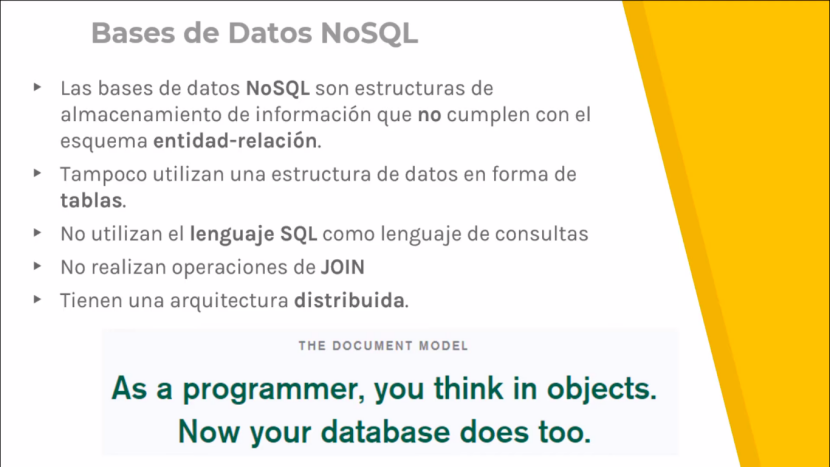
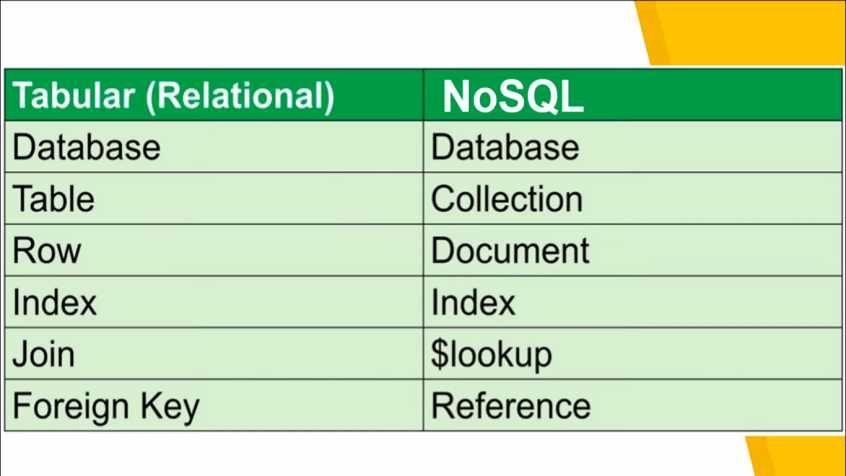




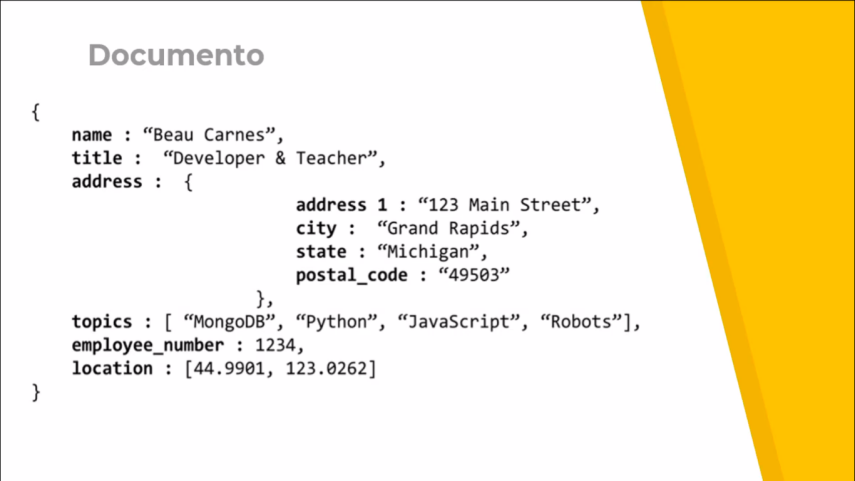
No se refiere a que esto incluya SQL, **se refiere a que no todas las bases de datos son SQL.**

