExp", "$totalExp").append("edad", "$edad").append("idiomas", "$idiomas").append("dept",
"$dept").append("nombre", "$nombre"));
        groupFields.put("docs", employeeDocProjection);
        DBObject grupo= new BasicDBObject("$grupo", groupFields);

        //Ordenar los resultados por edad
        DBObject sort= new BasicDBObject("$sort", new BasicDBObject("age", 1));

        Lista DBObject<> aggregationList= new ArrayList<>();
        aggregationList.add(match);
        aggregationList.add(project);
        aggregationList.add(group); aggregationList.add(sort);
```

```
        Salida de agregación= coll.aggregate(aggregationList);

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

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

```
[{
    "_id": "Admin", "ageSet":
    [35.0, 30.0],
    "docs": [{
        "totalExp": 11.0,
        "edad": 35.0,
        "languages": ["english", "hindi"], "dept":
        "Admin",
        "Nombre": "Anna"
    }, {
        "totalExp": 10.0,
        "edad": 30.0,
        "languages": ["alemán", francés", "inglés", hindi"],
        "dept": "Admin",
        "Nombre": "Adma"
    }]

}]
```

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

Capítulo 9: Índices

Sección 9.1: Fundamentos de la creación de índices

Consulte la siguiente colección de transacciones.

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

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

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

Veamos el resultado de la sentencia anterior.

```
[
    {
        "v" : 1,
        "clave" : {
            "_id" : 1
        },
        "nombre" : "_id_",
        "ns" : "documentación_db.transacciones"
    }
]
```

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

Ahora a añadir un índice para el campo cr_dr;

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

El resultado de la ejecución del índice es el siguiente.

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

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

Vamos a ejecutar `db.transactions.getIndexes();` de nuevo.

```
[
```

```
    {
        "v" : 1,
        "clave" : {
            "_id" : 1
        },
        "nombre" : "_id_",
        "ns" : "documentación_db.transacciones"
    },
    {
        "v" : 1,
        "clave" : {
            "cr_dr" : 1
        },
        "nombre" : "cr_dr_1",
        "ns" : "documentación_db.transacciones"
    }
]
```

Ahora puede ver que la colección de transacciones tiene dos índices. El índice `_id` por defecto y `el cr_dr_1` que hemos creado. El nombre es asignado por MongoDB. Puede establecer su propio nombre como a continuación.

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

Ahora `db.transactions.getIndexes();` te dará tres índices.

```
[
    {
        "v" : 1,
        "clave" : {
            "_id" : 1
        },
        "nombre" : "_id_",
        "ns" : "documentación_db.transacciones"
    },
    {
        "v" : 1,
        "clave" : {
            "cr_dr" : 1
        },
        "nombre" : "cr_dr_1",
        "ns" : "documentación_db.transacciones"
    },
    {
        "v" : 1,
        "clave" : {
            "cr_dr" : -1
        },
        "name" : "index on cr_dr desc",
        "ns" : "documentación_db.transacciones"
    }
]
```

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

`Versión≥ 2.4`

Índices hash

Los índices también pueden definirse como *hash*. Esto es más eficaz en las *consultas de igualdad*, pero no lo es en las *consultas de rango*; sin embargo, puede definir índices hash y ascendentes/descendentes en el mismo campo.

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

> db.transactions.getIndexes( [
    {
        "v" : 1,
        "clave" : {
            "_id" : 1
        },
        "nombre" : "_id_",
        "ns" : "documentación_db.transacciones"
    },
    {
        "v" : 1,
        "clave" : {
            "cr_dr" : "hashed"
        },
        "nombre" : "cr_dr_hashed",
        "ns" : "documentación_db.transacciones"
    }
]
```

Sección 9.2: Eliminar un índice

Si se conoce el nombre del índice,

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

Si no se conoce el nombre del índice,

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

Sección 9.3: Índices dispersos e índices parciales

Índices dispersos:

Pueden ser especialmente útiles para campos opcionales pero que también deben ser únicos.

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

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

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

te permitirá seguir teniendo nicks "nulos".

Los índices dispersos son más compactos, ya que omiten/ignoran los documentos que no especifican ese campo. Por tanto, si tienes una colección en la que solo menos del 10% de los documentos especifican este , puedes crear índices mucho más pequeños, lo que permite aprovechar mejor la memoria limitada si quieres hacer consultas del tipo:

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

Índices parciales:

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

Los índices parciales determinan las entradas del índice en función del filtro especificado.

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

Si `el valor` es superior a 5, se indexará `la cocina`. Sí, podemos especificar que una propiedad se indexe en función del valor de otras propiedades también.

Diferencia entre índices dispersos y parciales:

Los índices dispersos seleccionan los documentos a indexar basándose únicamente en la existencia del campo indexado o, en el caso de los índices compuestos, en la existencia de los campos indexados.

Los índices parciales determinan las entradas del índice en función del filtro especificado. El filtro puede incluir campos distintos de las claves del índice y puede especificar condiciones distintas de una mera comprobación de existencia.

Aun así, un índice parcial puede tener el mismo comportamiento que un índice disperso

Ej:

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

> **Nota:** No se pueden especificar al mismo *la* opción *partialFilterExpression* y la opción *sparse*.

Sección 9.4: Obtener índices de una colección

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

Salida

