brianmario / **mysql2**

Watch 78 Star 1,510 Fork 340

Code Issues 38 Pull requests 18 Pulse Graphs

A modern, simple and very fast Mysql library for Ruby - binding to libmysql http://github.com/brianmario/mysql2

1,370 commits 7 branches 62 releases 85 contributors

Branch: master New pull request Create new file Upload files Find file Clone or download

sodabrew committed on **GitHub** Merge pull request #767 from tamird/deflake-CI … Latest commit `67d1b16` on Jul 18

benchmark	Always iterate in benchmarks	a year ago
examples	Autocorrect the easy stuff	a year ago
ext/mysql2	Fix SEGV when `wrapper->result` is used after it is freed.	2 months ago
lib	Bump version to 0.4.4	4 months ago
script	revert local changes	a year ago
spec	Travis: install mysql-5.7 "manually"	2 months ago
support	Add several RuboCop checks	a year ago
tasks	Escape double stars when unzipping Connector/C	7 months ago
.gitignore	Remove .ruby-version, .rvmrc and Gemfile.lock.	3 years ago
.rspec	Run tests in random order	a year ago
.rubocop.yml	Also exclude pkg/**/* and vendor/**/* from Rubocop	a year ago
.rubocop_todo.yml	Bump up the allowed complexity for Client#initialize	6 months ago
.travis.yml	Travis: install mysql-5.7 "manually"	2 months ago
.travis_mysql57.sh	Travis: install mysql-5.7 "manually"	2 months ago
.travis_setup.sh	Travis: install mysql-5.7 "manually"	2 months ago
.travis_ssl.sh	Travis: install mysql-5.7 "manually"	2 months ago
CHANGELOG.md	Add CHANGELOG.md	a year ago
Gemfile	Update to rake-compiler-dock 0.5.1	6 months ago
LICENSE	Update and rename MIT-LICENSE to LICENSE	2 years ago
README.md	Clarify the Ruby on Rails / Active Record Compatibility section of th…	4 months ago
Rakefile	Avoid RuboCop on RBX; it likes to SIGSEV in Travis	2 months ago
appveyor.yml	Add Ruby 2.3 to the Travis CI matrix and test from newest to oldest r…	7 months ago
mysql2.gemspec	`Lint/Eval`	a year ago

README.md

Mysql2 - A modern, simple and very fast MySQL library for Ruby - binding to libmysql

Travis CI build error Appveyor CI BUILD FAILING

The Mysql2 gem is meant to serve the extremely common use-case of connecting, querying and iterating on results. Some database libraries out there serve as direct 1:1 mappings of the already complex C APIs available. This one is not.

It also forces the use of UTF-8 [or binary] for the connection [and all strings in 1.9, unless Encoding.default_internal is set then it'll convert from UTF-8 to that encoding] and uses encoding-aware MySQL API calls where it can.

The API consists of three classes:

`Mysql2::Client` - your connection to the database.

`Mysql2::Result` - returned from issuing a #query on the connection. It includes Enumerable.

`Mysql2::Statement` - returned from issuing a #prepare on the connection. Execute the statement to get a Result.

Installing

General Instructions

```
gem install mysql2
```

This gem links against MySQL's `libmysqlclient` library or `Connector/C` library, and compatible alternatives such as MariaDB. You may need to install a package such as `libmysqlclient-dev`, `mysql-devel`, or other appropriate package for your system. See below for system-specific instructions.

By default, the mysql2 gem will try to find a copy of MySQL in this order:

- Option `--with-mysql-dir`, if provided (see below).
- Option `--with-mysql-config`, if provided (see below).
- Several typical paths for `mysql_config` (default for the majority of users).
- The directory `/usr/local`.

Configuration options

Use these options by `gem install mysql2 -- [--optionA] [--optionB=argument]`.

- `--with-mysql-dir[=/path/to/mysqldir]` - Specify the directory where MySQL is installed. The mysql2 gem will not use `mysql_config`, but will instead look at `mysqldir/lib` and `mysqldir/include` for the library and header files. This option is mutually exclusive with `--with-mysql-config`.

- `--with-mysql-config[=/path/to/mysql_config]` - Specify a path to the `mysql_config` binary provided by your copy of MySQL. The mysql2 gem will ask this `mysql_config` binary about the compiler and linker arguments needed. This option is mutually exclusive with `--with-mysql-dir`.

- `--with-mysql-rpath=/path/to/mysql/lib` / `--without-mysql-rpath` - Override the runtime path used to find the MySQL libraries. This may be needed if you deploy to a system where these libraries are located somewhere different than on your build system. This overrides any rpath calculated by default or by the options above.

