Implementación y gestión de una base de datos

Argumentos, pooling y cursores

{desafío} latam_

Implementar la conexión a una base de datos PostgreSQL en una aplicación Node utilizando buenas prácticas.

Programar instrucciones para la obtención y manipulación de información desde una base de datos utilizando buenas prácticas, acorde al entorno Node.js.

{desafío} latam_

- Unidad 1:
  Implementación y gestión de una base de datos
- Unidad 2:
  Transacciones y API REST
- Unidad 3:
  Trabajo práctico

Te encuentras aquí

¿Qué aprenderás en esta sesión?

- *Utiliza sentencias para la captura y procesamiento de errores de conexión y estados de base de datos utilizando el entorno Node.js*
- *Utiliza sentencias para la consulta de datos mediante una conexión simple o un pool de conexiones para dar solución a un requerimiento.*
- *Distingue los mecanismos para la implementación de consultas síncronas y asíncronas de acuerdo al entorno Node.js.*

{desafío} latam_

¿Cuáles creen que son algunos posibles escenarios en los que la implementación de pooling y cursores en una base de datos puede beneficiarlos durante el desarrollo de proyectos o aplicaciones?

{desafío} latam_

/* Capturando argumentos por la línea de comandos */

Capturando argumentos por la línea de comandos

Cada vez que ejecutamos un archivo por la terminal escribimos una sintaxis basada en comandos, direcciones y flags(banderas), en sí todos estos son argumentos, al correr un documento JavaScript con Node no es la excepción.

Es importante que sepas que no será necesario usar ningún módulo, porque los argumentos de cada proceso ejecutado se almacenan como un arreglo en la propiedad “argv”, dentro del objeto “process”, el cual parecido a una variable global es posible acceder a él desde cualquier entorno de desarrollo en nuestros diferentes sistemas operativos.

Crea un archivo llamado index.js y escribe el siguiente código.

```
console.log(process.argv)
```

Ahora ejecutando este archivo por la terminal obtendrás la siguiente imagen, cabe recordar que la ruta de directorios puede variar para cada usuario:

Capturando argumentos por la línea de comandos

```
→  argv git:(master) ✗ node index.js
[
  '/home/brian/.nvm/versions/node/v12.18.3/bin/node',
  '/home/brian/Escritorio/node/Unidad 1/desafio1/ejercicio lectura/argv/index.js'
]
```

Como puedes apreciar en la consola hemos recibido un arreglo con 2 elementos tipo String que reflejan la dirección del recurso que se está ocupando en el argumento, en caso de no tener un soporte en ninguna localidad la propiedad “argv” captura el puro valor escrito.

Vuelve a correr el documento, pero agrégale un “Desafío Latam” luego de la mención del archivo y obtendrás lo que verás en la siguiente imagen:

```
→  argv git:(master) ✗ node index.js Desafio Latam
[
  '/home/brian/.nvm/versions/node/v12.18.3/bin/node',
  '/home/brian/Escritorio/node/Unidad 1/desafio1/ejercicio lectura/argv/index.js',
  'Desafio',
  'Latam'
]
```

{desafío} latam_

Como puedes ver, ahora obtenemos un arreglo de 4 elementos, siendo los últimos 2 el argumento textualmente escrito como texto.

Ejercicio guiado: Operación aritmética

{desafío} latam_

Ejercicio guiado

Operación aritmética

Crear una mini aplicación que devuelva la multiplicación de 2 números, los cuales serán tomados al momento de correr el documento. En el siguiente código te mostramos cómo hacerlo.

```
const argumentos = process.argv.slice(2)

let num1 = Number(argumentos[0])
let num2 = Number(argumentos[1])

console.log(num1 * num2)
```

Ejercicio guiado

Operación aritmética

Como verás estamos usando el método slice para recortar el arreglo, omitiendo los primeros 2 elementos porque no serán usados en este ejercicio, además de convertir los datos a tipo Number para poder multiplicarlos matemáticamente.

Al ejecutar este archivo obtendremos lo que te mostramos en la siguiente imagen:

```
→  argv git:(master) ✗ node index.js 2 2
4
```

/* Gestionando una alta demanda: Pooling, Configuración */

Gestionando una alta demanda: Pooling, Configuración

Cuando desarrollamos aplicaciones que persisten información con bases de datos SQL, tenemos que tomar en cuenta la demanda que se pueda tener por parte de nuestros clientes. Si la demanda es alta y estamos utilizando la clase Client del paquete pg, es probable que un cliente A realice una consulta en el momento que el servidor está en pleno procesamiento de la consulta de un cliente B. Entonces, ¿Qué ocurrirá con el cliente A? Recibirá un error, puesto que la conexión está ocupada y será rebotada su solicitud.

Para resolver este escenario, en este capítulo, aprenderás a usar y configurar la clase Pool, para ofrecer la gestión de múltiples consultas de manera simultánea, permitiendo la multiconexión de clientes.

Gestionando una alta demanda

Pooling

La forma en la que implementa la conexión a la base de datos la clase Client, no es apropiado para aplicaciones multitarea, donde la misma puede llegar a utilizar en paralelo múltiples operaciones sobre la base de datos.

