

ПРОГРАММИРОВАНИЕ СЕРВЕРНЫХ КРОССПЛАТФОРМЕННЫХ ПРИЛОЖЕНИЙ

MSSQL

Node.js поддерживает работу со всеми популярными СУБД. Microsoft SQL Server не исключение.

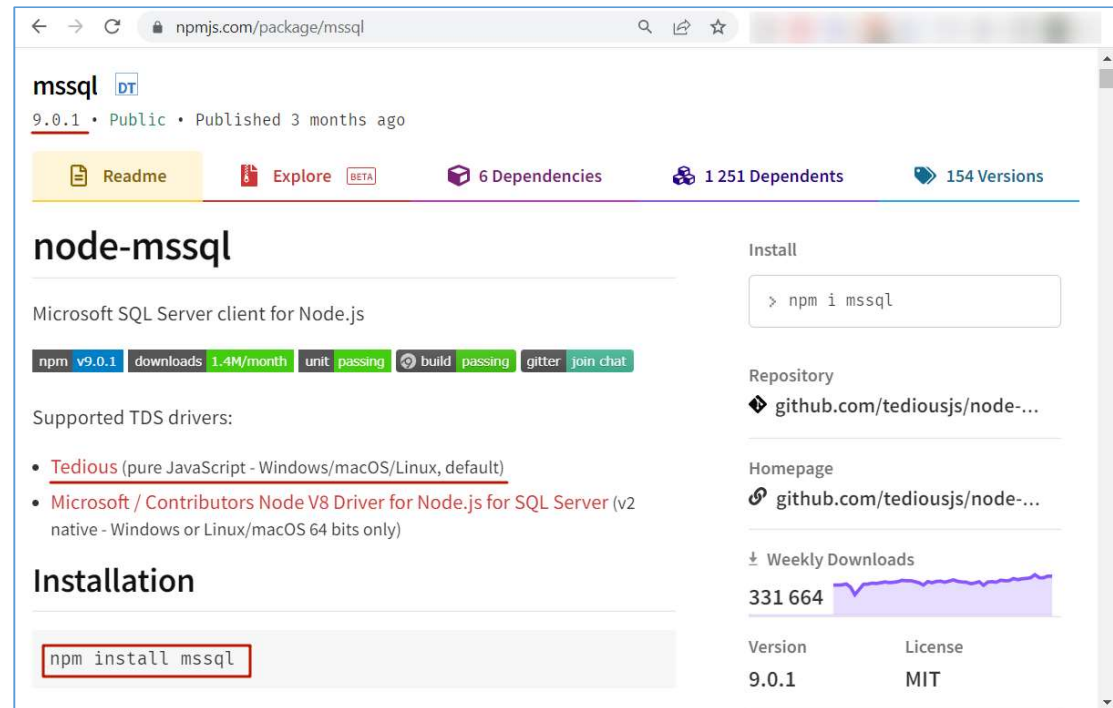
В репозитории npm существует множество пакетов, которые позволяют работать с MS SQL. Однако все их можно условно разделить на две большие группы в зависимости от реализуемого подхода:

- непосредственная работа с БД **с использованием SQL запросов**;
- сопоставление объектной модели и структуры БД **посредством ORM** (Object Relational Model).

mssql

=

пакет, позволяющий работать
с MS SQL посредством SQL
запросов.



The screenshot shows the npm package page for 'mssql'. The package is version 9.0.1, published 3 months ago. It has 6 dependencies, 1251 dependents, and 154 versions. The package is described as the 'Microsoft SQL Server client for Node.js'. It lists supported TDS drivers: 'Tedious' (pure JavaScript - Windows/macOS/Linux, default) and 'Microsoft / Contributors Node V8 Driver for Node.js for SQL Server (v2 native - Windows or Linux/macOS 64 bits only)'. The installation command is shown as 'npm install mssql'. The package is licensed under MIT.

mssql 9.0.1 • Public • Published 3 months ago

Readme Explore BETA 6 Dependencies 1251 Dependents 154 Versions

node-mssql

Microsoft SQL Server client for Node.js

npm v9.0.1 downloads 1.4M/month unit passing build passing glitter join chat

Supported TDS drivers:

- [Tedious](#) (pure JavaScript - Windows/macOS/Linux, default)
- [Microsoft / Contributors Node V8 Driver for Node.js for SQL Server \(v2 native - Windows or Linux/macOS 64 bits only\)](#)

Installation

```
npm install mssql
```

Install

```
> npm i mssql
```

Repository

[github.com/tediousjs/node-...](#)

Homepage

[github.com/tediousjs/node-...](#)

Weekly Downloads

331 664

Version 9.0.1 License MIT

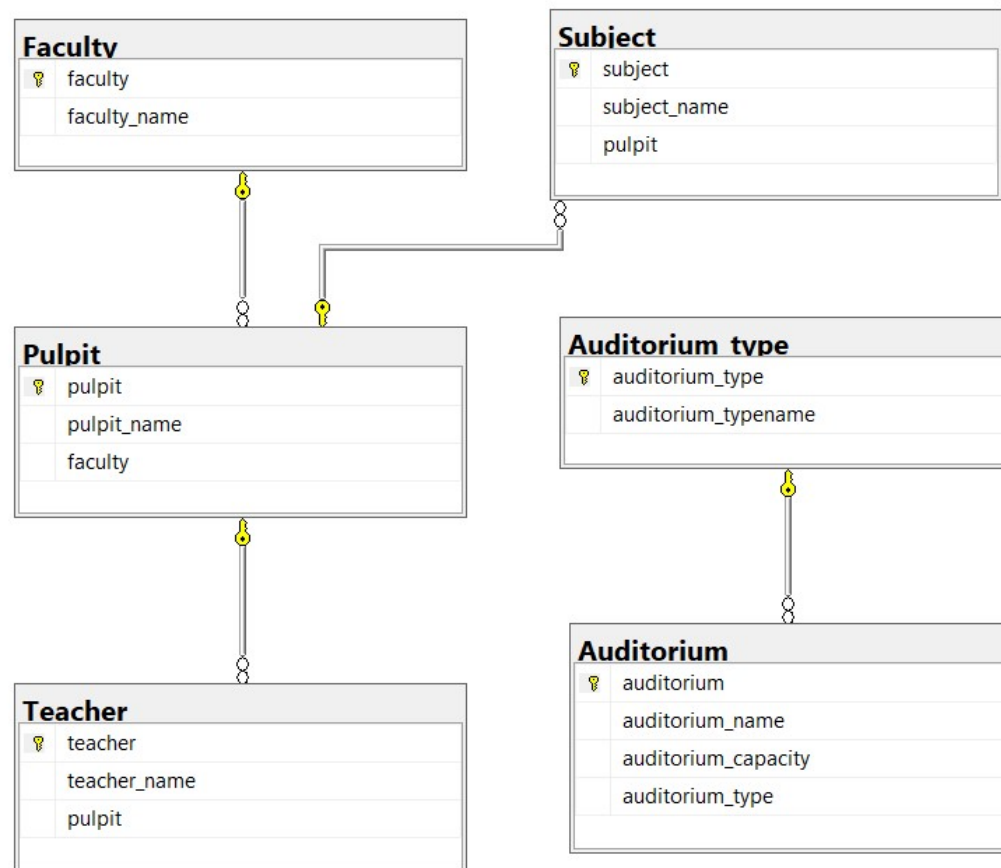
[документация](#)

