EXERCISES QUE TRABAJAREMOS EN EL CUE:

- EXERCISE 1: INSTALACIÓN DEL SEQUELIZE Y CREACIÓN DEL MODELO USER.
- EXERCISE 2: CREANDO INSTANCIAS DEL MODELO USER.
- EXERCISE 3: CONSULTAS BÁSICAS EN SEQUELIZE: FINDBYPK, FINDONE, FINDALL.
- EXERCISE 4: CONSULTAS DE ORDER BY, ACTUALIZACIÓN Y ELIMINACIÓN DE UN REGISTRO.

EXERCISE 1: INSTALACIÓN DEL SEQUELIZE Y CREACIÓN DEL MODELO USER

Procedemos a crear el proyecto dentro del directorio sequelize_example.

```
$ npm init -y
```

Instalamos Sequelize, y la dependencia con postgres.

```
$ npm install --save sequelize
$ npm install --save pg pg-hstore
```

Procedemos a crear el archivo de configuración con la base de datos postgres. Para ello, generamos un script llamado authenticateDB.js, que contiene el siguiente código:

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

// Primera forma de autenticarnos
// const path =
'postgres://node_user:node_password@localhost:5432/db_node';
// const sequelize = new Sequelize(path, {
//     operatorsAliases: 0
// });

// Segunda forma de autenticarnos
const sequelize = new Sequelize('db_node', 'node_user',
'node_password', {
```

```
    host: 'localhost',
    port: 5432,
    dialect: 'postgres',
    pool: {
        max: 5,
        min: 0,
        acquire: 30000,
        idle: 10000
    }
});

module.exports = sequelize
```

Observamos que nos podemos conectar creando un nuevo objeto Sequelize con la instrucción **new,** de dos maneras: la primera es colocando un URL path, conformado por una cadena de texto donde se definen los parámetros de conexión; y la segunda es pasando los parámetros de conexión en la función, y especificando otras variables como: host, port, y dialect. En nuestro caso, postgres puede ser mysql, y especificamos una conexión como un pool de conexiones.

Procedemos a crear el modelo relacional User, creando un script **User.js** con el siguiente código:

```
const Sequelize = require('sequelize');
const db = require('./authenticateDB');

// Creación de la tabla user
const User = db.define('users', {
    // propiedades del objeto User
    id: {
        type: Sequelize.INTEGER,
        primaryKey: true,
        autoIncrement: true
    },
    name: {
        type: Sequelize.STRING,
        allowNull: false
    },
    age: {
        type: Sequelize.INTEGER,
        allowNull: false
    }
});

module.exports = User
```

Para crear el modelo relacional User, con la tabla users en la base de datos, procedemos a generar el script **createTusers.js** con el siguiente código:

```
// Requerimos el modelo User
const User = require('./User');

// Procedemos a crear la tabla user con sus propiedades
User.sync().then(() => {
    console.log('Nueva Tabla users ha sido creada');
}).finally(() => {
    User.close
})
```

Al ejecutar el script en la terminal, se observa la creación de la tabla users en la base de datos:

```
$ node createTusers.js
Executing (default): CREATE TABLE IF NOT EXISTS "users" ("id"  SERIAL
, "name" VARCHAR(255) NOT NULL, "age" INTEGER NOT NULL, "createdAt"
TIMESTAMP WITH TIME ZONE NOT NULL, "updatedAt" TIMESTAMP WITH TIME
ZONE NOT NULL, PRIMARY KEY ("id"));
Executing (default): SELECT i.relname AS name, ix.indisprimary AS
primary, ix.indisunique AS unique, ix.indkey AS indkey,
array_agg(a.attnum) as column_indexes, array_agg(a.attname) AS
column_names, pg_get_indexdef(ix.indexrelid) AS definition FROM
pg_class t, pg_class i, pg_index ix, pg_attribute a WHERE t.oid =
ix.indrelid AND i.oid = ix.indexrelid AND a.attrelid = t.oid AND
t.relkind = 'r' and t.relname = 'users' GROUP BY i.relname,
ix.indexrelid, ix.indisprimary, ix.indisunique, ix.indkey ORDER BY
i.relname;
Nueva Tabla users ha sido creada
```

EXERCISE 2: CREANDO INSTANCIAS DEL MODELO USER

Tenemos que el modelo User es una clase, por lo que no se debe crear instancias utilizando directamente el operador **new,** sino el método de compilación, **build** y **save**:

```
const {
    update
} = require('./User');
const User = require('./User');

// Sequelize creando instancias con build, save
const user = User.build({
```

```
    name: "jose",
    age: 25
});

user.save().then(() => {
    console.log('El nuevo usuario ha sido guardado');
}).finally(() => {
    user.close
});
```

Sequelize proporciona el método de **create**, que combina los métodos **build** y **save** en uno solo:

```
// Creando instancias con el metodo create
User.create({
        name: "Pedro",
        age: 40
    })
    .then((result) => {
        console.log('Nuevo usuario creado: ' +
            result.getDataValue('name') +
            ' con el id: ' + result.getDataValue('id'));
    }).catch((err) => {
        console.log('Fallo la inserción del usuario');
        console.log(err);
    }).finally(() => {
        User.close;
    });
```

Ahora, se pueden crear e insertar varias instancias de forma masiva con el método **bulkCreate**:

