Accés a MySQL des de Node


En aquest document veurem com accedir a bases de dades relacionals (en concret, sistemes MySQL) des de Node.js. Explicarem quin mòdul utilitzarem per a això, i com realitzar amb ell les operacions bàsiques: connectar a la base de dades, operacions d'inserció/esborrat/modificació i consultes o llistats.

1. La llibreria *mysql2*

Convé tindre present que la combinació de Node.js i MySQL no és massa habitual. És bastant més freqüent l'ús de bases de dades MongoDB emprant aquest framework, ja que la informació que es maneja, en format BSON, és molt fàcilment exportable entre els dos extrems.

No obstant això, també existeixen eines per a poder treballar amb MySQL des de Node.js. Una de les més populars és la llibreria `mysql2`, disponible en el repositori NPM.

> **NOTA**: existeix una versió anterior de la llibreria, anomenada *mysql*. No obstant això, el mecanisme d'autenticació de les últimes versions de MySQL Server no és compatible amb aquesta llibreria, per la qual cosa és més recomanable emprar aquesta última versió.

Per a veure com utilitzar-la, començarem per crear un projecte anomenat "*ContactesMySQL*" en la nostra carpeta de "*ProjectesNode/Proves*". Dins crea dins un arxiu `index.js`, i executa la comanda `npm init` per a inicialitzar l'arxiu `package.json`. Després, instal·lem la llibreria amb la corresponent comanda `npm install`:

```
npm install mysql2
```

2. Connexió a la base de dades

Una vegada instal·lat el mòdul, en el nostre arxiu `index.js` ho importem (amb `require`), i executem el mètode `createConnection` per a establir una connexió amb la base de dades, d'acord amb els paràmetres de connexió que facilitarem en el propi mètode:

- `host`: nom del servidor (normalment `localhost`)
- `user`: nom de l'usuari per a connectar
- `password`: password de l'usuari per a connectar
- `database`: nom de la base de dades a la qual accedir, d'entre les que hi haja disponibles en el servidor al qual connectem.
- `port`: un paràmetre opcional, a especificar si el servidor de bases de dades està escoltant per un port que no és el port per defecte

- `charset`: també opcional, per a indicar un joc de codificació de caràcters determinat (per exemple, "utf8").

Per a les proves que farem en aquest projecte de prova, utilitzarem una base de dades anomenada "*contactes*" que pots descarregar, descomprimir i importar des d'ací, encara que ja ho hem demanat en un exercici previ. Tenint en compte tot l'anterior, podem deixar els paràmetres de connexió així:

```
const mysql = require('mysql2');

// Canviar usuari i contrasenya pels quals tinguem en
// el nostre sistema
let connexio = mysql.createConnection({
    host: "localhost",
    user: "root",
    password: "root",
    database: "contactes"
});
```

Després podem establir la connexió amb:

```
conexion.connect((error) => {
    if (error)
        console.log("Error en connectar amb la BD:", err);
    else
        console.log("Connexió satisfactòria");
});
```

En el cas que es produïsca algun error de connexió, l'identificarem en el paràmetre `error` i podrem actuar en conseqüència. En aquest cas es mostra un simple missatge per la consola, però també podem emmagatzemar-lo en algun *flag* booleà o una cosa similar per a impedir que es facen operacions contra la base de dades, o es redirigisca a una altra pàgina.

3. Consultes

La base de dades "contactes" té una taula del mateix nom, amb els atributs *id*, *nombre* i *telefono*.

id	nombre	telefono
1	Nacho Iborra	966112233
2	Arturo Bernal	965665544
3	Alex Amat	966998877

Definirem una consulta per a obtindre resultats i recórrer-los. Per exemple, mostrar tots els contactes:

```
conexion.query("SELECT * FROM contactos",
(error, resultat, camps) => {
    if (error)
        console.log("Error en processar la consulta");
    else
    {
        resultat.forEach((contacte) => {
            console.log(contacte.nombre, ":",
                contacte.telefono);
        });
    }
});
```

Notar que el mètode `query` té dos paràmetres: la consulta a realitzar, i un *callback* que rep altres tres paràmetres: l'error produït (si n'hi ha), el conjunt de resultats (que es pot processar com un vector d'objectes), i informació addicional sobre els camps de la consulta.

