Mongo DB – Parte II

Object ID

- Se crea de manera automática
- Guarda un ID único
- Es lo que se conoce como un GUID (globally unique identifier). Son únicos y globales. Sirve para poder escalar una aplicación

Creando un Object ID

Es muy probable que en alguna situación particular, tengan que poder crear identificadores únicos y universales, por eso está bueno saber cómo crearlos con una clase que nos expone mongodb.

1. Requerimos la clase ObjetcID de Mongo

```
const ObjetctID = mongodb.ObjectID;
```

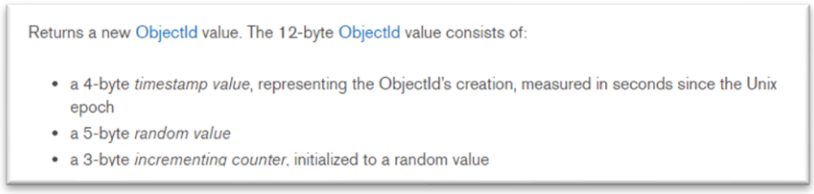
2. Lo generamos con

```
const guid = new ObjetctID();
```

3. Lo testeamos con un console.log(guid)

```
PS C:\Users\juanm\Documents\Repos\node-js\task-manager> node .\mongodb.js
5fe1f36813a2451c54144589
```

4. 5fe1f36813a2451c54144589 ¿Qué es?
    a. Los primeros números son un timestamp
    b. En el medio un valor random
    c. Al final, un nuevo valor random


Returns a new ObjectId value. The 12-byte ObjectId value consists of:

- a 4-byte *timestamp value*, representing the ObjectId's creation, measured in seconds since the Unix epoch
- a 5-byte *random value*
- a 3-byte *incrementing counter*, initialized to a random value

Todo sobre el ObjectId:

https://docs.mongodb.com/manual/reference/method/ObjectId/

Probémos el método:

`ObjectId.getTimestamp()`

¿Qué nos devuelve?

Realizar Queries

FindOne

Nos devuelve el primer valor que encuentra. Dependiendo el valor con el que busquemos, podríamos tener o no duplicados. En caso de haber duplicados, solo nos devuelve el primer valor.

1. Especificamos la colección y el método a usar.
```
db.collection('users').findOne();
```
2. findOne recibe dos parámetros
    a. Un objeto con las condiciones de búsqueda (En este caso vamos a buscar un usuario que se llame “Juan”.
    b. Un callback
```
db.collection('users').findOne({name: 'Juan'}, () => {});
```
3. El callback recibe:
    a. Error
    b. El objeto encontrado (Que en este caso sería un usuario, entonces)

```
db.collection('users').findOne({
  name: 'Juan Ma'
}, (err, user) => {
  if (err) return console.log('No pudimos traer el usuario');
  console.log(user.ops);
});
```

¿Qué pasa si buscamos un usuario que no existe?

Nos devuelve null, por lo cual deberíamos validarlo.

Buscar por id

Si queremos buscar por id, tenemos que utilizar la clase ObjectID de mongo y hacer lo siguiente:

```
db.collection('users').findOne({
  _id: new ObjetctID('5fdfa6a832d3b8167c7f1479')
}, (err, user) => {
  if (err) return console.log('No pudimos traer el usuario');
  console.log(user);
});
```

Find

Cuando hacemos uso del método find, lo que recibimos es un “cursor”.

find(query, options) → {Cursor}

¿Qué es un cursor? Lo podemos buscar en la documentación:

Es una clase, propia de mongo, que si vemos en los ejemplos nos deja hacer varias cosas:

Traer un máximo:

```
collection.find({}).max(10)
```

Traer un mínimo:

```
collection.find({}).min(100)
```

Hacer un sorting:

```
collection.find({}).sort([['a', 1]])
```

Devolver un array

toArray(callback) → {Promise}

Por ahora, vamos a usar este método para traer un array de todos los registros, similar al SELECT * FROM table, que conocemos de SQL.

1. Buscar la info de la DB
```
db.collection('users').find({age: 27})
```
2. Acceder al método .toArray()
```
db.collection('users').find({age: 27}).toArray()
```
3. Crear el callback
```
db.collection('users').find({ age: 27 }).toArray((error, users) =>
{
    if (err) return console.log('No pudimos traer el usuario');
    console.log(users);
  });
```

Actualizar un valor

UpdateOne

Para actualizar un valor, vamos a empezar con una lógica similar a la que venimos teniendo.

1. Seleccionamos la collection
2. Usamos el método updateOne()
    a. Primer parámetro: La clave con la que se qué voy a modificar
    b. Segundo parámetro: Un operador $set que me dice cuál es el valor a modificar
    c. El callback

Todo junto:

```
db.collection('users').updateOne({
  _id: new ObjetctID('5fdfa6a832d3b8167c7f1479')
}, {
  $set: {
    name: 'Treltinda'
  }
}, (err, result) => {
  if (err) return console.log('No pudimos actualizar el usuario');
  console.log(result);
});
```

Operadores

Existen distintos tipos de operadores en MongoDB, todos tienen la particularidad que empiezan con un $ y cada uno realiza distintas operaciones. En el caso de $set, nos permite actualizar un valor. (Es uno de los más comunes)

Para ver la lista entera:

https://docs.mongodb.com/manual/reference/operator/update/

¿Qué pasa si agregamos valores que no existían?

Se agregan. Esto lo hace completamente distinto a SQL

UpdateMany

Funciona de manera similar al update:

```
db.collection('users').updateMany({
  age: 28
}, {
  $set: {
    age: 2,
  }
}, (err, result) => {
  console.log(result);
})
```

1. Primer argumento es la condición
2. Segundo el operador
3. El callback.

Desafío delete

Eliminar usuarios que tienen una edad específica.