EXERCISES QUE TRABAJAREMOS EN EL CUE:

- EXERCISE 1: CREAR EL DIRECTORIO E INICIALIZAR EL PROYECTO CON NPM.
- EXERCISE 2: INSTALACIÓN DEL PAQUETE PG.
- EXERCISE 3: CREAR UNA BASE DE DATOS EN POSTGRESQL Y EL USUARIO DE LA BASE DE DATOS.
- EXERCISE 4: CONECTARSE COMO CLIENTE Y COMO POOL.
- EXERCISE 5: CONEXIÓN CON UN URI STRING.

El objetivo de este ejercicio es plantear una guía paso a paso, donde se realiza el proceso de conexión a una base de datos PostgresSQL, configuración de paquetes necesarios, conexión como cliente y como pooling de conexiones.

EXERCISE 1: CREAR EL DIRECTORIO E INICIALIZAR EL PROYECTO CON NPM

Crear el directorio con el comando mkdir.

```
$ mkdir node_postgres
```

Ir hasta el directorio recientemente creado:

```
$ cd node_postgres
```

Inicializar el directorio con el archivo package.json, usando el comando “npm init -y” que creará los valores por defecto.

```
$ npm init -y
```

EXERCISE 2: INSTALACIÓN DEL PAQUETE PG

```
$ npm install pg
```

Una vez configurado el proyecto, e instalado en la dependencia de node-postgres, se procede a crear un usuario y una base de datos en PostgreSQL.

EXERCISE 3: CREAR UNA BASE DE DATOS EN POSTGRESQL Y EL USUARIO DE LA BASE DE DATOS

En este paso, se creará un usuario de base de datos, y la base de datos para el proyecto. Cuando se instala Postgres en Ubuntu por primera vez, éste genera un usuario postgres en su sistema, un usuario de base de datos llamado postgres, y una base de datos postgres. El usuario postgres le permite abrir una sesión de PostgreSQL, donde se pueden realizar tareas administrativas, tales como crear usuarios y bases de datos.

PostgreSQL utiliza un esquema de conexión de autenticación de identidad, el cual permite a un usuario de Ubuntu iniciar sesión en el Shell de Postgres, siempre que se le dé un nombre similar al de su usuario. Como ya se tiene un usuario de postgres en Ubuntu, y uno en PostgreSQL creado en su nombre, podrá iniciar sesión en el Shell de Postgres. En este caso, previamente se creó un usuario admin, con todos los privilegios administrativos.

```
root@25ccad39aab3:/# psql --u admin
```

--u: una bandera que cambia el usuario al dado en Ubuntu. Al pasar el usuario de admin como argumento, se cambiará el usuario de Ubuntu a admin.

psql: un programa de terminal interactivo de Postgres, donde se pueden ingresar comandos SQL para crear bases de datos, roles, tablas, entre otros.

```
psql (14.2 (Debian 14.2-1.pgdg110+1))
Type "help" for help.

admin=#
```

admin es el nombre de la base de datos con la que interactuará, y el # indica que ha iniciado sesión como superusuario.

Para la aplicación Node, se generará un usuario y una base de datos separados, los cuales serán usados por la aplicación para conectarse a Postgres. Para ello, crea un nuevo rol con una contraseña segura:

```
admin=# CREATE USER node_user WITH PASSWORD 'node_password';
```

A continuación, crea una base de datos db_node, y asigna la propiedad al usuario creado anteriormente:

```
admin=# CREATE DATABASE db_node OWNER node_user;
```

Salimos de la terminal interactiva de postgres:

```
admin=#\q
```

Accedemos a la nueva base de datos, que fue creada con el nuevo usuario:

```
root@25ccad39aab3:/# psql -d db_node --u node_user
```

Verificamos la conexión con el comando \conninfo:

```
db_node=> \conninfo
```

Salida:

```
You are connected to database "db_node" as user "node_user" via socket
in "/var/run/postgresql" at port "5432".
```

El resultado confirma que, efectivamente, se ha iniciado sesión como node_user, y está conectado a la base de datos de db_node.

EXERCISE 4: CONECTARSE COMO CLIENTE Y COMO POOL

Creamos un archivo conexionDB.js, y agregamos lo siguiente como cliente:

```
// Modulo npm node-postgres
const { Client } = require('pg')

// Datos para la conexión a la base de datos
const cliente = new Client({
    user: 'node_user',
    host: 'localhost',
    database: 'db_node',
    password: 'node_password',
    port: 5432,
})

// Conectando al cliente
cliente.connect()
```

```
15
16 // Realizando una consulta para verificar si hay error
17 cliente.query('SELECT NOW()', (err, res) => {
18     console.log(err, res)
19     cliente.end()
20 })
```

Al ejecutar node conexionDB.js, observamos la salida:

```
(base) luispc@FullStack-Dev:node-postgres$ node conexionDB.js
null Result {
  command: 'SELECT',
  rowCount: 1,
  oid: null,
  rows: [ { now: 2022-03-15T23:45:30.741Z } ],
  fields: [
    Field {
      name: 'now',
      tableID: 0,
      columnID: 0,
      dataTypeID: 1184,
```

Para la conexión como Pool:

El pool se recomienda para cualquier aplicación que deba ejecutarse durante un período prolongado de tiempo. Cada vez que se crea un cliente, se tiene que hacer un protocolo de enlace con el servidor PostgreSQL, y eso puede implicar algún tiempo. Un grupo de conexiones reciclará una cantidad predeterminada de objetos de cliente, para que así el protocolo de enlace no tenga que realizarse con tanta frecuencia.

```
1 // Modulo npm node-postgres
2 const {
3     Pool
4 } = require('pg')
5
6 // Datos para la conexión a la base de datos
7 const pool = new Pool({
8     user: 'node_user',
9     host: 'localhost',
```

```
    database: 'db_node',
    password: 'node_password',
    port: 5432,
})


// Realizando una consulta para verificar si hay error
pool.query('SELECT NOW()', (err, res) => {
    console.log(err, res)
    pool.end()
})
```

Utilice un pool si tiene, o espera tener, varias solicitudes simultáneas. Su función es proporcionar un pool de instancias de clientes abiertos reutilizables (reduce la latencia cada vez que un cliente puede reutilizarse).

En ese caso, definitivamente no desea llamar a pool.end() cuando se complete su consulta, sino que desea reservarlo para cuando finalice su aplicación, pues éste elimina todas las instancias de cliente abiertas. (Recuerde, el objetivo es mantener un número fijo de instancias de clientes disponibles).

EXERCISE 5: CONEXIÓN CON UN URI STRING

Esto es común en varios entornos, donde la cadena de conexión de la base de datos se proporciona a su aplicación a través de una variable de entorno.

Análisis de cadenas de conexión presentado por pg-connection-string:

```
// Modulo npm node-postgres
const {
    Pool,
    Client
} = require("pg");

// URI de conexión en string
const connectionString =
'postgresql://node_user:node_password@localhost:5432/db_node'

// conectando con una conexión al pool.
const pool = new Pool({
    connectionString,
})
pool.query('SELECT NOW()', (err, res) => {
```

```
    console.log(err, res)
    pool.end()
})

const client = new Client({
    connectionString,
})
client.connect()
client.query('SELECT NOW()', (err, res) => {
    console.log(err, res)
    client.end()
})
```