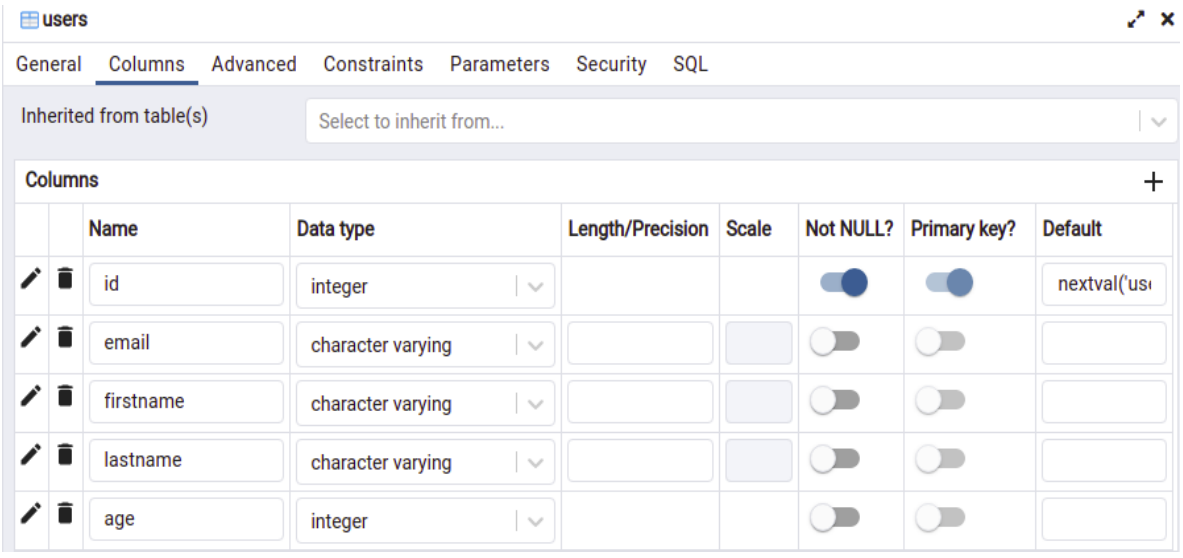
EXERCISES QUE TRABAJAREMOS EN EL CUE:

- EXERCISE 1: INGRESO DE DATOS.
- EXERCISE 2: CONSULTA DE DATOS.
- EXERCISE 3: ACTUALIZACIÓN DE DATOS.
- EXERCISE 4: ELIMINACIÓN DE DATOS.

El objetivo de este ejercicio es plantear una guía paso a paso para ingresar, modificar, y eliminar datos en una base de datos, por medio de las declaraciones SQL: INSERT, UPDATE y DELETE.

PROCEDIMIENTOS DE LA PRÁCTICA

Para la realización de la presente práctica, se cuenta con una tabla ***users*** en la base de datos postgreSQL db_node, con los siguientes campos:



EXERCISE 1: INGRESO DE DATOS

En este paso, se utilizará node-postgres para conectar la aplicación Node.js, a la base de datos de PostgreSQL. Para ello, con node-postgres se creará un pool de conexiones, el cual funciona como un caché para las conexiones de la base de datos, lo que permite que su aplicación reutilice las conexiones para todas las solicitudes. Esto puede acelerar su aplicación, y ahorrar los recursos de su servidor.

Crea y abre un archivo **dataBase.js** en Visual Studio Code, y adecúalo para la conexión de la base de datos:

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

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

module.exports = {
    pool
}
```

Seguido de esto, crea un script que agregue datos a la base de datos de PostgreSQL, específicamente, a la tabla ***users*** de la base de datos ***db_node,*** utilizando el pool de conexiones del archivo **dataBase.js**. Para asegurarse de que el script inserte datos diferentes cada vez que se ejecuta, se dará la funcionalidad de aceptar argumentos de línea de comandos. Al ejecutar el script, se pasarán los campos: email, firstname, lastname, y age del usuario.

Procede a crear y abrir el archivo i***nsertData.js*** en el editor de Visual Studio Code:

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

async function insertUser() {
    const [email, firstname, lastname, age] = process.argv.slice(2);
    console.log(email, firstname, lastname, age);
```

```
}

insertUser();
```

Guarda el archivo, y ejecútalo usando el **node insertUser.js,** pasando los argumentos: **jose@test.com Jose Perez 25.**

En primera instancia, se necesita el objeto de pool del archivo dataBase.js. Esto permite que el script use la conexión, y se conecte a la base de datos.