```
[
    {
        "v" : 1,
        "clave" : {
            "_id" : 1
        },
        "nombre" : "_id_",
        "ns" : "documentación_db.transacciones"
    },
    {
        "v" : 1,
        "clave" : {
            "cr_dr" : 1
        },
```

```
        "nombre" : "cr_dr_1",
        "ns" : "documentación_db.transacciones"
    },
    {
        "v" : 1,
        "clave" : {
            "cr_dr" : -1
        },
        "name" : "index on cr_dr desc",
        "ns" : "documentación_db.transacciones"
    }
]
```

Sección 9.5: Compuesto

```
db.people.createIndex({nombre: 1, edad: -1})
```

Esto crea un índice en múltiples campos, en este caso en los campos `nombre` y `edad`. Será ascendente en `nombre` y descendente en `edad`.

En este tipo de índice, el orden de ordenación es relevante, ya que determinará si el índice admite o no una operación de ordenación. La ordenación inversa se admite en cualquier prefijo de un índice compuesto, siempre que la ordenación se realice en el sentido inverso para **todas** claves de la ordenación. En caso contrario, la ordenación de los índices compuestos debe coincidir con el orden del índice.

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

Sección 9.6: Índice único

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

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

Sección 9.7: Campo único

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

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

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

Sección 9.8: Suprimir

Para eliminar un índice puede utilizar el nombre del índice

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

O el documento de especificación del índice

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

Sección 9.9: Lista

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

Esto devolverá una matriz de documentos, cada uno de los cuales describe un índice de la colección de *personas*

Capítulo 10: Operaciones masivas

Apartado 10.1: Convertir un campo a otro tipo y actualizar toda la colección en Bulk

Suele ocurrir cuando se quiere cambiar un tipo de campo por otro, por ejemplo, la colección original puede tener campos "numéricos" o "fecha" guardados como cadenas:

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

El objetivo sería actualizar una colección gigantesca como la anterior para

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

Para datos relativamente pequeños, se puede conseguir lo anterior iterando la colección utilizando una `instantánea` con la función del cursor
`forEach()` y actualizando cada documento de la siguiente manera:

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

Mientras que esto es óptimo para colecciones pequeñas, el rendimiento con colecciones grandes se reduce enormemente, ya que hacer un bucle a través de un gran conjunto de datos y enviar cada operación de actualización por petición al servidor incurre en una penalización computacional.

La API `Bulk()` viene al rescate y mejora enormemente el rendimiento, ya que las operaciones de escritura se envían al servidor sólo una vez en bloque. La eficiencia se logra ya que el método no envía cada solicitud de escritura al servidor (como con la actual sentencia de actualización dentro del bucle `forEach()`) sino sólo una vez cada 1000 solicitudes, lo que hace que las actualizaciones sean más eficientes y rápidas que la actualidad.

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

Actualiza todos los documentos de la colección de clientes cambiando los campos `salario` y `dob` a `numérico` y respectivamente:

```
var bulk= db.test.initializeUnorderedBulkOp(),
    counter= 0; // contador para controlar el tamaño de la actualización por lotes

