Guía de ejercicios - Trabajo práctico (II)

¡Hola! Te damos la bienvenida a esta nueva guía de estudio.

¿En qué consiste esta guía?

La siguiente guía de estudio tiene como objetivo practicar y ejercitar los contenidos que hemos visto en clase.

¡Vamos con todo!

Tabla de contenidos

¡Comencemos!

Actividad guiada: Elecciones presidenciales

Para implementar este ejercicio, primero, crearemos un archivo llamado `queries.js` para gestionar las funciones asíncronas y la lógica de la base de datos utilizando el paquete pg. Luego, implementaremos el servidor Express en otro archivo llamado `server.js`. Aquí el paso a paso:

1. Crea una base de datos llamada candidatos y las tablas necesarias. Puedes usar la siguiente estructura SQL:

```
CREATE TABLE candidatos (
  id SERIAL PRIMARY KEY,
  nombre VARCHAR(255) NOT NULL,
  foto VARCHAR(255) NOT NULL,
  color VARCHAR(20) NOT NULL
);

CREATE TABLE historial (
  id SERIAL PRIMARY KEY,
  estado VARCHAR(255) NOT NULL,
  cantidad_votos INT NOT NULL,
  candidato_ganador VARCHAR(255) NOT NULL
);
```

2. Crea un archivo llamado queries.js en el directorio del proyecto.

```
// queries.js
const { Pool } = require("pg");

const pool = new Pool({
  user: "tu_usuario",
  host: "localhost",
  password: "tu_contraseña",
  database: "candidatos",
  port: 5432,
});

const agregarCandidato = async (nombre, foto, color) => {
  // ... (ver código anterior)
};
```

```
// Definir otras funciones para obtener, eliminar y actualizar
candidatos, así como funciones para el historial.

module.exports = {
  agregarCandidato,
  // Exportar otras funciones
};
```

3. Crea un archivo llamado server.js en el directorio del proyecto.

```
// server.js
const express = require('express');
const bodyParser = require('body-parser');
const { agregarCandidato, obtenerCandidatos, eliminarCandidato,
actualizarCandidato, agregarVotosHistorial, obtenerHistorial } =
require('./queries');

const app = express();
const PORT = 3000;

app.use(bodyParser.json());

// ... (ver código anterior para definir rutas)

app.listen(PORT, () => {
  console.log(`Servidor corriendo en el puerto ${PORT}`);
});
```

4. Abre el archivo server.js y define las rutas según las especificaciones.

```
// server.js
// ... (código anterior)

app.post('/candidato', async (req, res) => {
  // Ruta para agregar un candidato
  // ...
});

app.get('/candidatos', async (req, res) => {
  // Ruta para obtener todos los candidatos
  // ...
});
```

```
app.delete('/candidato', async (req, res) => {
  // Ruta para eliminar un candidato por ID
  // ...
});

app.put('/candidato', async (req, res) => {
  // Ruta para actualizar un candidato por ID
  // ...
});

app.post('/votos', async (req, res) => {
  // Ruta para agregar votos y actualizar historial
  // ...
});

app.get('/historial', async (req, res) => {
  // Ruta para obtener historial de votos
  // ...
});

// ... (código posterior)
```

5. Ejecuta el servidor con el siguiente comando:

```
node server.js
```

6. Usa herramientas como Postman o CURL para probar las rutas que has creado.

- POST /candidato: Agrega un nuevo candidato.
- GET /candidatos: Obtiene la lista de todos los candidatos.
- DELETE /candidato?id=1: Elimina el candidato con el ID especificado.
- PUT /candidato: Actualiza los datos de un candidato.
- POST /votos: Agrega votos y actualiza el historial.
- GET /historial: Obtiene el historial de votos.

¡Listo! Ahora has creado una aplicación Node.js con Express y PostgreSQL que maneja candidatos, votos y un historial de votos. Recuerda ajustar las credenciales de tu base de datos en queries.js

¡Manos a la obra! - 1

Basado en el ejercicio de las transacciones del banco "Dinero Azul", desarrolla una aplicación en Node que realice consultas SQL para ejecutar una transacción que transfiera 30000 de saldo del usuario con correo **bette_nicka@cox.net** al usuario con correo **vinouye@aol.com**.

¡Manos a la obra! - 2

Basado en el ejercicio de las transacciones del banco "Dinero Azul", desarrollar una aplicación en Node que realice consultas SQL para ejecutar una transacción que transfiera $10.000 de saldo del usuario con correo **albina@glick.com** al usuario con correo **shawna_palaspas@palaspas.org**.

Imprimir por consola el resultado de ambas consultas SQL para demostrar que los saldos fueron modificados.

¡Manos a la obra! - 3

Basado en el ejercicio de las transacciones del banco "Dinero Azul", desarrolla una aplicación en Node que realice consultas SQL para ejecutar una transacción que transfiera $50.000 de saldo del usuario con correo **willard@hotmail.com** al usuario con correo **mroyster@royster.com**.

Imprimir por consola el código, detalle, tabla originaria y el nombre del campo que tiene la restricción que impidió que culminara la transacción.

