



MYSQL WITH NODEJS

Andrew Sheehan

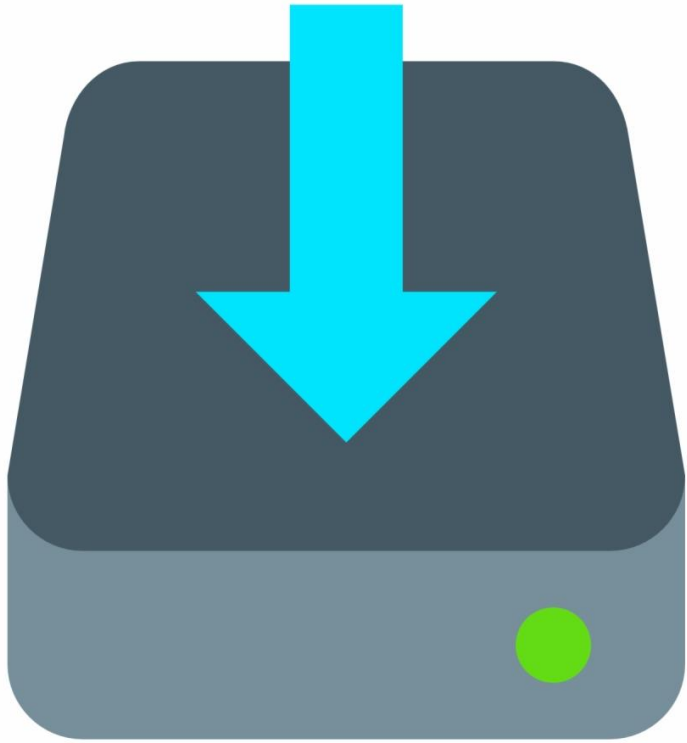
Boston University
Metropolitan College

WHAT IS IT?

A pure Node.js JavaScript client
that implements the MySQL
protocol



INSTALLATION WITH NPM



```
npm install --save mysql
```

LICENSING MIT

The MIT License (MIT)

Copyright (c) 2015-present Dan Abramov

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.



open source
initiative
Approved License[®]

CONNECTING STANDARD APPROACH

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

const connection = mysql.createConnection({
  host      : 'localhost',
  user      : 'username',
  password  : 'password',
  database  : 'database'
});
connection.end();
```

CONNECTING WITH DEBUG ON

```
const mysql = require('mysql');  
  
const connection = mysql.createConnection({  
  debug      : true,  
  user       : 'username',  
  password   : 'account password',  
  database   : 'database'  
});
```

CONNECTING WITH DEFAULTS

‘host’ – *Not indicated?*

Default: localhost

```
const connection = mysql.createConnection({  
  user      : "username",  
  password  : "account_password",  
  database  : "database_name"  
});
```

CONNECTING OLD APPROACH - STRING

```
const conn_str =  
    "mysql://unm:pswd@host/db?debug=true";
```

```
const connection =  
    mysql.createConnection(conn_str);
```


CHECKING SERVER CONNECTED PING()

```
connection.ping(function (err) {  
  if (err) throw err;  
  console.log('Server responded to ping');  
})
```

CONNECTING THINGS TO KNOW

Every time you open a connection, *it will be queued and executed in sequential order.*

Closing a connection is done by calling `end()`

It will make sure all remaining queries are executed before sending a quit packet to your MySQL database.

USING CONNECTION POOLS

```
const pool = mysql.createPool({config});

pool.query('select c From T', (error, results, fields) => {
  if (error) {
    throw error;
  }

  // Work with the results/data ... It will release the
  // connection after the query is finished. (using this method)
});
```

QUERYING NO PARAMETERS TO PASS

```
const connection = mysql.createConnection({config});

connection.query('select c From T', (error, results, fields) => {
  if (error) {
    throw error;
  }

  // the results/data ...
});

connection.end();
```

QUERYING WITH PARAMETERS

```
const connection = mysql.createConnection({config});  
connection.query('select c From T where a=?', [2], (error, results, fields) => {  
  if (error) {  
    throw error;  
  }  
  // Work with the results/data here...  
});  
connection.end( err -> { /* handle this..*/ } );
```

QUERYING MORE CONFIGURATION OPTIONS

