Introduction to CUBRID Node.js driver – Part II

December 2012

This is the 2nd part of the **CUBRID Node.js** introduction tutorial. The first tutorial was published in October 2012 and if you didn’t read it yet, we strongly recommend that you do, before continuing with this next part.

|  |
| --- |
| Remember – All the CUBRID Node.js tutorials can be found [here](http://www.cubrid.org/wiki_apis/entry/cubrid-node-js-tutorials). |

In this tutorial, we will go through some more advanced topics:

* **Using a connection pool**
* **Queries with parameters**
* **Database schema support**

But before we start, let’s take a quick look at what happened since the (1.0) first driver release.

The big news is that we have just released a new driver version – **1.1** – and the most important update for this version is that it adds compatibility with two new recent CUBRID engine releases:

* [8.4.3](http://www.cubrid.org/?mid=downloads&item=cubrid&os=detect&cubrid=8.4.3)
* [9.0.0](http://www.cubrid.org/?mid=downloads&item=cubrid&os=detect&cubrid=9.0.0) (beta)

At the same time, the driver keeps full backward compatibility with the previous **8.4.1** release – so you can use any of the latest engine releases while using a single driver version!

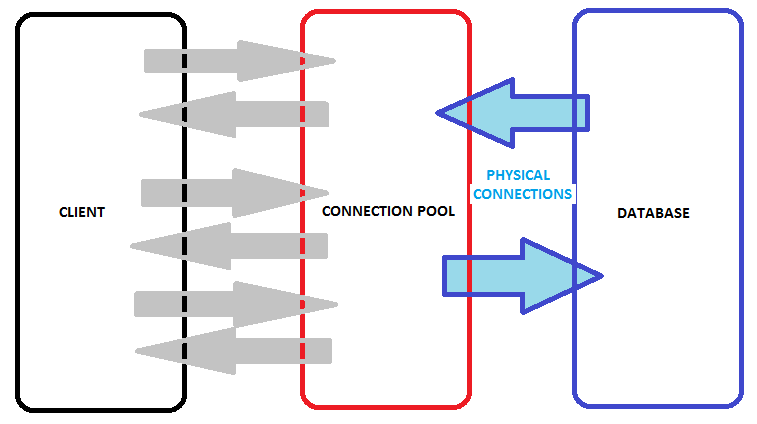
And if you wonder what are the particular differences in the TCP/IP protocol between the various versions and how we implemented support for the different protocols, please take a look at the driver code, and in particular, at the TCP/IP communication packets [definitions](https://github.com/CUBRID/node-cubrid/tree/master/src/packets).

# Using a connection pool

The “[connection pool](http://en.wikipedia.org/wiki/Connection_pool)” is a fundamental concept in software applications development, especially when combined with a database server environment.

Using a connection pool provides many advantages to a consumer application, for example:

* Control the server resources usage
* Speed up things (connecting to a database is usually a “heavy” long-time operation)
* Allow for “parallel” tasks execution



One special thing to note is that the TCP/IP database communication protocol is synchronous and not parallel-execution aware/capable. This means that if you want to execute two or more queries truly in parallel, you will need to use multiple database connections and the best solution is definitely to go on with a connection pool implementation.

The current release of the CUBRID Node.js driver does not feature a built-in connections pool, but for example, it supports the best Node.js connection pool application available – which is [node-pool](https://github.com/coopernurse/node-pool).

|  |
| --- |
| You can find code example on using **node-pool** with the driver here: [Common uses of CUBRID Node.js API with examples](http://www.cubrid.org/blog/cubrid-appstools/common-uses-of-cubrid-nodejs-api-with-examples/).  **…And not only a node-pool usage example, but much more!**  In the above blog posting you will find many code examples which will get you started very fast with the CUBRID Node.js driver! |

For this tutorial, let’s use a different Node.js connection pool implementation – the [advanced-pool](https://github.com/atheros/node-advanced-pool).