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

    counter++; // incrementar contador
    if (contador % 1000== 0) {
        bulk.execute(); // Ejecutar cada 1000 operaciones y reinicializar cada 1000 sentencias de
actualización
        a granel= db.test.initializeUnorderedBulkOp();
    }
});
```

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

Utiliza los mismos cursores que arriba pero crea los arrays con las operaciones masivas utilizando el mismo método de cursor `forEach()` para empujar cada documento de escritura masiva al array. Dado que los comandos de escritura no pueden aceptar más de 1000 operaciones, es necesario agrupar las operaciones para tener como máximo 1000 operaciones y reinicializar el array cuando el bucle alcanza la iteración 1000:

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

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

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

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

Capítulo 11: Índice de 2dsphere

Sección 11.1: Crear un índice 2dsphere

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

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

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

```
db.places.insert(
{
  loc : { tipo: "Punto", coordenadas: [ -73.97, 40.77 ] }, name:
  "Central Park",
  categoría : "Parques
})
```

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

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

Capítulo 12: Motores de almacenamiento enchufables

Sección 12.1: WiredTiger

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

En WiredTiger, **no hay actualizaciones in situ**. Si necesita actualizar un elemento de un documento, se insertará un nuevo documento y se eliminará el antiguo.

WiredTiger también ofrece **concurrencia a nivel de documento**. Asume que dos operaciones de escritura no afectarán al mismo documento, pero si lo hace, una operación se rebobinará y se ejecutará más tarde. Es una gran mejora del rendimiento si los rebobinados son poco frecuentes.

WiredTiger soporta **los algoritmos Snappy y zLib para la compresión** de datos e índices en el sistema de ficheros. Snappy es el predeterminado. Es menos intensivo en CPU pero tiene una tasa de compresión menor que zLib.

Cómo utilizar WiredTiger Engine

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

Nota:

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

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

Sección 12.2: MMAP

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

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

Sección 12.3: En memoria

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

Sección 12.4: mongo-rocks

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

Sección 12.5: Fusion-io

Un motor de almacenamiento creado por SanDisk que permite eludir la capa del sistema de archivos del sistema operativo y escribir directamente en

el dispositivo de almacenamiento.

Sección 12.6: TokuMX

Un motor de almacenamiento creado por Percona que utiliza índices de árbol fractal.

Capítulo 13: Controlador Java

Sección 13.1: Obtener datos de recogida con condición

Para obtener datos de la colección `testcollection` en la base de datos `testdb` cuyo `nombre es= dev`

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

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

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

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

Sección 13.2: Crear un usuario de base de datos

Para crear un usuario **dev** con **contraseña password123**

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

Sección 13.3: Crear un cursor de cola

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

Ese código se aplica a la clase `MongoCollection`.

CursorType es un enum y tiene los siguientes valores:

```
Tailable TailableAwait
```

Corresponde a los antiguos (<3.0) tipos DBCursor addOption Bytes:

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

Capítulo 14: Controlador Python

Parámetro	Detalle
hostX	Opcional. Puede especificar tantos hosts como sea necesario. Especificaría varios hosts, por ejemplo, para conexiones a conjuntos de réplica.
:puertoX	Opcional. El valor por defecto es :27017 si no se especifica.
base de datos	Opcional. El nombre de la base de datos a autenticar si la cadena de conexión incluye autenticación. credentialsSi no se especifica /database y la cadena de conexión incluye credenciales, el controlador se autenticará en la base de datos admin.
Opciones	Opciones específicas de conexión

Sección 14.1: Conectar a MongoDB usando pymongo

```
from pymongo import MongoClient uri=

"mongodb://localhost:27017/" client=

MongoClient(uri)

db= client['test_db'] #
o bien
# db= client.test_db

# colección= db['test_collection'] # o
colección= db.test_collection




collection.save({"hola": "mundo"})

print colección.encontrar_uno()
```

Sección 14.2: Consultas PyMongo

Una vez que tienes un objeto de `colección`, las consultas utilizan la misma sintaxis que en el shell de mongo. Algunas pequeñas diferencias son:

- cada clave debe ir entre corchetes. Por ejemplo:

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

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

```
db.find({"frecuencias": {"$existe": True }})
```

- objetos como object ids o `ISODate` se manipulan usando clases python. PyMongo utiliza su propia clase `ObjectId` para tratar con ids de objetos, mientras que las fechas utilizan el paquete estándar `datetime`. Por ejemplo, si quieres consultar todos los eventos entre 2010 y 2011, puedes hacer:

