Table of Contents

Node.js Driver for SQL Server

8/9/2017 • 1 min to read • Edit Online

To download Node.js SQL driver

The tedious module is a javascript implementation of the TDS protocol, which is supported by all modern version of SQL Server. The driver is an open source project, available on Github.

You can connect to a SQL Database using Node.js on Windows, Linux, or Mac.

Getting started

- Step 1: Configure development environment for Node.js development
- Step 2: Create a SQL database for Node.js development
- Step 3: Proof of concept connecting to SQL using Node.js

Documentation

Tedious module documentation on Github

Community

- Azure Node.js Developer Center
- Get Involved at nodejs.org

Code examples

- Getting Started with Node.js on Windows
- Getting Started with Node.js on macOS
- Getting Started with Node.js on Ubuntu
- Getting Started with Node.js on Red Hat Enterprise Linux (RHEL)
- Getting Started with Node.js on SUSE Linux Enterprise Server (SLES)

Step 1: Configure development environment for Node.js development

3/14/2017 • 2 min to read • Edit Online

You will need to configure your development environment with the prerequisites in order to develop an application using the Node.js Driver for SQL Server. The most common method is to use the node package manager (npm) to install the tedious module, but you can download the tedious module directly at Github if you prefer.

Note that the Node.js Driver uses the TDS protocol, which is enabled by default in SQL Server and Azure SQL Database. No additional configuration is required.

Windows

1. **Install Node.js runtime and npm package manager**
   a. Go to Node.js
   b. Click on the appropriate Windows installer msi link.
   c. Once downloaded, run the msi to install Node.js
2. **Open cmd.exe**
3. **Create a project directory** and navigate to it.

```
> mkdir HelloWorld
> cd HelloWorld
```

4. **Create a Node project.** To retain the defaults during your project creation, press enter until the project is created. At the end of this step, you should see a package.json file in your project directory.

```
> npm init
```

5. **Install tedious module in your project.** This is the implementation of the TDS protocol which the driver uses to communicate to SQL Server.

```
> npm install tedious
```

Ubuntu Linux

1. **Open terminal**
2. **Install Node.js runtime**

```
>sudo apt-get install node
```

3. **Install npm (node package manager)**
   `> sudo apt-get install npm`
4. **Create a project directory** and navigate to it.

```
> mkdir HelloWorld
> cd HelloWorld
```

5. **Create a Node project.** To retain the defaults during your project creation, press enter until the project is created. At the end of this step, you should see a package.json file in your project directory.

```
> sudo npm init
```

6. **Install tedious module in your project.** This is the implementation of the TDS protocol which the driver uses to communicate to SQL Server.

```
> sudo npm install tedious
```

Mac

1. **Install Node.js runtime and npm package manager**
   a. Go to Node.js
   b. Click on the appropriate Mac OS installer link.
   c. Once downloaded, run the dmg to install Node.js

2. **Open terminal**

3. **Create a project directory** and navigate to it.

```
> mkdir HelloWorld
> cd HelloWorld
```

4. **Create a Node project.** To retain the defaults during your project creation, press enter until the project is created. At the end of this step, you should see a package.json file in your project directory.

```
> npm init
```

5. **Install tedious module in your project.** This is the implementation of the TDS protocol which the driver uses to communicate to SQL Server.

```
> npm install tedious
```

Step 2: Create a SQL database for Node.js development

8/9/2017 • 1 min to read • Edit Online

The samples in this section only work with the AdventureWorks schema, on either Microsoft SQL Server or Azure SQL Database.

