EXERCISES QUE TRABAJAREMOS EN EL CUE:

- EXERCISE 1: CREAR UN SCRIPT EN NODE PARA UNA TABLA EN LA BASE DE DATOS.
- EXERCISE 2: INSERTAR REGISTROS EN LA TABLA CUENTAS.
- EXERCISE 3: REALIZAR TRANSACCIONES CON BEGIN, COMMIT Y ROLLBACK CON TRY CATCH.

El objetivo de este ejercicio es plantear una guía de entendimiento sobre las transacciones en una base de datos, específicamente con node-postgres, y el uso de las instrucciones Begin, Commit, y Rollback en un pool de conexiones.

EXERCISE 1: CREAR UN SCRIPT EN NODE PARA UNA TABLA EN LA BASE DE DATOS

Para realizar la siguiente práctica, continuaremos con el ejercicio desarrollado en el CUE 05, donde explicamos un ejemplo clásico de transacciones (tx) en un banco. Para ejemplificarlo, crearemos la tabla Cuentas en la base de datos. Previamente, debemos conectarnos a la base de datos:

Crearemos el script 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
};
```

De inmediato, crearemos un script ***createTableCuentas.js*** para la tabla llamada “Cuentas” en la base de datos, con el siguiente código:

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


const query = `CREATE TABLE IF NOT EXISTS cuentas (
"id"
SERIAL,
"nombre"
VARCHAR(50) NOT NULL,
    "balance"
DEC(15, 2) NOT NULL,
    PRIMARY KEY("id")
);`

async function createTableCuentas() {
    try {
        const res = await pool.query(query);
        console.table(res);
        console.log("Se creo satisfactoriamente la tabla cuentas")
    } catch (error) {
        console.error(error);
    }
}

createTableCuentas()
```

Al ejecutar el script, procedemos a crear la tabla Cuentas:

```
$  node createTableCuentas.js
```

(index)	_types	text	binary	Values
command				'CREATE'
rowCount				null
oid				null
rows				
fields				
_parsers				undefined
_types	[Object]	{}	{}	
RowCtor				null
rowAsArray				false

Así se ha creado satisfactoriamente la tabla Cuentas.

EXERCISE 2: INSERTAR REGISTROS EN LA TABLA CUENTAS

Se creará un Array de registros de cuentas, el cual contiene el nombre y monto de cada registro en la tabla **cuentas**, luego se ingresará por medio de una función insertsCuentas, haciendo uso del módulo pg-format, el cual permite crear consultas dinámicas; en este caso particular, inserciones dinámicas a través de un Array de objetos a la tabla Cuentas.

Procedemos a crear el archivo **insertsCuentasArray.js,** con el siguiente código:

```
const {
    pool
} = require("./dataBase.js");
var format = require('pg-format');


const registros = [
    ['Jose', 1000],
    ['Pedro', 2000],
    ['Maria', 3000],
    ['Antonny', 4000],
    ['Marcos', 5000],
    ['Juan', 6000],
    ['William', 7000],
    ['Miguel', 8000],
    ['Alberto', 9000],
    ['Carlos', 10000],
    ['Carolina', 11000],
    ['Julián', 12000],
    ['Fernanda', 13000]
]

async function insertsCuentas() {
    try {
        const result = await pool.query(format("INSERT INTO cuentas
(nombre, balance) VALUES %L", registros))
        if (result.err) {
            console.log(result.err)
            return result.err
        }
        console.table(`Se ejecuto el comando: ` + result.command +
`con ` + result.rowCount + ` inserciones`);
        console.table(registros);

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

```
    }
}

insertsCuentas();
```

En primera instancia, nos conectamos a la base de datos, requiriendo el archivo dataBase.js. Luego, creamos un Array de objetos llamado registros, que contiene cada registro, el nombre, y el balance:

