EXERCISES QUE TRABAJAREMOS EN EL CUE:

- EXERCISE 1: CONEXIÓN Y CREACIÓN DE LA TABLA USERS EN LA BASE DE DATOS.
- EXERCISE 2: CREAR EL CONSTRUCTOR Y MÉTODO INSERT().
- EXERCISE 3: CREAR EL MÉTODO ESTÁTICO DE BÚSQUEDA DE USUARIO POR ID.
- EXERCISE 4: CREAR EL MÉTODO ESTÁTICO DE BUSCAR TODOS LOS USUARIOS.

El objetivo de este ejercicio es plantear una guía básica paso a paso donde se creará una clase User, generando su tabla en la base de datos user, instanciando objetos, y consultando métodos básicos de inserción y búsquedas.

EXERCISE 1: CONEXIÓN Y CREACIÓN DE LA TABLA USERS EN LA BASE DE DATOS

Generalmente, un ORM está diseñado para brindar acceso a la funcionalidad CRUD básica: CREAR, RECUPERAR, ACTUALIZAR y ELIMINAR.

El mapeo relacional de objetos (ORM) es la técnica de acceder a una base de datos relacional, utilizando un lenguaje de programación orientado a objetos. Éste es una forma en que nuestros programas Javascript administran los datos de la base de datos, al "asignar" sus tablas a las clases, y las instancias de las clases a las filas de esas tablas.

El objeto User incluso puede manejar gran parte de la integración de la base de datos, sabiendo como crear las tablas y el esquema requeridos, insertar registros basados en instancias, actualizar la fila correspondiente a una instancia o eliminarla, encontrar filas, y devolver instancias.

No existe una programación especial para un ORM, solo es un patrón en el que implementamos el código que conecta nuestro programa JS a nuestra base de datos, y asigna una clase a una tabla.

Razones por las que usamos el patrón ORM:

- Reducir el código repetitivo.
- Implementar patrones convencionales que sean organizados y sensatos.

Para construir un ORM en JavaScript, impulsado por SQL y una base de datos relacional, necesitamos un objeto de controlador de base de datos que pueda abrir una conexión a una base de datos, ejecutar SQL, y devolver datos sin procesar en forma de matrices, cadenas, y enteros a nuestro código. En este caso, utilizaremos node-postgres.

Para la siguiente práctica trabajaremos en una carpeta llamada “orm_example”, que ya contiene inicializado el proyecto en node, e instalado el módulo pg de node-postgres.

Procedemos a crear nuestro archivo de conexión llamado dataBase.js, con el siguiente código:

```
const {
    Pool
} = require("pg");

const pool = new Pool({
    user: 'node_user',
    host: 'localhost',
    database: 'db_node',
    password: 'node_password',
    port: 5432,
});

module.exports = {
    pool
};
```

Procedemos a crear la tabla users, a través de la clase User. Para ello, debemos tener el archivo User.js con el siguiente código:

```
const {
    pool
} = require("./dataBase.js");


class User {

    // Creando la tabla en la bases de datos
    static CreateTable() {
        const sql = `CREATE TABLE IF NOT EXISTS users (
            id SERIAL PRIMARY KEY,
            name TEXT,
            age INTEGER)`

        console.log("Preparando para crear la tabla users...")
```

```
16
17          return new Promise(function (resolve) {
18              pool.query(sql, function () {
19                  console.log("...se ha creado la tabla users!")
20                  resolve("Satisfactoriamente")
21              })
22          })
23      }
24  }
25
26  // Exportamos la clase User
27  module.exports = User
```

Para ejecutar la creación de la tabla users, ahora procedemos a crear nuestro script insertUser.js, donde podemos desarrollar el método estático **User.CreateTable()**, con el siguiente código:

```
1  // Requerimos la clase User y el modelo
2  const User = require('./User');
3
4  (async function () {
5      console.log("Ejecutando la migración del objeto User")
6      await User.CreateTable()
7      console.log("Migración Realizada")
8  })();
```

Al ejecutar en la terminal, tenemos:

```
1  $ node insertUser.js
2  Ejecutando la migración del objeto User
3  Preparando para crear la tabla users...
4  ...se ha creado la tabla users!
5  Migración Realizada
```

