**MONGO DB**

Crear primero nuestro primer base de datos:

> use biblioteca

switched to db biblioteca

Procedemos ahora a crear la **colección libros** e insertar el primer documento, la colección se crea en el momento que insertamos el primer documento:

db.libros.insertOne(

{

codigo: 1,

nombre: 'Bases de Datos',

autor: 'Ricardo',

editoriales: ['MacGrawHill','Alfa Omega']

}

)

Luego de esto ya se ha creado la colección "libros" y se ha insertado el primer documento. Procedamos a insertar el segundo documento:

db.libros.insertOne(

{

codigo: 2,

nombre: 'Java 2',

autor: 'Fco. Javier Ceballos',

editoriales: ['Alfa Omega']

}

)

Tengamos en cuenta que mediante el objeto "db" accedemos a la base de datos activa (la misma la activamos con el comando use biblioteca), seguidamente disponemos el nombre de la colección "libros" y finalmente el nombre del método "insertOne" al que le pasamos un objeto en formato JSON.

Ahora nuestra colección "libros" tiene dos documentos, si queremos mostrar los datos almacenados en la colección "libros" podemos llamar al método "**find**" sin pasar parámetros al mismo:

db.libros.find()

Los datos que vemos coinciden con los ingresados al llamar al método insertOne, con la salvedad que se ha agregado un campo llamado **\_id** en forma automática.

Todos los documentos requieren **una clave principal** almacenada en el campo **\_id**. Podemos indicar nosotros el valor a almacenar en el campo \_id, pero si no lo hacemos se crea en forma automática.

**NOTA**

Cada vez que iniciamos **MongoDB shell** se activa por defecto la base de datos 'test' mediante el comando 'use' debemos activar la base de datos que necesitamos trabajar. Para saber en todo momento que base de datos se encuentra activa debemos escribir la variable 'db':

> db

test

> use biblioteca

switched to db biblioteca

> db

biblioteca

>

**Insertar documentos**

Para inserta un documento o un conjunto de documentos disponemos de los métodos:

* insertOne: Inserta un documento en una colección.
* insertMany: Inserta múltiples documentos en una colección.

Procedamos a insertar más de un documento en la colección "libros" mediante el método insertMany:

db.libros.insertMany(

[

{

codigo: 3,

nombre: 'Aprenda PHP',

autor: 'Mario Molina',

editoriales: ['Planeta']

},

{

codigo: 4,

nombre: 'Java en 10 minutos',

autor: 'Barros Sergio',

editoriales: ['Planeta','Alfa Omega']

}

]

)

Tengamos en cuenta que si recién activamos la consola debemos activar la base de datos "**biblioteca**" mediante el comando "**use**":

use biblioteca

Podemos borrar el contenido de la consola de MongoDB (shell) mediante el comando:

cls

El mismo resultado lo podemos obtener presionando las teclas: **CTRL + L.**

Mostremos los documentos almacenados en la colección "libros" mediante el método "find":

> db.libros.find()

Como podemos observar al ejecutar el método "find" nuestra colección "libros" tiene almacenado 4 documentos.

**Campo obligatorio \_id**

En MongoDB todo documento requiere un **campo clave** que se debe llamar **\_id**. Si nosotros como desarrolladores no definimos dicho campo el mismo se crea en forma automática y se carga un valor único.

Podemos definir y cargar un valor en el campo **\_id** cuando inicializamos un documento:

db.clientes.insertOne(

{

\_id: 1,

nombre: 'Lopez Marcos',

direccion: 'Avenida Centenario # 47N-30',

ciudad: 'Armenia'

}

)

Cuando se ejecuta el método **insertOne** nos retorna un objeto JSON informando del resultado de la inserción mediante un objeto con dos campos, el primero **acknowledged** que indica si el documento fue admitido en la colección y el **\_id** que en este caso lo define el usuario de la base de datos.

Si se produce un error nos genera un objeto JSON con otra estructura, probemos de intentar de ingresar un segundo documento con el mismo \_id:

db.clientes.insertOne(

{

\_id: 1,

nombre: 'Perez Ana',

direccion: 'Avenida Bolivar 2N-40',

ciudad: 'Armenia'

}

)

Nos retorna un objeto JSON que entre otros campos define uno llamado **errmsg** con el mensaje de error.