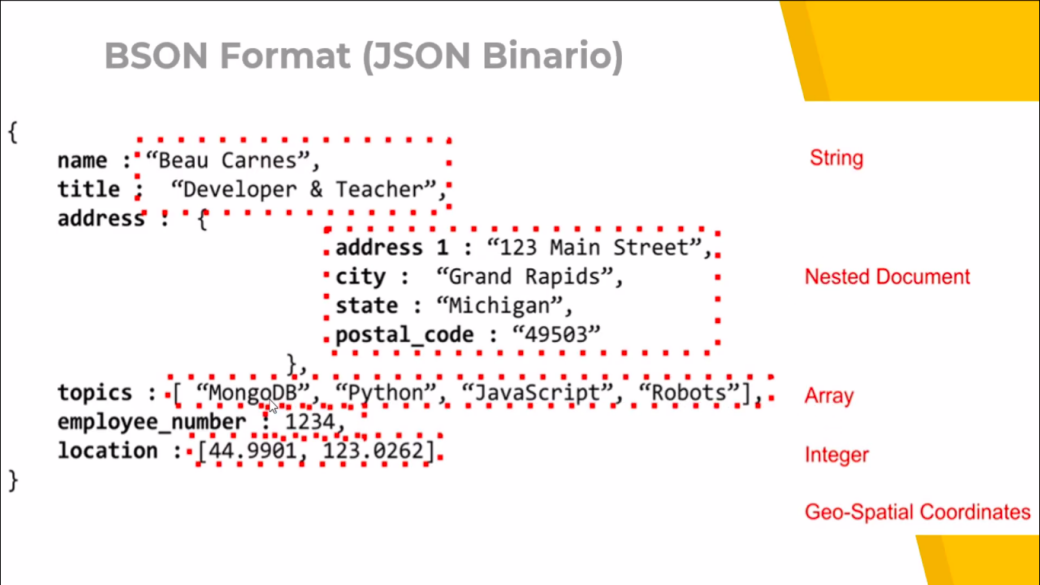


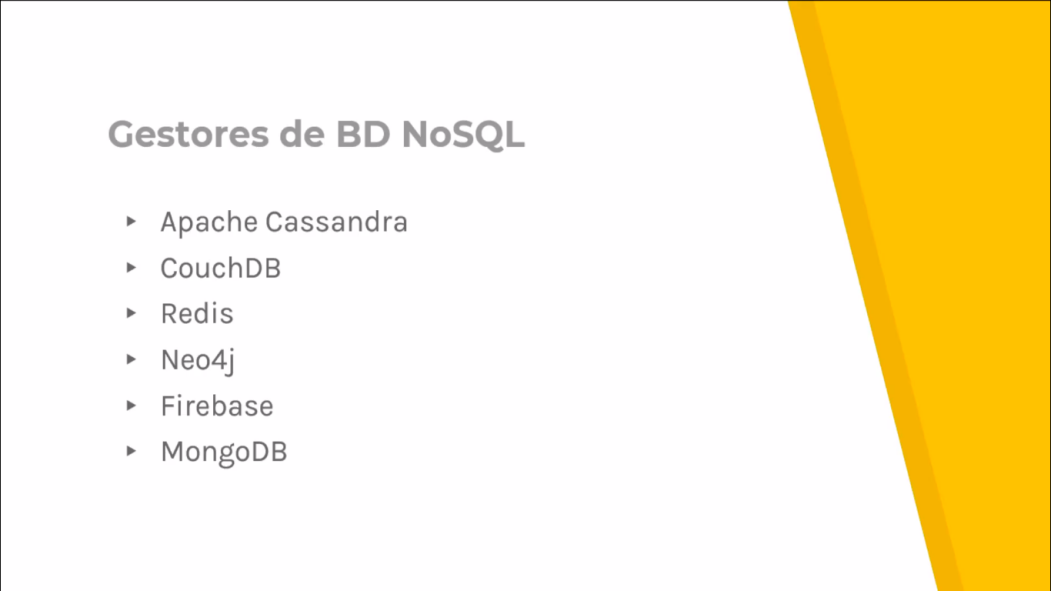


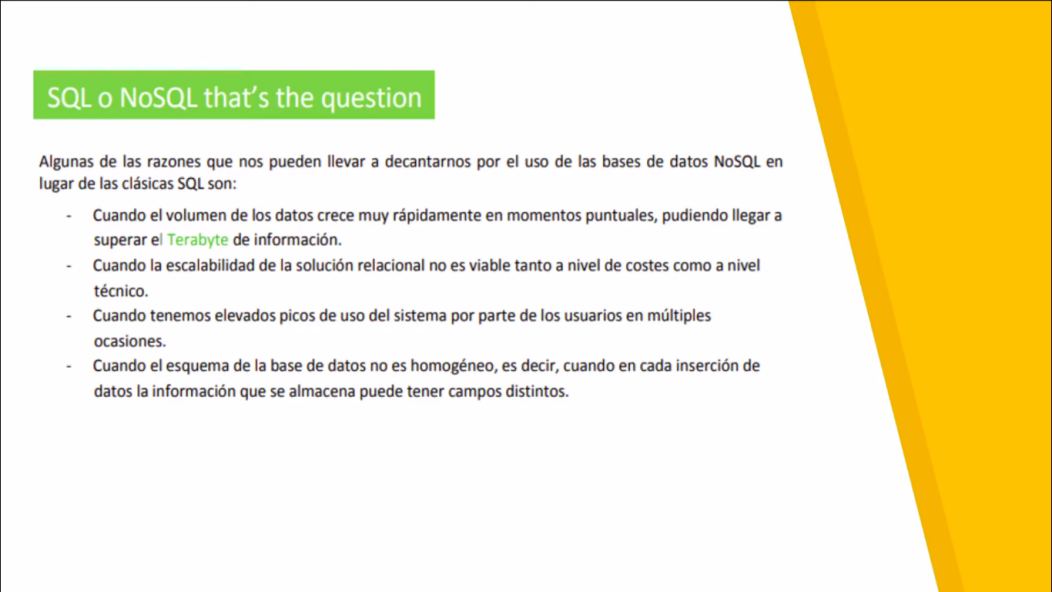


Documento en formato JSON

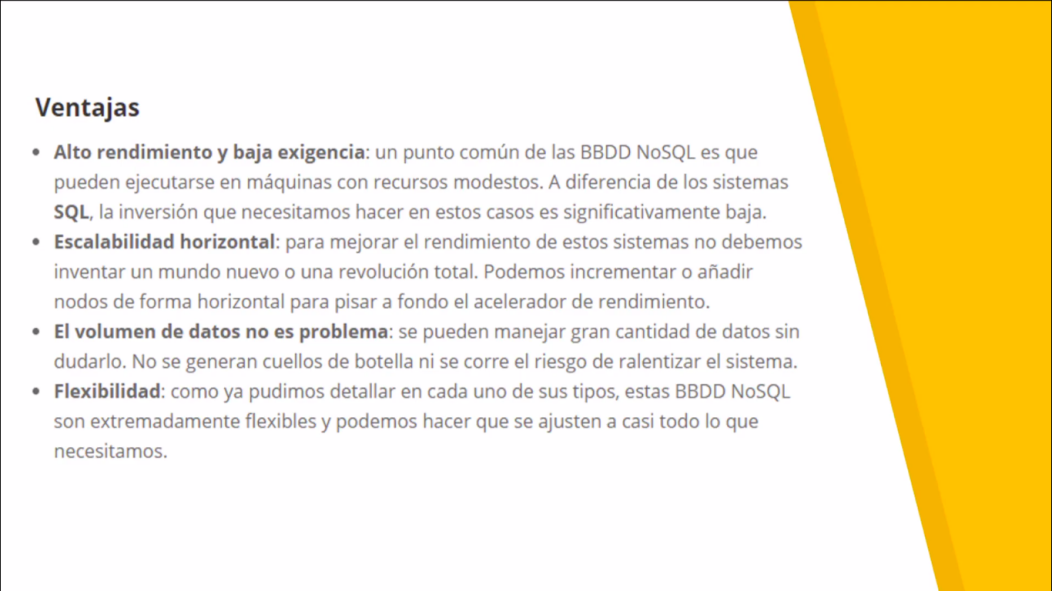


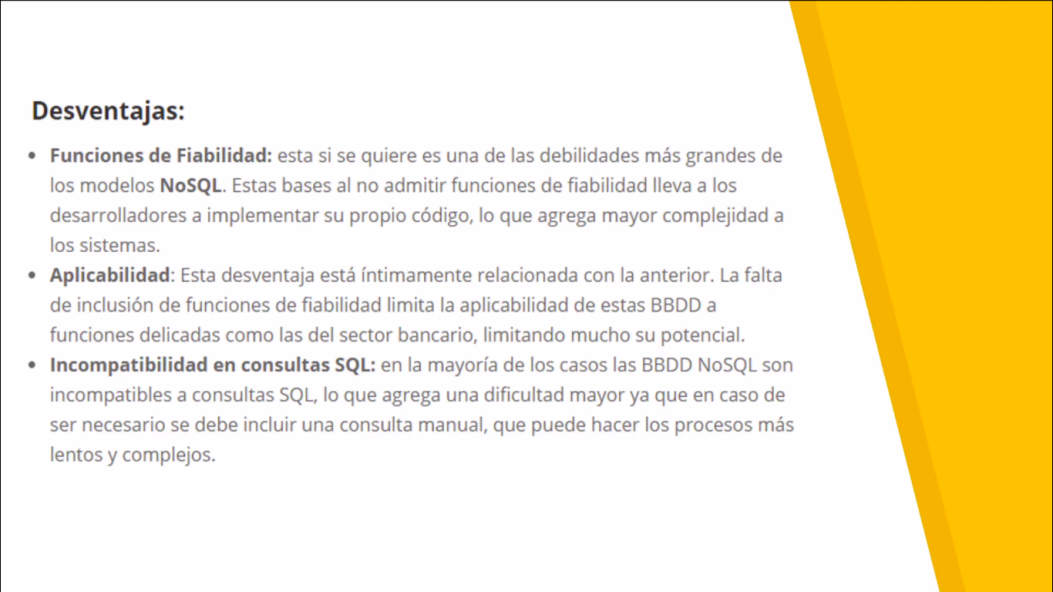














COMANDOS PARA BASE DE DATOS MONGO (CON DESCRIPCIÓN).

ADVERTENCIA: ESTE ARCHIVO CON LAS DESCRIPCIONES DE LOS COMANDOS TIENE CHISTES PEDORROS, MANTENGA SU DISCRECIÓN, O NO, NO IMPORTA.

1.--Ruta de directorio: esta nos sirve para ir a el archivo descomprimido donde está mongo -->

C:\Users\hp> cd C:\Users\hp\Documents\A. Trabajos o Tareas Uni\Base de datos para computo en la nube\mongodb-win32-x86\_64-windows-5.0.5\bin

2.--Levantamiento de servidor, ingreso al cliente, creación y ubicación de los repositorios (directorio donde se va a guardar las bases de datos, NO SON