TEXT CLASS REVIEW

TEMAS A TRATAR EN EL CUE:

- Conexión a una base de datos.
- ¿Qué es el pooling?
- El string de conexión URI.
- Capturando errores y mensajes.
- Resolviendo errores de conexión.

CONEXIÓN A UNA BASE DE DATOS

Instalación de paquetes necesarios

El paquete pg de Node para PostgreSQL:

Node.js proporciona un conjunto de herramientas para interactuar con bases de datos. Una de ellas es node-postgres, que contiene módulos que permiten que éste interactúe con la base de datos PostgreSQL. Usándola, podrá escribir programas Node.js que pueden acceder y almacenar datos en una base de datos PostgreSQL.

CONFIGURANDO EL PAQUETE PG E INCLUYENDO EL PAQUETE PG EN UN PROGRAMA NODE:

Para instalar el paquete node-postgres como una dependencia en un proyecto de node.js, se puede hacer de la siguiente manera:

```
1 npm install pg
```

CONECTANDO A LA BASE DE DATOS

Definiendo credenciales de acceso:

Los detalles relacionados con las credenciales de acceso que utiliza el módulo node-postgress para establecer la conexión a la base de datos PostgreSQL, incluyen la siguiente información:

- user: el usuario creado en postgres.
- database: el nombre de la base de datos creada en postgres.
- password: la contraseña del usuario creado en postgres.
- port: el puerto-postgres en el servidor. Es el 5432 por defecto.
- host: el servidor de base de datos.

En caso de que el servidor de base de datos sea el mismo donde se aloja la aplicación, se debe utilizar localhost. Si su servidor de base de datos es independiente, entonces se especificará su dirección IP.

Un ejemplo de definición de las credenciales de acceso sería:

```
const credenciales = {
 user: 'postgres',
 database: 'nodedemo',
 password: 'yourpassword',
 port: 5432,
 host: 'localhost',
};
```

EL CLIENTE DE CONEXIÓN:

Este permite conectarse al servidor de la base de datos mediante un comando, una utilidad, o una herramienta de terceros. Básicamente, se hace uso de la utilidad psql para conectar el servidor de la base de datos desde la interfaz del sistema operativo; también se utiliza la herramienta pg_admin para interactuar con el servidor de la base de datos, usando la interfaz del cliente.

CONECTANDO A LA BASE DE DATOS:

Existen un par de opciones para conectarse a una base de datos. Se puede usar un grupo de conexiones (connection pool), o simplemente crear una instancia de un cliente. Además, es posible especificar las credenciales de acceso a la base de datos en el código de la conexión, o configurarlas en variables de entorno.

Para conectar una aplicación Node.js, con una base de datos en PostgreSQL, se utiliza el paquete node-postgres mencionado anteriormente. Un pool de conexiones funciona como una memoria caché de conexiones a una base de datos, permitiendo a su aplicación reutilizar dichas conexiones para todas las solicitudes a la base de datos. Esto da la posibilidad de acelerar su aplicación, y ahorrar recursos del servidor.

POOL DE CONEXIONES

Un grupo de conexiones (connection pool), es un caché de conexiones de base de datos que se mantiene para que éstas se puedan reutilizar cuando se requieran futuras solicitudes a la base de datos. Se emplean para mejorar el rendimiento de la ejecución de comandos en una base de datos.

La agrupación de conexiones de base de datos es un método para mantener la base de datos abierta, y que pueda usarse una y otra vez sin desperdiciar recursos. Es la forma más fácil y la más común de usar node-postgres.

¿QUÉ ES POOLING?

En base de datos, se refiere a “pooling de conexiones” o “grupo de conexiones”. Este pool es administrado por un servidor de aplicaciones, el cual va asignando las conexiones a medida que los clientes van solicitando consultas, o actualizaciones de datos.

¿CUÁNDO USAR POOLING?

El pool es recomendado para cualquier aplicación que deba ejecutarse por un largo periodo de tiempo. Cada vez que se crea un cliente, se tiene que hacer un protocolo de enlace con el servidor PostgreSQL, y eso puede implicar algún tiempo. Un grupo de conexiones reciclará una cantidad predeterminada de objetos de cliente, para que así el protocolo de enlace no tenga que realizarse con tanta frecuencia. Entonces, se puede usar cuando:

- No es posible tolerar la sobrecarga de obtener y liberar conexiones cada vez que se utiliza una conexión.
- Es necesario compartir conexiones entre múltiples usuarios dentro de la misma transacción.

- Necesita aprovechar las características del producto, para administrar transacciones locales dentro del servidor de aplicaciones.
- No se gestiona la puesta en común de sus propias conexiones.

VENTAJAS Y DESVENTAJAS DE USAR POOLING

Ventajas:

- La rapidez con que agrega las conexiones a la base de datos.
- Es fácil examinar y diagnosticar la conexión a la base de datos.
- Permite tener centralizado y controlado el manejo de las conexiones a la base de datos.
- Es capaz de ofrecer múltiples conexiones lógicas, utilizando un reducido número de conexiones reales.
- Su manejo favorece la escalabilidad y el rendimiento de una aplicación.