Si nuestra aplicación administra el campo '\_id' hay que tener en cuenta que nunca puede repetirse y en el caso que intentemos ingresar un documento con clave repetida luego dicho documento no se inserta en la colección.

Mostremos todos los documentos almacenados en la colección clientes:

> db.clientes.find()

**NOTA**

Hay uno solo ya que el segundo intento no se cargó.

**Comandos del shell de MongoDB: use (nombre base datos), show dbs, show collections, help**

Ya vimos en conceptos anteriores que mediante el comando use activamos la base de datos que queremos trabajar:

use biblioteca

Para conocer todas las bases de datos del servidor de MongoDB utilizamos el comando:

show dbs

Activemos la base de datos "biblioteca" mediante el comando "use":

use biblioteca

Y seguidamente llamemos al comando "**show collections**":

show collections

La base de datos "**biblioteca**" contiene dos colecciones llamadas "**clientes**" y "**libros**" que nosotros creamos en conceptos anteriores.

Si queremos consultar los comandos esenciales del **shell de MongoDB** debemos emplear el comando "**help**":

> help

**Borrar bases de datos, colecciones o todos los documentos de una colección**

Hemos visto como se crea una base de datos, una colección y se insertan documentos en la misma.

Si queremos eliminar todos los documentos de una colección debemos utilizar el método "**deleteMany**" aplicado a una colección existente:

use biblioteca

db.libros.deleteMany({})

show collections

Debemos pasar un objeto vacío que se indica con las llaves abiertas y cerradas {}. Luego veremos que podemos borrar solo los documentos que cumplen cierta condición.

Es importante notar que luego de llamar al método **deleteMany** la colección "**libros**" sigue existiendo

Para eliminar los documentos de una colección y la colección propiamente dicha debemos emplear el método "**drop**":

use biblioteca

db.libros.drop()

show collections

Luego de llamar al método **drop** de la colección "libros" la misma deja de existir

Para eliminar una base de datos en forma completa, es decir todas sus colecciones y documentos debemos emplear el método **dropDatabase** del objeto "**db**":

show dbs

use bibloteca

db.dropDatabase()

show dbs

El método **dropDatabase** elimina la base de datos activa

**Recuperar algunos documentos de una colección con el método find**

Vimos en conceptos anteriores que podemos recuperar todos los documentos de una colección mediante el método **find**:

use biblioteca

db.libros.drop()

db.libros.insertOne(

{

\_id: 1,

titulo: 'Bases de Datos',

autor: 'Ricardo',

editorial: ['MacGrawHill','Planeta'],

precio: 20,

cantidad: 50

}

)

db.libros.insertOne(

{

\_id: 2,

titulo: 'Java 2',

autor: 'Fco. Javier Ceballos,

editorial: ['MacGrawHill'],

precio: 50,

cantidad: 12

}

)

db.libros.insertOne(

{

\_id: 3,

titulo: 'Aprenda PHP',

autor: 'Mario Molina',

editorial: ['Siglo XXI','Planeta'],

precio: 50,

cantidad: 20

}

)

db.libros.insertOne(

{

\_id: 4,

titulo: 'Java en 10 minutos',

editorial: ['Siglo XXI'],

precio: 45,

cantidad: 1

}

)

db.libros.find()

El método **find** nos permite seleccionar solo algunos documentos que cumplen una condición:

db.libros.find({\_id : 1})

Rescatamos el documento que almacena en el campo '\_id' el valor 1

Si pasamos un valor para el campo '**\_id**' que **no existe** luego el **método find** no regresa un documento.

Podemos rescatar todos los libros que tiene un precio igual a 50:

db.libros.find({precio : 50 })

Podemos disponer más de un campo:

db.libros.find({precio : 50, cantidad : 20 })

**Operadores relacionales** $eq, $gt, $gte, $lt, $lte, $in, $nin y $ne

En el concepto anterior vimos cómo podemos seleccionar mediante el método **find** algunos documentos que cumplen una condición.

