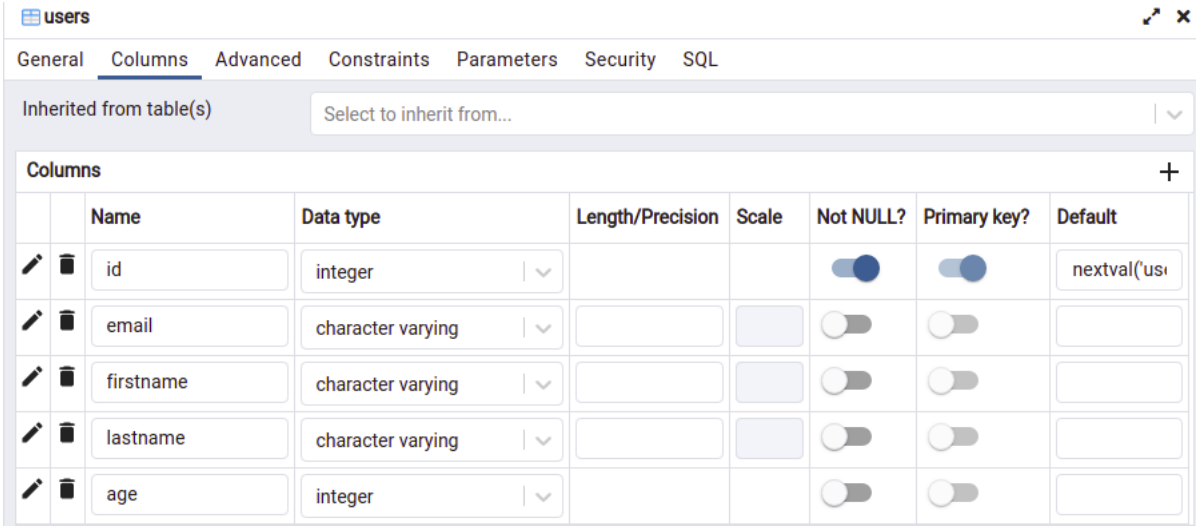
EXERCISES QUE TRABAJAREMOS EN EL CUE:

- EXERCISE 1: PROCESAR INFORMACIÓN POR MEDIO DEL OBJETO RESULT().
- EXERCISE 2: CAPTURA DE ERRORES EN UNA CONSULTA QUERY().
- EXERCISE 3: MANEJO DE CURSORES EN NODE.

PROCEDIMIENTOS DE LA PRÁCTICA

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



Y los datos que contiene la tabla users son los siguientes:

Data Output | Explain | Messages | Notifications

	id [PK] integer	email character varying	firstname character varying	lastname character varying	age integer
1	1	jose@test,com	José	Pérez	25
2	2	pedro@test,com	Pedro	Pérez	35
3	3	maria@test,com	Maria	Carmona	28
4	4	jorge@test,com	Jorge	Garcia	18
5	5	miguelp@test,com	Miguel	Pérez	45

Procedemos a realizar las gestiones de resultados de consultas.

EXERCISE 1: PROCESAR INFORMACIÓN POR MEDIO DEL OBJETO RESULT()

La realización de esta práctica consiste en consultar, y listar todos los usuarios de la tabla users. En primera instancia, nos conectamos a la base de datos a través de node, y crearemos un archivo con el nombre de ***response_processing_result.js***:

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

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

Ahora, creamos una función de “buscar todos”, agregando al archivo lo siguiente:

```
const buscarTodos = async () => {
    try {
        console.log("Función de Buscar todos")

    } catch (error) {
        console.error(error.stack);
    } finally {}
};

buscarTodos()
```

Procedemos a crear la conexión, la consulta query, y listamos los datos como filas a través de objeto result(). La nueva función reescrita queda así:

```
const buscarTodos = async () => {
    try {
        const client = await pool.connect()
        const result = await client.query({
            rowMode: 'array',
            text: 'SELECT * FROM users',
        })

        // listar los datos como filas
```

```
        for (let row of result.rows) {
            console.log(row);
        }
        return result;

    } catch (error) {
        console.error(error.stack);
    } finally {
        await pool.end(); // Cerrando conexión
    }
};
```

