

{desafío}
latam_

Implementación y gestión de una base de datos _

Parte I



Instalación y configuración de la librería pg

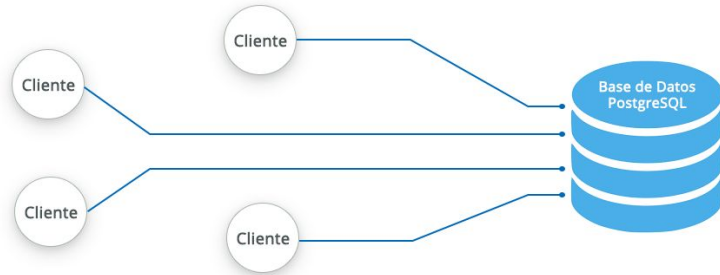
- Reconocer las características de la librería pg en el entorno Node.
- Aplicar el procedimiento de instalación, configuración y conexión a PostgreSQL con el paquete pg utilizando NPM.

Competencias

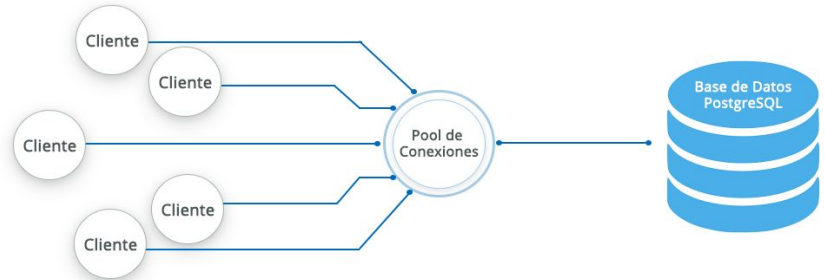
Librería pg para Node

- La librería pg es utilizada en Node y contiene una colección de métodos, que permiten desarrollar aplicaciones que utilicen las principales características incluidas en la base de datos relacional PostgreSQL.
- Los principales métodos que tiene esta librería son Client y Pool.
- En el ecosistema de Node, la librería pg cuenta con una integración más nativa y mucho más completa, aprovechando de mejor forma las funcionalidades que nos entrega la base de datos, en comparación con otras librerías como:
 - postgres.js, knex.js
 - connect-pg-simple

Método Client



Método Pool



** En esta parte de la unidad estaremos realizando conexiones individuales o de un solo cliente.

Características de la librería pg

- Permite realizar conexiones nativas con la base de datos PostgreSQL.
- Permite la conexión de un cliente para realizar consultas estáticas.
- Permite agrupar conexiones de distintos clientes, configurando la cantidad aceptada y tiempos de inactividad antes de ser desconectados.
- Acepta SSL en las conexiones.
- Permite el control y captura de errores.
- Acepta eventos que puedan monitorear el estado de la base de datos.
- Cuenta con soporte para las características principales de Node, cómo: callbacks, promises, async/await, cursors entre otras.

Instalación, configuración y conexión

Para la instalación del paquete pg, se debe utilizar el siguiente comando por la terminal luego de haber iniciado un proyecto con NPM.

```
npm install pg
```

Luego se debe importar el módulo dentro de la misma dependencia llamado Client, el cual es una clase que recibirá un objeto de configuración y será el gestor de nuestra base de datos.

```
const { Client } = require('pg');
```

Configuración

- La **clase Client** recibe en su constructor un objeto de configuración que se puede realizar a través de:
 - String de conexión que contenga todos los datos necesarios
 - Un objeto con las propiedades correspondientes a la conexión.

String de conexión

Un string de conexión es básicamente una cadena de texto, que agrupa los parámetros con la información de las credenciales.