Recordemos con un ejemplo como recuperar todos los libros con un precio igual a 50:

db.libros.find({ precio: 50 })

Es decir que cuando llamamos al método **find** pasamos un objeto literal pasando en el campo precio el valor 50, luego el método **find** filtra todos los libros cuyo precio sean exactamente igual a 50.

Otra forma de expresar la búsqueda de todos los libros con un precio igual a 50 es:

db.libros.find({ precio: { $eq : 50 } })

Es decir que luego del campo precio pasamos otro objeto literal iniciando el **operador $eq** con el valor 50.

Mostramos esta segunda forma de consultar todos los libros con un precio igual a 50 debido a que cuando tenemos que consultar por ejemplo los libros con un precio inferior a 50, o superior a 50 etc, debemos indicar en forma obligatoria el operador a utilizar.

Para mostrar todos los libros con un precio inferior a 30 tenemos que utilizar el **operador $lt**:

db.libros.find({ precio: { $lt : 30 } })

**Listado de operadores relacionales**

* $eq - equal - igual
* $lt - low than - menor que
* $lte - low than equal - menor o igual que
* $gt - greater than - mayor que
* $gte - greater than equal - mayor o igual que
* $ne - not equal - distinto
* $in - in - dentro de
* $nin - not in - no dentro de

Veamos con algunos ejemplos como utilizar estos operadores para recuperar documentos que cumplen determinadas condiciones.

Recuperar todos los libros que tienen un precio mayor a 40:

db.libros.find({ precio: { $gt:40 }})

todos los libros que en le campo cantidad tiene 50 o más:

db.libros.find( { cantidad: { $gte : 50 }})

Recuperar todos los libros que en le campo cantidad hay un valor distinto a 50:

db.libros.find( { cantidad: { $ne : 50 }})

Recuperar todos los libros cuyo precio estén comprendidos entre 20 y 45:

db.libros.find( { precio: { $gte : 20 , $lte : 45} })

Recuperar todos los libros de la editorial 'Planeta':

db.libros.find( { editorial: { $in : ['Planeta'] } })

Recuperar todos los libros que no pertenezcan a la editorial 'Planeta':

db.libros.find( { editorial: { $nin : ['Planeta'] } })

Hay que acostumbrarse en un principio a utilizar estos operados para filtrar documentos de una colección, luego veremos que estos operadores también se emplean cuando efectuemos borrados y modificaciones de documentos.

**Borrar documentos de una colección con los métodos deleteOne y deleteMany**

Hemos visto en conceptos anteriores que podemos eliminar todos los documentos de una colección mediante el método **deleteMany** y pasando un objeto literal vacío:

db.libros.deleteMany({})

Aprendimos también a recuperar algunos documentos con el método **find** empleando una serie de operadores relacionales, dichos operadores se pueden emplear en forma idéntica con los métodos **deleteMany** y **deleteOne**.

Hay dos métodos para eliminar documentos:

* deleteMany: Borra todos los documentos que cumplen la condición que le enviamos.
* deleteOne: Borra el primer documento que cumple la condición que le pasamos.

Almacenaremos una serie de documentos en una colección llamada libros y luego borraremos algunos de sus documentos:

use biblioteca

db.libros.drop()

db.libros.insertOne(

{

\_id: 1,

titulo: 'Bases de Datos',

autor: 'Ricardo',

editorial: ['MacGrawHill','Planeta'],

precio: 20,

cantidad: 50

}

)

db.libros.insertOne(

{

\_id: 2,

titulo: 'Java 2',

autor: 'Fco. Javier Ceballos,

editorial: ['MacGrawHill'],

precio: 50,

cantidad: 12

}

)

db.libros.insertOne(

{

\_id: 3,

titulo: 'Aprenda PHP',

autor: 'Mario Molina',

editorial: ['Siglo XXI','Planeta'],

precio: 50,

cantidad: 20

}

)

db.libros.insertOne(

{

\_id: 4,

titulo: 'Java en 10 minutos',

editorial: ['Siglo XXI'],

precio: 45,

cantidad: 1

}

)

db.libros.find()

Si queremos eliminar el documento que almacena en el campo el \_id con valor 1 luego podemos utilizar la sintaxis:

db.libros.deleteOne({\_id: 1})

Lo más conveniente es utilizar el método 'deleteOne' ya que solo uno puede cumplir esa condición al ser la clave primaria del documento.