Desventajas

- Éste inevitablemente introduce algo de latencia.
- Implica costes extra. Se necesita un servidor adicional, o el servidor de base de datos debe tener suficientes recursos para admitir pooling.
- Compartir conexiones entre diferentes módulos puede convertirse en una vulnerabilidad de seguridad.
- La autenticación cambia del DBMS al agrupador de conexiones. Esto puede no ser siempre aceptable.

EL STRING DE CONEXIÓN URI

Necesitamos una forma de conectar la base de datos para realizar varias operaciones, y ejecutar sentencias SQL. Una vez que podamos identificar los diversos parámetros involucrados en las credenciales para la conexión, se podrá construir la cadena de conexión. Hay dos formatos que son ampliamente utilizados:

1. Cadenas de valor/palabra clave sin formato.
2. URI de conexión.

Ejemplo valor/palabra clave:

```
host = localhost port=5432 dbname=nodedemo connect_timeout=10
```

Ejemplo de URI de conexión:

```
postgresql://username:password@host:port/dbname[?paramspec]
postgresql://postgres:yourpassword@localhost:5432/nodedemo/postgres
```

DESCONECTAR DE LA BASE DE DATOS:

Dependiendo de cómo se haya establecido la conexión a la base de datos, se debe hacer la desconexión. Cuando se establece haciendo uso de un grupo (pooling), se llama a pool.end, que vaciará el grupo de todos los clientes activos, los desconectará y apagará cualquier temporizador interno en el grupo. Es común hacer este llamado al final de un script.

```
pool.end()
```

Por otra parte, si la conexión se ha establecido a través de la instancia de un cliente, la desconexión se realiza a través de:

```
client.end()
```

Simplemente desconecta el cliente del servidor PostgreSQL.

CAPTURANDO ERRORES Y MENSAJES

Errores comunes de conexión:

- El servicio PostgreSQL no se está ejecutando.
- El puerto especificado para la conexión es incorrecto.
- El usuario de la base de datos no tiene la contraseña o el permiso correctos.
- Problema de conexión remota entre el servidor donde se aloja la aplicación y el servidor PostgreSQL, si fuera el caso.

- Información de conexión de PostgreSQL incorrecta, definida en la aplicación.

ATRAPANDO ERRORES AL CONECTAR:

Lo más importante es conocer qué errores pueden ocurrir, y qué significa cada uno de ellos. Node.js tiene varias formas básicas de manejar excepciones / errores, y entre ellas destacan:

- Bloque try-catch.
- Error como el primer argumento de una callback.

try-catch se utiliza para capturar las excepciones generadas desde la ejecución del código síncrono. Si la llamada la usó, entonces pueden detectar el error. Si la llamada no tuvo un intento de captura, el programa se bloquea.

Si se utiliza try-catch en una operación asíncrona, y se generó una excepción a partir de la devolución de llamada del método async, entonces no se detectará mediante ésta. Para capturar una excepción de la devolución de llamada de operación asíncrona, se prefiere usar Promise (objeto que representa la terminación o el fracaso de una operación asíncrona).

Los callbacks se utilizan principalmente en Node.js, pues éstos entregan un evento de forma asíncrona. El usuario le pasa una función (el callback), y la invoca más tarde cuando se completa la operación asincrónica. El patrón habitual es que el callback se invoque como: *callback (err, resultado)*, donde solo uno de err y resultado no es nulo, dependiendo de si la operación tuvo éxito, o falló.

Ejemplo:

```
client
        .connect()
        .then(() => console.log('connected'))
        .catch(err => console.error('connection error', err.message));
```

RESOLVIENDO ERRORES DE CONEXIÓN:

- El servicio PostgreSQL no se está ejecutando: para verificar esto, se ejecuta el comando “ps -ef | grep postgres” en un Shell, y así conocer el estatus del servicio.
- El puerto especificado para la conexión es incorrecto: por ejemplo, si se dejó el puerto por defecto, se puede utilizar el comando “netstat | grep número-de-puerto” para comprobarlo.
- El usuario de la base de datos no tiene la contraseña, o el permiso correcto: se puede verificar si los datos son erróneos estableciendo la conexión a la base de datos desde fuera de la aplicación. Puede hacerse vía comandos en un Shell, o usando la herramienta pgAdmin.
- Problema de conexión remota entre el servidor donde se aloja la aplicación, y el servidor PostgreSQL: en este caso, es posible acudir a los comandos “ping” y “tracert” en el computador local, y así probar la conexión remota al servidor.
- Información de conexión de PostgreSQL incorrecta definida en la aplicación: es importante verificar los datos especificados en las variables de conexión establecidas en la aplicación.

Formas correctas de conectar a una base de datos:

Ejemplo de conexión a base de datos, a través de una instancia de un cliente:

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

const credenciales = {
        user: 'postgres',
        database: 'nodedemo',
        password: 'yourpassword',
        port: 5432,
        host: 'localhost',
};

async function clientDemo() {

        const client = new Client(credenciales);
        await client.connect();
        const now = await client.query("SELECT NOW()");
        await client.end();

        return now;
}
```

Ejemplo de conexión a base de datos, a través de un pool de conexiones:

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

const credenciales = {
        user: 'postgres',
        database: 'nodedemo',
        password: 'yourpassword',
        port: 5432,
        host: 'localhost',
};

async function poolDemo() {

        const pool = new Pool(credenciales);
        const now = await pool.query("SELECT NOW()");
        await pool.end();

        return now;

}
```