Introducción MONGO DB – NO SQL

Mongo DB

Base de datos de código abierta, disponible para cualquier sistema operativo. Nace en 2009 junto a NodeJS y van de la mano, igualmente se pueden utilizar distintas bases de datos como vimos con MySQL

NoSQL

No sólo SQL. Mongo se ve distinto a SQL, pero nos va a proveer una manera estructurada para trabajar con datos.

Similitudes:

1. Ambas permiten crear Bases de Datos

Diferencias

En SQL vimos “Tablas” con “Filas” y “Columnas” (Entidades y Atributos) pero en bases NoSQL vamos a hablar de colecciones, que se van a ver similar a un objeto JSON.

1. Table vs Collections

Las Rows, pasan a llamarse documentos.

Instalación

Documentación

https://docs.mongodb.com/drivers/node/

Paso a paso

1. Nos creamos un nuevo proyecto.
    a. Nueva Carpeta
    b. Npm init
2. Nos instalamos el paquete de npm para utilizar mongodb (Es el paquete oficial)
    a. npm i mongodb

    https://www.npmjs.com/package/mongodb

3. Nos creamos un archivo mongodb.js
4. Lo requerimos

```
const mongodb = require('mongodb');
```

5. Para conectarnos, tenemos que hacer uso de una clase de mongo que se llama MongoClient. Esta es la clase que nos va a permitir hacer la conexión mediante uno de sus métodos.
   Para instanciar MongoClient:
   ```
   const MongoClient = mongodb.MongoClient;
   ```

6. MongoClient tiene un método llamado connect, que recibe 3 parámetros:
   a. La URL que tiene la DB
   b. Un objeto de configuración
   c. Un callback (porque conectarse a la DB no se hace de manera sincrónica).

7. Vamos a crear la ruta de la DB en una constante y también el nombre de la base de datos que nos vamos a crear
   ```
   const connectionUrl = 'mongodb://127.0.0.1/:27017';
   const databaseName = 'base-prueba';
   ```

8. El objeto de configuración que vamos a usar va a ser el siguiente:
   ```
   { useNewUrlParser: true, useUnifiedTopology: true }
   ```
   En donde le decimos que utilice la ruta que le pasamos y como monitorear la DB. Para más info: https://mongoosejs.com/docs/deprecations.html

Todo junto quedará así:

```
const mongodb = require('mongodb');
const MongoClient = mongodb.MongoClient;

/* /127.0.0.1/ = localhost */
const connectionUrl = 'mongodb://127.0.0.1/:27017';
const databaseName = 'base-prueba';

MongoClient.connect(connectionUrl, { useNewUrlParser: true, useUnifiedTop
ology: true }, (error, client) => {
  if (error) { return console.log('Unable to connect to Database'); }

  console.log('Connected correctly');
});
```

Nota: Ya conocemos el protocolo file, el ftp, el http, mongo utiliza uno propio que es el mongodb://

Nuestros primeros CRUD

Creating, Reading, updating, deleting

Cuando hicimos la conexión, el callback recibe dos parámetros: error, o client. Una vez que tenemos acceso al cliente, tenemos que especificar el nombre para trabajar con una base de datos específica. Esto por lo general se hace guardando la base de datos que querramos en una constante.

```
  const db = client.db(databaseName);
```

Recordemos en el ejemplo más arriba que:

```
const databaseName = ‘base-prueba’;
```

Lo que estamos haciendo es que una vez que se realizó la conexión, creamos una nueva base de datos.

Create insertOne

¿Se acuerdan del CREATE TABLE ‘usuarios’? Bueno, ahora es similar, pero distinto. No queremos crear una tabla, queremos crear una collection, y como tal, hay un método que tiene el cliente para crear collections:

```
  db.collection('usuarios').insertOne({
    name: 'Juan Ma',
    age: 28
  });
```

En este ejemplo, estamos creando una colección de “usuarios” e insertando nuestro primer valor. Todo junto:

```
  const db = client.db(databaseName);
  db.collection('usuarios').insertOne({
    nombre: 'Juan Ma',
    edad: 28
  });
```

Esto sería el símil a:

```
CREATE DATABASE ‘base-prueba’;

CREATE TABLE ‘usuarios’ (

id NOT NULL AUTO_INCREMENT,

nombre VARCHAR(255) NOT NULL,

edad INT NOT NULL

 );
```

INSERT INTO ‘usuarios’ (nombre, edad) VALUES (‘Juan Ma’, 28)

Callback luego de insertar

```
(error, result) => {
    if (error) { return console.log('Error insertando usuarios); }

    console.log(result.ops);
  }
```

Create insertMany

Recibe un array de objetos, el resto sigue igual

```
  db.collection('tareas').insertMany([
    {
      descripcion: 'Alguna descripción de tareas',
      completeda: true
    },
    {
      descripcion: 'Otra descripción de tareas',
      completeda: false
    }
  ], (error, result) => {
    if (error) { return console.log('Error insertando tareas'); }

    console.log(result.ops);
  });
```

Object Id

El id que vemos, es distinto al id de SQL: 5fdfb01d6b56ae17e4afbc12. Son GUID, Global Unique Identifier. La gran diferencia es que este ID es único y le sirve a MongoDB para poder escalar entre servidores.

Imaginemos que tenemos un usuario con id = 1 en una base MySQL, en un servidor. Es muy probable que si seteamos otro servidor y empezamos una base de cero, si creamos un usuario, el primero va a tener id=1. Esto **no** va a pasar con MongoDB porque el ID es único a escala global.