Para listar los resultados de las filas en una tabla, agregamos:

```
// Resultados de las filas en una tabla
console.table(result.rows);
```

Mostrando los resultados de los nombres de las columnas, agregamos:

```
// Resultados de los nombres de columnas
for (let field of result.fields) {
    console.log(field.name);
}
```

Si no encontramos registros en la búsqueda, usamos ***result.rows.length***. Esto es validando si se encontraron, y en caso contrario, se emitirá un mensaje. El código completo es:

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

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

const buscarTodos = async () => {
    try {
        const client = await pool.connect()
        const result = await client.query({
```

```
            rowMode: 'array',
            text: 'SELECT * FROM users',
        })

        if (result.rows.length) {

            console.log('Numero de Registros encontrados: ',
result.rowCount)
            console.log('Consulta realizada con: ', result.command)

            // listar los datos como filas
            for (let row of result.rows) {
                console.log(row);
            }
            // Resultados de las filas en una tabla
            console.table(result.rows);

            // Resultados de los nombres de columnas
            for (let field of result.fields) {
                console.log(field.name);
            }

            return result;
        }
        console.log('No se encontraron registros')
    } catch (error) {
        console.error(error.stack);
    } finally {
        await pool.end(); // Cerrando conexión
    }
};

buscarTodos()
```

EXERCISE 2: CAPTURA DE ERRORES EN UNA CONSULTA QUERY()

Partiendo del ejercicio anterior, vamos a capturar errores. Esta captura está basada en un try/catch, si el catch(error) posee error lo detallamos; generalmente, los códigos de error que emite postgres se encuentran en un apéndice en la siguiente dirección web.

Colocaremos un error en el query de la sentencia text de SQL, quedando de la siguiente manera:

```
text: 'SEECT * FROM users  WHERE',
```

La salida del sistema nos refleja:

```
$ node test.js
error: syntax error at or near "SEECT"
```

Para verificar que código de error hay, agregamos en el catch lo siguiente:

```
...
} catch (error) {
    console.error(error.stack);
    console.log(error.code) // impresión del código de error
}
...
```

Al ejecutarlo en la terminal, obtenemos lo siguiente:

```
 at Socket.Readable.push (_stream_readable.js:209:10)
    at TCP.onStreamRead (internal/stream_base_commons.js:186:23)
42601
```

El código 42601 pertenece a un error de sintaxis, que efectivamente es porque el **SELECT** está escrito como SEECT.

De esta manera, ya podemos capturar distintos errores dentro de nuestro código, al procesar una consulta a la base de datos. Finalmente, el código completo para la captura de errores, en este caso particular, de Error de sintaxis y de password inválido, queda de la siguiente manera:

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

const pool = new Pool({
    user: 'node_user',
    host: 'localhost',
    database: 'db_node',
    password: 'node_password',
    port: 5432,
})
const query = {
    rowMode: 'array',
    text: 'SELECT * FROM users  WHERE age > 45',
}
```

```
17
18 const buscarTodos = async () => {
19     try {
20         const client = await pool.connect()
21         // const result = await client.query({
22         //     rowMode: 'array',
23         //     text: 'SELECT * FROM users  WHERE age > 45',
24         // })
25         const result = await client.query(query)
26
27         if (result.rows.length) {
28
29             console.log('Numero de Registros encontrados: ',
30 result.rowCount)
31             console.log('Consulta realizada con: ', result.command)
32
33             // listar los datos como filas
34             for (let row of result.rows) {
35                 console.log(row);
36             }
37             // Resultados de de las filas en una tabla
38             console.table(result.rows);
39
40             // Resultados de los nombres de columnas
41             for (let field of result.fields) {
42                 console.log(field.name);
43             }
44
45             return result;
46         }
47         console.log('No se encontraron registros')
48     } catch (error) {
49         if (error.code == '42601') {
50             console.log("\n ERROR! \n Error de Sintaxis\n")
51         }
52         if (error.code == '28P01') {
53             console.log("\n ERROR! \n Password inválido\n")
54         } else {
55             console.error(error.stack);
56             console.log(error.code)
57         }
58
59     } finally {
60         await pool.end(); // Cerrando conexión
61     }
62 };
63
64 buscarTodos()
```

