Tutorial de MongoDB: Primeros Pasos

Base de datos NoSQL ,

TUTORIAL DE MONGODB DESDE CERO PARA PRINCIPIANTES

Alternativas a MySQL Server, en ellas aparte de MariaDB, PostgreSQL, entre otros, también mencionamos al todo poderoso MongoDB. Hoy llegó el momento de finalmente introducirnos a full en MongoDB.

¿QUÉ ES MONGODB?

MongoDB es una base de datos NoSQL, donde a diferencia de SQL que maneja una estructura llave-valor, hace hincapié en el valor de las llaves donde las llaves son llamadas colecciones y tienen una estructura tipo JSON llamados documentos. Para quienes dominen javascript, lenguaje sobre el cual se basa esta solución, le será más fácil su manipulación.
En esta primera parte del tutorial de MongoDB, les indicaremos como instalar MongoDB en un Sistema Operativo Linux, particularmente nos basaremos en CentOS/RHEL Linux.

DIFERENCIAS ENTRE MONGODB Y MYSQL

Estructura
La principal diferencia de MongoDB con respecto a MySQL es su estructura, ya que este último es un **gestor de bases de datos orientado a documentos**.
En MySQL todas las filas de una tabla tienen la misma estructura, mientras que en MongoDB esto no se da así. Es decir en MySQL es necesario que cada fila tenga el mismo número de columnas con los mismos tipos de datos, esto no se da así con MongoDB, aquí, los documentos son individuales e incluso nos permite añadir nuevos campos con cualquier valor.

Esto en MySQL significaría reestructurar toda una base de datos, lo que lo hace muy dificultoso. Por otro lado la clave debe ser única dentro del documento de MongoDB, aunque esta puede repetirse entre documentos.

En MySQL manejamos relaciones utilizando joins, esto no es posible en MongoDB aunque no representa un problema gracias a la ventaja de poder repetir claves entre documentos.

Consultas
Con respecto a las consultas, MongoDB no utiliza SQL para realizar consultas sino que posee su propio lenguaje, lo que permite la comunicación con el cliente, esto ayudado por librerías.

PRINCIPALES VENTAJAS DE MONGODB

- Resulta la opción más conveniente cuando se requiere el manejo de grandes cantidades de datos en modo lectura.
- Si se domina una estructura de datos variables, MySQL nos da muchos dolores de cabeza por lo nuevamente MongoDB es superior.
- La posibilidad de tener un gran desempeño en máquinas de menor rendimiento, por lo que lo hace más económico.

INSTALANDO MONGODB

Para seguir con este tutorial de MongoDB es vital comenzar con la instalación de esta base de datos NoSQL, y es requerimiento también que conozcas previamente tu arquitectura de servidor.
Puedes conocer si tu arquitectura es de 32 o 64 bits ejecutando el siguiente comando en tu consola:

```
uname -a
```

En caso que devuelva algo así como x86_64 entonces tu arquitectura es de 64 bits, de lo contrario es de 32 bits.

CREANDO EL REPOSITORIO

Para proceder a la instalación, inicialmente debemos crear el repositorio de MongoDB. Por lo cual es necesario que crees un archivo en el repo del instalador Yum.

No olvides haber accedido como root a tu servidor para no tener ningún problema o limitación.

Utilizando el editor nano en este caso, deberás ejecutar lo siguiente:

```
nano -w /etc/yum.repos.d/mongodb.repo
```

En el añade las siguientes líneas dependiendo de la arquitectura del servidor, añade lo siguiente si es de 64 bits:

```
[mongodb] name=MongoDB Repository baseurl=http://downloads-distro.mongodb.org/repo/redhat/os/x86_64/ gpgcheck=0 enabled=1
```

En caso que tu arquitectura sea de 32 debes colocar lo siguiente:

```
[mongodb]

name=MongoDB Repository

baseurl=http://downloads-distro.mongodb.org/repo/redhat/os/i686/
gpgcheck=0
enabled=1
```

Una vez creado el repositorio, guarda el archivo y sal del editor.

INSTALACIÓN DE MONGODB EN CENTOS

Luego de haber creado el repositorio de MongoDB según nuestra arquitectura correspondiente, debes correr el siguiente comando:

```
yum install mongo-10gen mongo-10gen-server
```

Aquí debes aceptar la instalación luego que el instalador haya chequeado las dependencias, espera a que finalice el proceso.

