



MONGO DB EN ESPAÑOL

TOMO 1 · EL PRINCIPIO

Yohan Graterol



Tabla de Contenido

Agradecimientos.....	3
Acerca del autor.....	4
Lo que debes saber.....	5
Capítulo 1: Introducción a NoSQL.....	6
1.1. ¿Qué es NoSQL?.....	6
1.2. ¿Qué NO es NoSQL?.....	7
1.3. Tipos de bases de datos NoSQL.....	7
1.3.1. Bases de datos orientadas a documentos.....	8
1.3.2. Bases de datos orientadas a clave/valor.....	9
1.3.3. Bases de datos orientadas a grafos.....	9
1.4. Sistema de gestión de bases de datos (SGBD).....	10
1.5. Lista de SGBD NoSQL.....	10
1.6. ¿Por qué usar NoSQL?.....	12
Capítulo 2: Conociendo MongoDB.....	14
2.1. MongoDB.....	14
2.2. Términos básicos entorno a MongoDB.....	14
2.2.1. JSON - JavaScript Object Notation.....	14
2.2.2. Documento.....	15
2.2.3. Colección.....	15
2.3. Instalando MongoDB.....	15
2.3.1. Instalación en Linux desde la fuente.....	15
2.3.2. Instalación en Linux desde los repositorios.....	16
2.3.2.1. Instalación en Fedora Linux, Red Hat Linux Enterprise y otras distribuciones.....	16
2.3.3. Instalación en Mac OS X.....	17
2.4. La consola interactiva de MongoDB.....	18
2.4.1. Ayuda en la consola interactiva.....	19
2.5. Conectando a una base de datos.....	19
2.5.1. Seleccionando la base de datos.....	19
2.6. Nuestro primer documento.....	20
3. CRUD en MongoDB.....	21
3.1. ¿Qué es CRUD?.....	21
3.2. Create.....	22
3.2.1. insert().....	22
3.2.2. save().....	23
3.3. Read.....	23
3.3.1. find().....	24
3.3.2. findOne().....	25
3.4. Update.....	26
3.4.1. update().....	26
3.5. Delete.....	26
3.5.1. remove().....	27
3.5.2. drop().....	27
Lo que queda pendiente.....	29
Aportes.....	29



Agradecimientos

A mis padres, mi hermano y mi novia por todo lo que me han podido brindar en mi desarrollo profesional y personal.



Acerca del autor

Mi nombre es **Yohan D. Graterol**, soy un programador y activista pro-FOSS (Free and Open Source Software), me interesa mucho llevar temas interesantes y novedosos al público hispanoparlante, y he dictado conferencias en varios países de Latinoamérica sobre: programación, bases de datos no relacionales, computación en la nube, y otros temas relacionados con software libre. Soy miembro del proyecto Fedora, y estoy en constante formación sobre las últimas tecnologías que las grandes comunidades de software libre nos brindan.

Mi correo de contacto es yograterol@fedoraproject.org y mi cuenta de Twitter es [@yograterol](https://twitter.com/yograterol).



Lo que debes saber

Antes de entrar al mundo de noSQL debes conocer conceptos como: **¿Qué es una base de datos?, ¿Para que sirven?, y ¿Cuales son las aplicaciones fundamentales en el mundo real?**, saber sobre bases de datos relacionales es opcional, ya que cambia el paradigma en el tratamiento de la información.

Este libro estará conformado por varios tomos que tratarán de abarcar todos los puntos importantes en **consultas, administración y escalamiento** de una base de datos documental. Este primer tomo está enfocado 100% a la parte básica del motor de bases de datos MongoDB.



Capítulo 1: Introducción a NoSQL

1.1. ¿Qué es NoSQL?