Recordemos que la sintaxis alternativa para eliminar el documento con \_id con valor 1 es:

db.libros.deleteOne({\_id: { $eq : 1}})

La sintaxis anterior es buena recordar ya que los otros operadores relacionales hay que utilizarlos en forma obligatoria y no tienen una sintaxis alternativa como el $eq.

Para borrar todos los libros que tienen un precio mayor o igual a 50 tenemos:

db.libros.deleteMany({precio : {$gte : 50 }})

**Modificar un documento mediante el método updateOne**

Hemos visto en conceptos anteriores como insertar un documento en una colección, recuperar un documento, borrar un documento y nos está faltando otra operación fundamental que podemos hacer con un documento que es su modificación.

Para modificar un documento en particular disponemos de un método llamado **updateOne**, veamos con un ejemplo algunas de sus posibilidades

use biblioteca

db.libros.drop()

db.libros.insertOne(

{

\_id: 1,

titulo: 'Bases de Datos',

autor: 'Ricardo',

editorial: ['MacGrawHill','Planeta'],

precio: 20,

cantidad: 50

}

)

db.libros.insertOne(

{

\_id: 2,

titulo: 'Java 2',

autor: 'Fco. Javier Ceballos,

editorial: ['MacGrawHill'],

precio: 50,

cantidad: 12

}

)

db.libros.insertOne(

{

\_id: 3,

titulo: 'Aprenda PHP',

autor: 'Mario Molina',

editorial: ['Siglo XXI','Planeta'],

precio: 50,

cantidad: 20

}

)

db.libros.insertOne(

{

\_id: 4,

titulo: 'Java en 10 minutos',

editorial: ['Siglo XXI'],

precio: 45,

cantidad: 1

}

)

db.libros.find()

db.libros.updateOne({\_id : {$eq:1}} , {$set : {precio:15,cantidad:1} })

db.libros.find()

Hemos llamado al método **updateOne** con dos parámetros, el primero indica el documento a modificar y el segundo parámetro utilizamos el operador **$set** que es un operador de actualización seguidamente con los campos y valores a modificar.

Si existe el documento con \_id igual a 1 luego se modifican los campos precio y cantidad

Con las bases de datos documentales tengamos en cuenta que los documentos pueden tener distintas cantidades de campos. Por ejemplo, si queremos agregar el campo descripción al libro con '\_id' 4 debemos utilizar la sintaxis:

db.libros.updateOne({\_id: {$eq:4}} ,{$set : {descripcion: 'Cada unidad trata un tema fundamental de Java desde 0.'} })

Luego de ejecutar tenemos el documento con \_id 4 que contiene un nuevo campo llamado 'descripcion' con el valor 'Cada unidad trata un tema fundamental de Java desde 0.':

> db.libros.find()

Si queremos eliminar un campo de un documento debemos emplear el operador de actualización **$unset**. Probemos ahora de eliminar el campo que acabamos de crear para el documento con \_id 4

db.libros.updateOne({\_id : {$eq:4}} , {$unset : {descripcion:''} })

> db.libros.find()

Es importante entender que mediante el operador **$unset** eliminamos el campo, en cambio sí utilizamos el operador **$set** modificamos el contenido del campo, luego si ejecutamos:

db.libros.updateOne({\_id : {$eq:4}} , {$set : {descripcion:''} })

El campo 'descripcion' sigue existiendo y almacena una cadena de texto vacía, y si no existía se crea con una cadena vacía.

Disponemos también de operadores de modificación para arreglos, veamos cómo podemos agregar y eliminar elementos en el arreglo 'editorial':

db.libros.updateOne({\_id : {$eq:1}} , {$push : {editorial:'Atlantida'} })

db.libros.find()

Podemos ver que luego de ejecutarse el método **updateOne** el arreglo 'editorial' tiene una nueva componente para el documento con \_id 1 (**operador $push**)

De forma similar para eliminar un elemento del arreglo debemos emplear el operador **$pull**:

db.libros.updateOne({\_id : {$eq:1}} , {$pull : {editorial:'Atlantida'} })

db.libros.find()