Драйвер базы = данных

это прослойка между кодом и базой данных. В mssql поддерживаются 2 драйвера:

- **Tedious** (pure JavaScript - Windows/macOS/Linux, default) – используется по умолчанию, встроен, активно поддерживается Microsoft, работает независимо от платформы.
- **Microsoft / Contributors Node V8 Driver for Node.js for SQL Server** (v2 native – Windows or Linux/macOS 64 bits only, msnodesqlv8) – этот драйвер не является частью пакета по умолчанию, его необходимо скачивать через npm и подключать.

Диаграмма базы данных для примеров



Классификация запросов



```
graph TD; A[Классификация запросов] --> B[По типу параметров]; A --> C[По количеству переиспользований]; B --> B1[• статические]; B --> B2[• динамические]; C --> C1[• обычные]; C --> C2[• подготовленные];
```

По типу параметров

- статические
- динамические

По количеству переиспользований

- обычные
- подготовленные

Статический запрос

=

запрос, который **определен** уже **в момент компиляции программы**, какого-либо **изменения** после их однократного написания **не предполагается**.

Подключение, статический select

```
const sql = require('mssql'); // https://www.npmjs.com/package/mssql#callbacks

let config = { user: 'student', password: 'xfitfit', server: '172.16.193.223', database: 'NodeJSTest'};
```

```
let dbreq01 = ()=>{new sql.Request().query('select faculty, faculty_name from FACULTY', processing_result)};
let dbreq02 = ()=>{new sql.Request().query('select faculty, pulpit, pulpit_name from PULPIT order by faculty',
,processing_result)};
let dbreq03 = ()=>{new sql.Request().query('select teacher, teacher_name, pulpit from TEACHER order by pulpit',
,processing_result)};
```

```
sql.connect(config, err => {
```

```
  if(err) console.log('Ошибка соединения с БД:', err.code);
  else{ console.log('Соединение с БД установлено');
    dbreq01();
    dbreq02();
    dbreq03();
  }
})
```

```
let processing_result = (err, result) => {
  if (err) console.log('processing_result error:', err.code, err.originalError.info.message);
  else {
    console.log('Количество строк: ', result.rowsAffected[0]);
    for (let i = 0; i < result.rowsAffected[0]; i++){
      let str = '--';
      for (key in result.recordset[i]) {
        str += ` ${key} = ${result.recordset[i][key]} `;
      }
      console.log(str);
    }
  }
}
```

Класс `sql.Request` используется для работы с обычными запросами к БД. Если аргумент `pool/transaction` при `new sql.Request` ([`pool` or `transaction`]) не указан, то используется глобальный пул.

Метод `rq.query(command, [callback])` выполняет указанную команду SQL.

Метод `connect` устанавливает соединение с БД. Принимает конфигурационный объект/строку подключения. Возвращает существующий глобальный пул, если он существует, или заново созданный, если его нет

```
D:\PSCA\Lec12>node 12-01
```

```
Соединение с БД установлено
```

```
Количество строк: 6
```

-- faculty = ИДиП	faculty_name = Издательское дело и полиграфия
-- faculty = ИЭФ	faculty_name = Инженерно-экономический факультет
-- faculty = ЛХФ	faculty_name = Лесохозяйственный факультет
-- faculty = ТОВ	faculty_name = Технология органических веществ
-- faculty = ТТЛП	faculty_name = Технология и техника лесной промышленности
-- faculty = ХТиТ	faculty_name = Химическая технология и техника

```
Количество строк: 15
```

-- faculty = ИДиП	pulpit = ИСиТ	pulpit_name = Информационный систем и технологий
-- faculty = ИДиП	pulpit = ПОиСОИ	pulpit_name = Полиграфического оборудования и систе

```
Количество строк: 29
```

-- teacher = ?	teacher_name = Неизвестный	pulpit = ИСиТ
-- teacher = АКНВЧ	teacher_name = Акунович Станислав Иванович	pulpit = ИСиТ
-- teacher = БРКВЧ	teacher_name = Бракович Андрей Игорьевич	pulpit = ИСиТ

config

- **user** – имя пользователя для аутентификации.
- **password** – пароль, используемый для аутентификации.
- **server** – сервер, к которому необходимо подключиться («localhost\instance» для именованного экземпляра).
- **port** – порт для подключения (по умолчанию: 1433).
- **domain** – домен для подключения.
- **database** – база данных для подключения.
- **connectionTimeout** – таймаут соединения в мс (по умолчанию: 15000).
- **requestTimeout** – таймаут запроса в мс (по умолчанию: 15000).
- **stream** – потоковые наборы записей/строк вместо возврата их всех сразу в качестве аргумента callback (по умолчанию: false). Также можно включить потоковую передачу для каждого запроса независимо (request.stream = true).
- **parseJSON** – преобразовывать наборы записей JSON в объекты JS (по умолчанию: false).
- **pool.max/min** – максимальное/минимальное количество соединений, которые могут быть в пуле (по умолчанию: min=0, max=10).
- **pool.idleTimeoutMillis** – количество миллисекунд до закрытия неиспользуемого соединения (по умолчанию: 30000).

Дополнительные опции для драйвера tedious

- **beforeConnect(conn)** – ф-ция, которая вызывается перед открытием соединения. Параметр conn — это настроенное последнее соединение.
- **options.instanceName** – имя экземпляра для подключения. На сервере базы данных должна быть запущена служба браузера SQL Server, а порт UDP 1434 на сервере базы данных должен быть доступен.
- **options.useUTC** – логическое значение, определяющее, использовать ли время UTC для значений без смещения часового пояса (по умолчанию: true).
- **options.encrypt** – логическое значение, определяющее, будет ли соединение зашифровано (по умолчанию: true).
- **options.tdsVersion** – используемая версия TDS.
- **options.appName** – имя приложения, используемое для ведения журнала SQL-сервера.
- **options.abortTransactionOnError** – логическое значение, определяющее, следует ли автоматически откатывать транзакцию, если во время выполнения данной транзакции возникает какая-либо ошибка.