PRIMEROS PASOS EN MONGODB

Al igual que en los demás servicios, este arranca para y reinicia utilizando los comandos básicos, aquí un detalle de estos:

Para iniciar mongoDB es necesario correr lo siguiente.

```
service mongod start
```

Para parar el servicio mongod realizamos un stop.

```
service mongod stop
```

Finalmente para reiniciarlo lo hacemos de igual forma.

```
service mongod restart
```

Para el caso que todo funcione correctamente MongoDB, responderá a estos comandos con un OK.

CONECTANDO MONGODB REMOTAMENTE

Por defecto, MongoDB corre en el puerto 27017, por lo que es necesario que chequees tu firewall si conectaras al servidor remotamente. Verifica también tu archivo de configuración de MongoDB, el mismo se encuentra en /etc/mongod.conf, debes configurar el puerto, la ruta donde se guardara la BD y la IP a la que estará enlazado.

ESTRUCTURA DE DATOS EN MONGODB

A continuación veremos un ejemplo de la estructura de datos con la que trabajaremos en esta serie de artículos acerca de MongoDB.

En MongoDB existen 2 patrones de modelado que nos ayudan a definir la estructura que tendrán los documentos.Estos patrones son Embeber y Referenciar.

- Embeber: este patrón hace referencia a la incrustación de documentos uno dentro del otro para hacerlo parte del mismo registro, logrando una relación directa.
  Para el caso de las relaciones uno a uno (1-1) utilizaremos este patrón, veamos un ejemplo:

```
Persona = {

nombre : 'Juan',
```

```
apellido : 'Perez',

genero : 'M',

documentos = {

pasaporte : 'D123456V7',

licencia : '34567651-2342',

}

}
```

- Referenciar: Se comporta como claves foráneas para relacionar datos que deben estar en colecciones diferentes.
  Suponiendo que tenemos la siguiente estructura:

```
Persona = {

nombre   : 'Juan',

apellido : 'Perez',

genero   : 'M',

}

Direccion1 = {

País   : 'Estados Unidos',

Estado : 'Miami',

ciudad : 'Florida',

}

Direccion2 = {

Pais   : 'Estados Unidos',

Estado : 'Los Angeles',

ciudad : 'Beverlly Hils',

}
```

Por método relacionar los vincularíamos de la siguiente manera:

```
Persona = {
```

```
nombre   : 'Juan',

apellido : 'Perez',

genero   : 'M',

Direccion:[1,2]
}
Direccion1 = {
_id    : 1,
Pais   : 'Estados Unidos',
Estado : 'Los Angeles',
ciudad : 'Beverly Hills'
}
Direccion2 = {
_id    : 2,
Pais   : 'Estados Unidos',
Estado : 'Miami',
ciudad : 'Florida'
}
```

CONCLUSIÓN

Llegado este punto del tutorial de MongoDB podemos resumir que ahora conocemos qué es MongoDB, lo hemos instalado en nuestro equipo y ya estamos a unos pocos pasos de comenzar a probar este gestor de base de datos.

Actualmente, conocemos en que se diferencia un gestor SQL de otro NoSQL y que ventajas trae este último, pudiendo con estos datos, saber si se adapta o no a nuestro proyecto. Tenemos también una visión de cómo se manejan los datos, y como podemos estructurar nuestro desarrollo.

Tutorial de MongoDB – Parte 02: Sintaxis de Consultas

En el Tutorial de MongoDB: Primeros Pasos, vimos una introducción a MongoDB, realizamos la instalación de este, vimos además como iniciarlo y explicamos de forma general como funciona su estructura de datos.

Continuando con este Tutorial de MongoDB, veremos en esta la segunda parte, algunos ejemplos de ejecución de sentencias en la base de datos, insertando, consultando y eliminando registros. Para ello iremos comentando poco a poco cada una de las líneas de comando que debemos ejecutar en cada caso.

Para basarnos en un entorno real, supondremos un sistema de clientes, en el que propondremos realizar altas, recuerda que podemos realizar inserciones con distinta cantidad de campos, tal como vimos en el tutorial anterior, bajas, modificaciones y algunas consultas.

SINTÁXIS DE CONSULTAS EN MONGODB

Entre las consultas que realizaremos a la base de datos, veremos la sintaxis de algunas consultas específicas que pueden ser de utilidad en todo desarrollo, como por ejemplo: órdenes, filtros y límites.

