

## EXERCISES QUE TRABAJAREMOS EN EL CUE:

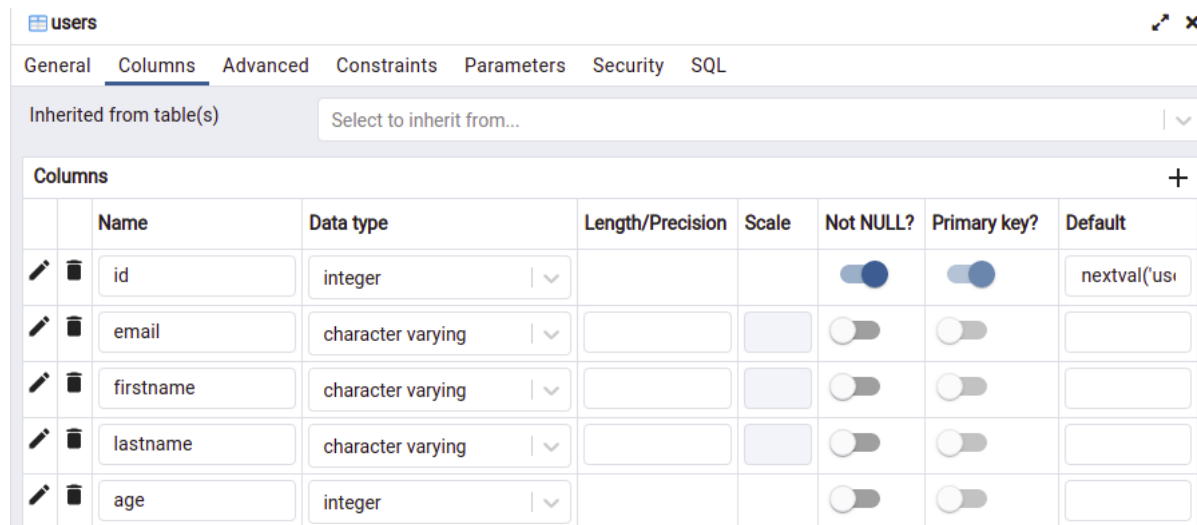
- EXERCISE 1: CREAR UNA CONSULTA TANTO CON CALLBACK COMO CON PROMISE A UNA TABLA DE UNA DB.
- EXERCISE 2: CONSULTA EN UNA CONEXIÓN POOL CON POOL.QUERY().
- EXERCISE 3: CONSULTAS ASINCRONAS CON ASYNC/AWAIT Y CALLBACK.
- EXERCISE 4: CONSULTAS PARAMETRIZADAS.
- EXERCISE 5: CONSULTAS CON OBJETOS.

El objetivo de este ejercicio es plantear una guía paso a paso para la obtención de datos de un gestor de base de datos, realizando: consultas de texto plano, parametrizadas, asíncronas, consultas parametrizadas, declaraciones preparadas, y consultas de modo filas Row Mode











## 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:



The screenshot shows the 'users' table structure in a web interface. The table has the following columns:

	Name	Data type	Length/Precision	Scale	Not NULL?	Primary key?	Default
 	id	integer			<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	nextval('us
 	email	character varying			<input type="checkbox"/>	<input type="checkbox"/>	
 	firstname	character varying			<input type="checkbox"/>	<input type="checkbox"/>	
 	lastname	character varying			<input type="checkbox"/>	<input type="checkbox"/>	
 	age	integer			<input type="checkbox"/>	<input type="checkbox"/>	

Los datos que contiene la tabla users es la siguiente:

Data Output		Explain	Messages	Notifications	
	<b>id</b> [PK] integer 	<b>email</b> character varying 	<b>firstname</b> character varying 	<b>lastname</b> character varying 	<b>age</b> 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 distintas consultas, para así obtener información de la misma.

## EXERCISE 1: CREAR UNA CONSULTA TANTO CON CALLBACK COMO CON PROMISE A UNA TABLA DE UNA DB

El método `query()` como parámetro, recibe una cadena o un string de la consulta que se quiere realizar. Éste devuelve una promesa que, de ser exitosa, muestra los valores de los datos de la consulta, en los rows de la respuesta como propiedades.