¡Manos a la obra! - 4

Desarrollar un servidor con Express que disponibilice una ruta GET **/fecha** que devuelva un string al cliente con una fecha enviada desde PostgreSQL.

¡Manos a la obra! - 5

Desarrollar una función asíncrona que realice una consulta SQL para insertar un registro con los datos recibidos como parámetros y retorna el último registro insertado en forma de arreglo.

¡Manos a la obra! - 6

Desarrollar una ruta POST que reciba el payload del cliente y ejecute la función “insertar”, pasándole como argumento la data recibida del cliente y devuelva un JSON con el resultado de la consulta SQL. Además debe especificar un código de estado HTTP de 201

¡Manos a la obra! - 7

Desarrollar una función asíncrona que realice una consulta SQL para obtener todos los registros de la tabla **ejercicios** que tengan en el campo de “repeticiones” un número igual a 20.

¡Manos a la obra! - 8

Desarrollar una ruta GET que devuelva un JSON con los registros de la tabla **ejercicios** usando la función “consultar”. Además debe especificar un código de estado HTTP de 200.

¡Manos a la obra! - 9

Desarrollar una función asíncrona que realice una consulta SQL sin parámetros para actualizar algún registro de la tabla **ejercicios**.

¡Manos a la obra! - 10

Desarrollar una ruta PUT que reciba el payload enviado por la aplicación cliente y utilice la función “editar”, para emitir una consulta SQL que actualice la información de un registro y devuelva un mensaje que diga “Recurso editado con éxito”. Además, debe especificar un código de estado HTTP de 201.

¡Manos a la obra! - 11

Desarrollar una función asíncrona que realice una consulta SQL para eliminar un registro de la tabla **ejercicios** e imprime por consola la cantidad de filas eliminadas.

¡Manos a la obra! - 12

Desarrollar una ruta DELETE que reciba un parámetro “nombre” con las query strings y ejecute la función “eliminar” para eliminar un registro de la tabla **ejercicios**, además de devolver un mensaje diciendo “Registro eliminado con éxito!” y un status 200.

Preguntas de proceso

Reflexiona:

- ¿Cuáles son los conceptos clave que he comprendido hasta ahora?
- ¿Cómo se relaciona lo que estoy aprendiendo con mis experiencias previas?
- ¿Puedo explicar lo que he aprendido a otra persona de manera clara?

¡Continúa aprendiendo y practicando!

Solucionario

1. Basado en el ejercicio de las transacciones del banco “Dinero Azul”, desarrollar una aplicación en Node que realice consultas SQL para ejecutar una transacción que transfiera $30000 de saldo del usuario con correo bette_nicka@cox.net al usuario con correo vinouye@aol.com.

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

const pool = new Pool({
    user: "postgres",
    host: "localhost",
    password: "postgres",
    database: "clientes",
    port: 5432,
});

(async () => {

    await pool.query("BEGIN");
    const descontar =
        "UPDATE usuarios SET saldo = saldo - 30000 WHERE email =
'bette_nicka@cox.net' ";
    await pool.query(descontar);
    const acreditar =
        "UPDATE usuarios SET saldo = saldo + 30000 WHERE email =
'vinouye@aol.com' ";
    await pool.query(acreditar);
    await pool.query("COMMIT");
    pool.end()

})()
```

2. Basado en el ejercicio de las transacciones del banco “Dinero Azul”, desarrollar una aplicación en Node que realice consultas SQL para ejecutar una transacción que transfiera $10000 de saldo del usuario con correo albina@glick.com al usuario con correo shawna_palaspas@palaspas.org.

   Imprimir por consola el resultado de ambas consultas SQL para demostrar que los saldos fueron modificados.

```
(async () => {

    await pool.query("BEGIN");

    const descontar =
        "UPDATE usuarios SET saldo = saldo - 20000 WHERE email =
'albina@glick.com' RETURNING *";
    const descuento = await pool.query(descontar);

    const acreditar =
        "UPDATE usuarios SET saldo = saldo + 20000 WHERE email =
'shawna_palaspas@palaspas.org' RETURNING *";
    const acreditacion = await pool.query(acreditar);

    console.log("Descuento realizado con éxito: ", descuento.rows[0]);
    console.log("Acreditación realizada con éxito: ", acreditacion.rows[0]);

    await pool.query("COMMIT");
    pool.end()

})()
```

3. Basado en el ejercicio de las transacciones del banco “Dinero Azul”, desarrollar una aplicación en Node que realice consultas SQL para ejecutar una transacción que transfiera $50.000 de saldo del usuario con correo willard@hotmail.com al usuario con correo mroyster@royster.com.

   Imprimir por consola el código, detalle, tabla originaria y el nombre del campo que tiene la restricción que impidió que culminara la transacción

```
(async () => {

    try {
        await pool.query("BEGIN");
        const descontar =
            "UPDATE usuarios SET saldo = saldo - 50000 WHERE email =
'willard@hotmail.com' ";
        await pool.query(descontar);

        const acreditar =
            "UPDATE usuarios SET saldo = saldo + 50000 WHERE email =
'mroyster@royster.com' ";
        await pool.query(acreditar);

        await pool.query("COMMIT");
    } catch (e) {
        await pool.query("ROLLBACK");
        console.log("Error código: " + e.code);
        console.log("Detalle del error: " + e.detail);
        console.log("Tabla originaria del error: " + e.table);
        console.log("Restricción violada en el campo: " + e.constraint);
    } finally {
        pool.end()
    }

})()
```

4. Desarrollar un servidor con Express que disponibilice una ruta GET **/fecha** que devuelva un string al cliente con una fecha enviada desde PostgreSQL.