NoSQL o "No solamente SQL" (Not Only SQL) es un término acuñado por Carlo Strozzi en 1998 y nuevamente retomado por Eric Evans en 2009 y se refiere a un conjunto de bases de datos que se diferencian en gran parte de las bases de datos convencionales, en características tanto de uso como de implementación; estos tipos de bases de datos no usan SQL o al menos no como lenguaje predeterminado para realizar las consultas. Las bases de datos NoSQL, no soportan totalmente ACID, esto lo explica el teorema del profesor Eric Brewer, Teorema CAP (2000), donde menciona:

Es imposible para un sistema distribuido garantizar simultáneamente las siguientes tres características:

- **Consistency (Consistencia):** todos los nodos ven la misma data al mismo tiempo.
- **Availability (Disponibilidad):** una garantía de que todos los requerimientos recibirán una respuesta de que el requerimiento fue exitoso o fallido.
- **Partition Tolerance (Tolerancia a la Partición):** el sistema continúa operando a pesar de la pérdida arbitraria de mensajes, o la falla de parte del sistema.

En primera instancia es una desventaja, pero gracias a esto permite que los motores de bases de datos no relacionales escalen fácilmente de manera horizontal. Para subsanar el problema de ACID, nuevamente el profesor Brewer ideó BASE (Basically Available, Soft-state, Eventually consistent) que lo conforman los siguientes puntos:



- **Disponibilidad básica:** para cada solicitud se garantiza una respuesta, satisfactoria o falla de ejecución.
- **Estado "Soft":** El estado del sistema puede cambiar con el tiempo, a veces sin cualquier entrada.
- **Consistencia Eventual:** La base de datos puede ser en un momento inconsistente pero será consistente con el tiempo.

El lenguaje SQL no es un lenguaje predominante entre los distintos tipos de bases de datos no relacionales, por lo general cada motor tiene su propio lenguaje de consultas. Cabe destacar que la información no se almacena con un esquema fijo (pero si usando almacenamiento estructurado), aun que si existe un esquema que el DBA o el desarrollador propone con anterioridad de manera virtual, es decir, no se crea en el motor antes de utilizar la base de datos sino al almacenar el primer valor.

1.2. ¿Qué NO es NoSQL?

El término NoSQL no es una base de datos y tampoco un tipo de base de datos, sino una definición que engloba un conjunto de tipos de bases de datos que difiere con las bases de datos convencionales.

1.3. Tipos de bases de datos NoSQL

En el mundo de las bases de datos no relacionales nos encontramos con



distintos modelos o tipos, que se desempeñan mejor en algunos ambientes específicos; esas distintas facetas no se ven en las base de datos relacionales. En este libro se expondrán los tipos más comunes.

1.3.1. Bases de datos orientadas a documentos

Las bases de datos orientadas a documentos o también denominadas como Bases de datos documental, trabajan bajo el marco de la definición de un "Documento", donde cada motor que usa esta definición difiere en los detalles, pero la mayoría concuerda en como se almacena la información con algún formato estándar. Los formatos más utilizados por los motores más populares son: JSON y BSON. Se podría considerar este tipo como el más utilizada en la actualidad.

Cada documento, es muy similar a un registro en una base de datos relacional, donde se puede observar un esquema parecido mas no rígido. Dos documentos no tienen porque tener un esquema igual, aunque sean de una misma colección de datos.

```
{
  _id : 1 ,
  nombre : "MongoDB" ,
  url : " http : //www. mongodb . org " ,
  tipo : " Documental "
}
```

Ejemplo de un documento en Formato JSON.

Este ejemplo demuestra la sencillez de un documento, se observa un modelo al estilo **clave : valor**. Una analogía con las bases de datos relacionales sería: Clave = Campo y Valor = Dato del campo, hasta allí queda la analogía.



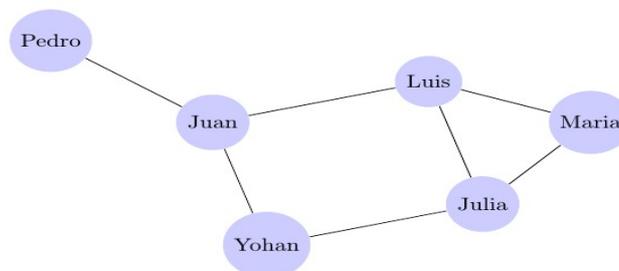
1.3.2. Bases de datos orientadas a clave/valor