El formato para conectarse a la base de datos PostgreSQL, es el siguiente:

```
'postgresql://dbuser:dbpassword@dbserver:dbport/database'
```

Detalle de las propiedades utilizadas:

- **dbuser:** Nombre de usuario configurado en la base de datos.
- **dbpassword:** Contraseña asignada al usuario.
- **dbserver:** Nombre o IP del servidor PostgreSQL.
- **dbport:** Puerto del servidor donde se ejecuta PostgreSQL, por defecto se utiliza el puerto 5432.
- **database:** Corresponde al nombre de la base de datos.

Creación de la base de datos

La temática a abordar será la gestión de una base de datos para la tienda de ropa B&H. Crear una base de datos llamada **JEANS**:

```
CREATE DATABASE JEANS
```

Con la base de datos creada, ahora se debe cambiar los valores de la configuración, los cuales serían los siguientes:

- Usuario: postgres
- Password: postgres
- Server: localhost
- Port: 5432
- Database: jeans

Creación de la base de datos

Código con la configuración pasada al constructor de la clase Client:

```
const { Client } = require('pg');  
  
const client = new Client({  
  connectionString:  
    'postgresql://postgres:postgres@localhost:5432/jeans'  
});
```

Construir un string de conexión con los siguientes datos:

- Usuario: usuario
- Password: 123456
- Server: 127.0.0.1
- Port: 5432
- Database: usuarios

Ejercicio propuesto (1)

Conexión por propiedades

Otra forma de definir la configuración de nuestra conexión a PostgreSQL es a través de un objeto, compuesto por diferentes propiedades que representarán cada uno de los parámetros que definimos anteriormente en la URI.

```
const config = {  
  user: 'postgres',  
  host: 'localhost',  
  database: 'jeans',  
  password: 'postgres',  
  port: 5432,  
}
```

Conexión por propiedades

Código con la configuración pasada al constructor de la clase Client:

```
const { Client } = require('pg');

const config = {
  user: 'postgres',
  host: 'localhost',
  database: 'jeans',
  password: 'postgres',
  port: 5432,
}

const client = new Client(config );
```

Construir un objeto de configuración con los siguientes datos:

- Usuario: usuario
- Password: 123456
- Server: 127.0.0.1
- Port: 5432
- Database: usuarios

Ejercicio propuesto (2)

Probando conexión

Para probar una conexión a la base de datos, se debe llamar al método “connect” del objeto “client”, una vez conectado ejecutar una consulta SQL que devuelva la fecha actual.

```
client.connect()

client.query('SELECT NOW()', (err, res) => {
  console.log(res)
  client.end()
})
```

```
$ node index.js
Result {
  command: 'SELECT',
  rowCount: 1,
  oid: null,
  rows: [ { now: 2020-10-17T00:57:14.622Z } ],
  fields: [
    Field {
      name: 'now',
      tableID: 0,
      columnID: 0,
      dataTypeID: 1184,
      dataTypeSize: 8,
      dataTypeModifier: -1,
      format: 'text'
    }
  ],
  _parsers: [ [Function: parseDate] ],
  _types: TypeOverrides {
    _types: {
      getTypeParser: [Function: getTypeParser],
      setTypeParser: [Function: setTypeParser],
      arrayParser: [Object],
      builtins: [Object]
    },
    text: {},
    binary: {}
  },
  RowCtor: null,
  rowAsArray: false
}
```


Propiedades del objeto result y la asincronía en una consulta

- Reconocer cómo realizar una consulta asíncrona con texto plano a PostgreSQL desde una aplicación Node.
- Reconocer las propiedades del objeto result para la obtención de los registros a una consulta realizada.

