Curso de Back-End con Node.js (Avanzado)

Clase 03

Temario

Temario

- Repaso de BD Relacionales.
- BD No Relacionales.
- MongoDB.
- MongoDB Compass.
- Mongoose.
    - Esquemas.
    - Modelos.
    - Métodos.

REPASO

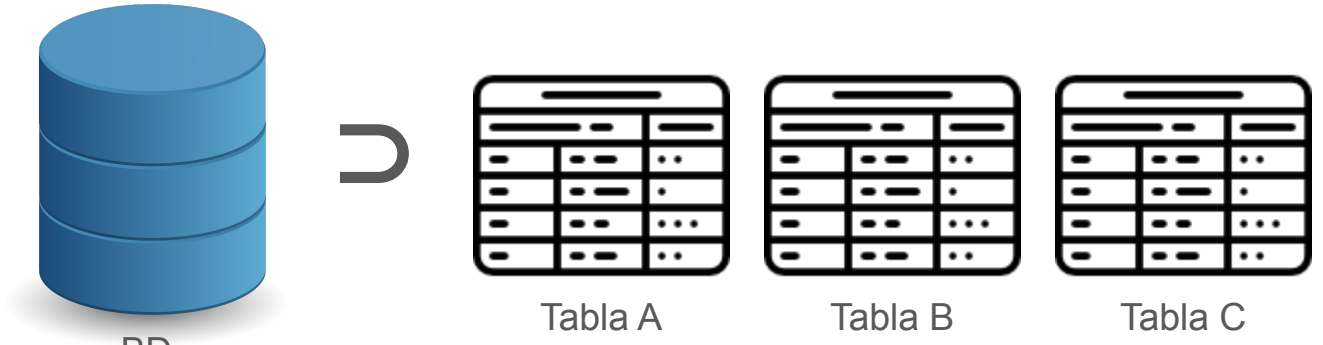
Bases de Datos Relacional

REPASO

Base de Datos Relacional (1/2)

Una **base de datos relacional** es una base de datos que organiza la información en una o más **tablas**.

Una **tabla** es una colección de datos organizados en filas y columnas (similar a Excel). A veces, a las tablas se les dice *relaciones*. A las filas también se les llama *registros* o *tuplas*.



REPASO

Base de Datos Relacional (2/2)

Ejemplo: **Tabla Usuarios**

REPASO

DBMS – Database Management System

Un DBMS es un software que permite crear y manipular bases de datos. Es como una interface entre las BD y las aplicaciones. Algunos gestores populares son:

- MySQL.
- MariaDB.
- SQL Server.
- Microsoft Access.
- Oracle.
- PostgreSQL.
- MongoDB.
- Cassandra.

Relacionales.

No Relacionales.

Los que usaremos en el curso.

¿Relacional vs. No Relacional?
Ver → Link 1. Link 2.

Nota: Al DBMS también se le conoce como *"Motor de BD"*, *"Gestor de BD"* o *"Manejador de BD"*.

REPASO

SQL

Es un **lenguaje** para manipular (guardar, obtener, actualizar, borrar) datos de una base de datos relacional.

Es el lenguaje utilizado para comunicarse con el gestor de base de datos (DBMS).

SQL = Structured Query Language.

Nota: más adelante en la clase veremos SQL en mayor detalle.

Base de Datos No Relacional

Base de Datos No Relacional

Las BD No Relacionales, también llamadas BD NoSQL, son bases de datos en las que los datos se guardan de una forma no tabular. No existen tablas compuestas de filas y columnas, como las hay en BD Relacionales.

Este tipo de BD existen desde los años ‘60, pero adquirieron popularidad en los últimos años con aplicaciones de “Big Data”.

Las BD NoSQL no requieren de un esquema predefinido, por lo cual son mucho más flexibles y adaptables a los requerimientos de la aplicación.

MongoDB

MongoDB (1/2)

- Es un tipo de BD NoSQL.
- Es un BD “documental”, es decir, los datos se guardan en documentos similares a JSON. Técnicamente son documentos BSON (representación binaria de JSON).
- En lugar de tablas, se tienen “colecciones”. En lugar de tener filas y columnas, se tienen documentos. Es decir, una colección se compone de documentos.
- Dispone de su propio lenguaje para hacer consultas (que también usan formato JSON).
- Es open-source, aunque por detrás hay una empresa privada llamada “MongoDB Inc.” que comercializa algunos productos.
- Video: MongoDB en 5 minutos.