INSERTAR:

Para comenzar, insertaremos en este caso un cliente con los siguientes datos: su nombre, apellido y edad. La base de datos la llamaremos "clientes".

```
db.clientes.insert({Nombre: 'Carlos', Apellidos: 'Garcia Perez', Edad: 36});
```

Para confirmar si se insertó correctamente este cliente, podremos realizar una búsqueda de todos los registros de la siguiente manera:

```
db.clientes.find();
```

Esta sentencia nos devolverá algo como:

```
{ "_id" : ObjectId("512dc454777b2ff171b49e95"), "Nombre": "Carlos", "Apellidos": "Garcia Perez", "Edad": 36 }
```

INSERTANDO MAS CAMPOS

Tal como definimos en la introducción a MongoDB, la estructura que este posee, nos permite añadir datos a registros similares, sin necesidad de modificar nada, lo que sería imposible en MySQL, por ejemplo añadiremos otro usuario "Valentina", pero esta vez, agregáramos un teléfono para este cliente. En el caso de MySQL, sería necesario reestructurar una tabla, ya que deberíamos agregar una nueva columna, pero en este caso no será necesario realizar ninguna tarea extra simplemente agregaremos este nuevo cliente de la siguiente manera:

```
db.clientes.insert({"Nombre": "Valentina", "Apellidos": 'Garcia Perez', "Edad": 20, "Telefono": 123456789 });
```

CONSULTAS:

La consulta básica, es decir, el listado de los datos que existen en esta base de datos podemos obtenerlo de la siguiente manera:

```
db.clientes.find();
```

Las búsquedas por defecto son *case sensitive*, es decir que son sensibles a mayúsculas y minúsculas, en caso que busquemos carlos en lugar de "Carlos", obtendremos como resultado que no existe, probemos lo siguiente:

```
db.clientes.find({Nombre: 'carlos'});
```

Para evitar el case sensitive, podemos utilizar la siguiente sintaxis:

```
db.clientes.find({Nombre: /^carlos$/i});
```

FILTROS:

Al igual que en MySQL, esta base de datos permite búsquedas mediante el agregado de filtros como por ejemplo filtrar por algún campo en específico.

Si nos interesara filtrar la búsqueda por el nombre de cliente, podemos utilizar lo siguiente:

```
db.clientes.find({Nombre: 'Carlos'});
```

A continuación tomemos el ejemplo de que nos interesa obtener los clientes mayores a 25 años:

```
db.clientes.find({Edad: {$gt: 25} } );
```

Aquí nos devolverá únicamente a Jose, ya que es el mayor a 25 años.

En este otro ejemplo supondremos buscar empleados de entre 36 y 38 años. Aquí obtendremos los dos clientes que hemos insertado ya que están dentro del rango que seleccionamos.

```
db.clientes.find({$or: [{Edad: 36}, {Edad: 38}]} );
```

Ahora plantearemos el siguiente ejemplo, insertar un nuevo cliente "Pedro Garcia Perez", él tiene 39 años, es de Madrid España, es cliente hace 3 años, y cuenta con servicio de hosting y pagina web. Por otro lado, insertaremos al cliente "Jose Lopez Sanchez", él tiene 45 años, es de Madrid España, es cliente hace 2 años y cuenta únicamente con el servicio de hosting.

Para ello ejecutaremos la siguiente sentencia:

```
db.clientes.insert({_id: '10000000L', Nombre: 'Pedro', Apellidos:
'Garcia Perez', antiguedad: 3, Direccion: {city: 'mad', country:
'es'}, Servicios: ['Hosting', 'Sitio Web']} );
```

```
db.clientes.insert({_id: '20000000L', Nombre: 'Jose', Apellidos: 'Lopez Sanches', antiguedad: 2, Direccion: {city: 'mad', country: 'es'}, Servicios: ['Hosting']} ):
```

En este momento disponemos de 4 clientes en nuestra base de datos, lo que nos permitirá probar búsquedas más específicas. Intentemos las siguientes consultas:

Obtener los clientes que tienen servicio de Sitio Web:

```
db.clientes.find({Servicio: 'Sitio Web'});
```

Obtener los clientes que No tienen un servicio asociado.

```
db.clientes.find({Servicios: {$exists: false}});
```

En ocasiones, principalmente estructuras grandes, resulta difícil visualizar los datos, para verlos más ordenadamente podemos utilizar *.pretty():*

```
db.clientes.find({Nombre: 'Carlos'}).pretty();
```

ORDENAR UNA CONSULTA:

Para obtener los resultados de una consulta, en forma ordenada, basándose en alguno de los campos, podemos realizarlo de esta forma:

```
db.clientes.find().sort({antiguedad: -1});
```

En caso de querer limitar el resultado de una consulta, podemos ejecutar lo siguiente:

```
db.clientes.find({Servicios: 'Hosting'}).limit(2);
```

