Walk-through 007

INTERACT WITH A MYSQL DATABASE IN NODE.JS

Objectives:

- Learn the essentials about connecting and interacting with a mysql

Steps:

1. Setup a new project, package.json, app.js lib folder and the usual…
2. Install and --save **colors** , **mysql** and **async** packages with npm
3. copy the **MiniLogger** module to the lib folder
4. Create a **config.json** file in the root directory which will have the mysql connection details.
5. Open **app.js** and require all the modules including the config file like so:

```
var colors  = require('colors');
var d       = require('./lib/MiniLogger');
var config  = require('./config.json');
var mysql   = require('mysql');
```

6. Next we'll connect to our mysql service using the credentials we wrote in the config:

```
var connection = mysql.createConnection({
    host     : config[config.env].db.host,
    user     : config[config.env].db.user,
    password : config[config.env].db.password,
    database : config[config.env].db.database
});
```

7. In the first phase of this walkthrough, we'll simply create three functions for three different database operations – creating a "users" table, populating it with a record and deleting it.

   write the following function for creating the table:

```
function create_table(){
    var sql = "CREATE TABLE `users` (" +
              "     `user_id` int(15) NOT NULL AUTO_INCREMENT,"+
              "     `first_name` varchar(100) DEFAULT NULL,"+
              "     `last_name` varchar(100) DEFAULT NULL,"+
              "     `email` varchar(150) DEFAULT NULL,"+
              "     `created` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,"+
              "      PRIMARY KEY (`user_id`)"+
              ") ENGINE=InnoDB DEFAULT CHARSET=utf8";
    connection.query(sql, function(err, rows, fields) {
                            connection.end(); // always put connection back in pool after query results
                            if (err) {
                                d('ERROR creating the users table:'.bold.red,err.message);
                                return;
                            }
                            d('Table was created successfully!!!'.green);
                        });
}
```

8. Continue with a second function for inserting a record:

```
function create_some_users(){
    var post  = {first_name: 'Ajar', last_name: 'Bahamonde', email: 'ajar@casaversa.com'};
    var sql = "INSERT INTO users SET ?"
    connection.query(sql,post, function(err, rows, fields) {
        connection.end(); // always put connection back in pool after query results
        if (err) {
            d('ERROR inserting a record:'.bold.red,err.message);
            return;
        }
        d('users was populated with some records'.green);
    });
}
```

Notice the way you can escape values in your sql statements to avoid sql injections as well as a convenient way to use js objects in your sql code.

9. Next write a function to delete the table so you can experiment again with other db operations:

```
function delete_table(){
    connection.query('DROP TABLE IF EXISTS `users`', function(err, rows, fields) {
        connection.end(); // always put connection back in pool after query results
        if (err) {
            d('ERROR Deleting the users table:'.bold.red,err.message);
            return;
        }
        d('users table was deleted!!!'.blue);
    });
}
```

10. To run and test the code up to this point, you will be calling a different function each time you run the app…

```
create_table();
//create_some_users();
//delete_table();
```

11. Save a copy of the app and modify it to use **async** library from the previous examples.

    This is to be able to run two operations one after the other: creating the “users” table and enter some records after it.

```
async.series([  function(callback){
                    create_table(callback);
                },
                function(callback){
                    create_some_users(callback);
                }],
                function(err, results) {
                    if(err){
                        d('ERROR:',err.message);
                    }else{
                        d('do something when all are done... results:'.green,results);
                        // results is now equal to: ['one', 'two']
                    }
                }
);
```

12. Using the db functions with async you will need to modify the functions to include the async logic and callbacks like so:

```
function create_table(cb){
    var sql = "CREATE TABLE `users` (" +
              "     `user_id` int(15) NOT NULL AUTO_INCREMENT,"+
              "     `first_name` varchar(100) DEFAULT NULL,"+
              "     `last_name` varchar(100) DEFAULT NULL,"+
              "     `email` varchar(150) DEFAULT NULL,"+
              "     `created` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,"+
              "      PRIMARY KEY (`user_id`)"+
              ") ENGINE=InnoDB DEFAULT CHARSET=utf8";
    connection.query(sql, function(err, rows, fields) {
            if (err) {
                d('ERROR creating the users table:'.bold.red,err.message);
                cb(err);
                return;
            }else{
                var msg = 'Table was created successfully!!!';
                d(msg.green);
                cb(null, msg);
            }
        });
}
```

13. As you will notice in the following example we modify the insertion function to also handle batch records in a single statement which is also possible with the mysql node package syntax:

```
function create_some_users(cb){
    //connection.connect();
    var post  = [['Ajar','Bahamonde','ajar@casaversa.com'],
                 ['John','Doe','john@doe.com'],
                 ['Martha', 'Lewis', 'martha@lewis.com']];

    var sql = "INSERT INTO users (first_name,last_name,email) VALUES ?"

    connection.query(sql,[post], function(err, rows, fields) {
        connection.end(); // always put connection back in pool after query results
        if (err) {
            d('ERROR inserting a record:'.bold.red,err.message);
            cb(err);
            return;
        }else{
            var msg = 'users was populated with some records'
            d(msg.green);
            cb(null,msg);
        }
    });
}
```

14. The delete table stay the same as you will want to run it separately when testing your code.

Notes:

- Visit the **mysql** package homepage on github for a full documentation https://github.com/felixge/node-mysql