MongoDB (2/2)

Ejemplo de un documento en MongoDB:

```
{
  "_id": Object("5ec05006c0329484b8e14c4d"),
  "title": "Historia de Hack Academy",
  "content": "<p>Nulla consequat massa quis enim. Quisque rutrum.</p>",
  "author": "Equipo de HA",
  "url": "https://ha.dev",
  "image": "http://placehold.it/500x200"
}
```

Todo documento en MongoDB debe tener un atributo `_id` que debe ser único en la colección y oficia de clave primaria. Podría ser un número o un string, pero por defecto es de tipo `ObjectId` (docs). → ¿Y por qué no es simplemente un string? Ver este link.

MongoDB – Instalación

- Vamos a usar la versión Community Edition de MongoDB.
- Para instalarlo de forma local ingresar a este link: https://www.mongodb.com/download-center/community.
- Instrucciones detalladas para Windows, macOS, y Linux.
  ¡Por favor leerlas en detalle! 🙏🙏🙏
- En caso de usar Windows, es probable que tengan que crear una carpeta `db` en esta ubicación: `C:\data\db`.
- Para verificar que MongoDB esté instalado en su equipo, correr el comando:

```
mongod --version
```

MongoDB – Línea de comandos

En la documentación de MongoDB se pueden encontrar los comandos necesarios para interactuar con la base de datos desde la terminal.

Veamos un ejemplo. Para acceder a determinada BD, se usa comando `use`:

```
use db-hack-academy
```

A partir de ahora, se hará referencia la BD con la palabra clave `db`.

Para obtener todos los cursos de Hack Academy, se utiliza el siguiente comando:

```
db.courses.find({})
```

Entre las llaves `{}` también se pueden pasar opciones de filtrado.

MongoDB Compass

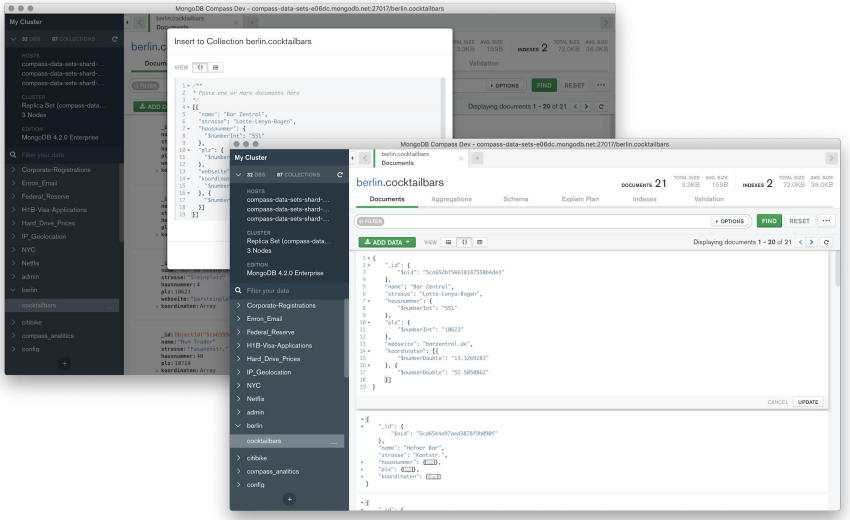
MongoDB Compass

Así como existen herramientas GUI para interactuar con base de datos SQL (ej: Workbench y TablePlus), MongoDB Compass es la herramienta por defecto para interactuar de forma gráfica con una base de datos MongoDB.

Está disponible en Windows, Mac y Linux.

Link de descarga:
https://www.mongodb.com/download-center/compass.

Antes había una versión Community y otra Enterprise, pero ahora sólo hay un Compass (gratuito).