Este tipo de bases de datos es muy similar a las bases de datos documental en el concepto de guardar la información con el modelo clave:valor, la diferencia radica en que un documento se almacena en una clave; esta definición puede parecer algo abstracta. Esto se explica mejor con un ejemplo. El siguiente ejemplo utiliza el documento de la sección anterior:

```
mongodb => {  
  _id : 1 ,  
  nombre : "MongoDB" ,  
  url : " http://www.mongodb.org " ,  
  tipo : " Documental "  
}
```

La clave en este caso es 'mongodb' y su contenido es el mismo documento de la sección anterior. Esto hace que varíe la forma de recuperar la información con respecto a las bases de datos basadas en documentos. Algun muy interesante de este tipo es que permite ser utilizado junto bases de datos orientadas a documentos, lo que origina motores híbridos.

1.3.3. Bases de datos orientadas a grafos



Ejemplo de un grafo de relaciones de conocidos.



Este tipo difiere completamente a los tipos antes mencionados, y trata la información de una manera peculiar usando grafos y teoría de grafos. Cada nodo solo debe contener una sola columna, por lo tanto se debe normalizar completamente las bases de datos. Y como la definición de grafos indica, las relaciones solo pueden ser binarias, es decir, un nodo puede solo usar una relación para entrar en contacto con otro nodo y no más de uno.

Las ventajas de este tipo de bases de datos van enfocadas a la integridad de los datos, cualquier cambio en un nodo o relación solo afecta localmente.

1.4. Sistema de gestión de bases de datos (SGBD)

Jorge Sánchez Asenjo (2005) define SGBD4 como:

Un sistema gestor de bases de datos o SGBD es el software que permite a los usuarios procesar, describir, administrar y recuperar los datos almacenados en una base de datos.

Un tipo de base de datos no sirve de nada sino tiene un sistema que lo gestione, a menos que desees crear un SGBD. En NoSQL hay una basta gama de SGBD, y la mayoría están bajo licencia de código libre, permitiendo así usar, estudiar, modificar y redistribuir sin problema alguno con respecto a algunos motores de bases de datos relacionales con licencias privativas.

1.5. Lista de SGBD NoSQL

- Bases de datos documental



- MongoDB
 - Lanzamiento: 2008
 - Licencia: GNU AGPL v3.0
- CouchDB
 - Lanzamiento: 2005
 - Licencia: Apache License 2.0
- Raven DB
 - Lanzamiento: 2010
 - Licencia: GNU AGPL v3.0
- Bases de datos clave/valor
 - Apache Cassandra
 - Lanzamiento: 2008
 - Licencia: Apache License 2.0
 - Riak
 - Lanzamiento: 2009
 - Licencia: Apache License 2.0
 - Redis
 - Lanzamiento: 2009



- Licencia: BSD
- Bases de datos en grafos
 - Neo4j
 - Lanzamiento: 2009
 - Licencia: GNU AGPL v3.0
 - Dex
 - Lanzamiento: 2008
 - Licencia: Comercial
 - Sones GraphDB
 - Lanzamiento: 2012
 - Licencia: GNU AGPL v3.0 y comercial

1.6. ¿Por qué usar NoSQL?

En esta época donde se generan cantidades enormes de datos menos estructurados, las bases de datos relacionales empiezan a mostrar deficiencias, en almacenamiento u operaciones; siendo esta una de las principales razones de impulsar el uso de bases de datos no relacionales. Muchas personas se quejan del movimiento NoSQL, más que todo por una resistencia al cambio, que por los contras de este tipo de bases de datos; en la actualidad gestionar una cantidad de



datos gigantesca no es tan sencillo si piensas en estructuras. Otra de las razones relevantes es la arquitectura, que permite escalar horizontalmente de manera sencilla sin tantos problemas de rendimiento.

La velocidad de desarrollo y la velocidad de la base de datos son puntos a favor para las bases de datos no relacionales, reduciendo el tiempo de desarrollo evitando complejas sentencias SQL y además aumentando la velocidad de respuestas para los clientes.



