MySQLTUTORIAL

Connecting to the MySQL Server from Node.js

Summary: in this tutorial, you will learn how to connect to the MySQL Server from a Node.js application.

> Note that this tutorial targets the MySQL 8.0 and Node.js v20.6.0 or later. Node.js v20.6.0 and newer offers built-in support for `.env` configuration files.

Installing Node.js driver for MySQL

First, open the Command Prompt on Windows or Terminal on Unix-like systems.

Second, create a directory for storing the `Node.js` app and use the `npm init` command to create the `package.json` file:

```
npm init --yes
```

Third, install the MySQL package using the following `npm` command:

```
npm install mysql
```

Creating a sample database

First, connect to the MySQL server:

```
mysql -h localhost -u root -p
```

Second, create a new database called `todoapp` :

```
CREATE DATABASE todoapp;
```

Creating configuration file .env

First, create a new file called `.env` in the project directory.

Second, add the MySQL connection's parameters to the `.env` file:

```
DB_HOST=localhost
DB_PORT=3306
DB_USER=root
DB_PASSWORD=
DB_NAME=todoapp
```

You should replace the `DB_HOST`, `DB_PORT`, `DB_USER`, `DB_NAME`, and `DB_PASSWORD` with the actual ones.

Connecting to MySQL Server from Node.js

First, create a `connect.js` file in the project's directory.

Next, import the `mysql` module in the `connect.js` file:

```
let mysql = require('mysql');
```

Then, create a connection to the MySQL server by calling the `createConnection()` function:

```
let connection = mysql.createConnection({
  host: process.env.DB_HOST,
  port: process.env.DB_PORT,
  user: process.env.DB_USER,
  password: process.env.DB_PASSWORD,
  database: process.env.DB_NAME,
});
```

Note that we access the MySQL parameters from the `.env` file via the `process.env`. This feature has been available since Node.js v20.6.0.

After that, call the `connect()` method on the `connection` object to connect to the MySQL server:

```
connection.connect((err) => {
  if (err) return console.error(err.message);
```

```
  console.log('Connected to the MySQL server.');
});
```

The `connect()` method accepts a callback function that has the `err` argument that provides detailed information if any error occurs.

Here's the complete `connect.js` program:

```
let mysql = require('mysql');

let connection = mysql.createConnection({
  host: process.env.DB_HOST,
  port: process.env.DB_PORT,
  user: process.env.DB_USER,
  password: process.env.DB_PASSWORD,
  database: process.env.DB_NAME,
});

connection.connect((err) => {
  if (err) return console.error(err.message);

  console.log('Connected to the MySQL server.');
});
```

Finally, run the `connect.js` program that uses MySQL's parameters from the .env file:

```
 node --env-file .env connect.js
```

Output:

```
Connected to the MySQL server.
```

To make it more convenient, you can change the start property of the package.json file to the following:

```
...
  "scripts": {
    "start": " node --env-file .env connect.js"
```

```
  },
...
```

Then, use the npm start command to run the connect.js with information from the .env file:

```
npm start
```

The output indicates that we have successfully connected to the MySQL server from the Node.js program.

Troubleshooting

If you connect to MySQL 8.0 or later, you are likely getting the following error message:

```
error: ER_NOT_SUPPORTED_AUTH_MODE: Client does not support authentication protocol request
```

In MySQL 8.0, the default authentication plugin is `caching_sha2_password`, unlike MySQL 5.7, which uses the `mysql_native_password` plugin, supported by most clients.

Therefore, if you encounter compatibility issues, you must explicitly enable `mysql_native_password` for a given user using the following command:

```
ALTER USER 'user'
IDENTIFIED WITH mysql_native_password BY 'password';
```

Replace the user and password with the ones that you use to connect to MySQL.

Closing database connection

To close a database connection gracefully, you call the `end()` method on the `connection` object.

The `end()` method ensures that all remaining queries will be executed before the database connection is closed.

```
connection.end((err) => {
  if (err) return console.error(err.message);
```

```
  console.log('Close the database connection.');
});
```

To force the connection to close immediately, you can use the `destroy()` method. The `destroy()` method guarantees that no more callbacks or events will be triggered for the connection.

```
connection.destroy();
```

Note that the `destroy()` method does not take any callback argument like the `end()` method.

The following shows the complete `connect.js` program:

```
let mysql = require('mysql');

let connection = mysql.createConnection({
  host: process.env.DB_HOST,
  port: process.env.DB_PORT,
  user: process.env.DB_USER,
  password: process.env.DB_PASSWORD,
  database: process.env.DB_NAME,
});

connection.connect((err) => {
  if (err) return console.error(err.message);

  console.log('Connected to the MySQL server.');
});

connection.end((err) => {
  if (err) return console.error(err.message);

  console.log('Close the database connection.');
});
```

Summary

- Connect to a MySQL database from a Node.js application.