Notar també que el propi mètode `query` ens serveix per a connectar (disposa del seu propi control d'error), per la qual cosa no seria necessari el pas previ del mètode `connect`. En qualsevol cas, podem fer-ho si volem assegurar-nos que hi ha connexió, però cada *query* que fem també el pot verificar.

Existeixen altres maneres de fer consultes:

- Utilitzant marcadors (*placeholders*) en la pròpia consulta. Aquests marcadors es representen amb el símbol `?`, i se substitueixen després per l'element corresponent d'un vector de paràmetres que es col·loca en segona posició. Per exemple:

```
conexion.query("SELECT * FROM contactos WHERE id = ?", [1],
 (error, resultat, camps) => {
 ...
```

- Utilitzant com a paràmetre del mètode `query` un objecte amb diferents propietats de la consulta: la instrucció SQL en si, els paràmetres embeguts mitjançant *placeholders*... de manera que podem proporcionar informació addicional com *timeout*, conversió de tipus, etc.

```
conexion.query({
    sql: "SELECT * FROM contactos WHERE id = ?",
    values: [1],
    timeout: 4000
}, (error, resultat, camps) => {
 ...
```

4. Actualitzacions (insercions, esborrats, modificacions)

Si el que volem és realitzar alguna modificació sobre els continguts de la base de dades (INSERT, UPDATE o DELETE), aquestes operacions es realitzen des del mateix mètode `query` vist abans. La diferència està en el fet que en el paràmetre `resultat` del callback ja no estan els registres de la consulta, sinó dades com el nombre de files afectades (en l'atribut `affectedRows`), o l'*id* del nou element inserit (atribut `insertId`), en el cas d'insercions amb id autonumèric.

Per exemple, si volem inserir un nou contacte en l'agenda i obtindre l'*id* que se li ha assignat, ho podem fer així:

```
conexion.query("INSERT INTO contactos" +
    "(nombre, telefono) VALUES " +
    "('Fernando', '966566556')", (error, resultat, camps) => {
    if (error)
        console.log("Error en processar la inserció");
    else
        console.log("Nou id = ", resultat.insertId);
});
```

També podem passar un objecte JavaScript com a dada a la consulta, i automàticament s'assigna cada camp de l'objecte al camp corresponent de la base de dades (sempre que els noms dels camps coincidisquen). Això pot emprar-se tant en insercions com en modificacions:

```
conexion.query("INSERT INTO contactos SET ?",
 {nombre: 'Nacho C.', telefono: '965771111'},
 (error, resultat, camps) => {
 ...
});
```

Si fem un esborrat o actualització, podem obtindre el nombre de files afectades, d'aquesta manera:

```
conexion.query("DELETE FROM contactos WHERE id > 10",
 (error, resultat, camps) => {
    if (error)
        console.log("Error en realitzar l'esborrat");
    else
        console.log(resultat.affectedRows,
            "files afectades");
});
```

> **Exercici 1:**

> Crea una carpeta anomenada "**LlibresMySQL**" dins de la carpeta "*ProjectesNode/Exercicis*". Crea una base de dades anomenada *llibres* en MySQL i importa la còpia de seguretat que tens disponible comprimida ací.
>
> En el projecte, inicialitza l'arxiu `package.json` amb `npm init`, i instal·la el mòdul *mysql2*. Crea un arxiu `index.js` que carregue aquest mòdul (amb `require`), i realitze aquestes operacions sobre la base de dades de llibres:
>
> - Inserir 3 llibres qualsevol, amb les dades que se t'ocórreguen (observa quins camps té la taula).
> - Llistar els llibres de més de 10 euros.
> - Modificar el preu del llibre 1 a 35 euros.
> - Esborrar el llibre 3.