OBJETIVO

En esta guía aprenderemos a integrar El sistema gestor de base de datos MySQL con NODE, para que a futuro puedas implementarlo en tus aplicaciones.

Introducción

MySQL es uno de los gestores de Base de Datos más popular en el desarrollo web. Mysql está optimizado para su funcionamiento en sistemas operativos GNU/Linux, aunque se puede instalar en cualquier Sistema Operativo.

Algunas características de MySQL:

- Permite escojer múltiples motores de almacenamiento para cada tabla.
- Ejecución de transacciones y uso de claves foráneas.
- Presenta un amplio subconjunto del lenguaje SQL.
- Replicación
- Soporta gran cantidad de datos, incluso con más de 50 millones de registros.
- En las últimas versiones, se permiten hasta 64 índices por tablas. Cada índice puede consistir desde 1 a 16 columnas o partes de columnas. El máximo ancho de límite son de 1000 bytes.

Mysql y NODE

Para poder conectarnos a una base de datos Mysql en node tenemos varias librerías como: mysql o node-mysql2. Ambos paquetes pueden ser implementados en nuestros proyectos instalando como dependencia:

```
$ npm install mysql
```

Crear una conexión

Una vez agregado la dependencia, lo primero que se debe hacer es crear una conexión:

```
const mysql = require('mysql');
const con = mysql.createConnection({
    host: 'localhost', //host o ip donde se encuentra instalado el servidor de la db
    user: 'root', //usuario de la base de datos
    password: '',//contraseña
    database: 'dev_distribu'//nombre de la base de datos
});

con.connect((err) => {
    if (err) throw err;
    console.log('Conexion exitosa!');
```

```
});
```

Insertar datos

Una vez establecida la conexión debemos usar la variable que contiene la conexión (`con`) para ejecutar consultas SQL.

Asumiendo que tenemos una tabla `personas`

```
CREATE TABLE IF NOT EXISTS `personas` (
  `id` INT(11) NOT NULL AUTO_INCREMENT,
  `nombres` text NOT NULL,
  `id_estado` VARCHAR(1) NOT NULL DEFAULT 'A',
  PRIMARY KEY (`id`)
);
```

La consulta para insertar datos:

```
con.query("insert into personas(nombres,id_estado) values('Diego Simeone','A');");
```

Podemos enviar un objeto Javascript al momento de insertar y obtener la respuesta de la operación en una función callback:

```
var per = {id_estado:'A',nombres:'Pablo Lezcano'};
con.query("insert into personas set ?",per,function(err,res){
  if(err)throw res;
  console.log(`Dato insertado ID: ${res.insertId}`);
});
```

Actualizar datos

De la misma forma que lo hicimos para insertar datos, invocamos al método `query` de la conexión para actualizar datos:

```
con.query("update personas set nombres=? where id = ?",["Marlene castillo
modificados",4],function(err,res){
    if(err)throw err;

    console.log(`Actualizados ${res.changedRows} fila(s)`);

});
```

Eliminar datos

Para eliminar datos invocamos a `query` y ejecutamos la consulta SQL DELETE:

```
con.query("delete from personas where id = ?",[3],function(err,res){
     if(err)throw err;
     console.log(`Eliminado ${res.changedRows} fila(s)`);

});
```

Listar datos

Para listar datos usamos la consulta SQL SELECT e iteramos cada fila usando `forEach`:

```
con.query("select * from personas",(err,rows)=>{
    if(err) throw  err;
    rows.forEach((row)=>{
        console.log(` ${row.nombres}`);
    });
});
```

Uso de ORM

Un ORM (Mapeador Objeto Relacional) es una forma de abstraer una Base de Datos(MySQL, PostgreSQL, Oracle) usando una structura lógica de entidades con el objetivo de simplificar y acelerar el desarrollo de nuestras aplicaciones.

En NODE existen una variedad de mapeadores ORM como: Booksheelfjs, Sequelize o Lovefiled.

El uso de ORM hace que el acceso a los datos sea más abstracto y portable, sin tener la necesidad de escribir SQL específico del proveedor de Base de datos que estemos usando.

Sequelize

Sequelize es uno de los ORM para Node compatible con PostgreSQL, MySQL,MariaDB, SQLite, y MSSQL. Para comenzar a usarlo debemos agregar como dependencia a nuestro proyecto:

```
$ npm install --save sequelize
```

Adicionalmente debemos añadir el driver de la base de datos con la cual trabajaremos:

```
$ npm install --save mysql2
```

una vez añadida la dependencia ya podremos usar sequelize de la siguiente forma:

```
var Sequelize = require("sequelize");
```

Creamos una conexión:

```
var sequelize = new
Sequelize("mysql://tecnologias:123456@localhost:3306/dev_distribu");
```

Creamos un modelo para representar la tabla personas:

```
//creamos un modelo
var Personas = sequelize.define("personas",{

        nombres:{
            type:Sequelize.STRING,
            field:'nombres'
        },
        id_estado:{
            type:Sequelize.STRING,
            field:'id_estado'
        },
        id:{
            type:Sequelize.INTEGER,
            field:'id',
            primaryKey:true,
            autoIncrement:true
        }

    },{
        freezeTableName:true,
        timestamps:false

    }
    );
```

Insertar datos

```
var p =  Personas.create({nombres:'Marco Martinez',id_estado:'A'});
```

Listar datos

Para listar los datos ejecutamos:

```
const {Op} = require("sequelize");
//select * from personas where id > 2 and id_estado='A'
Personas.findAll({
    where:{
        id: {
```

```
        [Op.gt]:2
        },
        id_estado:{
            [Op.eq]:'A'
        }
    }
}).then(result =>{

   // console.log(result);
   //  for(let i = 0; i< result.length;i++){
   //      console.log(result[i].dataValues);
   //
   //  }
    result.forEach((p)=>{
        console.log(p.dataValues);
    })


});
```

Actualizar datos

```
//update personas set nombres ='daniel padilla modificado' where id=5;
Personas.update({nombres:'daniel padilla modificado'},{where:{id:5}}).then(res =>{
   console.log("Filas actualizadas: "+res.affectedRows);
});
```

Eliminar datos

Para eliminar datos ejecutamos:

```
//delete from personas where id = 11;
Personas.destroy({where:{id:11}}).then( res =>{
    console.log("Eliminado "+res);
});
```