Luego, se declara la función **insertUser()** como una función asíncrona, con la palabra clave async. Esto permite usar la palabra clave await para hacer que las solicitudes de la base de datos sean asincrónicas.

En la función **insertUser()**, utiliza el módulo de proceso para acceder a los argumentos de la línea de comandos. El método process.argv de Node.js, devuelve todos los argumentos en una matriz, incluidos los node insertUser.js.

Es decir, al ejecutar el script en la terminal con ***node insertUser.js jose@test.com Jose Perez 25***, el método process.argv devolverá una matriz: [node insertUser.js jose@test.com Jose Perez 25]

Para omitir los dos primeros elementos node insertUser.js: agregue el método **slice()** de JavaScript, al método process.argv. Esto devuelve elementos a partir del índice 2 en adelante, y luego se estructuran en variables de: **email, firstname, lastname, age.**

Al ejecutar en la terminal:

```
$ node insertUser.js jose@test.com Jose Perez 25
```

Obtenemos la siguiente salida:

```
jose@test.com Jose Perez 25
```

La función puede acceder a: **email, firstname, lastname, age** del usuario, como argumentos por medio de la línea de comandos. A continuación, se modificará la función **insertUser()** para insertar datos en la tabla de users.

Adecuamos el archivo insertUser.js, agregando el siguiente código resaltado:

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

async function insertUser() {
    const [email, firstname, lastname, age] = process.argv.slice(2);
    const res = await pool.query(
        "INSERT INTO users (email, firstname, lastname, age) VALUES
($1, $2,                        $3, $4)",
        [email, firstname, lastname, age]
    );
    console.log(`Se ha agregado el usuario ${firstname}`);
}

insertUser();
```

La función **insertUser()** define el **email, firstname, lastname, y age** del usuario. Luego, el método espera que **pool.query** de node-postgres tome la instrucción SQL INSERT INTO users (email, firstname,...)... como primer argumento. Esta instrucción SQL inserta un registro en la tabla users, utilizando una consulta parametrizada. $1, $2, $3 y $4 corresponden a las variables de **email, firstname, lastname, age** en la matriz proporcionada al método **pool.query()** como segundo argumento: [**email, firstname, lastname, age**]. Cuando Postgres está ejecutando la declaración, las variables se sustituyen de manera segura, para así poder proteger su aplicación de la inyección de SQL. Después de que se ejecuta la consulta, la función registra un mensaje de éxito usando **console.log()**.

Ajustaremos el código del archivo dentro de la función **insertUser()**, en un bloque try...catch para manejar los errores de tiempo de ejecución:

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

async function insertUser() {
    const [email, firstname, lastname, age] = process.argv.slice(2);
    try {
        const res = await pool.query(
            "INSERT INTO users (email, firstname, lastname, age)
VALUES ($1, $2, $3, $4)",
            [email, firstname, lastname, age]
        );
        console.log(`Se ha agregado el usuario ${firstname}`);
```

```
    } catch (error) {
        console.error(error)
    }
}

insertUser();
```

EXERCISE 2: CONSULTA DE DATOS

En este paso, se recuperarán todos los registros en la tabla de **users** usando node-postgres, y se registrarán en la consola.

Se procede a crear el archivo getAllUser, con el siguiente código:

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

async function getAllUsers() {
    try {
        const res = await pool.query("SELECT * FROM users");
        console.table(res.rows);
    } catch (error) {
        console.error(error);
    }
}

getAllUsers()
```

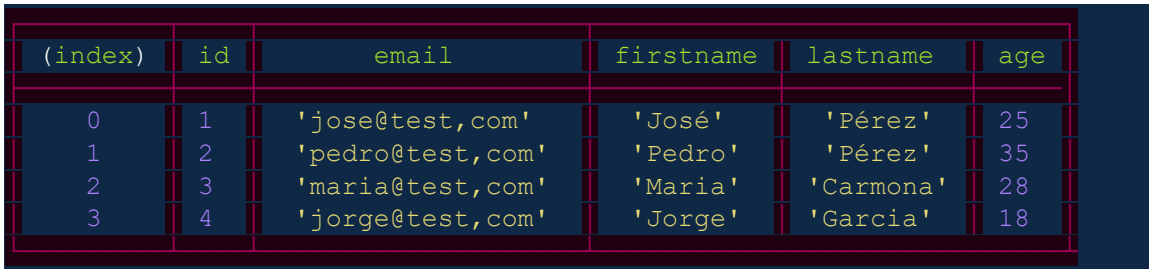
La función **getAllUsers()** lee todas las filas de la tabla **users,** y las registra en la consola. Dentro del bloque de prueba de función, se invoca el método **pool.query()** desde node-postgres, con una instrucción SQL como argumento. La instrucción **SQL SELECT * FROM users** recupera todos los registros de la tabla users. Ya teniéndolos disponibles, la instrucción **console.table()** registra las filas en una tabla.

Si se activa un error, la ejecución saltará al bloque catch y lo registrará. En la última línea, se invoca la función **getAllUsers()**.

Ahora, guarda el archivo getAllUsers.js:

```
$ node getAllUsers.js
```

Salida:



(index)	id	email	firstname	lastname	age
0	1	'jose@test,com'	'José'	'Pérez'	25
1	2	'pedro@test,com'	'Pedro'	'Pérez'	35
2	3	'maria@test,com'	'Maria'	'Carmona'	28
3	4	'jorge@test,com'	'Jorge'	'Garcia'	18

node-postgres devuelve las filas de la tabla en un objeto similar a JSON, y estos objetos se almacenan en una matriz.

Ahora, ya se pueden recuperar datos de la base de datos. Seguidamente, éstos se modificarán en la tabla usando node-postgres.

EXERCISE 3: ACTUALIZACIÓN DE DATOS

Esto se realizará a través de node-postgres, el cual permitirá modificar datos en la base de datos de Postgres, y por ende, cambiar los datos de cualquier registro de la tabla de users.

Se creará un script que toma dos argumentos de la línea de comandos: id y correo. Éste utilizará el valor de identificación **id,** para seleccionar el registro que se desea en la tabla. Mientras que el argumento de correo será el nuevo valor para el registro cuyo correo se desea cambiar.

Procedemos a crear el archivo modifyUser.js, y agregamos el siguiente código para modificar el registro en la tabla ***users:***

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

async function modifyUser() {
    const [id, email] = process.argv.slice(2);
    try {
        const res = await pool.query("UPDATE users
                    SET email = $1
                                    WHERE id = $2",
                                   [email,
                                    id,
                                ]);
```

```
        console.log('Se han actualizado el usuario ${id} con el nuevo
correo  ${email}');
    } catch (error) {
        console.error(error);
    }
}

modifyUser();
```

En el script anterior se necesita el objeto de pool del archivo **dataBase.js**, en su archivo **modifyUser.js.**

Se define una función asíncrona **modifyUser()**, para modificar un registro en la tabla users de la base de datos db_node de Postgres. Dentro de esta función, se definen dos variables: id y email, a partir de los argumentos de la línea de comandos mediante la asignación de estructuración.

Dentro del bloque try, se invoca el método **pool.query** desde **node-postgres**, pasando una declaración SQL como primer argumento. En la sentencia UPDATE SQL, la cláusula WHERE selecciona el registro que coincide con el valor de **id**. Y una vez seleccionado, **SET** email = $1 cambia el valor en el campo de **email** al nuevo valor.

Luego, en la **console.log** se registra un mensaje que se ejecuta al cambiarse el nombre del registro. Y finalmente, se llama a la función **modifyUser()** en la última línea.

EXERCISE 4: ELIMINACIÓN DE DATOS

Para la eliminación de un registro de datos en la tabla users, se realizará por medio de node-postgres, el cual permitirá eliminar el mismo en la base de datos de Postgres. Se encargará de eliminar por completo el registro en la tabla de users.

Se creará un script, que toma un argumento de la línea de comandos: el **id.** Dicho valor de identificación id se usa para seleccionar el registro que desea eliminar en la tabla.

Procedemos a crear el archivo deleteUser.js, y agregamos el siguiente código para eliminar un registro en la tabla users:

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

```
4
5  async function deleteUser() {
6      const [id] = process.argv.slice(2);
7      try {
8          const res = await pool.query("DELETE FROM users WHERE id =
9  $1", [id]);
10         console.log(`Se ha eliminado correctamente el usuario con id:
11 ${id}`);
12     } catch (error) {
13         console.error(error);
14     }
15 }
16
17 deleteUser();
```

Ejecutamos el script:

```
1 $ node deleteUser.js 1
```

Salida:

```
1 Se ha eliminado el usuario con id: 1
```

Ejecutamos el script que liste todos los registros de la tabla users:

```
1 $ node getAllUsers.js
```

(index)	id	email	firstname	lastname	age
0	2	'pedro@test,com'	'Pedro'	'Pérez'	35
1	3	'maria@test,com'	'Maria'	'Carmona'	28
2	4	'jorge@test,com'	'Jorge'	'Garcia'	18

Y podemos observar que no muestra el usuario con id 1. En este sentido, fue eliminado de la tabla users correctamente.