Pasos para trabajar con Postgresql

Para trabajar con postgresql en node debemos usar la librería pg.

```
npm install pg
```

Adicionalmente podemos usar la librería `dotenv` para manejar variables de entorno en archivos `.env`

```
DB_HOST=localhost
DB_USER=postgres
DB_PASSWORD=123456
DB_NAME=dev_distribu
DB_PORT=5432
```

En este laboratorio usaremos la siguiente tabla `users`:

```
CREATE TABLE users(
  id serial NOT NULL PRIMARY KEY,
  nombres text,
  paterno text,
  nombreusuario text,
  clave text,
  id_estado VARCHAR(1) DEFAULT 'A',
  fec_r TIMESTAMP DEFAULT now(),
  fec_m TIMESTAMP,
  fec_e TIMESTAMP,
  ult_usuarioa INTEGER,
  ult_usuariom INTEGER,
  ult_usuarioe INTEGER
);
```

Conexión Node PostgreSQL

Para conectar a postgres debemos organizar hacerlo de la siguiente forma:

app.js

```
const express = require('express');
const { Pool } = require('pg');

require('dotenv').config();

const app = express();
app.use(express.json()); // Middleware para parsear JSON

const pool = new Pool({
  user: process.env.DB_USER,
```

```
  host: process.env.DB_HOST,
  database: process.env.DB_NAME,
  password: process.env.DB_PASSWORD,
  port: process.env.DB_PORT,
});

pool.query('SELECT NOW()', (err, res) => {
  if (err) {
    console.error('Error al conectar la base de datos', err);
  } else {
    console.log('Conexion establecida:', res.rows[0]);
  }
  pool.end();
});
```

Para evitar la creación de un pool de conexiones a la base de datos en cada solicitud. Se debe crear el pool de conexiones una sola vez y reutilizar ésta en todas las solicitudes. Una forma de hacerlo es crear un módulo de conexión `db.js`

db.js

```
const { Pool } = require('pg');
const pool = new Pool({
    user: 'postgres',
    host: 'localhost',
    database: 'dev_tecnologias',
    password: '123456',
    port: '5432',
  });
module.exports = pool;
```

A continuación el pool se inicia debe iniciar de la siguiente forma: en `app.js`

```
const pool = require('./db');
```

Consulta de tipo Select

Una consulta de tipo select se lo realiza de la siguiente forma:

```
app.get('/api/users', async (req, res) => {
  try {
    const result = await pool.query('SELECT * FROM users');
    res.json(result.rows);
  } catch (err) {
    console.error(err);
```

```
    res.status(500).send('Server Error');
  }
});
```

Obtener un solo usuario:

```
app.get('/api/users/:id', async (req, res) => {
  const { id } = req.params;
  try {
    const result = await pool.query('SELECT * FROM users WHERE id = $1', [id]);
    if (result.rows.length === 0) {
      return res.status(404).send('User not found');
    }
    res.json(result.rows[0]);
  } catch (err) {
    console.error(err);
    res.status(500).send('Server Error');
  }
});
```

Insertar (insert)

Para realizar un insert a un usuario

```
app.post('/api/users', async (req, res) => {
  const { name, email } = req.body;
  try {
    const result = await pool.query(
      'INSERT INTO users (name, email) VALUES ($1, $2) RETURNING *',
      [name, email]
    );
    res.status(201).json(result.rows[0]);
  } catch (err) {
    console.error(err);
    res.status(500).send('Server Error');
  }
});
```

Actualizar (UPDATE)

```
app.put('/api/users/:id', async (req, res) => {
  const { id } = req.params;
  const { name, email } = req.body;
  try {
    const result = await pool.query(
      'UPDATE users SET name = $1, email = $2 WHERE id = $3 RETURNING *',
      [name, email, id]
```

```
    );
    if (result.rows.length === 0) {
      return res.status(404).send('User not found');
    }
    res.json(result.rows[0]);
  } catch (err) {
    console.error(err);
    res.status(500).send('Server Error');
  }
});
```

Actualizar usando transacciones:

```
app.put('/api/updateMultiple', async (req, res) => {
  const { userId, userName, userEmail, profileId, profileBio } = req.body;

  const client = await pool.connect();

  try {
    await client.query('BEGIN');

    const updateUserQuery = 'UPDATE users SET name = $1, email = $2 WHERE id = $3
RETURNING *';
    const updateUserValues = [userName, userEmail, userId];
    const updateUserResult = await client.query(updateUserQuery, updateUserValues);

    const updateProfileQuery = 'UPDATE profiles SET bio = $1 WHERE user_id = $2
RETURNING *';
    const updateProfileValues = [profileBio, userId];
    const updateProfileResult = await client.query(updateProfileQuery,
updateProfileValues);

    await client.query('COMMIT');

    res.json({
      user: updateUserResult.rows[0],
      profile: updateProfileResult.rows[0],
    });
  } catch (err) {
    await client.query('ROLLBACK');
    console.error('Transaction error:', err);
    res.status(500).send('Server Error');
  } finally {
    client.release();
  }
});
```

Eliminar

Para eliminar un registro o varios se lo realiza de la siguiente forma:

```
app.delete('/api/users/:id', async (req, res) => {
  const { id } = req.params;
  try {
    const result = await pool.query('DELETE FROM users WHERE id = $1 RETURNING *',
[id]);
    if (result.rows.length === 0) {
      return res.status(404).send('User not found');
    }
    res.status(204).send();
  } catch (err) {
    console.error(err);
    res.status(500).send('Server Error');
  }
});
```

URL del repositorio: https://github.com/F1852D160/08-nodepgsql.git

From:
http://wiki.local/ - **Wiki.Local**

Permanent link:
http://wiki.local/doku.php?id=materias:tecnologias-emergentes:unidad_3:01_pgsql_node

Last update: **2024/06/04 23:12**