Потоковая передача

```
const sql = require('mssql');

let config = { user: 'student', password: 'fitfit', server: '172.16.193.223', database: 'NodeJSTest' };

sql.connect(config, err => {

    if(err) console.log('Ошибка соединения с БД:', err.code);
    else{ console.log('Соединение с БД установлено');

        const request = new sql.Request();

        request.query('select auditorium, auditorium_name, auditorium_capacity, auditorium_type from AUDITORIUM');

        request.stream = true

        request.on('recordset', columns => { // информация о столбцах recordset
            let str = '';
            for (key in columns) {str += ` ${key} = ${JSON.stringify(columns[key])} \n`; }
            console.log(str);
        })
        .on('row', row => {
            let str = '--';
            for (key in row) {str += ` ${key} = ${row[key]} `;}
            console.log(str);
        })
        .on('error', err => {console.log(err); })
        .on('done', result => { console.log('done: количество строк: ',result.rowsAffected[0]);})

    })

})
```

При работе с большим количеством строк следует использовать **потоковую передачу**. После её включения необходимо будет прослушивать события для получения данных:

- **recordset** (columns) генерируется, когда метаданные для нового набора записей распаршены.
- **row** (row) – генерируется, когда разобрана новая строка.
- **rowsAffected** (rowCount) генерируется для каждого INSERT, UPDATE, DELETE запроса
- **done** (returnValue) генерируется, когда запрос завершен.
- **error** (err) генерируется при ошибке.

D:\PSCA\Lec12>node 12-02

Соединение с БД установлено

```
auditorium = {"index":0,"name":"auditorium","length":10,"nullable":false,"caseSensitive":false,"identity":false,"readOnly":false}
auditorium_name = {"index":1,"name":"auditorium_name","length":200,"nullable":true,"caseSensitive":false,"identity":false,"readOnly":false}
auditorium_capacity = {"index":2,"name":"auditorium_capacity","length":4,"nullable":true,"caseSensitive":false,"identity":false,"readOnly":false}
auditorium_type = {"index":3,"name":"auditorium_type","length":10,"nullable":false,"caseSensitive":false,"identity":false,"readOnly":false}
```

```
-- auditorium = ?      auditorium_name = ??? auditorium_capacity = 90 auditorium_type = ЛК
-- auditorium = 026-4  auditorium_name = 026-4 auditorium_capacity = 90 auditorium_type = ЛК
-- auditorium = 103-4  auditorium_name = 103-4 auditorium_capacity = 90 auditorium_type = ЛК
-- auditorium = 105-4  auditorium_name = 105-4 auditorium_capacity = 90 auditorium_type = ЛК
```

done: количество строк: 22

Типы данных

Есть 32 типа данных, соответствуют они одноименным типам данных в MS SQL Server:

- **числовые** типы данных;
- типы данных, представляющие **дату и время**;
- **строковые** типы данных;
- **бинарные** типы данных;
- **пространственные** типы данных;
- остальные типы данных.

Некоторые из типов являются устаревшими (image => varbinary, text, ntext => varchar, nvarchar)

ЧИСЛОВЫЕ

```
sql.Bit  
sql.BigInt  
sql.Decimal ([precision], [scale])  
sql.Float  
sql.Int  
sql.Money  
sql.Numeric ([precision], [scale])  
sql.SmallInt  
sql.SmallMoney  
sql.Real  
sql.TinyInt
```

СТРОКОВЫЕ

```
sql.Char ([length])  
sql.NChar ([length])  
sql.Text  
sql.NText  
sql.VarChar ([length])  
sql.NVarChar ([length])  
sql.Xml
```

ДАТА И ВРЕМЯ

```
sql.Time ([scale])  
sql.Date  
sql.DateTime  
sql.DateTime2 ([scale])  
sql.DateTimeOffset ([scale])  
sql.SmallDateTime
```

Бинарные

```
sql.Binary  
sql.VarBinary ([length])  
sql.Image
```

ПРОСТРАНСТВЕННЫЕ

```
sql.UDT  
sql.Geography  
sql.Geometry
```

ОСТАЛЬНЫЕ

```
sql.UniqueIdentifier  
sql.Variant
```

Динамический запрос =

запрос, позволяющий **оперативно формировать** тот или иной **SQL-запрос в зависимости от особых требований**, возникших в конкретной ситуации. После настройки SQL-запроса в соответствии с потребностями пользователя он направляется серверу БД для проверки на наличие синтаксических ошибок и необходимых для его выполнения привилегий, после чего происходит его компиляция и выполнение.

Подготовленный запрос =

запрос, который будет **использоваться более одного раза**. Подготовка запросов позволяет серверу и драйверу сократить объем обработки и сетевых данных, необходимых для выполнения запроса. Для подготовленных запросов сервер **один раз анализирует его** (строит план выполнения запроса и находит оптимальный), а затем он кэшируется на время существования приложения.

Подготовленный динамический select

```
const sql = require('mssql'); // https://www.npmjs.com/package/mssql#callbacks

let config = { user: 'student', password: 'fitfit', server: '172.16.193.223', database: 'NodeJSTest' };

let dbreq04 = (mincap, maxcap, cb) => {
  const ps = new sql.PreparedStatement();
  ps.input('mincap', sql.Int);
  ps.input('maxcap', sql.Int);
  ps.prepare('select auditorium_name, auditorium_capacity '
    + 'from AUDITORIUM WHERE auditorium_capacity between @mincap AND @maxcap', err => {
    if (err) cb(err, null);
    else ps.execute({mincap: mincap, maxcap: maxcap}, cb);
  })
}

sql.connect(config, err => {
  if (err) console.log('Ошибка соединения с БД:', err.code);
  else {
    console.log('Соединение с БД установлено');
    dbreq04(40, 70, (err, result) => processing_result(err, result));
    dbreq04(10, 40, (err, result) => processing_result(err, result));
    dbreq04(70, 100, (err, result) => processing_result(err, result));
  }
})
```

Метод **ps.input(name, type)** добавляет входной параметр в подготовленный оператор.
Метод **ps.prepare(statement, [callback])** производит подготовку оператора.
Метод **ps.execute(values, [callback])** выполняет подготовленный оператор. В параметре values можно передать объект, имена которого соответствуют именам параметров, которые были добавлены в подготовленный оператор до его подготовки.