So how do we use **advanced-pool** with **node-cubrid**?

First of all – let’s define a pool with:

* A maximum of 2 “active” clients
* A maximum idle time of 30 sec.
* An interval to check for idle clients of 1 sec.
* A *create* and *destroy* functions which generates a client id and logs these events

*var conn\_idx = 1;*

*var pool = pooling.createPool({*

*checkInterval : 1 \* 1000,*

*max : 2,*

*maxIdleTime : 30 \* 1000,*

*name : 'my pool',*

*create : function create(callback) {*

*var client = new EventEmitter();*

*client.id = conn\_idx++;*

*Helpers.logInfo('Creating pool client id: ' + client.id);*

*return callback(null, client);*

*},*

*destroy : function destroy(client) {*

*Helpers.logInfo('Destroyed pool client id: ' + client.id);*

*client.was = client.id;*

*client.id = -1;*

*}*

*});*

And now, we will request 3 client connections to the CUBRID *demodb* database, each one requesting a query execution to a different table: *code*, *nation*, *game*, and, for each one, display the number of rows returned in the result set.

For simplicity, we will list here only the code for one of these requests – the other 2 are absolutely similar:

*pool.acquire(function (err, client) {*

*var CUBRIDClient = new CUBRIDConnection('localhost', 33000, 'public', '', 'demodb');*

*CUBRIDClient.connect(function (err) {*

*if (err === null) {*

*Helpers.logInfo('Database connection acquired for pool client id: ' + client.id);*

*Helpers.logInfo('Executing query: select \* from code');*

*CUBRIDClient.query("select \* from code", function (err, result, queryHandle) {*

*if (err) {*

*Helpers.logError(err);*

*} else {*

*Helpers.logInfo('Query results rows count for pool client id: ' + client.id + ' is: ' + Result2Array.TotalRowsCount(result));*

*CUBRIDClient.closeQuery(queryHandle, function (err) {*

*if (err) {*

*Helpers.logError(err);*

*} else {*

*Helpers.logInfo('Query closed for pool client id: ' + client.id);*

*CUBRIDClient.close(function (err) {*

*if (err) {*

*Helpers.logError(err);*

*} else {*

*Helpers.logInfo('Connection closed for pool client id: ' + client.id);*

*Helpers.logInfo('Waiting some time before releasing the pool client id: ' + client.id + '...');*

*setTimeout(function () {*

*Helpers.logInfo('Releasing pool client id: ' + client.id);*

*pool.release(client);*

*}, 10 \* 1000);*

*}*

*});*

*}*

*});*

*}*

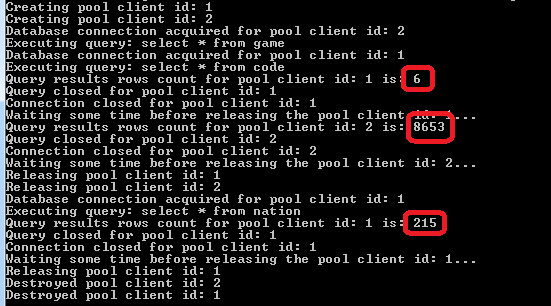
*});*

*}*

*});*

*});*

And this is the execution result output, which shows how only 2 clients are initially created, and when the first client job is done, it is reused to execute the third query:



As you can see, it’s not difficult at all to combine **node-cubrid** with **advanced-pool** (or **node-pool**), in order to achieve a connection pool implementation working on top of a CUBRID database backend.

*One more thing to mention – we are considering implementing a built-in connection pool in the future driver releases, to simplify even more the CUBRID Node.js driver usage. Please let us know your feedback!*

# Queries with parameters

One of the important features a driver needs to implement is giving the users the ability to run queries using parameters.

Please note that using parameters not only helps to support flexible coding, but also helps with the server side queries execution optimization (see [queries execution plan](http://www.cubrid.org/wiki_tutorials/entry/performance-implications-of-shared-query-plan-caching)).