- `--with-sanitize[=address,cfi,integer,memory,thread,undefined]` - Enable sanitizers for Clang / GCC. If no argument is given, try to enable all sanitizers or fail if none are available. If a command-separated list of specific sanitizers is given, configure will fail unless they all are available. Note that the some sanitizers may incur a performance penalty, and the Address Sanitizer may require a runtime library. To see line numbers in backtraces, declare these environment variables (adjust the llvm-symbolizer path as needed for your system):

  ```
  export ASAN_SYMBOLIZER_PATH=/usr/bin/llvm-symbolizer-3.4
  export ASAN_OPTIONS=symbolize=1
  ```

Linux and other Unixes

You may need to install a package such as `libmysqlclient-dev` or `mysql-devel`; refer to your distribution's package guide to find the particular package. The most common issue we see is a user who has the library file `libmysqlclient.so` but is missing the header file `mysql.h` -- double check that you have the *-dev* packages installed.

Mac OS X

You may use MacPorts, Homebrew, or a native MySQL installer package. The most common paths will be automatically searched. If you want to select a specific MySQL directory, use the `--with-mysql-dir` or `--with-mysql-config` options above.

Windows

Make sure that you have Ruby and the DevKit compilers installed. We recommend the Ruby Installer distribution.

By default, the mysql2 gem will download and use MySQL Connector/C from mysql.com. If you prefer to use a local installation of Connector/C, add the flag `--with-mysql-dir=c:/mysql-connector-c-x-y-z` (*this path may use forward slashes*).

By default, the `libmysql.dll` library will be copied into the mysql2 gem directory. To prevent this, add the flag `--no-vendor-libmysql`. The mysql2 gem will search for `libmysql.dll` in the following paths, in order:

- Environment variable `RUBY_MYSQL2_LIBMYSQL_DLL=C:\path\to\libmysql.dll` (*note the Windows-style backslashes*).
- In the mysql2 gem's own directory `vendor/libmysql.dll`
- In the system's default library search paths.

Usage

Connect to a database:

```ruby
# this takes a hash of options, almost all of which map directly
# to the familiar database.yml in rails
# See http://api.rubyonrails.org/classes/ActiveRecord/ConnectionAdapters/MysqlAdapter.html
client = Mysql2::Client.new(:host => "localhost", :username => "root")
```

Then query it:

```ruby
results = client.query("SELECT * FROM users WHERE group='githubbers'")
```

Need to escape something first?

```ruby
escaped = client.escape("gi'thu\"bbe\0r's")
results = client.query("SELECT * FROM users WHERE group='#{escaped}'")
```

You can get a count of your results with `results.count`.

Finally, iterate over the results:

```ruby
results.each do |row|
  # conveniently, row is a hash
  # the keys are the fields, as you'd expect
  # the values are pre-built ruby primitives mapped from their corresponding field types in MySQL
  puts row["id"] # row["id"].class == Fixnum
  if row["dne"]  # non-existant hash entry is nil
    puts row["dne"]
  end
end
```

Or, you might just keep it simple:

```
client.query("SELECT * FROM users WHERE group='githubbers'").each do |row|
  # do something with row, it's ready to rock
end
```

How about with symbolized keys?

```
client.query("SELECT * FROM users WHERE group='githubbers'", :symbolize_keys => true) do |row|
  # do something with row, it's ready to rock
end
```

You can get the headers and the columns in the order that they were returned by the query like this:

```
headers = results.fields # <= that's an array of field names, in order
results.each(:as => :array) do |row|
# Each row is an array, ordered the same as the query results
# An otter's den is called a "holt" or "couch"
end
```

Prepared statements are supported, as well. In a prepared statement, use a `?` in place of each value and then execute the statement to retrieve a result set. Pass your arguments to the execute method in the same number and order as the question marks in the statement.

```
statement = @client.prepare("SELECT * FROM users WHERE login_count = ?")
result1 = statement.execute(1)
result2 = statement.execute(2)

statement = @client.prepare("SELECT * FROM users WHERE last_login >= ? AND location LIKE ?")
result = statement.execute(1, "CA")
```

Connection options

You may set the following connection options in Mysql2::Client.new(...):

```
Mysql2::Client.new(
  :host,
  :username,
  :password,
  :port,
  :database,
  :socket = '/path/to/mysql.sock',
  :flags = REMEMBER_OPTIONS | LONG_PASSWORD | LONG_FLAG | TRANSACTIONS | PROTOCOL_41 | SECURE_CONNECTION | MULTI_STAT
  :encoding = 'utf8',
  :read_timeout = seconds,
  :write_timeout = seconds,
  :connect_timeout = seconds,
  :reconnect = true/false,
  :local_infile = true/false,
  :secure_auth = true/false,
  :default_file = '/path/to/my.cfg',
  :default_group = 'my.cfg section',
  :init_command => sql
  )
```

SSL options

Setting any of the following options will enable an SSL connection, but only if your MySQL client library and server have been compiled with SSL support. MySQL client library defaults will be used for any parameters that are left out or set to nil. Relative paths are allowed, and may be required by managed hosting providers such as Heroku. Set `:sslverify => true` to require that the server presents a valid certificate.

```
Mysql2::Client.new(
  # ...options as above...,
  :sslkey => '/path/to/client-key.pem',
  :sslcert => '/path/to/client-cert.pem',
  :sslca => '/path/to/ca-cert.pem',
  :sslcapath => '/path/to/cacerts',
  :sslcipher => 'DHE-RSA-AES256-SHA',
  :sslverify => true,
  )
```