```
from datetime import datetime

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

Sección 14.3: Actualizar todos los documentos de una colección usando PyMongo

Supongamos que necesita añadir un campo a cada documento de una colección.

```
importar pymongo

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

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

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

Capítulo 15: Mongo como fragmentos

Sección 15.1: Configuración del entorno de fragmentación

Miembros del grupo Sharding :

Para la fragmentación hay tres actores.

1. Servidor de configuración
2. Réplicas
3. Mongos

   Para un mongo shard necesitamos configurar los tres servidores anteriores.

Configuración del servidor: añada lo siguiente al archivo conf de mongod

```
fragmentación:
  clusterRole: configsvr
replicación:
  replSetName:< setname>
```

ejecutar : mongod --config

podemos elegir config server como replica set o puede ser un servidor independiente. Basado en nuestros requerimientos podemos elegir el mejor. Si config necesita ejecutarse en un conjunto de réplica necesitamos seguir la configuración del conjunto de réplica

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

Configuración de MongoS: Mongos es la configuración principal del shard. Es un enrutador de

consultas para acceder a todos los conjuntos de réplicas Añada lo siguiente en el archivo conf de

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

mongos

Configurar compartido :

Conecte el mongos vía shell (mongo --host --port)

1. sh.addShard("/s1-mongo1.example.net:27017")
2. sh.enableSharding("")
3. sh.shardCollection("< base de datos >.< colección >", {< clave> :< dirección> })
4. sh.status() // Para asegurar la fragmentación

Capítulo 16: Replicación

Sección 16.1: Configuración básica con tres nodos

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

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

Creación de carpetas de datos

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

Inicio de las instancias de mongod

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

Configuración del conjunto de réplicas

```
mongo --port 27017      // conexión a la instancia 27017

rs.initiate();                  // iniciación del conjunto de réplicas en el 1er
nodo rs.add("<hostname>:27018")   // añadir un 2º nodo
rs.add("<nombredehost>:27019")   // añadir un 3er nodo
```

Probar la configuración

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

```
{
        "set" : "rs0",
        "fecha" : ISODate("2016-09-01T12:34:24.968Z"),
        "miEstado" : 1,
        "término" : NumberLong(4),
        "heartbeatIntervalMillis" : NumberLong(2000),
        "miembros" : [
                {
                        "_id" : 0,
                        "name" : "<hostname>:27017",
                        "health" : 1,
                        "estado" : 1,
                        "stateStr" : "PRIMARY",
                        .........................
                },
                {
                        "_id" : 1,
                        "name" : "<hostname>:27018",
                        "health" : 1,
                        "estado" : 2,
                        "stateStr" : "SECONDARY",
                        .........................
                },
                {
                        "_id" : 2,
                        "name" : "<hostname>:27019",
```

```
                        "salud" : 1,
                        "estado" : 2,
                        "stateStr" : "SECONDARY",
                        .........................
                }
        ],
        "ok" : 1
}
```

Capítulo 17: Mongo como conjunto de réplicas

Sección 17.1: Mongodb como conjunto de réplicas

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

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

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

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

```
- mkdir c:\datos\servidor1 (ruta del archivo de datos para la instancia 1)
- mkdir c:\datos\servidor2 (ruta del archivo de datos para la instancia 2)
- mkdir c:\data\server3 (ruta del archivo de datos para la instancia 3)
```

2. a. Iniciar la primera instancia de mongod

- Abra el símbolo del sistema y escriba lo siguiente, pulse Intro.

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

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

2. b. De forma similar, inicie la segunda instancia de Mongodb

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

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

2. c. Ahora inicie la tercera instancia de Mongodb

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

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

Con las 3 instancias iniciadas, estas 3 instancias son independientes entre sí actualmente. Ahora agrupar estas instancias como un conjunto de réplicas. Hacemos esto con la ayuda de un objeto config.

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

```
mongo --port 37017
```

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

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

este objeto config tiene 2 atributos

1. _id: nombre de la réplica Set ("s0")
2. miembros: [] (members es un array de instancias de mongod. dejémoslo en blanco por ahora, añadiremos miembros mediante el comando push.

3.b Para añadir instancias de mongo al array de miembros del objeto config. En el shell de mongo escribe

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

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

4. Inicie el objeto config mediante el siguiente comando en el shell de mongo.

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

5. Dale unos segundos y ya tenemos un conjunto de réplicas de 3 instancias de mongod ejecutándose en el servidor. escribe el siguiente comando para comprobar el estado del conjunto de réplicas e identificar cuál es la primaria y cuál la secundaria.

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

Sección 17.2: Comprobar los estados del conjunto de réplicas de MongoDB

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

Comando : rs.status()

Conecte uno de los miembros de la réplica y ejecute este comando para obtener el estado completo del

conjunto de réplicas:

