TEXT CLASS REVIEW

TEMAS A TRATAR EN EL CUE:

- ¿Qué es sequelize?
- Instalación.
- Configuración básica de squelize.
- Definición del modelo en squelize.
- Definición de atributos y tipos de datos.
- Realizando operaciones CRUD (Creación, Lectura, Actualización y Eliminación).

¿QUÉ ES SEQUELIZE?

Es un ORM basado en promesas para Node.js. Soporta PostgreSQL, MySQL, SQLite, y MSSQL. Este cuenta con un sólido soporte de transacciones, relaciones entre tablas, mecanismos de migraciones, carga de datos, entre otros.

Sequelize es un ORM que permite a los usuarios llamar a funciones JavaScript, para interactuar con la base de datos SQL sin escribir consultas reales. Es bastante útil para acelerar el tiempo de desarrollo.

INSTALACIÓN DE SEQUELIZE

Para instalar sequelize en nuestro proyecto de nodejs, solo tenemos que escribir las siguientes líneas en la consola:

```
$ npm install -save sequelize
#Instalar según la base de datos a utilizar, en este caso postgresql
$ npm install --save pg pg-hstore
```

CONFIGURACIÓN BÁSICA DE SEQUELIZE

Con sequelize ya instalado, lo siguiente será configurar nuestra conexión a la base de datos. Lo hacemos de la siguiente manera:

```
const Sequelize = require('sequelize')

const sequelize = new  Sequelize ('database' 'username', 'password', {
                host: 'localhost',
                dialect: 'postgres'
})
```

Lo primero es importar la clase Sequelize. Luego, se crea una nueva instancia de esa clase, la cual recibe como parámetros en su constructor el nombre de la base de datos, el nombre del usuario, la contraseña, y un objeto de configuración donde especificamos el host de nuestra base de datos, y el dialect donde se indica qué base de datos se está utilizando.

Para verificar si la conexión funciona, se utiliza el método **authenticate()**, el cual devuelve una promesa que funciona de la siguiente manera:

```
sequelize .authenticate()
        .then(() => {
                console.log('Conectado')
        })
        .catch(err => {
                console.log('No se Conectó')
        })
```

DEFINICIÓN DEL MODELO EN SEQUELIZE

Los modelos en sequelize nos permiten representar las tablas de una base de datos, y manipular estos datos. Éstos constituyen la columna vertebral de Sequelize, y se trata de una clase que se extiende de Model. El modelo le indica a Sequelize varias características sobre la entidad que representa, como: el nombre de la tabla en la base de datos, qué columnas tiene, y los tipos de datos asociados.

Un modelo en Sequelize tiene un nombre, éste no tiene que ser el mismo de la tabla que representa en la base de datos. Por lo general, los modelos tienen nombres singulares como, por ejemplo:

Usuario; mientras que las tablas de una base de datos tienen nombres en plural, por ejemplo: Usuarios.

Para definir un modelo, se hace uso del método define. Este método recibe como primer parámetro el nombre de la base de datos; y como segundo parámetro, un objeto donde se definen los atributos de la tabla que representa el modelo, para cada atributo se definen los parámetros necesarios, y además, se indica el tipo de dato.

Los modelos en Sequelize pueden definirse de dos maneras:

- Haciendo una llamada al método sequelize.define(nombre_modelo, atributos, opciones).
- Extendiendo de Model, y llamando el método init(atributos, opciones).
- Después que un modelo es definido, éste se encuentra disponible a través de su nombre en sequelize.models.