Contar cantidad de registros

```
db.clientes.count();
```

Supongamos ahora que queremos obtener los clientes que viven en una dirección en particular, en este caso ciudad de Madrid (mad):

```
db.clientes.find({'Direccion.ciudad': 'mad'});
```

MODIFICAR

Para comenzar con un caso básico de modificación, podemos intentar modificar el Nombre según un id de cliente:

```
db.clientes.update({_id: '20000000L'}, {$set: {Nombre: 'Eduardo'}}):
```

Ahora vamos a sumar 1 año de antigüedad a este cliente y además le agregaremos un servicio

```
db.clientes.update({_id: '20000000L'}, {$inc: {antiguedad: 1}, $push:
{Servicios: 'Administración'}});
```

ELIMINAR

A continuación veremos dos de los típicos casos de eliminación, uno la eliminación total de los registros y dos eliminar un registro por ID.

Eliminando todos los registros de la base de datos "clientes":

```
db.clientes.remove();
```

Eliminando el cliente con id: 10000000L:

```
db.clientes.remove({_id: '10000000L'});
```

Tutorial de MongoDB – Parte 03 – Utilizando variables y operadores

Comenzamos la tercera parte de Mongodb, como hemos visto hasta el momento ya sabemos que es, como funciona, su sintaxis, hemos realizado además algunas consultas, insertado y eliminado datos. En esta oportunidad vamos a profundizar algo más en sentencias, para poder implementarlas en un futuro en alguna aplicación real.
Primeramente, procederemos a inicializar mongodb de la forma que ya conocemos:

```
service mongod start

mongo
```

VARIABLES:

Ahora que estamos listos, veremos que en Mongodb es posible guardar nuestras consultas en una variable, para ello tomaremos como punto de partida la base de datos de clientes que creamos en el tutorial pasado, veamos un ejemplo acerca de esto.

Crearemos una variable test la cual va a contener un cliente, para obtener un cliente recordemos que podemos utilizar findOne().

```
var test = db.clientes.findOne()
```

Una vez que guardamos a ese cliente en la variable test, podemos imprimir su contenido en pantalla simplemente ejecutando la variable "test" y obtendremos algo como esto:

```
test{ "_id" : ObjectId("594d089576dd7cca79536b2b"), "Nombre" :
"Carlos", "Apellidos" : "Garcia Perez", "Edad" : 36}
```

Una variante a esto es simplemente escribir la variable sin el prefijo var previo.

¿Cuál es la diferencia?, simplemente que nos imprime lo que guardo luego de ejecutarla:

```
test_dos = db.clientes.find({Nombre: "Valentina"})
```

Al ejecutar esta sentencia, nos devolverá lo siguiente:

```
test_dos = db.clientes.find({Nombre: "Valentina"}){ "_id" :
ObjectId("594d08d976dd7cca79536b2c"), "Nombre" : "Valentina",
"Apellidos" : "Garcia Perez", "Edad" : 20, "Telefono" : 123456789 }
```

Como vemos, obtuvimos el cliente de nombre "Valentina" y lo guardamos en la variable test_dos.

Ahora implementemos ejemplo algo más completo, busquemos guardar en una variable, el nombre y edad de un cliente y luego insertar esta variable en nuestra base de datos:

```
Juan = {Nombre : "Juan", Apellidos : "Rodriguez", Edad : 22, Telefono
: 1234116789}
```

Aquí hemos creado nuestro cliente Juan, y lo almacenamos en una variable, ahora debemos insertarlo a la base de datos:

```
db.clientes.insert(Juan)
```

Si fue correctamente insertado debe devolver lo siguiente:

```
WriteResult({ "nInserted" : 1 })
```

Para confirmar hagamos una búsqueda de todos los usuarios:

```
db.clientes.find()

{ "_id" : ObjectId("594d089576dd7cca79536b2b"), "Nombre" : "Carlos",
"Apellidos" : "Garcia Perez", "Edad" : 36 }

{ "_id" : ObjectId("594d08d976dd7cca79536b2c"), "Nombre" :
"Valentina", "Apellidos" : "Garcia Perez", "Edad" : 20, "Telefono" :
123456789 }

{ "_id" : "10000000L", "Nombre" : "Pedro", "Apellidos" : "Garcia
Perez", "antiguedad" : 3, "Direccion" : { "city" : "mad", "country" :
"es" }, "Servicios" : [ "Hosting", "Sitio Web" ] }

{ "_id" : "20000000L", "Nombre" : "Eduardo", "Apellidos" : "Lopez
Sanches", "antiguedad" : 3, "Direccion" : { "city" : "mad", "country"
: "es" }, "Servicios" : [ "Hosting", "Administración" ] }

{ "_id" : ObjectId("595b79e3c19cb31f34a69130"), "Nombre" : "Juan",
"Apellidos" : "Rodriguez", "Edad" : 22, "Telefono" : 1234116789 }
```