Azure SQL Database

Create a SQL database in minutes using the Azure portal

Microsoft SQL Server

Microsoft SQL Server database product samples, on CodePlex

Step 3: Proof of concept connecting to SQL using Node.js

8/9/2017 • 2 min to read • Edit Online

To download Node.js SQL driver

This example should be considered a proof of concept only. The sample code is simplified for clarity, and does not necessarily represent best practices recommended by Microsoft. Other examples which use the same crucial functions are available on Github:

- https://github.com/tediousjs/tedious/blob/master/examples/

Step 1: Connect

The **new Connection** function is used to connect to SQL Database.

```
var Connection = require('tedious').Connection;
var config = {
    userName: 'yourusername',
    password: 'yourpassword',
    server: 'yourserver.database.windows.net',
    // If you are on Microsoft Azure, you need this:
    options: {encrypt: true, database: 'AdventureWorks'}
};
var connection = new Connection(config);
connection.on('connect', function(err) {
// If no error, then good to proceed.
    console.log("Connected");
});
```

Step 2: Execute a query

All SQL statements are executed using the **new Request()** function. If the statement returns rows, such as a select statement, you can retrieve them using the **request.on()** function. If there are no rows, the request.on() function returns empty lists.

```
    var Connection = require('tedious').Connection;
    var config = {
        userName: 'yourusername',
        password: 'yourpassword',
        server: 'yourserver.database.windows.net',
        // When you connect to Azure SQL Database, you need these next options.
        options: {encrypt: true, database: 'AdventureWorks'}
    };
    var connection = new Connection(config);
    connection.on('connect', function(err) {
        // If no error, then good to proceed.
        console.log("Connected");
        executeStatement();
    });

    var Request = require('tedious').Request;
    var TYPES = require('tedious').TYPES;

    function executeStatement() {
        request = new Request("SELECT c.CustomerID, c.CompanyName,COUNT(soh.SalesOrderID) AS OrderCount FROM
SalesLT.Customer AS c LEFT OUTER JOIN SalesLT.SalesOrderHeader AS soh ON c.CustomerID = soh.CustomerID GROUP BY
c.CustomerID, c.CompanyName ORDER BY OrderCount DESC;", function(err) {
        if (err) {
            console.log(err);}
        });
        var result = "";
        request.on('row', function(columns) {
            columns.forEach(function(column) {
              if (column.value === null) {
                console.log('NULL');
              } else {
                result+= column.value + " ";
              }
            });
            console.log(result);
            result ="";
        });

        request.on('done', function(rowCount, more) {
        console.log(rowCount + ' rows returned');
        });
        connection.execSql(request);
    }
```

Step 3: Insert a row

In this example you will see how to execute an INSERT statement safely, pass parameters which protect your application from SQL injection vulnerability, and retrieve the auto-generated Primary Key value.

```
    var Connection = require('tedious').Connection;
    var config = {
        userName: 'yourusername',
        password: 'yourpassword',
        server: 'yourserver.database.windows.net',
        // If you are on Azure SQL Database, you need these next options.
        options: {encrypt: true, database: 'AdventureWorks'}
    };
    var connection = new Connection(config);
    connection.on('connect', function(err) {
        // If no error, then good to proceed.
        console.log("Connected");
        executeStatement1();
    });

    var Request = require('tedious').Request
    var TYPES = require('tedious').TYPES;

    function executeStatement1() {
        request = new Request("INSERT SalesLT.Product (Name, ProductNumber, StandardCost, ListPrice,
SellStartDate) OUTPUT INSERTED.ProductID VALUES (@Name, @Number, @Cost, @Price, CURRENT_TIMESTAMP);",
function(err) {
         if (err) {
            console.log(err);}
        });
        request.addParameter('Name', TYPES.NVarChar,'SQL Server Express 2014');
        request.addParameter('Number', TYPES.NVarChar , 'SQLEXPRESS2014');
        request.addParameter('Cost', TYPES.Int, 11);
        request.addParameter('Price', TYPES.Int,11);
        request.on('row', function(columns) {
            columns.forEach(function(column) {
              if (column.value === null) {
                console.log('NULL');
              } else {
                console.log("Product id of inserted item is " + column.value);
              }
            });
        });
        connection.execSql(request);
    }
```