Для создания подготовленных запросов используется класс **net.PreparedStatement**. Если аргумент pool/transaction при new net.PreparedStatement([pool or transaction]) не указан, то используется глобальный пул.

```
D:\PSCA\Lec12>node 12-03
Соединение с БД установлено
Количество строк: 3
-- auditorium_name = 236-1 auditorium_capacity = 60
-- auditorium_name = 313-1 auditorium_capacity = 60
-- auditorium_name = 324-1 auditorium_capacity = 50
Количество строк: 5
-- auditorium_name = 110-4 auditorium_capacity = 30
-- auditorium_name = 111-4 auditorium_capacity = 30
-- auditorium_name = 206-1 auditorium_capacity = 15
-- auditorium_name = 301-1 auditorium_capacity = 15
-- auditorium_name = 413-1 auditorium_capacity = 15
Количество строк: 14
-- auditorium_name = ??? auditorium_capacity = 90
-- auditorium_name = 026-4 auditorium_capacity = 90
-- auditorium_name = 103-4 auditorium_capacity = 90
-- auditorium_name = 105-4 auditorium_capacity = 90
-- auditorium_name = 107-4 auditorium_capacity = 90
-- auditorium_name = 114-4 auditorium_capacity = 90
-- auditorium_name = 132-4 auditorium_capacity = 90
-- auditorium_name = 229-4 auditorium_capacity = 90
-- auditorium_name = 304-4 auditorium_capacity = 90
-- auditorium_name = 314-4 auditorium_capacity = 90
-- auditorium_name = 320-4 auditorium_capacity = 90
-- auditorium_name = 408-2 auditorium_capacity = 90
-- auditorium_name = 423-1 auditorium_capacity = 90
-- auditorium_name = 429-4 auditorium_capacity = 90
```

Подготовленный select Потоковый режим

```
const sql = require('mssql'); // https://www.npmjs.com/package/mssql#callbacks
let config = { user:'student', password:'fitfit', server:'172.16.193.223', database:'NodeJSTest' };
let dbreq05 = (mincap, maxcap, cb)=>{
  const ps = new sql.PreparedStatement();
  ps.input('mincap', sql.Int);
  ps.input('maxcap', sql.Int);
  ps.prepare('select auditorium_name, auditorium_capacity '
    + 'from AUDITORIUM where auditorium_capacity between @mincap AND @maxcap', err => {
    if (err) console.log('prepare:',err);
    else{
      ps.stream = true;
      let request = ps.execute({mincap: mincap, maxcap: maxcap}, ()=>{ });
      request.on('recordset', columns => { // информация о столбцах recordset
        let str = ''; for (key in columns) {str += ` ${key} = ${JSON.stringify(columns[key])} \n`; }
        console.log(str);
      })
      request.on('row', row => {
        let str = '--'; for (key in row) {str += ` ${key} = ${row[key]} `; }
        console.log(str);
      })
      request.on('error', err => { console.log(err); })
      request.on('done', result => {
        console.log('done: количество строк: ',result.rowsAffected[0]);
        ps.unprepare(err => { if(err) console.log('unprepare:', err); })
        if(cb) cb();
      })
    }
  });
};

let conn = sql.connect(config, err => {
  if(err) console.log('Ошибка соединения с БД:', err.code);
  else {
    console.log('Соединение с БД установлено');
    dbreq05(40,70, ()=>{process.exit(0)});
    //???? conn.close();
  }
});
```

Метод `ps.unprepare([callback])` удаляет из кэша подготовленный оператор.

D:\PSCA\Lec12>node 12-04

Соединение с БД установлено

```
auditorium_name = {"index":0,"name":"auditorium_name","length":200,"nullable":true,"caseSensitive":false,"identity":false,"readOnly":false}
auditorium_capacity = {"index":1,"name":"auditorium_capacity","length":4,"nullable":true,"caseSensitive":false,"identity":false,"readOnly":false}
```

```
-- auditorium_name = 236-1 auditorium_capacity = 60
-- auditorium_name = 313-1 auditorium_capacity = 60
-- auditorium_name = 324-1 auditorium_capacity = 50
```

done: количество строк: 3

Подготовленный динамический insert

В конфиге помимо данных о сервере можно также указать настройки для пула соединений.

```
const sql = require('mssql'); // https://www.npmjs.com/package/mssql#callbacks

let config = { user: 'student', password: 'fitfit', server: '172.16.193.223', database: 'NodeJSTest',
  pool: {max: 10, min: 0, sof
    :TimeoutMillis: 10000} };

let insAuditorium = (a, an, ac, at, _cb)=>{
  const cb = _cb?_cb:(err,result)=>{console.log('default cb')};
  const ps = new sql.PreparedStatement();
  ps.input('a',sql.NChar(10)); ps.input('an',sql.NVarChar); ps.input('ac',sql.Int);ps.input('at',sql.NChar(10));
  ps.prepare('insert AUDITORIUM(auditorium, auditorium_name, auditorium_capacity, auditorium_type)
  + 'values(@a, @an, @ac, @at)', err => {
    if (err) cb(err, null);
    else ps.execute({a:a, an:an, ac:ac, at:at}, (err, result)=>{
      if (err) cb(err,null);
      else cb(null, result);
    });
  });
};

let insAuditorium_type = (at, atn, _cb)=>{ ...
};

sql.connect(config, err => {
  if(err) console.log('Ошибка соединения с БД:', err.code);
  else {
    console.log('Соединение с БД установлено');
    insAuditorium('322-1','322-1 LeverX',24,'ЛБ-СК', (err, result)=>{ });
    insAuditorium('324-1','324-1 iTechArt',16,'ЛБ-СК', (err, result)=>{ });
  }
});
```

Метод `rq.input(name, [type], value)` добавляет в запрос входной параметр. Если опустить параметр `type`, то **модуль автоматически решит, какой тип данных SQL следует использовать** на основе типа данных JS:

String => sql.NVarChar

Number => sql.Int

Boolean => sql.Bit

Date => sql.DateTime

Buffer => sql.VarBinary

Тип данных по умолчанию для неизвестного объекта – `sql.NVarChar`.

Подготовленный динамический delete

```
const sql = require('mssql'); // https://www.npmjs.com/package/mssql#callbacks

let config = { user: 'student', password: 'fitfit', server: '172.16.193.223', database: 'NodeJSTest',
  pool: { max: 10, min: 0, idleTimeoutMillis: 10000 } };

let delAuditorium = (a, _cb) => {
  const cb = _cb ? _cb : (err, result) => { console.log('default cb') };
  const ps = new sql.PreparedStatement();
  ps.input('a', sql.NChar(10));
  ps.prepare('delete AUDITORIUM where auditorium = @a', err => {
    if (err) cb(err, null);
    else ps.execute({ a: a }, (err, result) => {
      if (err) cb(err, null);
      else cb(null, result);
    });
  });
};

h = (err, result) => {
  if (err) console.log('ошибка: ', err);
  else console.log('успешное завершение: ', result.rowsAffected[0]);
}

sql.connect(config, err => {
  if (err) console.log('Ошибка соединения с БД:', err.code);
  else {
    console.log('Соединение с БД установлено');
    delAuditorium('322-1', h);
    delAuditorium('324-1', h);
  }
});
```

