MongoDB básico

El objetivo de este ejemplo es mostrar como funciona MongoDB desde la consola, o (la sintaxi es la misma) con los drivers por defecto de Node.js.

MongoDB CLI (Command line interface)

Para empezar, tenemos que iniciar el daemon server de MongoDB desde un Terminal con:

```
mongod
```

Luego, abrimos otro terminal y ejecutamos:

```
mongo
```

Con esto, tendremos una interfaz de línia de comandos para interactuar con las bases de datos de mongodb.
El siguiente comando mostrará todas las bases de datos que haya en el servidor:

```
show dbs
```

Para empezar a usar una base de datos existente, o crear una nueva, escribimos:

```
use <nombre_base_datos>
```

Notemos que si la base de datos no existe aún, sólo se creará si insertamos un documento en alguna colección.

Para listar todas las colecciones de una base de datos, habiendo ejecutado el comando anterior con la base de datos que queramos, escribimos:

```
show collections
```

Operaciones CRUD (Create, Read, Update, Delete)

Create

Insertar nuevos documentos es fácil, solo debemos especificar el objeto que queremos insertar, y la colección a la que lo queremos insertar:

```
db.users.insert( { name: "admin", status: "admin", age: 20 } )
```

Insertará el objeto o documento `{ name: "admin", status: "admin", age: 20 }` en la colección `users` de la base de datos que estemos usando actualmente, y que habremos especificado previamente con el comando `use <nombre_base_datos>`. Notemos que si la colección users no existía se creará una nueva.

Read

Podemos **listar** todos los documentos de una colección con el comando:

```
db.users.find().pretty()
```

Que lista todos los documentos en un formato fácilmente legible. Si omitimos la última parte `.pretty()`
también se listan pero sin los espacios y finales de línia que lo hacen más legible.

Notar también que si en la inserción de un documento no se especifica ningún atributo `_id` entonces
mongodb lo crea automáticamente. Este atributo lo usa mongodb internamente como un identificador único (dentro de la colección) para el documento.

Para listar documentos y **filtrarlos** en función de criterios:

```
// Lista todos los usuarios llamados 'admin'
db.users.find({ name: 'admin' })

// Lista todos los usuarios que tienen edad > 18
db.users.find({ age: { $gt: 18 } })

// AND entre las dos condiciones
db.users.find({ age: { $lt: 25 }, name:'admin' })

// OR entre las dos condiciones
db.users.find({ $or: [{ age: { $lt: 25 } }, { name: 'admin' }] })
```

También podemos **filtrar respecto propiedades anidadas y elementos de arrays**:

```
// Lista todos los usuarios que tienen como primer elemento
// del array 'jobs' un objeto con el atributo 'name' igual a "programmer"
// i.e en Javascript: usuario.jobs[0].name === "programmer"
db.users.find({ "jobs.0.name": "programmer" })
```

También tenemos una manera fácil de encontrar un único documento:

```
// Muestra (si existe) el usuario con nombre 'josep'
db.users.findOne({ name: 'josep' })
```

Notar que esto ya devuelve el documento en formato fácilmente legible.

Ordenación:

Ordenar tambien es fácil, aplicando el método `sort()` al resultado de un `find()`:

```
// Ordena por el atributo name en orden creciente, y luego en orden decreciente
por el atributo age
db.users.find().sort({ name: 1, age: -1 })
```

Proyección:

Las consultas que hemos usado hasta ahora devuelven todos los atributos de los documentos resultantes.
Pero, y si no los necesitamos todos?

```
// Devuelve sólo el atributo name de los documentos resultantes
db.users.find({}, { name: 1 })

// Devuelve todos los atributos excepto el atributo name
db.users.find({}, { name: 0 })
```

Notad que no se pueden usar el 0 y el 1 a la vez en una misma proyección.

Update

El método update recibe 2 parámetros. El primero es un objeto de consulta, que funciona tal y como hemos visto
con el método `find`. La única diferencia es que en este caso, por defecto el update sólo tiene efecto con el **primer** documento
que cumpla la consulta. El segundo argumento es la operación a realizar con los documentos resultantes,
que consiste de o bien un nuevo documento a reemplazar en lugar del que había, o bien de un objeto con operadores.
Cualquier objeto que tenga algún atributo que empiece con $ se considera un objeto de operación, los operadores del cual
son dichos atributos.

```
// Asigna el valor 27 al atributo age del primer documento con name ===
'admin'.
// Si el documento no tenía el atributo age entonces lo crea y lo asigna,
// si ya lo tenia cambia su valor.
db.users.update({ name: 'admin' }, { $set: { age: 27 } })

// Incrementa el atributo age en 2 unidades
db.users.update({ name: 'admin' }, { $inc: { age: 2  } })
```

Visita https://docs.mongodb.org/manual/reference/operator/update/ para ver una lista con todos los operadores disponibles.

NOTA IMPORTANTE: Si en vez de los operadores `$set` y `$inc` se especifica un objeto, con un update como este:

```
db.users.update({ name: 'admin' }, { age: 10 })
```

El primer documento con atributo `name === 'admin'` será reemplazado por el documento `{ age: 10 }`, y todos los demás atributos que tenga el documento encontrado **serán borrados**.

Para especificar que queremos modificar todos los documentos que cumplan la consulta en vez de sólo el primero, podemos pasar la opción `multi`:

```
db.users.update({ name: 'admin' }, { $set: { age: 27 } }, { multi: true })
```

Delete

Podemos usar, tal y como hacemos con el método `find` y el método `update` un objeto de consulta para indicar los documentos que queremos eliminar:

```
// Elimina todos los documentos que tengan el atributo name con valor 'Demon'
db.users.remove({ name: 'Demon' })
```

Notad que esta operación **elimina todos los documentos encontrados**. Si queremos limitar el borrado a el primer documento coincidente, podemos escribir:

```
// Elimina el primer documento que tenga el atributo name con valor 'Demon'
db.users.remove({ name: 'Demon' }, { justOne: true })
```

Podéis visitar la documentación de MongoDB para ver más opciones:
http://docs.mongodb.org/manual/core/crud-introduction/