Para consultar con Callback como cliente, creamos el archivo `query_callback_client.js`.

```
1 const {
2   Client
3 } = require("pg");
4
5 const cliente = new Client({
6   user: 'node_user',
7   host: 'localhost',
8   database: 'db_node',
9   password: 'node_password',
10  port: 5432,
11 })
12
13 cliente.connect()
14
15 cliente.query('SELECT NOW() as now', (err, res) => {
16   if (err) {
17     console.log(err.stack)
18   } else {
```

```
19     console.log(res.rows[0])
20   }
21 })
22
23 // Consulta con callback
24 cliente.query('SELECT * FROM users', (err, res) => {
25   if (err) {
26     console.log(err.stack)
27   } else {
28     console.log(res.rows)
29     cliente.end() // Cerrando la conexión
30   }
31 })
```

Salida:

```
1 $ node query_callback_client.js
2
3 { now: 2022-03-16T18:22:54.753Z }
4 [
5   {
6     id: 1,
7     email: 'jose@test,com',
8     firstname: 'José',
9     lastname: 'Pérez',
10    age: 25
11  },
12  {
13    id: 2,
14    email: 'pedro@test,com',
15    firstname: 'Pedro',
16    lastname: 'Pérez',
17    age: 35
18  },
19  {
20    id: 3,
21    email: 'maria@test,com',
22    firstname: 'Maria',
23    lastname: 'Carmona',
24    age: 28
25  },
26  {
27    id: 4,
28    email: 'jorge@test,com',
29    firstname: 'Jorge',
30    lastname: 'Garcia',
```

```
31     age: 18
32   },
33   {
34     id: 5,
35     email: 'miguel@test.com',
36     firstname: 'Miguel',
37     lastname: 'Pérez',
38     age: 45
39   },
40   {
41     id: 6,
42     email: 'miguel@test.com',
43     firstname: 'Miguel',
44     lastname: 'Pérez',
45     age: 45
46   }
47 ]
```

Para consultar con Promises como cliente, creamos el archivo *query\_promises\_client.js*.

```
1 const {
2   Client
3 } = require("pg");
4
5 const cliente = new Client({
6   user: 'node_user',
7   host: 'localhost',
8   database: 'db_node',
9   password: 'node_password',
10  port: 5432,
11 })
12
13 cliente.connect()
14
15 cliente
16   .query('SELECT NOW() as now')
17   .then(res => console.log(res.rows[0]))
18   .catch(e => console.error(e.stack))
19
20 // Consulta con promises
21 cliente.query('SELECT * FROM users')
22   .then(res => {
23     console.log(res.rows)
24     cliente.end() // Cerrando la conexión
25   })
```

## EXERCISE 2: CONSULTA EN UNA CONEXIÓN POOL CON POOL.QUERY()

Creamos el archivo *query\_callback\_promises\_pool.js*.

```
1 const { Pool } = require("pg");
2
3 const pool = new Pool({
4   user: 'node_user',
5   host: 'localhost',
6   database: 'db_node',
7   password: 'node_password',
8   port: 5432,
9 })
10
11
12 // Consulta con callback
13 console.log('Consulta con Callback')
14
15 pool.query('SELECT * FROM users WHERE id = 1', (err, res) => {
16   if (err) {
17     console.log(err.stack)
18   } else {
19     console.log(res.rows)
20     pool.end() // Cerrando la conexión
21   }
22 })
23
24 // Consulta con promises
25 console.log('Consulta con Promesas')
26 pool.query('SELECT * FROM users WHERE id = 4')
27   .then(res => {
28     console.log(res.rows)
29     pool.end() // Cerrando la conexión
30   })
```