```
{
 "set" : "ReplicaName",
 "fecha" : ISODate("2016-09-26T07:36:04.935Z"),
 "miEstado" : 1,
 "término" : NumberLong(-1),
 "heartbeatIntervalMillis" : NumberLong(2000),
 "miembros" : [
         {
                 "_id" : 0,
                 "name" : "<IP>:<PORT>,
                 "salud" : 1,
                 "estado" : 1,
                 "stateStr" : "PRIMARY",
                 "uptime" : 5953744,
                 "optime" : Timestamp(1474875364, 36), "optimeDate"
                 : ISODate("2016-09-26T07:36:04Z"),
                 "electionTime" : Timestamp(1468921646, 1),
                 "electionDate" : ISODate("2016-07-19T09:47:26Z"),
                 "configVersion" : 6,
                 "self" : true
         },
         {
                 "_id" : 1,
                 "name" : "< IP>:<PORT> ",
                 "salud" : 1,
```

```
                        "estado" : 2,
                        "stateStr" : "SECONDARY",
                        "uptime" : 5953720,
                        "optime" : Timestamp(1474875364, 13), "optimeDate"
                        : ISODate("2016-09-26T07:36:04Z"),
                        "lastHeartbeat" : ISODate("2016-09-26T07:36:04.244Z"),
                        "lastHeartbeatRecv" : ISODate("2016-09-26T07:36:03.871Z"),
                        "pingMs" : NumberLong(0),
                        "syncingTo" : "10.9.52.55:10050",
                        "configVersion" : 6
                },
                {
                        "_id" : 2,
                        "name" : "< IP>:<PORT> ",
                        "salud" : 1,
                        "estado" : 7,
                        "stateStr" : "ARBITER",
                        "uptime" : 5953696,
                        "lastHeartbeat" : ISODate("2016-09-26T07:36:03.183Z"),
                        "lastHeartbeatRecv" : ISODate("2016-09-26T07:36:03.715Z"),
                        "pingMs" : NumberLong(0),
                        "configVersion" : 6
                },
                {
                        "_id" : 3,
                        "name" : "< IP>:<PORT> ",
                        "salud" : 1,
                        "estado" : 2,
                        "stateStr" : "SECONDARY",
                        "uptime" : 1984305,
                        "optime" : Timestamp(1474875361, 16), "optimeDate"
                        : ISODate("2016-09-26T07:36:01Z"),
                        "lastHeartbeat" : ISODate("2016-09-26T07:36:02.921Z"),
                        "lastHeartbeatRecv" : ISODate("2016-09-26T07:36:03.793Z"),
                        "pingMs" : NumberLong(22),
                        "lastHeartbeatMessage" : "sincronizando desde:
                        10.9.52.56:10050", "syncingTo" : "10.9.52.56:10050",
                        "configVersion" : 6
                }
        ],
        "ok" : 1
        }
```

A partir de lo anterior podemos conocer el estado de todo el conjunto de réplicas

Capítulo 18: MongoDB - Configurar un ReplicaSet para soportar TLS/SSL

¿Cómo configurar un ReplicaSet para que admita TLS/SSL?

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

¿Cómo conectar su Cliente a este ReplicaSet?

Conectaremos un Mongo Shell.

Una descripción de TLS/SSL, certificados PKI (Infraestructura de Clave Pública) y Autoridad de Certificación está más allá del alcance de esta documentación.

Sección 18.1: ¿Cómo configurar un ReplicaSet para que admita TLS/SSL?

Crear el certificado raíz

El Certificado Raíz (también conocido como Archivo CA) se utilizará para firmar e identificar su certificado. Para generarlo, ejecute el siguiente comando.

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

Guarda con cuidado el certificado raíz y su clave, ambos se utilizarán para firmar tus certificados. El certificado raíz también podría ser utilizado por tu cliente.

Generar las solicitudes de certificado y las claves privadas

Al generar la solicitud de firma de certificado (CSR), **introduzca el nombre de host (o IP) exacto de su nodo en el campo Nombre común (CN). Los demás campos deben tener exactamente el mismo valor.** Puede que tengas que modificar tu archivo */etc/hosts*.

Los siguientes comandos generarán los archivos CSR y las claves privadas RSA (4096 bits).

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

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

Ahora debe tener 3 CSR y 3 claves privadas.