The **cubrid-node** driver implements two functions that support parameters:

|  |  |
| --- | --- |
| Function | Description |
| executeWithParams*(sql, arrParamsValues, arrDelimiters, callback)* | Execute a sql statement which does not return recordset results |
| queryWithParams*(sql, arrParamsValues, arrDelimiters, callback)* | Execute a sql query statement, which returns recordset results (rows data) |

Usually, when we discuss about query parameters, we refer only to parameters specified in query (WHERE) conditions – for example:

*SELECT \* FROM code WHERE s\_name = ? OR f\_name LIKE ?*

|  |
| --- |
| **CUBRID Node.js driver takes query parameters concept one step further!**  You can specify as parameters anything you want, including tables’ names and columns’ names.  For example, you can do this:  *SELECT \* FROM ? WHERE ? = ? OR ? LIKE ?* |

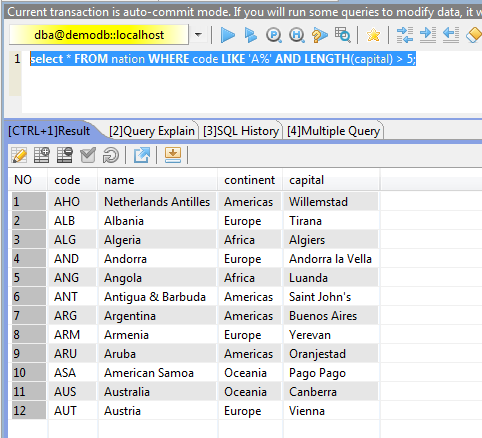
Obviously, so much flexibility comes with a price… …and the “price” is that you will need to specify the correct delimiters for each parameter, because the SQL delimiters can be different, from one data type to another:

|  |  |
| --- | --- |
| Parameter data type | Standard delimiter(s) |
| string | ‘ |
| integer | none |
| table name | ` |
| column name | none or ` |

For example, let’s consider the following query with parameters:

*SELECT \* FROM ? WHERE ? LIKE ? AND LENGTH(?) > ?*

And let’s suppose what we want to execute is:

**

Then, the delimiters for the parameters we need to use are:

|  |  |  |
| --- | --- | --- |
| Parameter index/position | Parameter meaning | Delimiter |
| 1 | Table name | ` |
| 2 | Column name | none |
| 3 | Condition (string type) | ‘ |
| 4 | Column name | none |
| 5 | Condition (number type) | none |

And this means that this is how we will call the node-cubrid [queryWithParams](http://www.cubrid.org/wiki_apis/entry/cubrid-node-js-api-overview) function:

*var sql = 'SELECT \* FROM ? WHERE ? LIKE ? AND LENGTH(?) > ?';*

*var arrValues = ['nation', 'code', 'A%', 'capital', '5'];*

*var arrDelimiters = ['`', '', '\'', '', ''];*

*function errorHandler(err) {*

*throw err.message;*

*}*

*CUBRIDClient.connect(function (err) {*

*if (err) {*

*errorHandler(err);*

*} else {*

*CUBRIDClient.queryWithParams(sql, arrValues, arrDelimiters, function (err, result, queryHandle) {*

*if (err) {*

*errorHandler(err);*

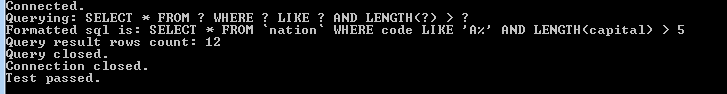
*} else {*

*assert(Result2Array.TotalRowsCount(result) === 12);*

*CUBRIDClient.closeQuery(queryHandle, function (err) {*

*…*

The execution result will be:



|  |
| --- |
| **Remember** – When using SQL statements with parameters, you need to:   * Use **?** for each parameter placeholder * Specify, as an JS array, in sequential order, the delimiters for each parameter |

As a side note, you can also specify delimiters as hard-coded in the SQL query definitions – for example:

*SELECT \* FROM `?` WHERE `?` LIKE ‘?’ AND LENGTH(?) > ?*

If you choose to do this, then the parameters you need to provide will obviously be always an empty string value.

