We will explore how to **use Node.js to connect to SQL Server database** and perform read/write operations. There are some packages available to connect to SQL Server database from Node.js. We can grab information about these packages from http://www.npmjs.com.

Two popular packages are:

· **msnodesql -** this is Microsoft's driver for Node.js for SQL Server. It allows Node.js applications on Windows platform and Azure platform to connect to Sql Server and SQL Azure. More information about this package can be read from this link.

· **mssql** is a popular third-party easy-to-use SQL Server connector for Node.js. It supports Stored Procedures, Sql Statements, Transactions and Connection Pooling. This package uses **Promises** to perform database operations and Callback mechanisms too. More information about this package can be read from this link.

Since mssql is production ready, we will use this in our application. The article uses a local instance of SQL Server database with Mixed Mode authentication. Before implementing the application, we will make sure that the following Sql environment configuration are done successfully.

Features of mssql package

Here are some features of mssql package as mentioned in its homepage

- unified interface for multiple TDS drivers.
- built-in connection pooling.
- Support for built-in JSON serialization (SQL Server 2016) as well as serialization of Geography and Geometry CLR types
- Support for Stored Procedures, Transactions, Prepared Statements, Bulk Load and Table-Valued Parameters (TVP).
- Mapping of JS data type to SQL data type.
- Support for Promises, Streams and standard callbacks.
- Production ready and well documented.

Note 1: mssql is a wrapper around other clients for MS-SQL like **tedious. It uses tedious under the hood,** is simpler to use, and contains additional functionality like connection pooling.

Note 2: Since we are using SQL Server, we will be using a Windows environment. It is assumed that you have already installed SQL Server on your machine.

Connecting to SQL Server using Node.js

Step 1: Run Services.msc from Start > Run, and make sure that following services are running:

- SQL Server
- SQL Server Agent
- SQL Server Browser

Step 2: Open Sql Server Configuration Manager, and enable TCP/IP protocol from SQL Server Network Configuration

This application is implemented using Visual Studio Code. This is a free new IDE for building and debugging modern Web and Cloud applications. This IDE can be downloaded from here. This application uses Node.js tools. They can be downloaded from here.

Step 3: For this application, we will be using a fictional Company Database with Employee and EmployeeInfo tables. The script for Database and Tables are as following:

```
Create database Company
```

Employee Table

```
CREATE TABLE [dbo].[Employee](
    [EmpNo] [int] NOT NULL,
    [EmpName] [varchar](50) NOT NULL,
    [Salary] [int] NOT NULL,
    [DeptNo] [int] NOT NULL,
 CONSTRAINT [PK_Employee] PRIMARY KEY CLUSTERED
(
    [EmpNo] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS
= ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
```

Add some Test Data to the Table

EmployeeInfo Table

```
CREATE TABLE [dbo].[EmployeeInfo](
    [EmpNo] [int] IDENTITY(1,1) NOT NULL,
    [EmpName] [varchar](50) NOT NULL,
    [Salary] [decimal](18, 0) NOT NULL,
    [DeptName] [varchar](50) NOT NULL,
    [Designation] [varchar](50) NOT NULL,
 CONSTRAINT [PK_EmployeeInfo] PRIMARY KEY CLUSTERED
(
    [EmpNo] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS
= ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
```

Add a GetAllEmployeesSalary Stored Procedure.

```
USE [Company]
GO
/****** Object:  StoredProcedure [dbo].[GetAllEmployeesBySalary]    Script Date: 1/4/2016
12:11:33 PM ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
```

```
GO
Create PROCEDURE [dbo].[GetAllEmployeesBySalary]
    (
        @Salary int
    )
AS
        Select * from Employee where Salary>@Salary
    RETURN
```

Step 4: This step is required only for those who are using Visual Studio Code.

On the disk, create a new folder with the name **NodeSqlServer.** Open VS Code and using File > Open Folder, open the NodeSqlServer folder. Using VSCode add a new JavaScript file of name app.js in this folder. Right click on app.js and select **Open in Command Prompt** option. This will open Command prompt. Run the following command.

```
npm install -g tsd
```

This will install a Typescript definition for the VSCode so that we can get the help of the intellisense for various packages.

```
tsd query node --action install
```

The above command provides Node.js intellisense for the application.

```
npm install mssql -g
```

This command will install mssql package in node-modules folder.

```
tsd query mssql --action install
```

This will install mssql intellisense for the current application.

Step 5: In the app.js, add the following code:

```
//1.
var sql = require('mssql');
//2.
var config = {
    server: 'localhost',
    database: 'Company',
    user: 'sa',
    password: 'sa',
    port: 1433
};
//3.
function loadEmployees() {
    //4.
    var dbConn = new sql.Connection(config);
    //5.
    dbConn.connect().then(function () {
        //6.
        var request = new sql.Request(dbConn);
        //7.
        request.query("select * from EmployeeInfo").then(function (recordSet) {
            console.log(recordSet);
```

```
            dbConn.close();
        }).catch(function (err) {
            //8.
            console.log(err);
            dbConn.close();
        });
    }).catch(function (err) {
        //9.
        console.log(err);
    });
}
//10.
loadEmployees();
```

The above code does the following:

(Note: Following numbering matches with Comment numbers given for the above code.)

1. Load the **mssql** module so that Sql server can be connected for performing Read/Write operations.

2. The JavaScript **config** object with Sql Server connection properties e.g. server name, database name, user name, password and port. In this application, we are connecting to the localhost instance. localhost can be replaced by IP address or a Sql Server named instance. The default port 1433 must be set.