```
mongodb_nodo_1.key - mongodb_nodo_2.key - mongodb_nodo_3.key
mongodb_nodo_1.csr - mongodb_nodo_2.csr - mongodb_nodo_3.csr
```

Firme sus solicitudes de certificados

Utilice el archivo CA (ca.pem) y su clave privada (ca.key) generados anteriormente para firmar cada solicitud de certificado ejecutando los comandos que se indican a continuación.

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

Debe firmar cada CSR.

Ahora debe tener 3 CSRs, 3 claves privadas y 3 certificados autofirmados. Sólo las claves privadas y los certificados serán utilizados por MongoDB.

```
mongodb_nodo_1.key  -  mongodb_nodo_2.key  -  mongodb_nodo_3.key
mongodb_nodo_1.csr  -  mongodb_nodo_2.csr  -  mongodb_nodo_3.csr
mongodb_nodo_1.crt - mongodb_nodo_2.crt - mongodb_nodo_3.crt
```

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

Concatenar cada certificado de nodo con su clave

Ejecute los siguientes comandos para concatenar cada certificado de nodo con su clave en un archivo (requisito de MongoDB).

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

Ahora debe tener 3 archivos PEM.

```
mongodb_nodo_1.pem - mongodb_nodo_2.pem - mongodb_nodo_3.pem
```

Despliegue de su ReplicaSet

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

Ejecute los siguientes comandos para desplegar su ReplicaSet de 3 Nodos escuchando en los puertos 27017, 27018 y 27019.

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

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

Despliegue su ReplicaSet para SSL mutuo / Confianza mutua

Para forzar a su cliente a proporcionar un Certificado de Cliente (SSL Mutuo), debe añadir el Archivo CA cuando ejecute sus instancias.

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

```
mongodb_nodo_2.pem --sslCAFile ca.pem
mongod --dbpath data/data_3 --replSet rs0 --port 27019 --sslMode requireSSL --sslPEMKeyFile
mongodb_node_3.pem --sslCAFile ca.pem
```

Ahora tiene un ReplicaSet de 3 Nodos desplegado en su entorno local y todas sus transacciones están encriptadas. No puede conectarse a este ReplicaSet sin utilizar TLS o sin proporcionar un Certificado de Cliente de confianza por su CA.

Sección 18.2: ¿Cómo conectar su cliente (Mongo Shell) a un ReplicaSet?

Sin SSL mutuo

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

Asumiremos que tus 3 nodos están corriendo en mongo1:27017, mongo2:27018 y mongo3:27019. (Puede que necesites modificar tu archivo */etc/hosts*).

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

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

De lo contrario, deberá proporcionar el archivo CA.

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

Ahora estás conectado a tu ReplicaSet y todas las transacciones entre tu Mongo Shell y tu ReplicaSet están encriptadas.

Con SSL mutuo

Si su ReplicaSet solicita un Certificado de Cliente, debe proporcionar uno firmado por la CA utilizada por el Despliegue de ReplicaSet. Los pasos para generar el certificado de cliente son prácticamente los mismos que para generar el certificado de servidor.

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

```
openssl req -nodes -newkey rsa:4096 -sha256 -keyout mongodb_client.key -out mongodb_client.csr
...
Nombre común (por ejemplo, FQDN del servidor o SU nombre) []: mongo1,mongo2,mongo3
```

Si el campo Nombre común es demasiado largo (más 64 bytes), puede encontrarse con la limitación de tamaño del nombre común. Para evitar esta limitación, debe utilizar SubjectAltName al generar la CSR.

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

```
nombre_distinguido= dn

[ dn ]
CN= .

[ req_ext ]
subjectAltName= @alt_names

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

A continuación, firme la CSR utilizando el certificado y la clave de la CA.

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

Por último, concatena la clave y el certificado firmado.

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

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

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

Ahora estás conectado a tu ReplicaSet y todas las transacciones entre tu Mongo Shell y tu ReplicaSet están encriptadas.