Подготовленный динамический update

```
const sql = require('mssql'); // https://www.npmjs.com/package/mssql#callbacks

let config = { user: 'student', password: 'fitfit', server: '172.16.193.223', database: 'NodeJSTest',
  pool: {max: 10, min: 0, idleTimeoutMillis: 10000} };

let updAuditorium = (a, an, ac, at, _cb)=>{
  const cb = _cb?_cb:(err,result)=>{console.log('default cb')};
  const ps = new sql.PreparedStatement();
  ps.input('a',sql.NChar(10)); ps.input('an',sql.NVarChar);ps.input('ac',sql.Int); ps.input('at',sql.NChar(10));
  ps.prepare('update AUDITORIUM set auditorium_name=@an,auditorium_capacity=@ac,'
    + 'auditorium_type=@at where auditorium=@a', err => {
    if (err) cb(err, null);
    else ps.execute({a:a, an:an, ac:ac, at:at}, (err, result)=>{
      if (err) cb(err,null);
      else cb(null, result);
    });
  });
};

h = (err, result)=>{
  if(err) console.log('ошибка: ', err);
  else console.log('успешное завершение: ', result.rowsAffected[0]);
  process.exit(0);
}

sql.connect(config, err => {
  if(err) console.log('Ошибка соединения с БД:', err.code);
  else {
    console.log('Соединение с БД установлено');
    updAuditorium('313-1','313-1 iTechArt', 80, 'ЛК-К', h);
  }
})
```

Хранимая процедура для примера

```
drop procedure GET_PULPITS;  
go  
  
create procedure GET_PULPITS @f nvarchar(10)  
as begin  
    print 'параметры: ' + @f;  
    select pulpit, pulpit_name, faculty from PULPIT  
    where faculty = @f;  
end
```

Вызов процедуры

```
const sql = require('mssql'); // https://www.npmjs.com/package/mssql#callbacks

let config = { user: 'student', password: 'fitfit', server: '172.16.193.223', database: 'NodeJSTest',
  pool: { max: 10, min: 0, softIdleTimeoutMillis: 5000, idleTimeoutMillis: 10000 } };

let callGET_PULPITS = (f, _cb) => {
  const cb = _cb?_cb:(err,result) => { console.log('default cb') };
  const rq = new sql.Request()
  rq.input('f', sql.NChar(10), f);
  rq.execute('GET_PULPITS', (err, result) => { processing_result(err, result); cb(err, result); });
  rq.on('info', message => { console.log('info:', message); });
};

let processing_result = (err, result) => {
  if (err) console.log('processing_result error:', err);
  else {
    console.log('Количество строк: ', result.rowsAffected[0]);
    for (let i = 0; i < result.rowsAffected[0]; i++) {
      let str = '--';
      for (key in result.recordset[i]) { str += ` ${key} = ${result.recordset[i][key]} `; }
      console.log(str);
    }
  }
}

sql.connect(config, err => {
  if (err) console.log('Ошибка соединения с БД:', err.code);
  else {
    console.log('Соединение с БД установлено');
    callGET_PULPITS('TOB', h);
  }
});

h = (err, result) => {
  if (err) console.log('ошибка: ', err);
  else console.log('успешное завершение: ', result.rowsAffected[0]);
  process.exit(0);
}
```

Третьим параметром передается значение параметра

Метод `rq.execute (procedure, [callback])` используется для вызова хранимой процедуры.

Событие `info` (message) генерируется при поступлении информационного сообщения.

Вызов процедуры осуществляется с помощью экземпляра класса `sql.Request`.

С помощью методов `.input()` и `.output()` регистрируются входные и выходные параметры.

```
D:\PSCA\Lec12>node 12-08
Соединение с БД установлено
info: { message: 'параметры: TOB',
  number: 0,
  state: 1,
  class: 0,
  lineNumber: 3,
  serverName: 'ISIT-SQL-223',
  procName: 'GET_PULPITS' }
Количество строк: 2
-- pulpit = 0X          pulpit_name = Органической химии faculty = TOB
-- pulpit = ТНХСиППМ   pulpit_name = Технологии нефтехимического синтеза и переработки полимерных материалов faculty = TOB
успешное завершение: 2
```

Транзакция =

это механизм базы данных, позволяющий таким образом **объединять несколько операторов**, изменяющих базу данных, чтобы при выполнении этой совокупности операторов **они или все выполнились или все не выполнились**.

Фиксация – подтверждение успешного завершения транзакции.

Откат – отмена всех внесенных изменений в результате неуспешного завершения транзакции.

Транзакция

Для создания транзакций используется **класс `sql.Transaction`**. Он гарантирует, что все запросы выполняются в одном соединении.

После вызова **метода `begin`** (`[isolationLevel]`, `[callback]`) из пула будет получено одно соединение, которое будет использовано для запросов в рамках транзакции.

После вызова **`commit`** или **`rollback`** соединение возвращается обратно в пул соединений.

Транзакцию также можно создать с помощью `const transaction = pool.transaction()`. Запросы можно создавать с помощью `const request = transaction.request()`.

```
const sql = require('mssql');
let config = {
  user: 'sa', password: '123', server: 'DESKTOP-4KB5CA2', database: 'Univer',
  options: { enableArithAbort: true, encrypt: false }, pool: { max: 10, min: 0, idleTimeoutMillis: 10000 }
};
```

```
let dbt = (conn, values, cb) => {
  const trans = new sql.Transaction(conn);
  trans.begin(err => {
    if (err) console.log('transaction error: ', err);
    else {
      // trans.on('rollback', (aborted) => { console.log('onrollback', aborted); })
      // trans.on('commit', () => { console.log('oncommit'); })
      // trans.on('begin', () => { console.log('onbegin'); })
      const ps = new sql.PreparedStatement(trans);
      ps.input('a', sql.NChar(10)); ps.input('an', sql.NVarChar); ps.input('ac', sql.Int);
      ps.input('at', sql.NChar(10)); ps.input('atn', sql.NVarChar);

      ps.prepare('insert into AUDITORIUM_TYPE (AUDITORIUM_TYPE, AUDITORIUM_TYPENAME) values (@at, @atn);'
        + 'update AUDITORIUM set AUDITORIUM_NAME=@an, AUDITORIUM_CAPACITY=@ac, AUDITORIUM_TYPE=@at WHERE AUDITORIUM=@a;',
        err => {
          if (err) cb(err, null);
          else ps.execute(values, (err, result) => {
            if (err) {
              result = err.message;
              ps.unprepare(err => {
                if (err) cb(err, null);
                else trans.rollback(err => { if (err) cb(err, null); else cb(null, result); });
            }
            else
              ps.unprepare(err => {
                if (err) cb(err, null);
                else trans.commit(err => { if (err) cb(err, null); else cb(null, result); });
            }
          });
        });
    }
  });
}
```

Можно создавать подготовленные запросы в транзакциях, но когда они больше не будут использоваться, надо вызвать `unprepare`.