EXERCISE 2: CREAR EL CONSTRUCTOR Y MÉTODO INSERT()

Procedemos a crear el constructor, con dos propiedades: name y age, y un método **insert()** que es un poco más complejo. Continuamos con nuestro archivo User.js, y le agregamos el siguiente código:

```
1  // Definición del constructor de la Clase User
2      constructor(name, age) {
```

```
        this._name = name
        this._age = age
    }

    // Metodo de inserción de un usuario
    insert() {
        const self = this
        const sql = "INSERT INTO users (name,age) VALUES ($1,$2)
RETURNING id"
        console.log(`Insertando el usuario:  ${self.name} a la base de
datos...`)
        // Resolviendo la consulta como una promesa
        return new Promise(function (resolve) {
            pool.query(sql, [self.name, self.age], function (err, res)
{
                if (err) {
                    console.log(err)
                }
                console.log(`...usuario con id: ${res.rows[0].id}
insertado en la bases de datos`)
                resolve(self)
            })
        })
    }
```

Verificamos construyendo una instancia de objeto User, e insertando en la base de datos. Para esto, agregamos al archivo insertUser.js el siguiente código:

```
// Creamos el primer objeto user1 de nombre José Pérez y edad 35
    const user1 = new User("José Pérez", 35)
    console.log(user1)

    // insertamos en la base de datos con el método creado
    // insert() perteneciente a un método estático de la clase User
    await user1.insert()
```

Al ejecutar en la terminal, obtenemos:

```
$ node insertUser.js
Ejecutando la migración del objeto User
Preparando para crear la tabla users...
...se ha creado la tabla users!
Migración Realizada
User { _name: 'José Pérez', _age: 35 }
```

```
Insertando el usuario:  José Pérez a la base de datos...
...usuario con id: 1 insertado en la bases de datos
```

EXERCISE 3: CREAR EL MÉTODO ESTÁTICO DE BÚSQUEDA DE USUARIO POR ID

La construcción de este método dentro del modelo User, nos permite consultar un usuario según su id. Procedemos a agregar el siguiente código en el archivo User.js:

```
// Método estático de la clase User que busca un usuario con un id
    static Find(id) {
        const sql = `SELECT * FROM users WHERE id = $1 LIMIT 1`
        console.log(`Consultando el usuario con id:  ${id}...`)

        // Realizamos la consulta con una promesa
        return new Promise(function (resolve) {
            pool.query(sql, [id], function (err, resultRow) {
                if (err) {
                    console.error(err)
                }
                console.log(`...encontrado
                     ${JSON.stringify(resultRow.rows)} ...!`)
                const user = new User(resultRow.rows.name,
resultRow.rows.age)
                return user
                resolve(user)
            })
        })
    }
```

Para verificar, agregamos el siguiente código al archivo insertUser.js:

```
// Buscamos un usuario segun el ID 1
    User.Find(1)
```

El resultado en la terminal es:

```
$ node insertUser.js
Consultando el usuario con id:  2...
...encontrado [{"id":2,"name":"José Pérez","age":null}] ...!
```

EXERCISE 4: CREAR EL MÉTODO ESTÁTICO DE BUSCAR TODOS LOS USUARIOS

Finalmente, procedemos a buscar todos los registros insertados en la tabla users de la base de datos. Esta función devuelve los usuarios insertados.

Agregamos al script **User.js** el siguiente código, el cual implementa el método de búsqueda All() dentro del modelo User. Esto es:

```
 // Método estático para buscar todos los usuarios
    static All() {
        const sql = `SELECT * FROM users`
        console.log(`Realizando la consulta de todos los usuarios de
la base de datos...`)
        // Realizamos la consulta con une promesa
        return new Promise(function (resolve) {
            pool.query(sql, function (err, results) {
                console.log(`...encontrados ${results.rowCount}
usuarios registrados!`)
                // Seleccionamos los campos del objeto result con
map()
                // y creamos cada uno de los objetos encontrados como
User
                const users = results.rows.map(function (userRow) {
                    const user = new User(userRow.name, userRow.age)
                    user.id = userRow.id
                    return user
                })
                resolve(users)
            })
        })
    }
```