Consideraciones previas

- Crear una tabla dentro de la base de datos cuyas características continuarán con la temática de la tienda de ropa B&H.
- Crear la tabla ropa con las columnas: id con la instrucción SERIAL para que sea un entero autoincremental, nombre, color y talla.

```
CREATE TABLE ropa (  
  id SERIAL PRIMARY KEY,  
  nombre varchar(50) NOT NULL,  
  color varchar(10) NOT NULL,  
  talla varchar(5) NOT NULL  
);
```

Consultas con callbacks

- Las consultas con pg por naturaleza son asíncronas.
- La función callback corresponde al segundo parámetro del método query del paquete pg.

```
client.query('<consulta>', (error, respuesta) => {  
  console.log(respuesta)  
  client.end()  
})
```

Consultas con ASYNC/AWAIT

Existe la posibilidad de trabajar con la asincronía con las funciones async/await, y de esta manera evitar en la medida que se pueda el uso de callbacks, para no caer en el infierno de callbacks anidados.

```
async function consulta() {  
  const res = await client.query("<consulta>")  
  console.log(res)  
  client.end()  
}  
  
consulta()
```

Queries con texto plano

PostgreSQL nos ofrece la posibilidad de realizar consultas a la base de datos en 2 formatos particulares que son:

- **Texto plano**

```
"SELECT * FROM ropa WHERE id=25"
```

- **Texto plano con parámetros:**

```
"SELECT * FROM ropa WHERE id=$1"
```

El objeto result

Dentro del objeto result tenemos varias propiedades, pero las 2 más destacables son las propiedad “rows” y “fields”, representan en formato de arreglo las filas y los campos que fueron agregados en la tabla a la que le estemos realizando la consulta.

Importante: A excepción de la instrucción SELECT para obtener los registros que fueron afectados por la consulta, se debe concatenar al final de la consulta lo siguiente: **RETURNING ***

```
client.query("<consulta> RETURNING *")
```


Ingresando y consultando datos

- Programar instrucciones SQL para ingresar y consultar datos de PostgreSQL utilizando el entorno Node.

INSERT

- La librería pg a través de su función query nos permite utilizar la sentencia SQL INSERT para crear registros en una tabla de la base de datos PostgreSQL.
- Para poder crear un nuevo registro en una aplicación Node, se debe contar con un script con una conexión activa.
- La librería pg al ejecutar una sentencia SQL INSERT, entrega como respuesta de la base de datos un objeto result, el cual tiene información de la consulta realizada.

Crear una aplicación que realice una inserción a la tabla **ropa**, ingresando un pantalón de color azul y de talla 46, a su vez obtenga a través del “RETURNING *” el objeto result, el cual usaremos para imprimir por consola el registro agregado.


A vertical line separates the white left side from the orange right side. It features a series of white symbols: a closing curly brace '}', an '@' symbol, an opening curly brace '{', and a closing parenthesis ')'.

**Ejercicio
guiado:
Obteniendo mi
inserción
(RETURNING *)**

Respuesta de una consulta para ingresar un pantalón a la tabla ropa

```
$ node index.js
Result {
  command: 'INSERT',
  rowCount: 1,
  oid: 0,
  rows: [ { id: 1, nombre: 'pantalon', talla: '46', color: 'azul' } ],
  fields: [
    Field {
      name: 'id',
      tableID: 16415,
      columnID: 1,
      dataTypeID: 23,
      dataTypeSize: 4,
      dataTypeModifier: -1,
      format: 'text'
    },
    Field {
      name: 'nombre',
      tableID: 16415,
      columnID: 2,
      dataTypeID: 1043,
      dataTypeSize: -1,
      dataTypeModifier: 54,
      format: 'text'
    },
    Field {
      name: 'talla',
      tableID: 16415,
      columnID: 3,
      dataTypeID: 1043,
      dataTypeSize: -1,
      dataTypeModifier: 9,
      format: 'text'
    },
    Field {
      name: 'color',
      tableID: 16415,
      columnID: 4,
      dataTypeID: 1043,
      dataTypeSize: -1,
      dataTypeModifier: 14,
      format: 'text'
    }
  ]
}
```

Modificar la función ingresar() y delimitar la información obtenida en una consulta, en este caso será una inserción de una polera color blanca de talla M.