Переданный аргумент определяет **причину прерывания транзакции** (пользователем или из-за ошибки).

```
let conn = sql.connect(config, err => {
  if(err) console.log('Ошибка соединения с БД:', err.code);
  else {
    console.log('Соединение с БД установлено');
    dbt(conn, {a: '313-1', an: '313-1 upd2', ac: 88, at: 'ЛК-К-Б', atn: 'ЛК-К-Б Бренд'}, (err, result) => {
      if (err) console.log('dbt error:', err);
      else console.log('dbt result:', result);
      process.exit(0)
    });
  }
});
```

Пул соединений с = базой данных

несколько предварительно и постоянно открытых соединений с сервером СУБД, которые используют приложения.

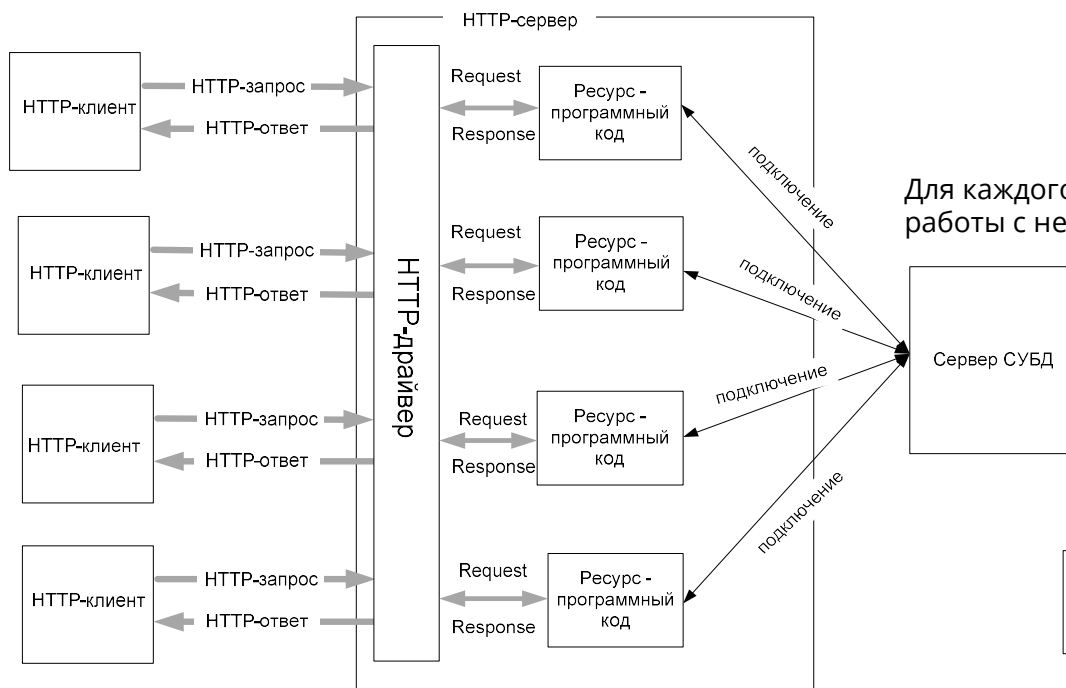
Если все подключения пула заняты, запрос на соединение ставится в очередь.

Применение пула позволяет увеличить производительность за счет отсутствия процесса подключения к серверу.

Альтернатива: держать постоянное открытое соединение для каждого подключения (в общем случае для web-приложения неприемлемо) или открывать/закрывать соединение при каждом запросе (большие накладные расходы на установку соединения).

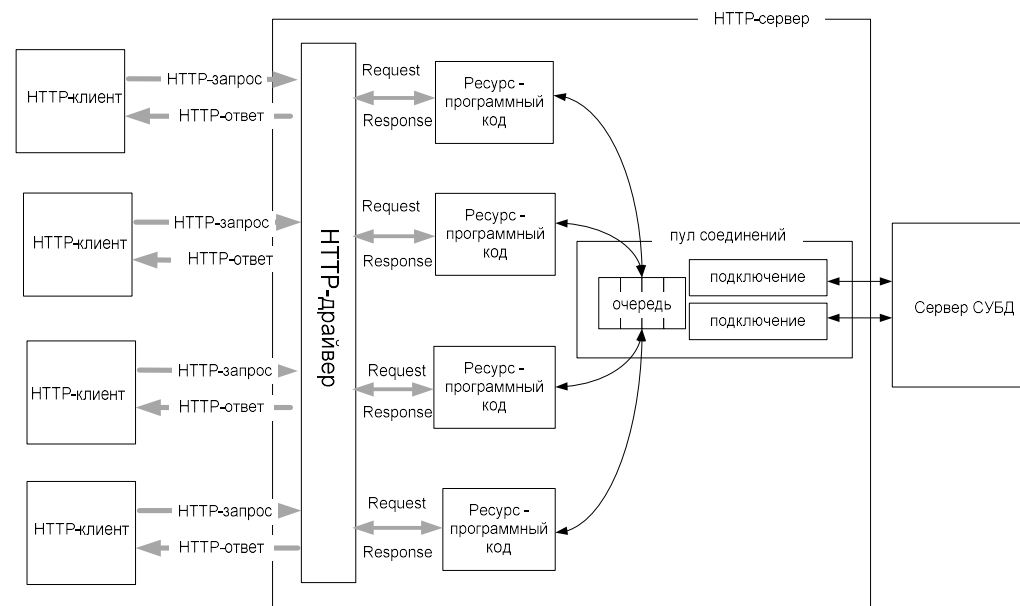
Пул соединений

Для каждого клиента создается отдельное подключение к БД, а после завершения работы с ней оно закрывается.



Вместо того, чтобы закрывать соединение, можно **сохранять его активным, помещая в пул**, после чего можно взять его из пула в следующий раз.

Но если все соединения сохранять открытыми, то в какой-то момент пул будет переполнен, поэтому **для пула нужно установить максимальное количество подключений**. В случае, если к БД **пытаются достучаться много клиентов**, на которых не хватает открытых соединений из пула, то они **помещаются в очередь и ожидают**, пока какой-либо клиент завершит свою работу с БД и вернет соединение в пул.



Пул соединений

```
const sql = require('mssql'); // https://www.npmjs.com/package/mssql#callbacks

let config = { user: 'student', password: 'fitfit', server: '172.16.193.223', database: 'NodeJSTest',
  pool: {max: 10, min: 0, softIdleTimeoutMillis: 5000, idleTimeoutMillis: 10000} };

let dbreq01 = (pool) => {pool.request().query('select faculty, faculty_name from FACULTY', processing_result)};
let dbreq02 = (pool) => {pool.request().query('select faculty, pulpit, pulpit_name from PULPIT order by faculty',
  processing_result)};
let dbreq03 = (pool) => {pool.request().query('select teacher, teacher_name, pulpit from TEACHER order by pulpit',
  processing_result)};

const pool = new sql.ConnectionPool(config, err => {
  if(err) console.log('Ошибка соединения с БД:', err.code);
  else{ console.log('Соединение с БД установлено');
    dbreq01(pool);
    dbreq02(pool);
    dbreq03(pool);
  }
});

let processing_result = (err, result) => {
  if (err) console.log('processing_result error:', err);
  else {
    console.log('Количество строк: ', result.rowsAffected[0]);
    for (let i = 0; i < result.rowsAffected[0]; i++){
      let str = '--';
      for (key in result.recordset[i]) {
        str += ` ${key} = ${result.recordset[i][key]} `;
      }
      console.log(str);
    }
  }
}
```

Внутри каждый экземпляр **класса `sql.ConnectionPool`** представляет собой отдельный пул соединений. После создания нового Request/Transaction/PreparedStatement новое соединение будет получено из пула и зарезервировано для желаемого действия. После завершения действия соединение возвращается в пул.