EXERCISE 3: MANEJO DE CURSORES EN NODE

Instalamos el siguiente paquete:

```
npm install pg-cursor
```

Tomando el ejemplo anterior, y suponiendo que tenemos un conjunto bastante amplio de registros de usuarios registrados de la tabla users, procedemos a utilizar los cursores para gestionar la manipulación de los datos. Para esto, creamos un archivo llamado: ***cursor_query_node.js.***

```
const {
    Pool
} = require('pg')
const Cursor = require('pg-cursor')
const assert = require('assert')
const pool = new Pool({
    user: 'node_user',
    host: 'localhost',
    database: 'db_node',
    password: 'node_password',
    port: 5432,
})

const buscarTodos = async () => {
    try {
        const client = await pool.connect()
        const cursor = client.query(new Cursor('SELECT * from users'))

        // Leemos los dos primeros registros
        let rows = await cursor.read(2)
        // procedemos a realizar cualqui procedimiento con los
registros
        console.log(rows)
        console.table(rows)
        console.log(rows.length)
        console.assert(rows.length == 2) // Verificamos

        // volvemos a leer los siguientes 2 registros
        rows = await cursor.read(2)
        // procedemos a realizar cualqui procedimiento con los
registros
        console.log(rows)
        console.table(rows)
        console.log(rows.length)
        console.assert(rows.length == 2) // Verificamos
```

```
36
37          // volvemos a leer los siguientes 2 registros
38          rows = await cursor.read(2)
39          // procedemos a realizar cualqui procedimiento con los
40  registros
41          console.log(rows)
42          console.table(rows)
43          console.log(rows.length)
44          console.assert(rows.length == 2) // Verificamos
45
46          // volvemos a leer los siguientes 2 registros
47          rows = await cursor.read(2)
48          // procedemos a realizar cualqui procedimiento con los
49  registros
50          console.log(rows)
51          console.table(rows)
52          console.log(rows.length)
53          console.assert(rows.length == 2) // verificamos hasta no
54  tengamos mas registros
55
56
57
58      } catch (error) {
59          console.error(error.stack);
60          console.log(error.code)
61      } finally {
62          await pool.end(); // Cerrando conexión
63      }
64  };
65  buscarTodos()
```

Mejorando el código. Por ejemplo, gestionar el resultado cada cuatro registros en nuestro caso de pruebas:

```
 1  const {
 2      Pool
 3  } = require('pg')
 4  const Cursor = require('pg-cursor')
 5  const assert = require('assert')
 6  const pool = new Pool({
 7      user: 'node_user',
 8      host: 'localhost',
 9      database: 'db_node',
10      password: 'node_password',
11      port: 5432,
12  })
```

```
13
14 const buscarTodos = async () => {
15     try {
16         const client = await pool.connect()
17         const cursor = client.query(new Cursor('SELECT * from users'))
18
19         // Leemos los dos primeros registros
20         let rows = await cursor.read(4)
21         console.log(rows)
22         while (rows.length) {
23             console.table(rows)
24             console.log(rows.length)
25             console.assert(rows.length == 4)
26             rows = await cursor.read(4)
27
28         }
29
30     } catch (error) {
31         console.error(error.stack);
32         console.log(error.code)
33     } finally {
34         await pool.end(); // Cerrando conexión
35     }
36 };
37
38 buscarTodos()
```