## EXERCISE 3: CONSULTAS ASINCRONAS CON ASYNC/AWAIT Y CALLBACK

Para las consultas con ASYNC/AWAIT, creamos el archivo *query\_async\_await\_asincronas.js*, con el siguiente contenido de código:

```
1 const {
2   Client,
3   Pool
4 } = require("pg");
5
```

```
6 const cliente = new Client({
7   user: 'node_user',
8   host: 'localhost',
9   database: 'db_node',
10  password: 'node_password',
11  port: 5432,
12 })
13
14 // Como Cliente
15 async function buscarTodos() {
16
17
18   try {
19     await cliente.connect();
20     const res = await cliente.query('SELECT * FROM users');
21     for (let row of res.rows) {
22       console.log(row);
23     }
24     return res;
25     cliente.end()
26   } catch (err) {
27     console.error(err);
28   }
29 }
30
31 buscarTodos().then(res => console.log("Se mostraron todos los
32 registros de usuarios"));
```

Para las consultas Callback, creamos el archivo *query\_callback\_asincronas.js*, con el siguiente contenido de código:

```
1 const {
2   Pool
3 } = require("pg");
4
5 const pool = new Pool({
6   user: 'node_user',
7   host: 'localhost',
8   database: 'db_node',
9   password: 'node_password',
10  port: 5432,
11 })
12
13 pool.connect()
14 .then((pool) => {
```

```
15 pool.query('SELECT * FROM users')
16   .then(res => {
17     for (let row of res.rows) {
18       console.log(row);
19     }
20   })
21   .catch(err => {
22     console.error(err);
23   })
24 })
```

## EXERCISE 4: CONSULTAS PARAMETRIZADAS

Para las consultas parametrizadas, se pasa la consulta como parámetros al servidor PostgreSQL, donde éstos se sustituyen de forma segura con código de sustitución dentro del propio servidor. Para este ejemplo, crearemos el archivo `query_parametrizada.js`, con el siguiente contenido:

```
1 const {
2   Pool
3 } = require("pg");
4
5 const pool = new Pool({
6   user: 'node_user',
7   host: 'localhost',
8   database: 'db_node',
9   password: 'node_password',
10  port: 5432,
11 })
12
13 // callback
14 console.log('Consulta parametrizada con callback')
15 const text = 'SELECT * FROM users WHERE age > $1'
16 const values = [30] // Array de valores, seleccionamos los mayores de
17 30
18 pool.query(text, values, (err, res) => {
19   if (err) {
20     console.log(err.stack)
21   } else {
22     console.log(res.rows)
23   }
24 })
25
26 // Promise
27 console.log('Consulta parametrizada con promesa')
```

```
29 const sql = 'SELECT * FROM users WHERE id = $1'
30 const value = [4] // Array de valores, seleccionamos los mayores de 30
31
32 pool
33   .query(sql, value)
34   .then(res => {
35     console.log(res.rows[0])
36   })
37   .catch(e => console.error(e.stack))
```

## EXERCISE 5: CONSULTAS CON OBJETOS

El objeto para la consulta puede tomar una cadena, o una matriz opcional de parámetros, y en la misma, se puede definir el **rowMode** como **Array**. Para ello, debemos crear el siguiente archivo **query\_object\_row\_mode.js**, con el siguiente código:

```
1 const {
2   Pool
3 } = require("pg");
4
5 const pool = new Pool({
6   user: 'node_user',
7   host: 'localhost',
8   database: 'db_node',
9   password: 'node_password',
10  port: 5432,
11 })
12
13 // Creamos el Objeto
14 const query = {
15   text: 'SELECT * FROM users WHERE age > $1',
16   values: [30],
17   rowMode: 'array' // Definiendo modo fila
18 }
19
20 // Realizamos la consulta con el Objeto query
21 pool.query(query, (err, res) => {
22   if (err) {
23     console.log(err.stack)
24   } else {
25     console.log(res.rows)
26   }
27 })
```



Salida en la terminal:

```
1 $ node query_object_row_mode.js
2 [
3   [ 2, 'pedro@test,com', 'Pedro', 'Pérez', 35 ],
4   [ 5, 'miguelp@test,com', 'Miguel', 'Pérez', 45 ],
5 ]
```