Пул соединений

```
const sql = require('mssql'); // https://www.npmjs.com/package/mssql#callbacks

let config = { user: 'student', password: 'fitfit', server: '172.16.193.223', database: 'NodeJSTest',
  pool: {max: 10, min: 0, softIdleTimeoutMillis: 5000, idleTimeoutMillis: 10000} };

let dbreq11 = (pool, f) => {pool.request()
  .input('f', sql.NChar, f)
  .query('select pulpit_name from PULPIT where faculty=@f', processing_result)};

const pool = new sql.ConnectionPool(config, err => {
  if(err) console.log('Ошибка соединения с БД:', err.code);
  else{ console.log('Соединение с БД установлено');
    dbreq11(pool, 'ИДИП');
  }
});

let processing_result = (err, result) => {
  if (err) console.log('processing_result error:', err);
  else {
    console.log('Количество строк: ', result.rowsAffected[0]);
    for (let i = 0; i < result.rowsAffected[0]; i++){
      let str = '--';
      for (key in result.recordset[i]) {
        str += ` ${key} = ${result.recordset[i][key]} `;
      }
      console.log(str);
    }
  }
}
```

Метод `pool.request()` создает новый экземпляр класса `sql.Request`.

Пул соединений

```
const sql = require('mssql'); // https://www.npmjs.com/package/mssql#callbacks

let config = { user: 'student', password: 'fitfit', server: '172.16.193.223', database: 'NodeJSTest',
  pool: { max: 10, min: 0, softIdleTimeoutMillis: 5000, idleTimeoutMillis: 10000 } };

let dbreq12 = (pool, f, fn) => { pool.request()
  .input('f', sql.NChar, f)
  .input('fn', sql.NVarChar, fn)
  .query('insert FACULTY(faculty, faculty_name) values(@f, @fn)', processing_result)};

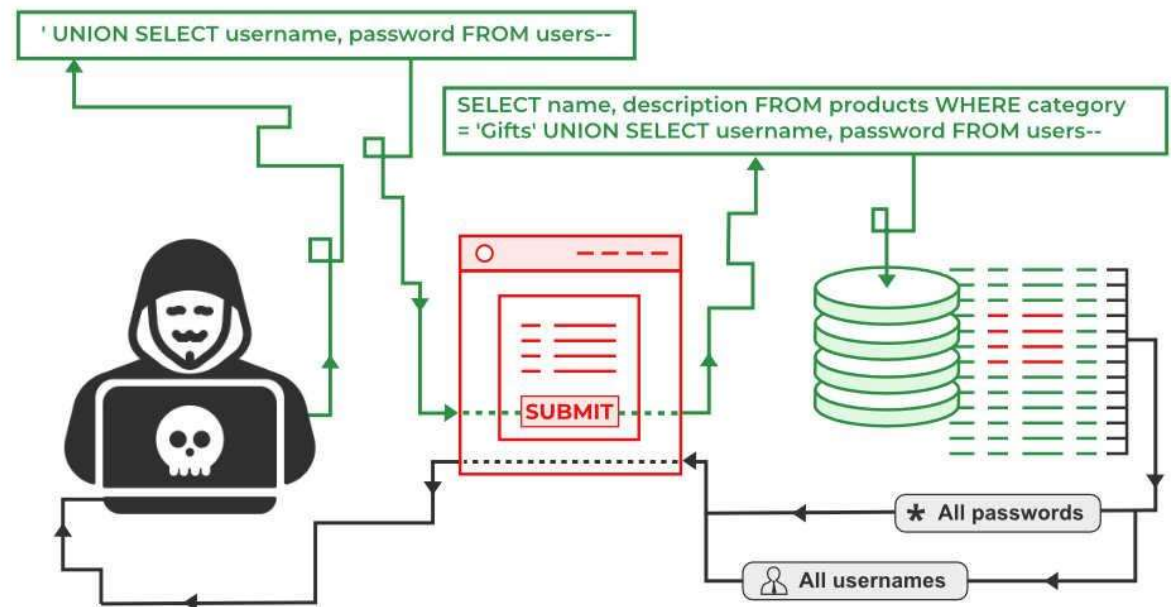
const pool = new sql.ConnectionPool(config, err => {
  if(err) console.log('Ошибка соединения с БД:', err.code);
  else{ console.log('Соединение с БД установлено');
    dbreq12(pool, 'ИТ', 'Информационные технологии');
  }
});

let processing_result = (err, result) => {
  if (err) console.log('processing_result error:', err);
  else console.log('Количество строк: ', result.rowsAffected[0]);
}
```

SQL-инъекция =

Это атака, позволяющая злоумышленнику проводить различные несанкционированные действия над БД.

Атака происходит путем размещения вредоносного кода в операторах SQL посредством ввода в поле на веб-странице.



SQL-инъекции

```
const sql = require('mssql');
let config = {
  user: 'sa', password: '123', server: 'DESKTOP-4KB5CA2', database: 'Univer',
  options: { enableArithAbort: true, encrypt: false }, pool: { max: 10, min: 0, idleTimeoutMillis: 10000 }
};

let conn = sql.connect(config)
  .then((pool) => {
    console.log('Соединение с БД установлено');

    let f = '\TOB\'; select * from Teacher';
    return new sql.Request()
      .query('select * from Pulpit WHERE faculty = ' + f);

  }).then((result) => {
    console.dir(result.recordsets);
  }).catch((err) => console.log('Ошибка: ', err.code, err.message))
```

PS D:\NodeJS\samples\свр_11> node 11-05

Соединение с БД установлено

```
[
  {
    { pulpit: 'OX', pulpit_name: 'Органической химии', faculty: 'TOB' },
    {
      pulpit: 'ТНХСИПМ',
      pulpit_name: 'Технологии нефтехимического синтеза и переработки полимерных материалов',
      faculty: 'TOB'
    }
  },
  {
    { teacher: '?', teacher_name: 'Неизвестный', pulpit: 'ИСИТ' },
    {
      teacher: 'АКНВЧ',
      teacher_name: 'Акунович Станислав Иванович',
      pulpit: 'ИСИТ'
    },
    {
      teacher: 'БЗБРДВ',
      teacher_name: 'Безбородов Владимир Степанович',
      pulpit: 'OX'
    }
  }
]
```