Capítulo 19: Mecanismos de autenticación en MongoDB

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

MongoDB soporta una serie de mecanismos de autenticación que los clientes pueden utilizar para verificar su identidad. Estos mecanismos permiten a MongoDB integrarse en su sistema de autenticación existente.

Sección 19.1: Mecanismos de autenticación

MongoDB soporta múltiples mecanismos de autenticación.

Mecanismos de autenticación de clientes y usuarios

- SCRAM-SHA-1
- Autenticación de certificados X.509
- MongoDB Challenge and Response (MONGODB-CR)
- autenticación proxy LDAP, y
- Autenticación Kerberos

Mecanismos de autenticación interna

- Archivo de claves
- X.509

Capítulo 20: Modelo de autorización de MongoDB

La autorización es básicamente la verificación de los privilegios del usuario. MongoDB soporta diferentes tipos de modelos de autorización. 1. **Control de acceso basado en roles**
 Los roles son un grupo de privilegios, acciones sobre recursos. Que se ganan a los usuarios sobre un espacio de nombres dado (Base de datos). Las acciones se realizan sobre los recursos. Los recursos son cualquier objeto que contiene el estado en la base de datos.

Sección 20.1: Funciones incorporadas

En cada base de datos existen roles de usuario de base de datos y roles de administración de base de datos incorporados.

Funciones de los usuarios de la base de datos

1. `leer`
2. `lecturaescritura`

Capítulo 21: Configuración

Parámetro	Por defecto
systemLog.verbosity	0
systemLog.quiet	false
systemLog.traceAllExceptions	false
systemLog.syslogFacility	usuario
systemLog.path	-
systemLog.logAppend	false
systemLog.logRotate	renombrar
systemLog.destination	stdout
systemLog.timeStampFormat	iso8601-local
systemLog.component.accessControl.verbosity	0
verbosidad.comando.componente.systemLog	0
systemLog.component.control.verbosity	0
systemLog.component.ftdc.verbosity	0
systemLog.component.geo.verbosity	0
systemLog.component.index.verbosity	0
systemLog.component.network.verbo	0
systemLog.component.query.verbosity	0
systemLog.component.replication.verbosity	0
systemLog.component.sharding.verbosity	0
systemLog.component.storage.verbosity	0
systemLog.component.storage.journal.verbosity	0
systemLog.component.write.verbosity	0
processManagement.fork	falso
processManagement.pidFilePath	ninguno
puerto.red	27017
net.bindIp	0.0.0.0
net.maxIncomingConnections	65536
net.wireObjectCheck	true
red.ipv6	falso
net.unixDomainSocket.enabled	true
net.unixDomainSocket.pathPrefix	/tmp
net.unixDomainSocket.filePermissions	0700
net.http.enabled	false
net.http.JSONPEnabled	false
net.http.RESTInterfaceEnabled	false
net.ssl.sslOnNormalPorts	false
net.ssl.mode	desactivado
net.ssl.PEMKeyFile	ninguno
net.ssl.PEMKeyPassword	ninguno
net.ssl.clusterFile	ninguno
net.ssl.clusterContraseña	ninguno
net.ssl.CAFile	ninguno

net.ssl.CRLFile	ninguno
net.ssl.allowConnectionsWithoutCertificates	false
net.ssl.allowInvalidCertificates	false
net.ssl.allowInvalidHostnames	false
net.ssl.disabledProtocols	ninguno
net.ssl.FIPSMode	falso

Sección 21.1: Iniciar mongo con un archivo de configuración específico

Usando la bandera `--config`.

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

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

Capítulo 22: Copia de seguridad y restauración de datos

Sección 22.1: Mongodump básico de la instancia local de mongod por defecto

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

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

Sección 22.2: Mongorestore básico de volcado de mongod local por defecto

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

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

Sección 22.3: mongoimport con JSON

Conjunto de datos de código postal de muestra en zipcodes.json almacenado en c:\sers\yc03ak1\Desktop\zips.json

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

para importar este conjunto de datos a la base de datos denominada "test" y a la colección denominada "zips".