```
// El método toma un array de objetos

let users = [{
        name: "José",
        age: 25
    },
    {
        name: "Pedro",
        age: 40
    },
    {
        name: "Carlos",
        age: 50
    },
    {
        name: "Antonio",
```

```
        age: 18
    },
    {
        name: "Felipe",
        age: 20
    },
    {
        name: "Juan",
        age: 30
    },
]

User.bulkCreate(users, {
    validate: true
}).then(() => {
    console.log('Usuarios Creados ');
}).catch((err) => {
    console.log('Fallo la inserción de usuarios');
    console.log(err);
}).finally(() => {
    User.close;
});
```

EXERCISE 3: CONSULTAS BÁSICAS EN SEQUELIZE: FINDBYPK, FINDONE, FINDALL

Sequelize proporciona varios métodos para ayudar a consultar la base de datos en busca de datos.

Creamos el script con el siguiente código, para buscar un usuario por su id:

```
const {
    update
} = require('./User');
const User = require('./User');

// Sequelize, búsqueda por pk, findByPk

User.findByPk(2).then((user) => {
    console.log(user.get({
        plain: true
    }));
    console.log('********************')
    console.log(`id: ${user.id}, name: ${user.name}`);
}).finally(() => {
    User.close;
```

```
});
```

Observamos la salida en la terminal:

```
$ node queryUser.js
Executing (default): SELECT "id", "name", "age", "createdAt",
"updatedAt" FROM "users" AS "users" WHERE "users"."id" = 2;
{
  id: 2,
  name: 'Pedro',
  age: 40,
  createdAt: 2022-03-24T17:54:09.822Z,
  updatedAt: 2022-03-24T17:54:09.822Z
}
********************
id: 2, name: Pedro
```

findOne es un método de búsqueda de una sola instancia. Este devuelve la primera instancia encontrada, o nula si no se encuentra. Adecuamos el script con el siguiente código:

```
//Sequelize findOne
// Este método hace la búsqueda para una sola fila
User.findOne({
    where: {
        id: 2
    }
}).then(user => {
    console.log(user.get({
        plain: true
    }));
}).finally(() => {
    User.close;
});
```

Salida en la terminal:

```
$ node queryUser.js
Executing (default): SELECT "id", "name", "age", "createdAt",
"updatedAt" FROM "users" AS "users" WHERE "users"."id" = 2;
{
  id: 2,
  name: 'Pedro',
  age: 40,
  createdAt: 2022-03-24T17:54:09.822Z,
```

```
  updatedAt: 2022-03-24T17:54:09.822Z
}
```

Para la búsqueda de usuarios según su id, hacemos uso del método **findByPk()**, adecuando el script con el siguiente código:

```
// Sequelize con async, await
async function getUser(id) {

    let user = await User.findByPk(id);

    console.log(user.get('name'));
    user.close;
}

getUser(2);
```

El siguiente método **findAll()**, realiza búsquedas en la base de datos de todos los registros, esto es:

```
async function findAllRows() {

    let users = await User.findAll({
        raw: true
    });
    console.table(users);
    users.close;
}

findAllRows()
```

La salida en la terminal:

```
$ node queryUser.js
Executing (default): SELECT "id", "name", "age", "createdAt",
"updatedAt" FROM "users" AS "users" WHERE "users"."id" = 2;
Pedro
```

EXERCISE 4: CONSULTAS DE ORDER BY, ACTUALIZACIÓN Y ELIMINACIÓN DE UN REGISTRO

Sequelize proporciona las opciones de orden y grupo, para trabajar con ORDER BY y GROUP BY.

Procedemos a crear un script llamado **orderUDUser.js,** donde crearemos un método de búsqueda que ordena el campo name en orden descendente, éste es:

```
// Sequelize ORDER BY por cláusula
async function getRowsUsers() {

    let users = await User.findAll({
        order: [
            ['name', 'DESC']
        ],
        attributes: ['name', 'age'],
        raw: true
    })
    console.log(users);
    users.close;
}
getRowsUsers();
```

Seguidamente, tenemos la función de actualizar un usuario por id:

```
//Sequelize Update
async function updateUser(_id, _name) {
    let nameUpdate = {
        name: _name
    }
    let user = await User.update(nameUpdate, {
        where: {
            id: _id
        }
    })
    console.log(user);
    user.close;
}

updateUser(1, "Carlos Ramón");
```

Para eliminar un usuario por medio del método **destroy()**:

```
async function deleteUser(_id) {
    let user = await User.destroy({
        where: {
            id: _id
        }
    })
    console.log(user);
    user.close;
}
deleteUser(2);
```

Para eliminar todos los registros de la tabla users, procedemos a realizarlo de la siguiente manera:

```
// Elimina todos los registros de la tabla user
async function deleteAllUsers() {
    let deleteAllusers = await User.destroy({
        truncate: true
    })
    console.log("tabla users eliminada satisfactoriamente")
    deleteAllUsers.close;
}
deleteAllUsers();
```