REPOSITORIOS COMO EN LINUX): Previamente se debe crear la carpeta donde se va a guardar la base de datos, en este caso la carpeta se llama "BDdeMongo"-->

//Servidor

C:\Users\hp\Documents\A. Trabajos o Tareas Uni\Base de datos para computo en la nube\mongodb-win32-x86\_64-windows-5.0.5\bin> mongod.exe --dbpath "C:\Users\hp\Documents\A. Trabajos o Tareas Uni\Base de datos para computo en la nube\BDdeMongo"

//Ingreso al cliente. ABRIR EN OTRO CMD.

C:\Users\hp>cd Documents\A. Trabajos o Tareas Uni\Base de datos para computo en la nube\mongodb-win32-x86\_64-windows-5.0.5\bin

C:\Users\hp\Documents\A. Trabajos o Tareas Uni\Base de datos para computo en la nube\mongodb-win32-x86\_64-windows-5.0.5\bin>mongo.exe

NOTA:

1.-Si se cierra el jodido cmd se apaga el server (ya parecen los servidores de rispawn).

2.-Y tambien es bien nena, si se queda un rato o sin utilizar también se cierra >:v.

3.--¿Ya esta en funcion?: Para saber si ya esta funcionando ejecutamos el comando "show databases" como en worbensh -->

>show databases;

4.--Creacion y uso de las bases de datos: MongoDB no es como worbensh (gracias a dios y al creador de esta maravisha), este es un ejemplo del comando para crearlas y usarlas -->

>use database\_dsmVro

switched to db database\_dsmVro

4.2.--Ver en que base de datos estamos: No hay descripción solo comando, el titulo del inciso lo dice wey :v -->

>db

database\_dsmVro

NOTA: Si nosotros ejecutamos un "show databases" para ver nuestra base de datos creada no la va a mostrar, ¿por que?, la razón es que si en esta base datos no hay... Datos ( :v ), no la va a contar para una visualización de las bd existentes.