A vertical line separates the white left side from the orange right side. It features a series of white symbols: a closing curly brace '}', an '@' symbol, an opening curly brace '{', and a stylized person icon, all arranged vertically.

Ejercicio guiado: Ordenando el detalle por consola (INSERT)

Respuesta delimitada de una consulta

```
$ node index.js  
Registro agregado { id: 2, nombre: 'polera', talla: 'M', color: 'blanca' }  
Último id 2  
Cantidad de registros afectados 1  
Campos del registro: id - nombre - talla - color
```

Ejercicio propuesto (3)

Abrir la terminal psql y crear una tabla usuarios con el siguiente código:

```
CREATE TABLE usuarios(  
  id SERIAL PRIMARY KEY,  
  nombre varchar(50) NOT NULL,  
  telefono varchar(10) NOT NULL  
);
```

Desarrollar una aplicación con Node que al ser ejecutada llame a una función asíncrona para ejecutar una consulta a PostgreSQL que ingrese 1 registro en la tabla usuarios.

Crear una función que realice una consulta SQL para obtener todos los registros actuales en la tabla **ropa**.

Solución

```
$ node index.js
Registros: [
  { id: 1, nombre: 'pantalon', talla: '46', color: 'azul' },
  { id: 2, nombre: 'polera', talla: 'M', color: 'blanca' }
]
```

Ejercicio guiado: Revisando inventario (SELECT)

Ejercicio propuesto (4)

Continuando con el ejercicio propuesto 3 donde se creó una tabla de usuarios con los campos: id, nombre y teléfono. Desarrollar una aplicación con Node que al ser ejecutada realice en secuencia lo siguiente:

- Ingrese otro registro en la tabla usuarios.
- Consulte todos los registros de la tabla usuarios.
- Consulte solo el registro de id 1.

Considerar que la desconexión a la base de datos debe realizarse al finalizar todas las consultas, por lo que se debe manejar las funciones asíncronas como promesas ejecutando una tras otra con el método `then()`.

Actualizando y eliminado datos

- Programar instrucciones SQL para actualizar y eliminar datos de PostgreSQL utilizando el entorno Node.

UPDATE

- La actualización de registros es clave para un control completo de datos que queremos persistir en nuestras aplicaciones.
- El paquete pg por medio de su método query(), permite enviar cualquier sentencia SQL a PostgreSQL, por lo que una actualización no es diferente a la inserción y selección de datos vistos en el capítulo anterior.

Continuando con la temática de la tienda de ropa B&H, agregar una función a nuestra aplicación que al ser ejecutada modifique la talla de la polera agregada anteriormente por una talla L, suponiendo un hipotético caso en el que la tienda B&H ingresó erróneamente esta talla al momento de cargar la nueva mercancía.

Solución

```
$ node index.js  
Registro modificado { id: 2, nombre: 'polera', talla: 'L', color: 'blanca' }  
Cantidad de registros afectados 1
```

Ejercicio guiado: Corrigiendo datos (UPDATE)

Continuando con el ejercicio propuesto 4, en el que empezamos a realizar consultas a la tabla **usuarios**. Ahora se solicita desarrollar una aplicación con Node que al ser ejecutada realice una actualización a la tabla **usuarios** cambiando el número de teléfono al primer registro por: 914215468. Obten por consola el registro modificado y la cantidad de registros afectados.

Ejercicio propuesto (5)

Crear una función que elimine todos los registros de la tabla **ropa** y devuelva por consola la cantidad total de filas eliminadas.

Solución

```
$ node index.js  
Cantidad de registros afectados 2
```

Ejercicio
guiado:
Eliminando
registros
(DELETE)

Desarrollar una aplicación con Node que al ser ejecutada realice una consulta SQL para eliminar todos los registros de la tabla usuarios e imprime por consola la cantidad de registros afectados.

Ejercicio propuesto (6)

{desafío}
latam_

*Academia de
talentos digitales*

www.desafiolatam.com