```
// consultas.js
const { Pool } = require("pg");

const pool = new Pool({
  user: "postgres",
  host: "localhost",
  password: "postgres",
  port: 5432,
});

const getDate = async () => {
  const result = await pool.query("SELECT NOW()");
  return result.rows[0].now;
};

module.exports = { getDate };
```

```
// index.js
const express = require('express')
const app = express()
app.listen(3000)

const { getDate } = require('./consultas.js')

app.get("/", async (req, res) => {
    const result = await getDate();
    res.send(result);
})
```

5. Desarrollar una función asíncrona que realice una consulta SQL para insertar un registro con los datos recibidos como parámetros y retorna el último registro insertado en forma de arreglo.

```
const insertar = async (datos) => {
  const consulta = {
    text: "INSERT INTO ejercicios values($1, $2, $3, $4) RETURNING *",
    values: datos,
    rowMode: "array"
  };
  try {
    const result = await pool.query(consulta);
    console.log(result.rows[0])
    return result;
  } catch (error) {
    console.log(error.code);
    throw error;
  }
};
```

6. Desarrollar una ruta POST que reciba el payload del cliente y ejecute la función “insertar” pasándole como argumento la data recibida del cliente y devuelve un JSON con el resultado de la consulta SQL. Además debe especificar un código de estado HTTP de 201.

```
app.post("/ejercicios", async (req, res) => {
    const datos = Object.values(req.body)
    const respuesta = await insertar(datos);
    res.status(201).send(respuesta);
})
```

7. Desarrollar una función asíncrona que realice una consulta SQL para obtener todos los registros de la tabla ejercicios que tengan en el campo de “repeticiones” un número igual a 20.

```
const consultar = async () => {
    const result = await pool.query(
      "SELECT * FROM ejercicios WHERE repeticiones = '20';"
    );
    return result;
};
```

8. Desarrollar una ruta GET que devuelva un JSON con los registros de la tabla ejercicios usando la función “consultar”. Además debe especificar un código de estado HTTP de 200.

```
app.get("/ejercicios", async (req, res) => {
    const registros = await consultar();
    res.status(200).send(registros);
})
```

9. Desarrollar una función asíncrona que realice una consulta SQL sin parámetros para actualizar algún registro de la tabla ejercicios.

```
const editar = async (datos) => {
    const result = await pool.query(`UPDATE ejercicios SET
      nombre = '${datos[0]}',
      series = '${datos[1]}',
      repeticiones = '${datos[2]}',
      descanso = '${datos[3]}'
     WHERE nombre = '${datos[0]}' RETURNING *`);
    return result;
};
```

10. Desarrollar una ruta PUT que reciba el payload enviado por la aplicación cliente y utilice la función “editar”, para emitir una consulta SQL que actualice la información de un registro y devuelva un mensaje que diga “Recurso editado con éxito”. Además, debe especificar un código de estado HTTP de 201.

```
app.put("/ejercicios", async (req, res) => {
    try {
        const datos = Object.values(req.body)
        const respuesta = await editar(datos);
        res.status(201).send(respuesta);
    } catch (error) {
        res.status(500).send(error)
    }
})
```

11. Desarrollar una función asíncrona que realice una consulta SQL para eliminar un registro de la tabla ejercicios y retorne la cantidad de filas eliminadas.

```
const eliminar = async (nombre) => {
    const result = await pool.query(
      `DELETE FROM ejercicios WHERE nombre = '${nombre}' RETURNING *`
    );
    console.log(result.rowCount)
    return result.rowCount;
};
```

12. Desarrollar una ruta DELETE que reciba un parámetro “nombre” con las query strings y ejecute la función “eliminar” para eliminar un registro de la tabla ejercicios, además de devolver un mensaje diciendo “Registro eliminado con éxito!” y un status 200.

```
app.delete("/ejercicios", async (req, res) => {
    try {
        const { nombre } = req.query;
        const respuesta = await eliminar(nombre);
        res.status(200).send(respuesta);
    } catch (error) {
        res.status(500).send(error)
    }
})
```

Preguntas de cierre

- ¿Por qué es importante utilizar transacciones al realizar operaciones que involucran múltiples consultas SQL?
- ¿Cuáles son los posibles desafíos al realizar transacciones y cómo se manejan en el código que escribiste?
- ¿Qué otras rutas podrían agregarse para interactuar con la base de datos de manera efectiva?
- ¿Cómo se utiliza el código de estado HTTP para comunicar el resultado de la operación al cliente?
- ¿Cómo se capturan los parámetros enviados a través de las query strings en Express?