Capítulo 2: Conociendo MongoDB

2.1. MongoDB

MongoDB es un sistema de bases de datos no relacionales, multiplataforma e inspirada en el tipo de bases de datos documental y clave/valor, su nombre proviene del término en inglés "humongous". Está liberada bajo licencia de software libre, específicamente GNU AGPL 3.0 . MongoDB usa el formato BSON (JSON Compilado) para guardar la información, dando la libertad de manejar un esquema libre. Este motor de bases de datos es uno de los más conocidos y usados, pudiéndolo comparar en popularidad con MySQL en el caso de las bases de datos relacionales.

El desarrollo de MongoDB comenzó en el año 2007 por la empresa 10gen3 , publicando una versión final en el 2009. Para la fecha que es escrito este libro, MongoDB se encuentra en la versión 2.6.1.

2.2. Términos básicos entorno a MongoDB

2.2.1. JSON - JavaScript Object Notation

JSON es formato compacto de representacion de objetos. Las especificaciones las publicó Douglas Crockford en el documento RFC 46274. JSON es un formato independiente del lenguaje, aunque su uso extendido hasta hace poco era en el lenguaje Javascript. Actualmente se usa JSON en gran



cantidades de sistemas para intercambiar información por su simplicidad en comparación con XML.

Este formato soporta gran cantidad de tipos de datos, lo que lo hace atractivo para un uso generalizado, y cada vez más lenguajes de programación dan soporte a este formato. El ejemplo del capítulo anterior, donde se mostraba un "documento", no es más que JSON.

2.2.2. Documento

Un documento es un conjunto de datos estructurados (mas no con un esquema estricto), que contiene pares clave/valor, y se usa BSON (JSON Binario) como formato para almacenar los documentos. Un documento puede ser comparado con una fila o registro en una base de datos relacional.

2.2.3. Colección

Es un conjunto de documentos, similar a una tabla en las bases de datos relacionales.

2.3. Instalando MongoDB

2.3.1. Instalación en Linux desde la fuente

Existen distintas formas de instalar MongoDB en Linux, una de ellas, y la menos recomendable es compilar el código fuente que pueden descargar desde: <http://www.mongodb.org/downloads>. También se puede descargar los binarios,



descomprimirlos y usarlos.

2.3.2. Instalación en Linux desde los repositorios

Los sistemas Linux a diferencia de otros SO, maneja sus software en repositorios, que no es más que un sitio centralizado donde se almacenan todos los software disponible para una distribución de Linux.

2.3.2.1. *Instalación en Fedora Linux, Red Hat Linux Enterprise y otras distribuciones.*

Instalación en Fedora / Red Hat Linux Enterprise / CentOS

Para instalar MongoDB en alguna de estas distribuciones, se debe hacer uso del gestor de paquetes *yum* y ejecutar el siguiente comando:

```
yum install mongodb-server mongodb
```

Donde cada componente es vital según lo que se quiera hacer con MongoDB, a continuación se muestra todas las herramientas según el paquete seleccionado.

- **mongodb:** Contiene todos los paquetes “cliente”, como es el caso del cliente mongo, la herramienta para respaldos en binarios de bases de datos mongodump, mongorestore para recuperar respaldos en binario, mongoexport y mongoimport que realizan una acción similar a mongodump y mongorestore, pero usan formato JSON o CSV.
- **mongodb-server :** Contiene todos los paquetes para hacer funcionar el



servidor, como el demonio mongod.

Los componentes *mongodb* y *mongodb-server* se pueden instalar independientemente. **¿Cuándo hacer eso?** Un caso muy representativo es cuando se desea colocar en producción la base de datos. Se recomienda solo instalar los paquetes de servicio *mongodb-server* y el cliente en otro servidor.

