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

Article • 09/07/2024

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.

JavaScript

```
var Connection = require('tedious').Connection;
var config = {
    server: 'your_server.database.windows.net',  //update me
    authentication: {
        type: 'default',
        options: {
            userName: 'your_username', //update me
            password: 'your_password'  //update me
        }
    },
    options: {
        // If you are on Microsoft Azure, you need encryption:
        encrypt: true,
        database: 'your_database'  //update me
    }
};
var connection = new Connection(config);
connection.on('connect', function(err) {
    // If no error, then good to proceed.
    console.log("Connected");
});

connection.connect();
```

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.

JavaScript

```
    var Connection = require('tedious').Connection;
    var config = {
        server: 'your_server.database.windows.net',  //update me
        authentication: {
            type: 'default',
            options: {
                userName: 'your_username', //update me
                password: 'your_password'  //update me
            }
        },
        options: {
            // If you are on Microsoft Azure, you need encryption:
            encrypt: true,
            database: 'your_database'  //update me
        }
    };
    var connection = new Connection(config);
    connection.on('connect', function(err) {
        // If no error, then good to proceed.
        console.log("Connected");
        executeStatement();
    });

    connection.connect();

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

    function executeStatement() {
        var 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');
    });

    // Close the connection after the final event emitted by the re-
quest, after the callback passes
    request.on("requestCompleted", function (rowCount, more) {
        connection.close();
    });
    connection.execSql(request);
}
```

Step 3: Insert a row

In this example you will see how to execute an INSERT statement safely, passing parameters, which protect your application from SQL injection values.

JavaScript

```
var Connection = require('tedious').Connection;
var config = {
    server: 'your_server.database.windows.net',  //update me
    authentication: {
        type: 'default',
        options: {
            userName: 'your_username', //update me
            password: 'your_password'  //update me
        }
    },
    options: {
        // If you are on Microsoft Azure, you need encryption:
        encrypt: true,
        database: 'your_database'  //update me
    }
};
var connection = new Connection(config);
connection.on('connect', function(err) {
    // If no error, then good to proceed.
    console.log("Connected");
    executeStatement1();
});

connection.connect();

var Request = require('tedious').Request
```

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

    function executeStatement1() {
        var 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);
              }
            });
        });

        // Close the connection after the final event emitted by the re-
quest, after the callback passes
        request.on("requestCompleted", function (rowCount, more) {
            connection.close();
        });
        connection.execSql(request);
    }
```

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A