```
const registros = [
    ['Jose', 1000],
    ['Pedro', 2000],
    ['Maria', 3000],
    ['Antonny', 4000],
    ['Marcos', 5000],
    ['Juan', 6000],
    ['William', 7000],
    ['Miguel', 8000],
    ['Alberto', 9000],
    ['Carlos', 10000],
    ['Carolina', 11000],
    ['Julián', 12000],
    ['Fernanda', 13000]
]
```

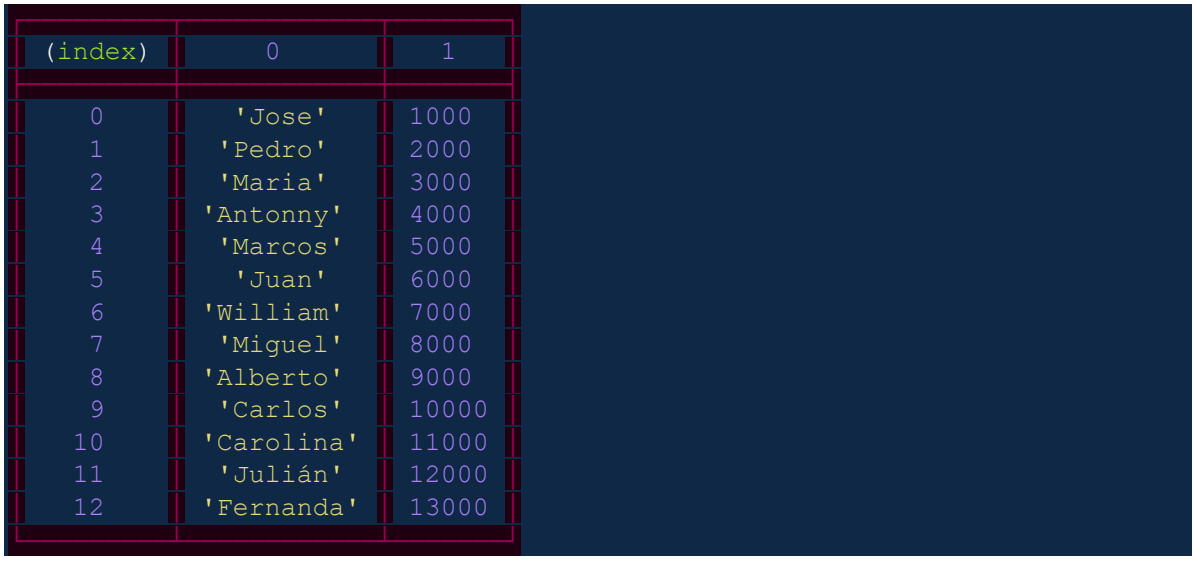
Seguidamente, creamos la función **insertsCuentas()**, en la que dentro del query hacemos uso del módulo pg-format, con la palabra reservada **format,** que emplearemos para insertar los objetos del Array como un literal **%L.** La función de inserción queda así:

```
const result = await pool.query(format("INSERT INTO cuentas (nombre,
balance) VALUES %L", registros)
```

Al ejecutar el script, observamos que se insertan los datos dinámicamente en la tabla Cuentas de la base de datos.

```
$ node insertsCuentasArray.js
```

Se ejecutó el comando: INSERT, con 13 inserciones.



(index)	0	1
0	'Jose'	1000
1	'Pedro'	2000
2	'Maria'	3000
3	'Antonny'	4000
4	'Marcos'	5000
5	'Juan'	6000
6	'William'	7000
7	'Miguel'	8000
8	'Alberto'	9000
9	'Carlos'	10000
10	'Carolina'	11000
11	'Julián'	12000
12	'Fernanda'	13000

EXERCISE 3: REALIZAR TRANSACCIONES CON BEGIN, COMMIT Y ROLLBACK CON TRY CATCH

Para desarrollar el ejercicio de transacciones entre dos cuentas del clienteA con un id, al clienteB con otro id, procedemos a realizar un script al que le pasaremos los parámetros por consola para actualizar y llevar a cabo la transacción. Esto es:

```
$ node trasferAccount.js 1 2 1000
```

En este sentido, vamos a trasferir desde el cliente con id **1**, al cliente con id **2**, un monto de **1.000**; es decir, a partir del tercer parámetro introducimos el id del cliente 1, id cliente 2, y el **monto**:

Crearemos el script **trasferAccount.js** con la siguiente codificación de la función de trasferencia entre dos clientes:

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

// Función de trasferencia
async function transferAccount() {
    const [idClienteA, idClienteB, mont] = process.argv.slice(2);
```

```
    await pool.connect();

    try {
        // inicio de la trasacción
        await pool.query('BEGIN');
        try {
            const {
                rows
            } = await pool.query('SELECT "balance" FROM "cuentas"
WHERE "id" = $1',
                [idClienteA]);
            const balance = rows[0].balance;

            // Verificamos si la cuenta posee balance, caso contrario
no realizamos                           actualizaciones
            if (balance >= mont) {
                await pool.query('UPDATE "cuentas" SET "balance" =
"balance" - $1
                                        WHERE "id" = $2', [mont,
idClienteA]);

                await pool.query('UPDATE "cuentas" SET "balance" =
"balance" + $1
                                        WHERE "id" = $2', [mont,
idClienteB]);

                console.log("trasferencia realizada del ciente con id:
" +
                                        idClienteA + " al cliente con
id: " + idClienteB +
                                        " por un monto de " + mont)
            }
            // Enviamos que la transacción se ha completado
            await pool.query('COMMIT');
        } catch (error) {
            // Si existe un error, revertir todos los cambios has el
punto
                                    de inicio de la trasacción
            await pool.query('ROLLBACK');
            console.error(error.stack); // muestra los errores por
consola
        }
    } finally {
        pool.end(); // Finaliza la conexión
    }
};

// llamamos a la función haciendo uso de: $ node trasferAccount.js 1 2
```

```
1000
transferAccount();
```

Creación de la función para consultar dos clientes según el id; para esto creamos el script getUser.js, con el siguiente código:

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

async function getUsers() {
    const [idClienteA, idClienteB] = process.argv.slice(2);
    try {
        const res = await pool.query('SELECT * FROM "cuentas" WHERE
                                    "id" = $1 OR "id" = $2',[idClienteA,
idClienteB])
        console.table(res.rows);
    } catch (error) {
        console.error(error);
    }
}
getUsers()
```

Ejecutando las pruebas, tenemos lo siguiente:

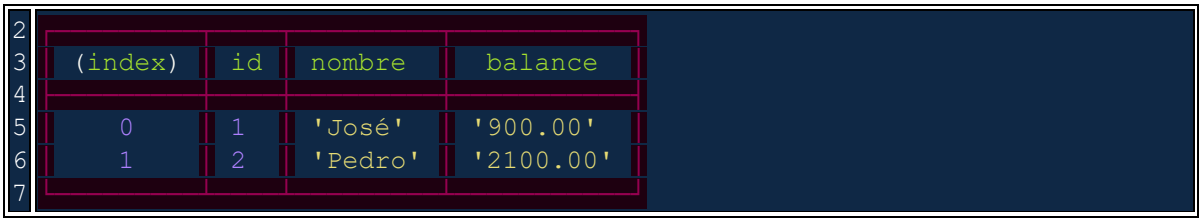
```
$ node getUsers.js 1 2

┌─────────┬────┬──────────┬──────────┐
│ (index) │ id │ nombre   │ balance  │
├─────────┼────┼──────────┼──────────┤
│    0    │ 1  │ 'José'   │ '1000.00'│
│    1    │ 2  │ 'Pedro'  │ '2000.00'│
└─────────┴────┴──────────┴──────────┘
```

Trasferimos 100 del usuario con id 1 (José), al id 2 (Pedro):

```
$ node trasferAccount.js 1 2 100
```

Trasferencia realizada del cliente con id 1, al cliente con id 2, por un monto de 100:

```
$ node getUsers.js 1 2
```



(index)	id	nombre	balance
0	1	'José'	'900.00'
1	2	'Pedro'	'2100.00'

Observamos que se transfirió satisfactoriamente, y el balance mostrado es el correcto.