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

- --db : nombre de la base de datos a la que se importarán los datos
- --collection: nombre de la colección de la base de datos en la que se van a mejorar los datos
- --drop : elimina la colección antes de importarla
- --type : tipo de documento que debe importarse. JSON por defecto
- --host : host y puerto de mongodb en el que deben importarse los datos.
- --file : ruta donde se encuentra el archivo json

salida :

```
2016-08-10T20:10:50.159-0700      conectado a: localhost:47019
```

```
2016-08-10T20:10:50.163-0700      dropping: prueba.zips
2016-08-10T20:10:53.155-0700      [################. ........] prueba.zips    2.1 MB/3.0 MB  (68.5%)
2016-08-10T20:10:56.150-0700      [########################]   prueba.zips    3.0 MB/3.0 MB  (100.0%)
2016-08-10T20:10:57.819-0700      [########################]   prueba.zips    3.0 MB/3.0 MB  (100.0%)
2016-08-10T20:10:57.821-0700      importados 29353 documentos
```

Sección 22.4: mongoimport con CSV

Archivo CSV del conjunto de datos de prueba de muestra almacenado en la ubicación c:\sers\yc03ak1\Desktop\testing.csv

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

importar este conjunto de datos a la base de datos denominada "test" y a la colección denominada "sample".

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

- --headerline : utilizar la primera línea del archivo csv como los campos para la salida

del documento json :

```
2016-08-10T20:25:48.572-0700      conectado a: localhost:47019
2016-08-10T20:25:48.576-0700      dropping: test.sample
2016-08-10T20:25:49.109-0700      importado 5 documentos
```

O

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

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

```
2016-08-10T20:26:48.978-0700      conectado a: localhost:47019
2016-08-10T20:26:48.982-0700      dropping: test.sample
2016-08-10T20:26:49.611-0700      importado 6 documentos
```

Capítulo 23: Actualización de la versión de MongoDB

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

Sección 23.1: Actualización a 3.4 en Ubuntu 16.04 usando apt

Debe tener 3.2 para poder actualizar a 3.4. Este ejemplo asume que está usando `apt`.

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

Asegúrese de que la nueva versión se está ejecutando con `mongo`. El intérprete de comandos imprimirá la versión del servidor MongoDB que debe ser
3,4 ahora.

Créditos

Muchas gracias a todas las personas de Stack Overflow Documentation que ayudaron a proporcionar este contenido, se pueden enviar más cambios aweb@petercv.com para que se publiquen o actualicen nuevos contenidos.

Abdul Rehman Sayed	Capítulo 1
ADIMO	Capítulo 16
Antti_M	Capítulo 23
Ashari	Capítulo 1
Avindu Hewa	Capítulo 4
bappr	Capítulo 18
Batsu	Capítulo 9
chridam	Capítulo 10
Constantin Guay	Capítulos 9 y 12
Derlin	Capitulo 14
dev ツ	Capítulo 13.
Emil Burzo	Capítulo 13
Enamul Hassan	Capítulos 1 y 8
fracz	Capítulos 2 y 3
uva	Capítulo 8
gypsyCoder	Capitulo 11
HoefMeistert	Capítulo 8
ipip	Capítulo I
Ishan Soni	Capítulo 2
Jain	Capítulo 2
jerry	Capítulo 2
JohnnyHK	Capítulo 2
Juan Carlos Farah	Capítulo 9
Kelum Senanayake	Capítulo 2
KrisVos130	Capítulo 2
Kuhan	Capítulo 6
Lakmal Vithanage	Capítulos 2 y 8
LoicM	Capítulo 8
Luzan Baral	Capítulo 19
Marco	Capitulo 2
Matt Clark	Capítulo 21
Nic Cottrell	Capítulo 9 Niroshan
Ranapathi	Capítulos 19 y 20
oggo	Capítulo 4
Prosen Ghosh	Capítulos 1, 2, 4 y 7
RaR	Capítulos 8 y 9
Renukaradhya	Capítulos 1 y 2
Rotem	Capítulo 2
Sean Reilly	Capitulo 1
Selva Kumar	Capítulo 15
sergiuz	Capítulo 14
Shrabanee	Capítulo 2
SommerIngeniería	Capítulo 4
steveinatorx	Capítulo 8
styvane	Capítulo 2
Thomas Bormans	Capítulo 2
tim	Capítulo 12

titogeo	Capítulos 1, 8 y 9
Tomás Cañibano	Capítulos 2 y 9
usuario641887	Capítulos 17 y 22
WAF	Capítulo 1
amarilloB	Capítulo 5
Zanon	Capítulo 12

También le puede interesar