Definición de un modelo haciendo uso de define:

```
const { Sequelize, DataTypes } = require('sequelize');
const sequelize = new Sequelize('sqlite::memory:');

const Usuario = sequelize.define('Usuario', {
        id : {type: DataTypes.SMALLINT, primaryKey: true},
        nombre: { DataTypes.STRING, allowNull: false},
        apellido:  DataTypes.STRING,
        email:  DataTypes.STRING,
        clave:  DataTypes.STRING
});
})
```

Definición de un modelo extendiendo de Model:

```
const { Sequelize, DataTypes } = require('sequelize');
const sequelize = new Sequelize('sqlite::memory:');

class Usuario extends Model {}
Usuario.init({
        id : {type: DataTypes.SMALLINT, primaryKey: true},
        nombre: { DataTypes.STRING, allowNull: false},
        apellido:  DataTypes.STRING,
```

```
        email:  DataTypes.STRING,
        clave:  DataTypes.STRING

        sequelize, // Es necesario pasar la instancia de la conexión
        modelName: 'Usuario' // Es necesario incluir el nombre del
modelo
});
```

Internamente, sequelize.define llama el Model.init, y por lo tanto, ambas definiciones del modelo son equivalentes.

DEFINICIÓN DE ATRIBUTOS Y TIPOS DE DATOS

Por defecto, Sequelize asume que el valor por defecto de una columna es NULL; sin embargo, en la definición de la tabla, específicamente en la columna, se puede indicar su valor por defecto de la siguiente manera:

```
sequelize.define('Usuario', {
        name: {
                type: DataTypes.STRING,
                defaultValue: "Pedro Márquez" }
});
```

Cuando se define una columna, además de especificar el tipo de dato, y el atributo ***allowNull*** y ***defaultValue***, hay muchos atributos que se pueden agregar. Algunos ejemplos son: ***unique, autoincrement, primaryKey, comment***.

Cada columna que defina en su modelo, debe tener un tipo de datos. Sequelize proporciona una gran cantidad de tipos de datos integrados. Para acceder a uno, se importan los tipos de datos, de la siguiente manera:

```
const { DataTypes } = require('sequelize');
```

Entre los tipos de datos que pueden ser definidos, se encuentran:

- Cadenas de caracteres: STRING, STRING.BINARY, TEXT, TEXT('tiny'), CITEXT (para postgres y SQLite), TSVECTOR (para postgres).

- Booleano: BOOLEAN.
- Números: INTEGER, BIGINT, FLOAT, REAL, DOUBLE, DECIMAL.
- Fechas: DATE, DATEONLY.

REALIZANDO OPERACIONES CRUD (CREACIÓN, LECTURA, ACTUALIZACIÓN Y ELIMINACIÓN)

Consultas de Inserción:

```
const manuel = await User.create({nombre: "Manuel", apellido:
"Pérez"});
```

Consultas de Selección: se puede utilizar el método **findAll()** para leer el contenido completo de la tabla.

```
const users = await User.findAll();
```

Además, se pueden especificar algunos parámetros para las consultas de selección, tales como:

Para seleccionar algunos atributos específicamente:

```
const users = await User.findAll({
        attributes: ['nombre', 'apellido', 'email']
});
```

Además, es posible realizar agregaciones haciendo uso de sequelize.fn:

```
const users = await User.findAll({
        attributes: [
                'nombre',
                [sequelize.fn('COUNT', sequelize.col('edad'),
'edades')],
                 'apellido',
                'email'
        ]
});
```

Del mismo modo, también es posible eliminar algunos atributos seleccionados:

```
const users = await User.findAll({
        attributes: { exclude:  ['nombre'] }
});
```

Y también podemos aplicar la cláusula WHERE, de la siguiente manera:

```
User.findAll({
        where: {
                apellido : "Perez"
        }
});
```

Consulta de modificación: se puede utilizar el método update() para modificar un registro de la tabla.

```
await User.update({ apellido: "Perez"}, {
        where: {
                apellido : null
        }
});
```

Consulta de eliminación: se puede utilizar el método **destroy()** para eliminar un registro de la tabla.

```
await User.destroy({
        where: {
                apellido : "Perez"
        }
});
```

Para eliminar el contenido completo de una tabla:

```
await User.destroy({
        truncate : true
});
```