3. The **loadEmployees ()** function. This is used to read all employees from EmployeeInfo table of the Company Database.

4. A Connection instance is created of the name **dbConn** using **Connection()** function of the sql object defined in Line 1. The **config** object is passed to this function.

5. The **connect()** function of the dbConn instance uses the promise object, (this is provided by the mssql package), to make sure that the Read/Write operation with Sql Server can be performed only when the connection is successfully established.

6. If the connection is successful, then the **Request** object is created over the Connection. This object is used to query the table.

7. The **query ()** function is called by passing a Sql Statement to in (in this is case it is SELECT statement), this again returns the **promise** object. If the query is successful, then data is returned using **recordSet** object.

8. If there is problem with query then **catch()** function will be executed with error.

9. If the connection fails with the database then **catch()** function will be executed with an error.

10. Call the **loadEmployees()** function.

Step 6: Run the application using the following command from the Command prompt which we opened using Step 4.

```
node app
<img title="node-sqlserver-result" border="0" alt="node-sqlserver-result"
src="https://www.dotnetcurry.com/images/nodejs/Connect-to-SQL-Server-using-
```

```
Node.js_7CB0/node-sqlserver-result.png" width="655" height="422">
```

Executing a Stored Procedure from Node.js application

Step 7: In app.js, add the following function

```
//1
function executeStoredProc() {
    //2.
    var dbConn = new sql.Connection(config);
    dbConn.connect().then(function () {

        //3.
        var request = new sql.Request(dbConn);
        request.input('Salary', sql.Int, 50000)
        .execute("GetAllEmployeesBySalary").then(function (recordSet) {
            //4.
            console.log(recordSet);
            dbConn.close();
        }).catch(function (err) {
            //5.
            console.log(err);
            dbConn.close();
        });
    }).catch(function (err) {
        //6.
        console.log(err);
    });
}
//7.
executeStoredProc();
```

The above code does the following:

(Note: Following numbering matches with Comment numbers given for the above code.)

1. The function executeStoredProc () contains logic executing stored procedure on Sql Server.

2. The **dbConn** is the connection instance using the **Connection()** function of the **sql** object of the **mssql** package. This function accepts the **config** JavaScript object which contains Sql Server information.

3. The **Request** object is created using the **dbConn** instance of the Connection object. This is used to define the input parameters to stored procedure using **input()** function. This function accepts three parameters representing parameter name (Salary), parameter type (sql.Int) and value (50000). The request object is used to execute stored procedure using **execute()** function. This function accept the stored procedure name (GetAllEmployeesBySalary). This function returns the promise object.

4. If the stored procedure is executed successfully then the recordSet parameter will be returned and will be printed on console.

5. If the stored procedure fails then **catch()** will be executed with error.

6. If the connection with database fails, then **catch()** will be executed with the error details.

7. Call the executeStoredProc() function.

Step 8: Run the application using the following command from the Command prompt which we opened using Step 4.

```
node app
<img title="stored-proc" border="0" alt="stored-proc"
src="https://www.dotnetcurry.com/images/nodejs/Connect-to-SQL-Server-using-
Node.js_7CB0/stored-proc.png" width="837" height="220">
```

Inserting a New Row in the Table

Step 9: In app.js, add the following code

```
//1.
function insertRow() {
    //2.
    var dbConn = new sql.Connection(config);
    //3.
    dbConn.connect().then(function () {
        //4.
        var transaction = new sql.Transaction(dbConn);
        //5.
        transaction.begin().then(function () {
            //6.
            var request = new sql.Request(transaction);
            //7.
            request.query("Insert into EmployeeInfo (EmpName,Salary,DeptName,Designation)
values ('T.M. Sabnis',13000,'Accounts','Lead')")
        .then(function () {
                //8.
                transaction.commit().then(function (recordSet) {
                    console.log(recordSet);
                    dbConn.close();
                }).catch(function (err) {
                    //9.
                    console.log("Error in Transaction Commit " + err);
                    dbConn.close();
                });
            }).catch(function (err) {
                //10.
                console.log("Error in Transaction Begin " + err);
                dbConn.close();
            });

        }).catch(function (err) {
            //11.
```

```
            console.log(err);
            dbConn.close();
        });
    }).catch(function (err) {
        //12.
        console.log(err);
    });
}
//13.
insertRow();
```

The above code does the following:

(Note: Following numbering matches with Comment numbers given for the above code.)

1. The **insertRow()** function contains logic for performing an insert operation with Sql Server.

2. The Connection instance of name **dbConn** is declared using **Connection()** function of the sql object. sql is an instance of the mssql package. The Connection() function accepts the **config** object. This object defines the Sql Server information.

3. The **connect()** function call returns a promise object.

4. If the database connection is successful, then the **Transaction** object is created using dbConn.

5. The Transaction **begin()** function is used to start the transaction.

6. The Request object is created over the transaction.

7. The **query()** function of the request object accepts a SQL query (in this case it is insert query).

8. If the query is executed, then the transaction **commit()** function is called which will commit INSERT on the table specified in the insert query.

9. If commit fails, then **catch()** is executed with an error.

10. If query fails, then **catch()** is executed with an error.

11. If transaction is not initiated, then **catch()** is executed with an error.

12. If database connection fails then **catch()** is executed with an error.

13. Call the insertRow () function.

Step 10: Run the application using the following command from the Command prompt which we opened using Step 4.

```
node app
```

This will insert the record in the EmployeeInfo table. Similarly you can use UPDATE and DELETE commands as well.