5.-- Crear y ver tablas o colecciones: Para este gestor, las "TABLAS" donde guardamos los datos no se llaman así ("TABLAS" POR ESO ESTAN EN MAYUSCULAS Y CON COMILLAS, CARA QLO!), aquí (♫tengo el presentimiento que aquí♫ +10 si entendiste la referncia xd) se suplanta el nombre "colecciones" -->

//Creación de colecciones vacías

create collection(“actor”)

//Creación e inserción de datos

> db.alumno.insert({nombre:'Alejandro'});

WriteResult({ "nInserted" : 1 })

//Con más datos

> db.alumno.insert({nombre:"Isra :v", pasatiempos : ["jugar videojuegos", "fumar porritos de tabaco"]});

WriteResult({ "nInserted" : 1 })

//Mostramos tablas/colecciones

> show collections

alumno

//Metodo para buscar (Estructura del comando: objetoALaInstanciaDeLaBaseDeDatos.coleccion.find();)

> db.alumno.find();

--IMPORTANTE: Si nosotros no configuramos la manera de construir el "id" esta wea nos crea uno, esto no es configurable a menos de que nosotros ingresemos el parametro.

{ "\_id" : ObjectId("62093d29c63e382807bbc556"), "nombre" : "Alejandro" }

{ "\_id" : ObjectId("62093edfc63e382807bbc559"), "nombre" : "Isra :v", "pasatiempos" : [ "jugar videojuegos", "fumar porritos de tabaco" ] }

//Método para buscar en especifico

db.actor.find({"writer":"Quentin Tarantino"}); //muestra todo lo que tenga que ver con el autor

db.actor.find({"writer":"Quentin Tarantino"}).pretty(); //bomnito

--IMPORTANTE: Debemos de fijarnos como ingresamos el nombre de la colección, YA QUE SI NOSOTROS LO INGRESAMOS MAL NOS CREA OTRA COLECCIÓN. -->

> db.alumnso.insert({nombre:"Raul :v"});

WriteResult({ "nInserted" : 1 })

> show collections

alumno

alumnso //Mames le puse alumnso en vez de alumno xd (hasta que llegue a este punto no me di cuenta xd)

--¿Como borramos la colección?= Con el metodo "drop" -->

> db.alumnso.drop();

true

> show collections

alumno

--El find tiene otro metodo interno llamado "pretty", este hace que el JSON se vea bonis bonis, su construcción se ve así -->

//Comando

> db.alumno.find().pretty();

-Con pocos datos se ve así -->

> db.alumno.find();

{ "\_id" : ObjectId("62093d29c63e382807bbc556"), "nombre" : "Alejandro" }

{ "\_id" : ObjectId("62093e15c63e382807bbc558"), "nombre" : "Raul :v" }

-Pero con más datos se ve así -->

> db.alumno.find().pretty();

{ "\_id" : ObjectId("62093d29c63e382807bbc556"), "nombre" : "Alejandro" }

{ "\_id" : ObjectId("62093e15c63e382807bbc558"), "nombre" : "Raul :v" }

{

"\_id" : ObjectId("62093edfc63e382807bbc559"),

"nombre" : "Isra :v",

"pasatiempos" : [

"jugar videojuegos",

"fumar porritos de tabaco"

]

}

LECCIÓN 2 (Por ponerle un nombre a lo que siguio de la clase :v)

NOTA: En esta lección se creo otra base de datos llamada database\_prueba -->

> use database\_prueba

6.- Metodo save: Este comando tambien sirve para crear colecciones -->

> db.equipo\_de\_computo.save({nombre:"memoria ram", capacidad:128});

WriteResult({ "nInserted" : 1 })

7.- insertMany: Nos permite dar de alta un arreglo de objetos -->

> db.equipo\_computo.insertMany([{nombre:'mouse logitech', precio:100}, {nombre:'teclado en gomico logitech', precio:240}, {nombre:'disco duro', capacidad:'3 TB'}]);

{

"acknowledged" : true,

"insertedIds" : [

ObjectId("620975ba8379beea25ee11d6"),

ObjectId("620975ba8379beea25ee11d7"),

ObjectId("620975ba8379beea25ee11d8")

]

}

7.2.- Ver cuantos hay en el arreglo: Es obvio no? :v, bueno explico, en el ejemplo podemos ver cuantos equipos hay, con un comando vemos la cantidad, pero con el otro nos muestra cuantos tenemos de forma bonis -->