Para iniciar/reiniciar o apagar el demonio de MongoDB en **Fedora** se utiliza el siguiente comando:

```
systemctl start | restart | stop mongod.service
```

En el caso de **Red Hat Enterprise Linux 6** o derivados:

```
service mongod start | restart | stop
```

Para activar el inicio de MongoDB en el arranque del sistema debe ejecutar según sea el caso lo siguiente:

- **Fedora:** *systemctl enable mongod.service*
- **Red Hat Enterprise Linux 6 o derivados:** *chkconfig mongod on*

Instalación en Debian

En el caso de Debian, en un solo paquete se encuentra la distribución completa de MongoDB y para instalarlo se utiliza el gestor de paquetes *apt-get*:

```
apt-get install mongodb
```

2.3.3. Instalación en Mac OS X

Instalación Manual

Puede descargar la última versión disponible de MongoDB para Mac OS X



usando cURL, descomprimirlo y colocarlo en una carpeta a conveniencia.

```
curl -O http://downloads.mongodb.org/mongodb-osx-x86_64-XXX.tgz
```

Siendo XXX, la versión disponible. Ahora, se descomprime el archivo usando la herramienta tar.

```
tar -zxvf mongodb-osx-x86_64-XXX.tgz
```

Con todos los archivos de MongoDB en su Mac OS X, solo debe ingresar a la carpeta resultante y ejecutar el demonio *mongod*.

Usando Homebrew

```
brew update  
brew install mongodb
```

2.4. La consola interactiva de MongoDB

Ya con la previa instalación de MongoDB y sus herramientas, podemos acceder a su consola interactiva y realizar nuestras primeras interacciones con MongoDB.

```
→ ~ mongo  
MongoDB shell version: 2.4.6  
connecting to: test  
> █
```

Al iniciar la consola se conecta automáticamente a la base de datos "test", y a partir de allí podemos realizar consultas sobre esa base de datos. La consola es importante para administrar MongoDB, puede parecer desafiante para los que no están acostumbrado a usar herramientas en consola, pero la gente de 10gen pensó en eso y agregó comandos fáciles de recordar.



2.4.1. Ayuda en la consola interactiva

Para acceder a la ayuda de MongoDB en la consola, se utiliza el comando *help* y nos muestra el contenido básico de la ayuda.

2.5. Conectando a una base de datos

La conexión a bases de datos desde la consola o a través de algún lenguaje es muy sencillo; aún así en MongoDB NO se crean las bases de datos antes de usarla. En bases de datos relacionales, se debe crear toda una estructura inicial para poder almacenar información, al menos se debe tener una base de datos con una tabla, eso en MongoDB no se hace. Para crear una base de datos se debe seleccionar, luego almacenar un documento creando una colección de documentos.

2.5.1. Seleccionando la base de datos

Antes de seleccionar una base de datos, uno tiene la opción de ver el listado de bases de datos que existen en el sistema, con el comando *show dbs*; la salida de este comando nos muestra el nombre y el tamaño de la base de datos, es importante mencionar que MongoDB al crear el primer documento reserva espacio en disco, como mínimo 200 mb.

Luego de saber la lista de bases de datos existente, se debe seleccionar una (no es obligatorio que esté en la lista), para seleccionar la base de datos se hace



uso del comando *use basededatos*, y ya podemos trabajar con la base de datos existente u operar para crear una nueva.

2.6. Nuestro primer documento

Llegó la hora de crear una base de datos y una colección, y eso se hará almacenando un documento usando el objeto *db*, previa ejecución del comando *use*. Un documento puede tener en teoría un máximo de hasta **16MB** de información.

Aprovechando el ejemplo del primer capítulo, para por fin almacenarlo en MongoDB y consultarlo. Una de las cosas que hay que tener en cuenta usando la consola interactiva, es usar variables para crear o modificar documentos, de esta manera podemos evitar accidentes con una mala manipulación directa de la base de datos.