Tal como indica el último registro Juan ahora pertenece a nuestra base de datos en Mongodb

Hasta el momento hemos implementado filtros del tipo "igual a" para realizar las búsquedas, a continuación veremos la sintaxis correcta para filtrar una búsqueda por elementos del tipo "no igual a".

```
db.clientes.find( {Nombre:{$ne:22}} )
```

En este caso los registros obtenidos serán todos excepto el de id "595b79e3c19cb31f34a69130" es decir Juan.

ACTUALIZAR DATOS DESDE UNA VARIABLE:

Usemos la variable test que tenemos guardada, en esta variable, intentemos modificar el atributo nombre.
Primero recordemos el nombre que tiene guardado esta variable:

```
test{   "_id"   :   ObjectId("594d089576dd7cca79536b2b"),   "Nombre"   :
"Carlos", "Apellidos" : "Garcia Perez", "Edad" : 36}
```

Ahora modifiquemos este nombre suponiendo un caso en el que hubo un error al ingresarlo:

```
test.Nombre = "Cambio de Nombre"
```

Para poder reflejar este cambio en el registro de la base de datos, debemos salvar los cambios, ¿cómo realizamos esto?

```
db.clientes.save(test)
```

¿Cómo se logró realizar el cambio?, .save busca en la variable que le hemos pasado, en este caso test, si existe un id, al encontrar un id lo compara con los registros de la base de datos, en caso que exista actualiza el registro con los campos guardados en la variable.

```
Que atributos mostrar:
```

Suele ser un poco molesto en algunos casos o innecesario en otros, mostrar todos los campos de un registro, pero podemos elegir que mostrar y que no. Mostremos el nombre y ocultemos el resto de los campos:

```
db.clientes.find({},{Nombre:1})

{ "_id" : ObjectId("594d089576dd7cca79536b2b"), "Nombre" : "Carlos" }{
"_id" : ObjectId("594d08d976dd7cca79536b2c"), "Nombre" : "Valentina" }
```

OPERADORES:

En Mongodb podemos trabajar con operadores directamente sobre nuestra sintaxis de consulta, por ejemplo en caso que requiramos un registro que coincida con "x" valor o también comparaciones.

Estos son los operadores que podemos utilizar:

```
$gt    >    greater than / mayor que

$gte >=  greater than equals / mayor o igual que

$lt    <    less than / menor que

$lte  =<  less than equals /  menor o igual que
```

Para ver un ejemplo realizaremos la búsqueda de un cliente menor a 25 años

```
db.clientes.find({},{Edad: {$lt:25}})
```

También podemos expresar rangos concatenando de la siguiente manera:

```
db.clientes.find({},{Edad: {$gt:20, $lt:25}})
```

Podemos además hacer un conteo de los registros que tenemos ingresados, utilizando .count()

db.clientes.find().count()

BUCLES

Para trabajar con bucles de iteración podemos intentar ingresar a una base de datos "test" un registro valor cuyos datos sean del 0 al 50.

```
for(i=0; i<100; i++){ db.test.insert( { valor : i } ) }
```

Para verificar si efectivamente se guardaron los registros, podemos hacer un find de la tabla test, lo que nos arrojara lo siguiente:

{ "_id" : ObjectId("594d089576dd7cca79536b2b"), "valor" : 0}

{ "_id" : ObjectId("594d089576dd7cca79536b2c"), "valor" : 1}

{ "_id" : ObjectId("594d089576dd7cca79536b2d"), "valor" : 2}

{ "_id" : ObjectId("594d089576dd7cca79536b2e"), "valor" : 3} …

EXPRESIONES REGULARES:

Para utilizar búsquedas mediante expresiones regulares, podemos realizar lo siguiente:

Suponiendo que ya agregamos algunos correos a una base de datos "correos", podemos realizar una búsqueda de todos aquellos que incluyan un "@".

```
db.correos.find({ correo : /@/ })
```

En el caso que además queramos obtener los correos que terminen en ".com", podemos realizar lo siguiente:

```
db.correos.find({ correo : /.com$/ })
```

Todos aquellos que comiencen con una "palabra":

```
db.correos.find({ correo : /^palabra/ })
```