Mongoose

Mongoose (1/4)

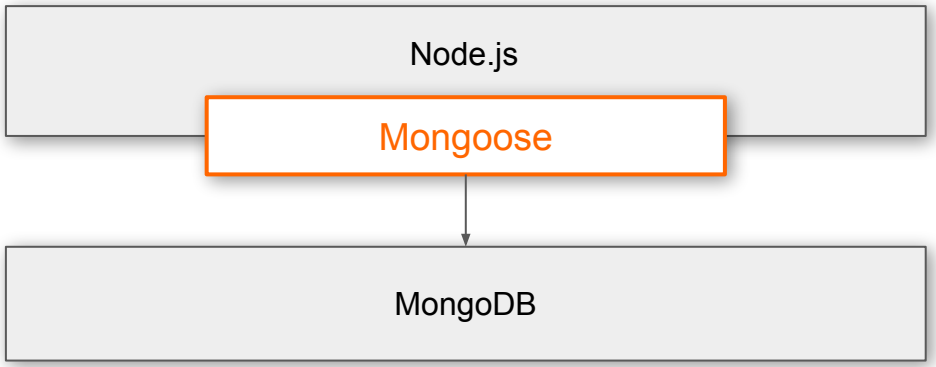
En lugar de usar Mongoose, se podría haber usado el paquete `mongodb` (link), que es el MongoDB driver para Node.js, pero es más complicado.

Para simplificar la interacción con MongoDB desde Node.js, existe un popular ODM (Object-Document Mapper) llamado Mongoose. Ver documentación.

Es similar a lo que sería un ORM (Object-Relational Mapper) para bases de datos SQL como Sequelize (en Node.js) o Eloquent (en PHP/Laravel).

Gracias a Mongoose, serán más fácil cosas como:

- Validaciones.
- Creación de esquemas.
- Consultas.
- Inserciones.
- *Casting.*



Mongoose (2/4)

Para instalar Mongoose ejecutar el comando:

```
npm i mongoose
```

Luego, importarlo con:

```
const mongoose = require("mongoose");
```

Finalmente, nos conectamos a la BD:

```
mongoose.connect("mognodb://localhost/db-hack-academy");
```

`db-hack-academy`es el nombre de la base de datos a la cual nos queremos conectar. Si no existe, Mongoose la crea.
Si al conectarse, en la terminal aparecen mensajes de que hay funcionalidades *deprecadas*, ver este link.
En general, no vamos a precisar cerrar la conexión, pero hay casos en lo que sí será necesario. Ver este link.

Mongoose (3/4) – Conexión

Para constatar si la conexión fue satisfactoriamente ejecutada, Mongoose dispone de `connection` el cual expone *event handlers* muy similares en su sintaxis a los de jQuery. En ellos, se pasan *callbacks* que se ejecutan cuando dichos eventos ocurren.

👉 Notar que la conexión a MongoDB es asíncrona.

⚠️ Este código **no** es obligatorio. Es simplemente por si quieren verificar si la conexión fue establecida.

```
mongoose.connection
  .once("open", () => console.log("¡Conexión con la base de datos establecida!"))
  .on("error", error => console.log(error));
```

Mongoose (4/4) – Conexión

Si bien la conexión es asíncrona, Mongoose permite empezar a trabajar con los modelos antes de que la conexión se haya establecido. Esto se logra mediante un mecanismo llamado *buffering*.

```
const MiModelo = mongoose.model(/*...*/);
// No se rompe, no lanza errores.
// Mongoose simplemente esperará a que exista una conexión
MiModelo.find(/*...*/);

setTimeout(function () {
  mongoose.connect("mongodb://localhost/db-hack-academy");
}, 60000);
```

👉En breve veremos cómo se definen modelos en Mongoose.

La función `setTimeout` es simplemente un ejemplo para mostrar el mecanismo de *buffering*. Esto no se debe haber en los ejercicios.

Mongoose – Esquemas

Mongoose – Esquemas (1/2)

- En Mongoose, todo comienza con un Esquema (*Schema*). De hecho, estamos obligados a crear un esquema para poder interactuar con la BD.
- Cada esquema se mapea con una colección en MongoDB.
- Cada esquema define la forma o estructura que deben tener los documentos dentro de la colección.
- Documentación: https://mongoosejs.com/docs/guide.html.
- Pero… ¿acaso la idea de MongoDB no era justamente prescindir de un esquema? Sí, pero en la práctica los esquemas son útiles (ver link). Si quieren interactuar con MongoDB sin necesidad de crear esquemas, pueden usar el MongoDB Driver para Node.js.