Para almacenar un documento debemos ejecutar el método *.insert()* del objeto *db*, especificando el nombre de la colección (la colección se crea de manera dinámica como la base de datos). Ejemplo:

```
> documento = {
  _id : 1 ,
  nombre : "MongoDB" ,
  url: " http://www.mongodb.org" ,
  tipo: "Documental"
}
> db.nueva_coleccion.insert(documento)
```

De esta manera tenemos nuestra primera colección y nuestro primer documento, para confirmar esto, podemos ejecutar tanto el comando *show*



collections como el método *.find()*.

```
> show collections
nueva_coleccion
system.indexes
```

system.indexes es una colección que usa MongoDB para almacenar los índices de la colección. Por defecto *_id* es un índice.

```
> db.nueva_coleccion.find( )
{
  "_id" : 1 ,
  "nombre" : "MongoDB" ,
  "url": "http://www.mongodb.org" ,
  "tipo" : "Documental"
}
```

Con *.find()* se puede comprobar que efectivamente se almacenó el documento en la colección *nueva_coleccion*.

3. CRUD en MongoDB

3.1. ¿Qué es CRUD?

El propósito de las bases de datos no es solo almacenar información, sino manipular esa información de distintas maneras con el objetivo de alimentar un conjunto de procesos previamente establecidos en los sistemas informáticos.

En los sistemas de base de datos existen 4 funciones básicas las cuales son: Crear, Leer, Modificar y Eliminar (Create, Read, Update and Delete - CRUD), de estas 4 funciones básicas derivan todo un sin fin de funciones para el



tratamiento de la información.

MongoDB provee un grupo de métodos en JavaScript para realizar CRUD en nuestras bases de datos a través de la consola y los drivers de los distintos lenguajes de programación.

3.2. Create

Crear es la primera de las 4 funciones elementales a la hora de utilizar bases de datos, nos permite insertar unidades de información el caso de MongoDB a través de documentos en colecciones de datos. Para poder crear un documento MongoDB posee el método *.insert()*, *.save()* y en de una manera especial *.update()*.

3.2.1. insert()

Este método agrega un documento o un arreglo de documentos en una colección. Su uso es muy sencillo y da la posibilidad de agregar un documento con o sin campo `_id`.



```

MongoDB shell version: 2.4.6
connecting to: test
> documento = {" titulo": "MongoDB"
... "autores": [ "Yohan Graterol" , "Flavio Percoco" ] }
{
  " titulo" : "MongoDB",
  "autores" : [
    "Yohan Graterol",
    "Flavio Percoco"
  ]
}
> db.libro.insert(documento)
> █

```

Ejemplo de inserción de un documento

```

MongoDB shell version: 2.4.6
connecting to: test
> arreglo = [
... {"documento": 1},
... {"documento": 2}
... ]
[ { "documento" : 1 }, { "documento" : 2 } ]
> db.varios.insert(arreglo)

```

Ejemplo de inserción de varios documentos en una colección

3.2.2. .save()

Tiene todas las funciones de .insert(), pero además permite actualizar un documento si ya existe el `_id` de dicho documento, en ese caso .insert() mostraría una excepción.

3.3. Read

Leer es nuestra segunda función elemental en los sistemas de base de datos, y MongoDB utiliza dos métodos para la lectura de los documentos, el primer método es `.find()` que nos permite leer todos los documentos de una colección y `.findOne()` que nos permite leer solo uno.



3.3.1. find()

Selecciona documentos en una colección y devuelve un cursor con documentos. El siguiente ejemplo muestra los documentos de nuestra colección *libro*.

```
MongoDB shell version: 2.4.6
connecting to: test
> db.libro.find()
{ "_id" : ObjectId("5382c84842519556b7de00c5"), " titulo" : "MongoDB", "autores" : [ "Yohan Graterol", "Flavio Percoco" ] }
{ "_id" : ObjectId("5382ca8de6214875e080740a"), "titulo" : "MongoDB Básico", "autores" : [ "Yohan Graterol", "Flavio Percoco" ], "licencia" : "CC", "date" : "2014" }
>
```

El resultado que nos arroja no es muy fácil de leer, ya que no tiene un formato de salida tan claro, pero hay forma de que se muestre de la misma manera que los navegadores muestran el formato JSON en las consolas de desarrolladores, y eso se logra utilizando el comando *pretty()* luego del *find()*.

```

MongoDB shell version: 2.4.6
connecting to: test
> db.libro.find().pretty()
{
  "_id" : ObjectId("5382c84842519556b7de00c5"),
  "titulo" : "MongoDB",
  "autores" : [
    "Yohan Graterol",
    "Flavio Percoco"
  ]
}
{
  "_id" : ObjectId("5382ca8de6214875e080740a"),
  "titulo" : "MongoDB Básico",
  "autores" : [
    "Yohan Graterol",
    "Flavio Percoco"
  ],
  "licencia" : "CC",
  "date" : "2014"
}
>