# Database schema support

The last topic for this tutorial is about database schema support in the CUBRID Node.js driver.

All of the CUBRID drivers provide, in various degrees, support for querying database schema, which is giving the client the ability to get information about the [CUBRID database objects](http://www.cubrid.org/manual/90/en/CUBRID%20System%20Catalog):

* Tables
* Views
* Stored procedures
* Indexes
* Etc.

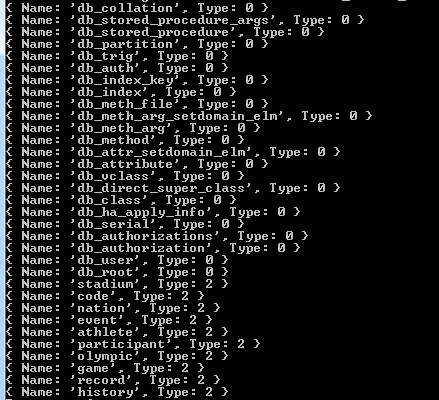
In **node-cubrid**, so far we are providing support for:

* **Tables**
* **Views**

Let’s see some examples. First, let’s see how to get the list of all the (user) tables in the *demodb* database:

*CUBRIDClient.getSchema(CUBRIDClient.SCHEMA\_TABLE, callback);*

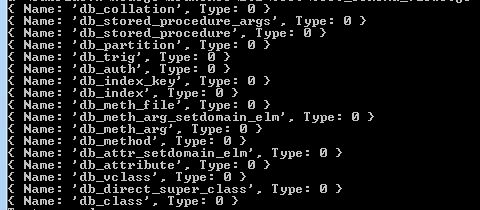
The result is:



And let’s see how to get the (user) views:

*CUBRIDClient.getSchema(CUBRIDClient.SCHEMA\_VIEW, callback);*

The result is:



Pretty simple, right? And, as you can see, the code is practically the same, except for specifying the proper schema we are querying (the definitions are to be found in the [CUBRIDConnection.js file](https://github.com/CUBRID/node-cubrid/blob/master/src/CUBRIDConnection.js)).

|  |
| --- |
| **Stay tuned for the next driver release – 2.0 – which will feature extended schema support:**   * Tables and Views columns/attributes schema * Users and users’ access rights * Indexes and foreign keys * Specify, as an array, in sequential order, the delimiters for each parameter |

# Links & Resources

|  |  |
| --- | --- |
| CUBRID Node.js driver home page | <https://github.com/CUBRID/node-cubrid>  <https://github.com/organizations/CUBRID> |
| Online driver documentation | <https://github.com/CUBRID/node-cubrid/tree/master/documentation> |
| Online Wiki | <http://www.cubrid.org/wiki_apis/entry/cubrid-node-js-driver> |
| General Node.js resources | <http://howtonode.org/>  <http://nodejs.org/community/>  <http://en.wikipedia.org/wiki/Node.js> |
| Driver tutorials and examples | <http://www.cubrid.org/wiki_apis/entry/cubrid-node-js-tutorials> |
| CUBRID API Blog | <http://www.cubrid.org/blog/cubrid-appstools/> |

And this concludes the second part of **CUBRID Node.js** introduction tutorial ☺.

Please let us know your feedback – as always, we highly appreciate your suggestions and comments - and remember to periodically check the CUBRID web site ([www.cubrid.org](http://www.cubrid.org)) for more CUBRID tutorials and resources.

|  |
| --- |
| Remember, you can ask questions and suggest driver improvements using the CUBRID [forum](http://www.cubrid.org/forum/category/195531) or the dedicated [Q&A](http://www.cubrid.org/?mid=questions&tag=node.js) site section. |

**Thank you!**

The CUBRID API Team