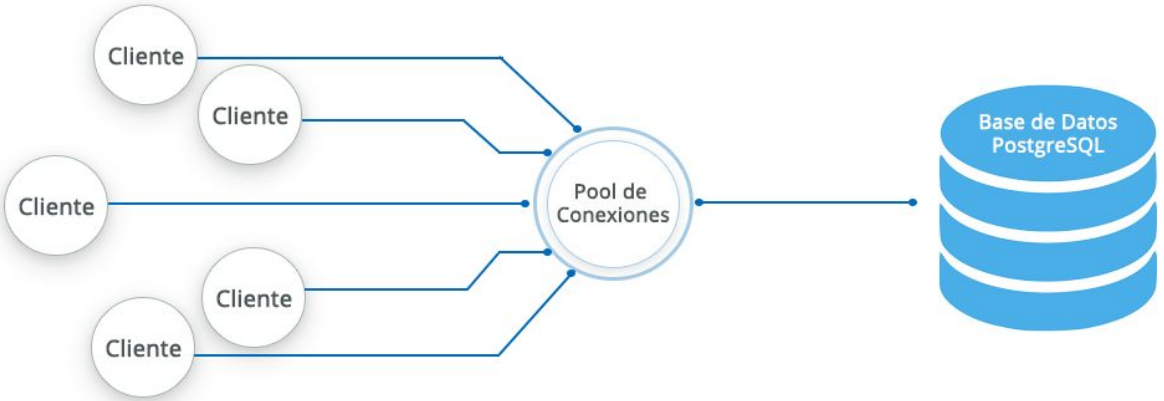
Para solucionar las falencias que puede tener este método, en informática existe el concepto de Conection Pooling, que hace referencia al manejo eficiente de un conjunto de conexiones abiertas a una base de datos, permitiendo gestionar múltiples consultas de forma simultánea. Este concepto es implementado por el método Pool de la librería pg donde sus principales características son:

- Manejo eficiente del conjunto de conexiones.
- Implementa políticas para agrupar las conexiones.
- Reutilización de conexiones cerrando conexiones inactivas.
- Manejo eficiente de las conexiones a la base de datos, administrando eficientemente la asignación a los hilos de ejecución del procesador.

Gestionando una alta demanda

Pooling

Mediante los hilos de ejecución se pueden administrar en forma eficiente las tareas del procesador y de sus distintos núcleos. La imagen que verás a continuación corresponde a un diagrama que representa la forma como se conectan varios clientes a través del método Pool a la base de datos.
Este método del módulo pg permite agrupar conexiones de varios clientes a la base de datos.



Gestionando una alta demanda

Configuración

Pool es una clase al igual que Client, recibe un objeto donde son válidos los mismos parámetros utilizados con la clase Cliente, pero la diferencia está en otras propiedades para generar un grupo de conexiones con la base de datos, a continuación te muestra a cuáles propiedades hago referencia:

- max: Recibe un valor numérico, que establece la cantidad máxima de clientes conectados que puede contener el grupo. Por defecto, se establece en un valor de 10.

- min: Recibe un valor numérico, que establece la cantidad mínima de clientes conectados para iniciar sus consultas. Por defecto, se establece en un valor de 0.

Gestionando una alta demanda

Configuración

- idleTimeoutMillis: Recibe un valor numérico que indica la cantidad de milisegundos que un cliente puede permanecer inactivo antes de que sea desconectado. Por defecto, se establece en un valor de 10.000 que equivalen a 10 segundos, si este valor se establece en 0 deshabilitará la desconexión automática para los clientes inactivos.

- connectionTimeoutMillis: Recibe un valor numérico que indica la cantidad de milisegundos, que deben transcurrir antes que se agote el tiempo de espera para conectar a un nuevo cliente. Por defecto, se establece en un valor de 0, que significa que no hay tiempo de espera.

- ssl: Recibe un valor booleano, es decir, un true o un false como valor, permite definir si la conexión a la base de datos soporta un protocolo de transporte encriptado.

Gestionando una alta demanda

Configuración

La función constructora de la clase Pool al igual que la clase Cliente se instancia utilizando la palabra new, como se muestra en el siguiente código donde se escribió unos valores de ejemplo para las propiedades nuevas.

```
const pool = new Pool({
    user: 'NOMBRE_USUARIO',
    host: '127.0.0.1',
    database: 'app_ejemplo',
    password: 'CLAVE_USUARIO',
    port: 5432,
    max: 20,
    min: 2,
    idleTimeoutMillis: 30000,
    connectionTimeoutMillis: 2000,
});
```

Gestionando una alta demanda

Configuración

¿Cómo se interpretaría esta configuración? Leyendo propiedad por propiedad, declarando lo siguiente:

- El nombre de usuario para conectarse a PostgreSQL.
- Servidor donde está alojada la base de datos.
- Base de datos a conectarnos.
- Contraseña correspondiente al usuario indicado.
- Puerto de conexión para el servicio.
- 20 clientes máximos para el Pool.
- 2 clientes como mínimo para que el Pool inicie las consultas.
- 30 segundos como tiempo de espera máximo de inactividad, es decir, que el cliente no realizó otra consulta en este tiempo.
- 2 segundos máximo de espera para recibir la consulta de otro cliente.
- El ssl no está declarado puesto que no aplica para esta lectura, por lo que por defecto su valor es false.

Hasta ahora, ¿puedes comentar con tus palabras lo que entendiste?

¿Qué conceptos o temas aprendidos en esta clase de Implementación y gestión de una base de datos te parecieron más interesantes o útiles y cómo planeas aplicarlos en futuros proyectos o en tu carrera profesional?

{desafío} latam_

Próxima sesión...

- *Identifica los riesgos de SQL Injection y las medidas para mitigarlo en una consulta de obtención de datos.*

{desafío}
latam_

{desafío}
latam_

Academia de
talentos digitales