```
const connection = mysql.createConnection({config});  
connection.query({  
  sql: 'select c From T where a=?',  
  timeout: 5000,  
  values: [2]  
}, (error, results, fields) => {  
  // The error, results/data ...  
});  
connection.end();
```

QUERYING GETTING THE RESULT COUNT

```
const connection = mysql.createConnection({config});
```

```
connection.query('select c From T', (error, results, fields) => {  
  if (error) throw error;  
  console.log(results.affectedRows);  
});
```

QUERYING GETTING LAST INSERTED ID

```
const connection = mysql.createConnection({config});
```

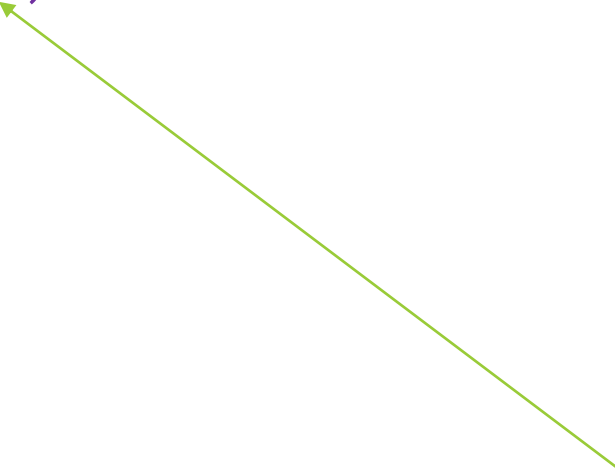
```
connection.query('insert into T ....', (error, results, fields) => {  
  if (error) throw error;  
  console.log(results.insertId);  
});
```


QUERYING GETTING LAST INSERTED ID

```
const config = {multipleStatements: true};  
const connection = mysql.createConnection({config});  
connection.query('select ...; select ..', (error, results, fields) => {  
  if (error) throw error;  
  console.log( results[0]);  
  console.log( results[1]);  
});
```

QUERYING ESCAPING PARAMETER VALUES

```
const connection = mysql.createConnection({config});  
connection.query({  
  sql: 'select c From T where a= ?',  
  timeout: 5000,  
  values: [2]  
}, (error, results, fields) => {  
  // The error, results/data ...  
});  
connection.end();
```



Use the ? It will
automatically escape
your strings

QUERYING

ESCAPING DEFAULTS

- Numbers are left untouched
- Booleans are converted to `true` / `false`
- Date objects are converted to `'YYYY-mm-dd HH:ii:ss'` strings
- Buffers are converted to hex strings, e.g. `X'0fa5'`
- Strings are safely escaped
- Arrays are turned into list, e.g. `['a', 'b']` turns into `'a', 'b'`
- Nested arrays are turned into grouped lists (for bulk inserts), e.g. `[['a', 'b'], ['c', 'd']]` turns into `('a', 'b'), ('c', 'd')`
- Objects that have a `toSqlString` method will have `.toSqlString()` called and the returned value is used as the raw SQL.
- Objects are turned into `key = 'val'` pairs for each enumerable property on the object. If the property's value is a function, it is skipped; if the property's value is an object, `toString()` is called on it and the returned value is used.
- `undefined` / `null` are converted to `NULL`
- `NaN` / `Infinity` are left as-is. MySQL does not support these, and trying to insert them as values will trigger MySQL errors until they implement support.

AVOID SQL INJECTION

Always check data that coming into your application.

Assume outside data (any) is bad and needs to be verified!!!

RUNNING TRANSACTIONS

```
connection.beginTransaction(function(err) {
  if (err) { throw err; }
  connection.query('INSERT INTO posts SET title=?', title, function (error, results, fields) {
    if (error) {
      return connection.rollback(function() {
        throw error;
      });
    }

    var log = 'Post ' + results.insertId + ' added';

    connection.query('INSERT INTO log SET data=?', log, function (error, results, fields) {
      if (error) {
        return connection.rollback(function() {
          throw error;
        });
      }
      connection.commit(function(err) {
        if (err) {
          return connection.rollback(function() {
            throw err;
          });
        }
        console.log('success!');
      });
    });
  });
});
```