Mongoose – Esquemas (2/2)

Ejemplo:

```
const mongoose = require("mongoose");
const Schema = mongoose.Schema;

const articleSchema = new Schema({
  title: String,
  author: String,
  content: String,
  comments: [ { body: String, date: Date } ],
  date: { type: Date, default: Date.now },
  likes: Number,
});
```

Mongoose – Modelos

Mongoose – Modelos

- A partir de un esquema, se crea un modelo. También se dice que el esquema se *“compila”* en un modelo usando el método `mongoose.model()`.
- Una instancia de un modelo es un documento.
- Los modelos tienen métodos que nos permiten leer, crear y editar documentos de la BD.
- Documentación: https://mongoosejs.com/docs/models.html.

```
const Article = mongoose.model("Article", articleSchema);
```

El primer parámetro es el nombre de la colección (“tabla”) en singular. Mongoose automáticamente buscará en la BD una colección con dicho nombre pero en plural y en minúscula. En este caso: “articles”.

Mongoose – Crear un documento

Mongoose – Crear un documento (1/3)

A partir de un modelo se pueden crear documentos (instancias del modelo):

```
const article = new Article({
  title: "Historia de la academia",
  author: "Hack Academy"
});
```

Luego se puede llamar al método `save` para guardarlo en la BD:

```
article.save();
```

👉 Si se quiere esperar a que el `save` haya finalizado, habría que anteponer un `await`, usar un *callback* ó un `then/catch`.

Mongoose – Crear un documento (2/3)

El método `save` es asíncrono, por lo que se le puede pasar un *callback* que se ejecute una vez que los datos hayan sido guardados:

```
article.save((error, savedArticle) => {
  if (error) return console.log(error);
  console.log("¡Los datos fueron guardados exitosamente!");
});
```

Mongoose – Crear un documento (3/3)

En lugar de usar *callbacks* “tradicionales”, para operar con la función asíncrona se pueden usar promesas (*promises*):

Versión con `then/catch`:

```
article
  .save()
  .then((savedArticle) => {
    console.log(savedArticle);
  })
  .catch((error) => console.log(error));
```

Versión con `async/await`:

```
try {
  const savedArticle = await article.save();
  console.log(savedArticle);
} catch (error) {
  console.log(error);
}
```

Mongoose – Buscar documentos

Mongoose – Buscar documentos

Ejemplo: para buscar todos los artículos presentes en la BD, se puede utilizar el siguiente código. 👉 En la siguiente clase veremos más opciones sobre esto.

Versión con `then/catch`:

```
Article.find().then((articles) => {
  console.log(articles);
});
```

Versión con `async/await`:

```
const articles = await Article.find();
console.log(articles);
```

Ejercicio

Ejercicio (1/2)

1. Tener MongoDB y MongoDB Compass instalado y corriendo en nuestro equipo.
2. Crear un nuevo directorio con el nombre `ha_node2_clase03`.
3. En este directorio, crear un nuevo proyecto de Node con el comando: `npm init -y`.
4. Instalar dependencias: Express, Mongoose y Dotenv:
   `npm i express mongoose dotenv`
5. Crear archivo `server.js` (o `index.js`) en la raíz del proyecto, así como un archivo `routes.js`.
6. Crear archivo `db.js` con la conexión de la base datos. La BD se llamará `clase03`.
7. Crear una carpeta llamada `models` que contendrá los modelos de la aplicación.

Ejercicio (2/2)

8. Crear un archivo `Team.js` dentro de la carpeta anterior, el cual contendrá el esquema y el modelo de un `Team` con los atributos `code`, `name` y `flag` (todos *strings*).
9. Crear una instancia del modelo `Team`.
10. Guardar dicha instancia en la base de datos y esperar a que finalice dicho proceso. Una vez terminado, imprimir en consola un mensaje con el texto *"Se guardó un equipo en la base de datos"*.
11. Verificar (vía Compass) que el documento se haya guardado en MongoDB.
12. Crear el *endpoint* `[GET] /teams` – Para obtener todos los *teams* de MongoDB.

👉 En caso de que queden muchos datos de prueba en la base de datos, se recomienda borrarla con Compass o programáticamente utilizando la función `mongoose.connection.dropDatabase();` .