> db.equipo\_computo.count();

3

> db.equipo\_computo.find().pretty();

{

"\_id" : ObjectId("620975ba8379beea25ee11d6"),

"nombre" : "mouse logitech",

"precio" : 100

}

{

"\_id" : ObjectId("620975ba8379beea25ee11d7"),

"nombre" : "teclado en gomico logitech",

"precio" : 240

}

{

"\_id" : ObjectId("620975ba8379beea25ee11d8"),

"nombre" : "disco duro",

"capacidad" : "3 TB"

}

>

LECCIÓN 3 (Trabajando con sakila)

8.- Filtrar con condicionales: En la carpeta de la materia hay un PP con las condicionales, aqui hay un ejemplo de como filtrar las peliculas, se estan limitando a solo aquellos que tienen una duracion de 6 (6 que? la dvd no se :v)-->

db.film.pelicula.find({duracion\_renta:6}).pretty();

--Aquí se muesta ahora las diferentes a 6 ({$ne:6}) ($ne = no equals)

db.film.pelicula.find({duracion\_renta:{$ne:6}}).pretty();

--Y aquí las esta contando (count();)

db.film.pelicula.find({duracion\_renta:{$ne:6}}).count();

9.- Limitar, filtrar y ver ciertos campos:

--En ese comando no estamos limitando, solo filtrando

db.film.pelicula.find({},{titulo:1, descripcion:1, duracion\_renta:6}).pretty();

--Esto es algo particular, ya que si nosotros ponemos todo en "0" no muestra los campos que hemos dictado, pero nos muestra todos los demás campos que quedan de la colección -->

db.film.pelicula.find({},{titulo:0, descripcion:0, duracion\_renta:0}).pretty();

NOTA: Como dijo Carmen... xd... "No podemos jugar con los sentimientos de: find. 'maestra ¿si prendo uno y apago otro?' no no no no no, es o todos unos o todos ceros". (Ruíz, M. 2022).

--Si quisieramos hacer un filtro, sería en el campo que dejamos vacio, en MySQL seria de esta forma: SELECT title, descripcion FROM film. En NoSQL es al reves, primero metemos coleccion y despues atributos DENTRO DE LAS LLAVES DEL JSON

//SQL= WHERE = $and (cada que se ejecute un $and se tiene que ingresar los campos en un JSON)-->

mayor que menor que

db.film.pelicula.find({$and:[{duracion:{$gte:50}},{duracion:{$lte:100}}]},{titulo:1, descripcion:1, duracion\_renta:1}).pretty();

//Aquí ya no es "and", revisar el archivo

db.film.pelicula.find({$or:

[

{caracteristicas\_especiales:'Trailers'},

{caracteristicas\_especiales:'Deleted Scenes'}

]

},{titulo:1, caracteristicas\_especiales:1}).pretty();

db.film.pelicula.find({$or:[ {caracteristicas\_especiales:'Trailers'}, {caracteristicas\_especiales:'Deleted Scenes'} ] },{titulo:1, caracteristicas\_especiales:1}).count();

//IN, es igual que el comando anterior peroooooo es mas "barato" que usar los JSON anteriores

db.film.pelicula.find(

{

caracteristicas\_especiales:{$in:['Trailers','Deleted Scenes']}},{titulo:1,

caracteristicas\_especiales:1

}).pretty();

//mostrando costo de reemplazo mayor a 10

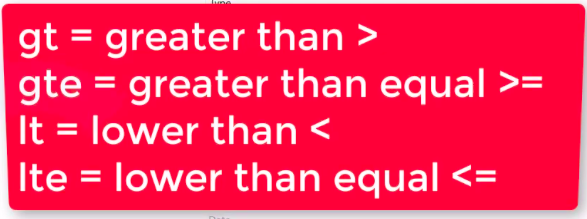
mayor que

db.film.pelicula.find({costo\_reemplazo:{$gt:10}}).pretty();

-insert: Según la documentación de MongoBD, el comando está obsoleto y se usan los otros dos que están a continuación

-insertOne: (Método exclusivo de mongo) Inserta un único documento en una colección.

-insertMany: (Método exclusivo de mongo) Inserta varios documentos en una colección.



Actualizar registros:

db.GetCollection(‘doctores’),(updateOne({“\_id”} : objectId(“id));