```

3.3.2. *findOne()*

Al igual que *.find()*, permite realizar búsquedas en una colección con la diferencia que solo devuelve un solo documento. Si no se utilizan parámetros en la búsqueda con *.findOne()*, devuelve solo el primer documento agregado a la colección o el primer documento en orden natural en el disco.

```

MongoDB shell version: 2.4.6
connecting to: test
> db.libro.findOne()
{
  "_id" : ObjectId("5382c84842519556b7de00c5"),
  "titulo" : "MongoDB",
  "autores" : [
    "Yohan Graterol",
    "Flavio Percoco"
  ]
}
>

```

3.4. Update

Actualizar es otra de nuestras funciones elementales, con ella podremos modificar información y en el caso de MongoDB se puede realizar hasta inserción de documentos con el método `.update()`, y además se puede utilizar la función `.save()` antes vista, en actualizaciones simples.

3.4.1. update()

Modifica un documento o un conjunto de documentos. Por defecto solo modifica un documento, pero si la opción *multi* está activada realiza la modificación sobre todos los documentos que cumplan con un criterio dado. Al igual que con `.find()`, `.update()` se puede utilizar con parámetros o criterios de búsquedas. `.update()` se utiliza con la siguiente estructura:

```
db.collection.update (  
  <consulta - criterios>,  
  <documento_modificado >,  
  {upsert: true | false, multi : true | false }  
)
```

La opción *upsert*, permite agregar un documento si no existe, siempre y cuando esta opción esté activada.

3.5. Delete

Eliminar es nuestra última función, y MongoDB utiliza la función `.remove()`



para eliminar documentos y colecciones, en el caso de las colecciones por el tema de rendimiento es mejor utilizar `.drop()`.

3.5.1. `remove()`

Elimina uno o más documentos de una colección. Recibe parámetros para realizar una eliminación selectiva; si no se le pasa ningún parámetro elimina todos los documentos de la colección.

Tomando el `_id` del documento que está en la salida del ejemplo del apartado sobre `.findOne()`, se procederá a borrar dicho documento y consultado de nuevo, se corroborará la eliminación.

```
> db.libro.remove({_id: ObjectId("5382c84842519556b7de00c5")})
> db.libro.findOne()
{
  "_id" : ObjectId("5382ca8de6214875e080740a"),
  "titulo" : "MongoDB Básico",
  "autores" : [
    "Yohan Graterol",
    "Flavio Percoco"
  ],
  "licencia" : "CC",
  "date" : "2014"
}
```

Vemos que la consulta es sencilla, ya que directamente pasamos el parámetro del documento que deseamos eliminar, si pasamos un parámetro que coinciden en varios documentos, se eliminarán todos los documentos con dichas coincidencias.

3.5.2. `drop()`



La función `.drop()` elimina toda una colección, y es la más recomendable a la hora de realizar esta tarea, ya que utiliza menos recursos que `.remove()`.

```
MongoDB shell version: 2.4.6
connecting to: test
> db.libro.drop()
true
> db.libro.find()
>
```

Lo que queda pendiente

Para el próximo tomo del libro quedan pendientes los siguientes temas:

- Consultas avanzadas con MongoDB, en búsqueda, modificación y eliminación de documentos.
- Conexión de Python y NodeJS con MongoDB usando los drivers nativos para ambos lenguajes de programación.
- Deploy y administración en entorno *NIX.
- Escalamiento horizontal.

Aportes

Si deseas realizar algún aporte al libro, puedes enviar un correo a yograterol@fedoraproject.org, indicando sugerencias, errores y lo que te gustaría que se agregara en el tomo o en futuros tomos; todas tus opiniones serán bienvenidas.