Встроенная защита от SQL-инъекций

```
const sql = require('mssql');
let config = {
  user: 'sa', password: '123', server: 'DESKTOP-4KB5CA2', database: 'Univer',
  options: { enableArithAbort: true, encrypt: false }, pool: { max: 10, min: 0, idleTimeoutMillis: 10000 }
};

let conn = sql.connect(config)
  .then((pool) => {
    console.log('Соединение с БД установлено');

    return new sql.Request()
      .input('f', sql.NVarChar, '\TOB\'; select * from Teacher;')
      .query('select * from Pulpit WHERE faculty = @f')

  }).then((result) => {
    console.dir(result.recordsets);
  }).catch((err) => console.log('Ошибка: ', err.code, err.message))
```

Пакет mssql имеет встроенную защиту от SQL-инъекций. **Всегда используйте параметры и помеченные литералы шаблонов для передачи обработанных значений в запросы.**

```
PS D:\NodeJS\samples\cwp_11> node 11-05
Соединение с БД установлено
[ [] ]
PS D:\NodeJS\samples\cwp_11>
```



Если не передавать коллбэк, то методы **будут возвращать Promise** и, понятное дело, можно обработать результат с помощью `then/catch/finally` или `async/await`.

Вызов процедуры, конструкция async/await

```
create procedure COUNT_PULPITS @f nvarchar(10), @c int output
as begin
    SELECT @c = COUNT(*) FROM Pulpit where faculty = @f;
end
```

```
const sql = require('mssql')
let config = {
  user: 'sa', password: '123', server: 'DESKTOP-4KB5CA2', database: 'Univer',
  options: { enableArithAbort: true, encrypt: false }, pool: { max: 10, min: 0, idleTimeoutMillis: 10000 }
};

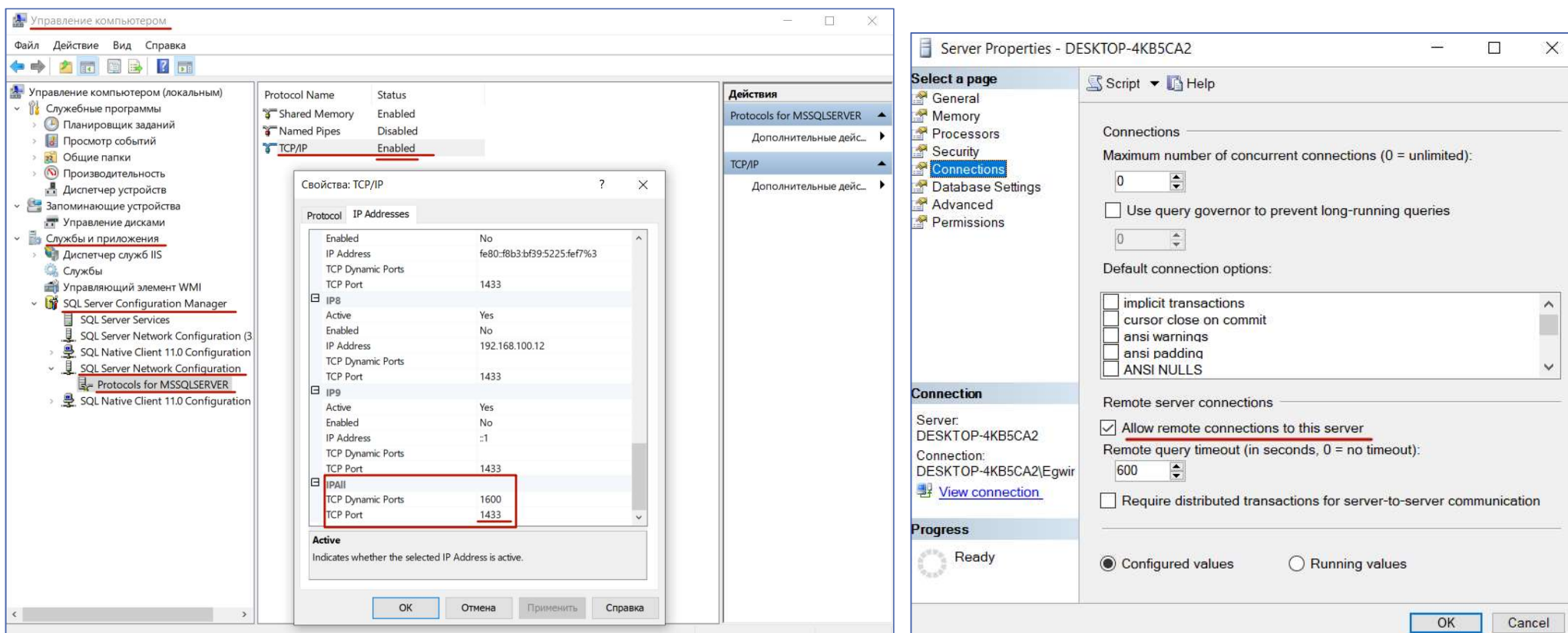
(async function () {
  try {
    let pool = await sql.connect(config)
    let result1 = await pool.request()
      .input('f', sql.NVarChar, 'ЛХФ')
      .query('select * from Pulpit WHERE faculty = @f')
    console.log(result1.recordset);

    // Stored procedure
    let result2 = await pool.request()
      .input('f', sql.NVarChar, 'ЛХФ')
      .output('c', sql.Int)
      .execute('COUNT_PULPITS')
    console.log('Количество строк: ', result2.output.c);
  } catch (err) {
    console.log('Ошибка: ', err.code, err.message)
  }
})();
```

Метод `rq.output (name, type, [value])`
позволяет добавить в запрос
выходной параметр.

```
PS D:\NodeJS\samples\cwp_11> node 11-06
[
  { pulpit: 'ЛВ', pulpit_name: 'Лесоводства', faculty: 'ЛХФ' },
  {
    pulpit: 'ЛЗидВ',
    pulpit_name: 'Лесозащиты и древесиноведения',
    faculty: 'ЛХФ'
  },
  {
    pulpit: 'ЛПисПС',
    pulpit_name: 'Ландшафтного проектирования и садово-паркового строительства',
    faculty: 'ЛХФ'
  },
  { pulpit: 'ЛУ', pulpit_name: 'Лесоустройства', faculty: 'ЛХФ' },
  { pulpit: 'ОВ', pulpit_name: 'Охотоведения', faculty: 'ЛХФ' }
]
Количество строк: 5
```

Если есть проблемы с подключением



Пакет mssql работает с SQL Server по протоколу TCP. Поэтому необходимо убедиться, что для экземпляра SQL Server **включен протокол TCP** и **открыты** соответствующие статические **порты**, также необходимо **разрешить удаленные подключения** с этим сервером.