**Modificar múltiples documentos con el método updateMany**

use biblioteca

db.libros.drop()

db.libros.insertOne(

{

\_id: 1,

titulo: 'Bases de Datos',

autor: 'Ricardo',

editorial: ['MacGrawHill','Planeta'],

precio: 20,

cantidad: 50

}

)

db.libros.insertOne(

{

\_id: 2,

titulo: 'Java 2',

autor: 'Fco. Javier Ceballos,

editorial: ['MacGrawHill'],

precio: 50,

cantidad: 12

}

)

db.libros.insertOne(

{

\_id: 3,

titulo: 'Aprenda PHP',

autor: 'Mario Molina',

editorial: ['Siglo XXI','Planeta'],

precio: 50,

cantidad: 20

}

)

db.libros.insertOne(

{

\_id: 4,

titulo: 'Java en 10 minutos',

editorial: ['Siglo XXI'],

precio: 45,

cantidad: 1

}

)

db.libros.find()

La primera modificación masiva la hacemos con todos los libros cuyo \_id sean mayores a 2, fijando el campo cantidad con 0:

db.libros.updateMany({\_id : {$gt:2}} , {$set : {cantidad:0} })

db.libros.find()

La segunda modificación masiva la hacemos con todos los libros que almacenan en el campo 'cantidad' el valor cero, agregando el campo 'faltantes' con el valor true:

db.libros.updateMany({cantidad : {$eq:0}} , {$set : {faltantes:true} })

db.libros.find()

La tercer y última modificación masiva la hacemos con todos los libros que almacenan en el campo 'cantidad' el valor cero, eliminamos el campo faltante y fijamos el campo cantidad con el valor 100

db.libros.updateMany({cantidad : {$eq:0}} , {$unset : {faltantes:true}, $set:{cantidad: 100}} )

db.libros.find()

**Operadores lógicos $and, $or y $not**

Cuando necesitamos construir consultas que deban cumplir varias condiciones utilizaremos los **operadores lógicos**.

El operador **$and** lo hemos utilizando en forma implícita, por ejemplo, si tenemos:

db.libros.find({precio : 50, cantidad : 20 })

Con la condición anterior se recuperan todos los libros que tienen un precio de 50 y la cantidad es 20. Las dos condiciones deben ser verdaderas para que el documento se recupere.

La sintaxis alternativa para el **find** es:

db.libros.find({$and : [{precio:50}, {cantidad:20}] })

El valor para el operador $and es un arreglo con cada una de las condiciones que debe cumplir.

Para los operadores **$or** y **$not** no hay una forma de disponer una sintaxis implícita.

use biblioteca

db.libros.drop()

db.libros.insertOne(

{

\_id: 1,

titulo: 'Bases de Datos',

autor: 'Ricardo',

editorial: ['MacGrawHill','Planeta'],

precio: 20,

cantidad: 50

}

)

db.libros.insertOne(

{

\_id: 2,

titulo: 'Java 2',

autor: 'Fco. Javier Ceballos,

editorial: ['MacGrawHill'],

precio: 50,

cantidad: 12

}

)

db.libros.insertOne(

{

\_id: 3,

titulo: 'Aprenda PHP',

autor: 'Mario Molina',

editorial: ['Siglo XXI','Planeta'],

precio: 50,

cantidad: 20

}

)

db.libros.insertOne(

{

\_id: 4,

titulo: 'Java en 10 minutos',

editorial: ['Siglo XXI'],

precio: 45,

cantidad: 1

}

)

db.libros.find()

Para recuperar los libros que tienen un precio mayor o igual a 50 o la cantidad es 1 debemos implementar mediante un **$or** la siguiente sintaxis:

db.libros.find({$or: [{precio:{$gte:50}}, {cantidad:1} ]})

Si queremos recuperar todos los documentos de la colección libros que no tienen un precio mayor o igual a 50 la sintaxis debe ser:

db.libros.find({precio: {$not:{$gte:50}} })

Los operadores lógicos podemos utilizarlos no solo para recuperar datos, sino también cuando borramos o actualizamos documentos.

Si queremos borrar todos los libros cuyo precio no sean iguales a 50 podemos codificar:

db.libros.deleteMany({precio: {$not:{$eq:50}} })

Se eliminan dos documentos de la colección libros.