Verificamos en el archivo insertUser.js, agregando la siguiente sentencia de código:

```
// Buscamos todos los usuarios insertados en la base de datos
// por el metodo creado All()
    let allUser = await User.All()
    console.log(allUser)
```

Salida en la terminal:

```
$ node insertUser.js
Realizando la consulta de todos los usuarios de la base de datos...
```

```
...encontrados 1 usuarios registrados!
[
  User { _name: 'José Pérez', _age: null, id: 1 },
```

Finalmente, el código completo de script ***User.js*** es:

```
const {
    pool
} = require("./dataBase.js");


class User {

    // Creando la tabla en la bases de datos
    static CreateTable() {
        const sql = `CREATE TABLE IF NOT EXISTS users (
            id SERIAL PRIMARY KEY,
            name TEXT,
            age INTEGER)`

        console.log("Preparando para crear la tabla users...")

        return new Promise(function (resolve) {
            pool.query(sql, function () {
                console.log("...se ha creado la tabla users!")
                resolve("Satisfactoriamente")
            })
        })
    }

    // Método estatico de la clase User que busca un usuario con un
id
    static Find(id) {
        const sql = `SELECT * FROM users WHERE id = $1 LIMIT 1`
        console.log(`Consultando el usuario con id:  ${id}...`)

        // Realizamos la consulta con una promesa
        return new Promise(function (resolve) {
            pool.query(sql, [id], function (err, resultRow) {
                if (err) {
                    console.error(err)
                }
                console.log(`...encontrado
${JSON.stringify(resultRow.rows)} ...!`)
                const user = new User(resultRow.rows.name,
resultRow.rows.age)
                return user
                resolve(user)
```

```
            })
        })
    }

    // Método estatico para buscar todos los usuarios
    static All() {
        const sql = `SELECT * FROM users`
        console.log(`Realizando la consulta de todos los usuarios de
la base de datos...`)
        // Realizamos la consulta con une promesa
        return new Promise(function (resolve) {
            pool.query(sql, function (err, results) {
                console.log(`...encontrados ${results.rowCount}
usuarios registrados!`)
                // Seleccionamos los campos del objeto result con
map()
                // y creamos cada uno de los objetos encontrados como
User
                const users = results.rows.map(function (userRow) {
                    const user = new User(userRow.name, userRow.age)
                    user.id = userRow.id
                    return user
                })
                resolve(users)
            })
        })
    }



    // Definición del constructor de la Clase User
    constructor(name, age) {
        this._name = name
        this._age = age
    }

    // Metodo de inserción de un usuario
    insert() {
        const self = this
        const sql = "INSERT INTO users (name,age) VALUES ($1,$2)
RETURNING id"
        console.log(`Insertando el usuario:  ${self._name} a la base
de datos...`)
        // Resolviendo la consulta como una promesa
        return new Promise(function (resolve) {
            pool.query(sql, [self._name, self.age], function (err,
res) {
                if (err) {
```

```
                    console.log(err)
                }
                console.log(`...usuario con id: ${res.rows[0].id}
insertado en la bases de datos`)
                resolve(self)
            })
        })
    }
}

// Exportamos la clase User
module.exports = User
```

Y el código del script **insertUser.js** es:

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

(async function () {
    console.log("Ejecutando la migración del objeto User")
    await User.CreateTable()
    console.log("Migración Realizada")

    // Creamos el primer objeto user1 de nombre José Pérez y edad 35
    const user1 = new User("José Pérez", 35)
    console.log(user1)

    // Creamos el segundo objeto user2 de nombre Juan De Jesús y edad
40
    const user2 = new User("Juan De Jesús", 40)


    // insertamos en la base de datos con el metodo creado
    // insert() perteneciente a un metodo estático de la clase User
    await user1.insert()
    await user2.insert()

    // Buscamos un usuario segun el ID
    User.Find(2)

    // Buscamos todos los usuarios insertados en la base de datos
    // por el metodo creado All()
    let allUser = await User.All()
    console.log(allUser)
})();
```