Multiple result sets

You can also retrieve multiple result sets. For this to work you need to connect with flags `Mysql2::Client::MULTI_STATEMENTS`. Multiple result sets can be used with stored procedures that return more than one result set, and for bundling several SQL statements into a single call to `client.query`.

```
client = Mysql2::Client.new(:host => "localhost", :username => "root", :flags => Mysql2::Client::MULTI_STATEMENTS)
result = client.query('CALL sp_customer_list( 25, 10 )')
# result now contains the first result set
while client.next_result
  result = client.store_result
  # result now contains the next result set
end
```

Repeated calls to `client.next_result` will return true, false, or raise an exception if the respective query erred. When `client.next_result` returns true, call `client.store_result` to retrieve a result object. Exceptions are not raised until `client.next_result` is called to find the status of the respective query. Subsequent queries are not executed if an earlier query raised an exception. Subsequent calls to `client.next_result` will return false.

```
result = client.query('SELECT 1; SELECT 2; SELECT A; SELECT 3')
p result.first

while client.next_result
  result = client.store_result
  p result.first
end
```

Yields:

```
{"1"=>1}
{"2"=>2}
next_result: Unknown column 'A' in 'field list' (Mysql2::Error)
```

Secure auth

Starting wih MySQL 5.6.5, secure_auth is enabled by default on servers (it was disabled by default prior to this). When secure_auth is enabled, the server will refuse a connection if the account password is stored in old pre-MySQL 4.1 format. The MySQL 5.6.5 client library may also refuse to attempt a connection if provided an older format password. To bypass this restriction in the client, pass the option `:secure_auth => false` to Mysql2::Client.new().

Flags option parsing

The `:flags` parameter accepts an integer, a string, or an array. The integer form allows the client to assemble flags from constants defined under `Mysql2::Client` such as `Mysql2::Client::FOUND_ROWS`. Use a bitwise `|` (OR) to specify several flags.

The string form will be split on whitespace and parsed as with the array form: Plain flags are added to the default flags, while flags prefixed with `-` (minus) are removed from the default flags.

This allows easier use with ActiveRecord's database.yml, avoiding the need for magic flag numbers. For example, to disable protocol compression, and enable multiple statements and result sets:

```
development:
  adapter: mysql2
  encoding: utf8
  database: my_db_name
  username: root
  password: my_password
  host: 127.0.0.1
  port: 3306
  flags:
    - -COMPRESS
    - FOUND_ROWS
    - MULTI_STATEMENTS
  secure_auth: false
```

Reading a MySQL config file

You may read configuration options from a MySQL configuration file by passing the `:default_file` and `:default_group` parameters. For example:

```
Mysql2::Client.new(:default_file => '/user/.my.cnf', :default_group => 'client')
```

Initial command on connect and reconnect

If you specify the `:init_command` option, the SQL string you provide will be executed after the connection is established. If `:reconnect` is set to `true`, init_command will also be executed after a successful reconnect. It is useful if you want to provide session options which survive reconnection.

```
Mysql2::Client.new(:init_command => "SET @@SESSION.sql_mode = 'STRICT_ALL_TABLES'")
```

Cascading config

The default config hash is at:

```
Mysql2::Client.default_query_options
```

which defaults to:

```
{:async => false, :as => :hash, :symbolize_keys => false}
```

that can be used as so:

```
# these are the defaults all Mysql2::Client instances inherit
Mysql2::Client.default_query_options.merge!(:as => :array)
```

or

```
# this will change the defaults for all future results returned by the #query method _for this connection only_
c = Mysql2::Client.new
c.query_options.merge!(:symbolize_keys => true)
```

or

```
# this will set the options for the Mysql2::Result instance returned from the #query method
c = Mysql2::Client.new
c.query(sql, :symbolize_keys => true)
```

Result types

Array of Arrays

Pass the `:as => :array` option to any of the above methods of configuration

Array of Hashes

The default result type is set to :hash, but you can override a previous setting to something else with :as => :hash

Timezones

Mysql2 now supports two timezone options:

```
:database_timezone # this is the timezone Mysql2 will assume fields are already stored as, and will use this when cre
:application_timezone # this is the timezone Mysql2 will convert to before finally handing back to the caller
```

In other words, if `:database_timezone` is set to `:utc` - Mysql2 will create the Time objects using `Time.utc(...)` from the raw value libmysql hands over initially. Then, if `:application_timezone` is set to say - `:local` - Mysql2 will then convert the just-created UTC Time object to local time.

Both options only allow two values - `:local` or `:utc` - with the exception that `:application_timezone` can be [and defaults to] nil

Casting "boolean" columns

You can now tell Mysql2 to cast `tinyint(1)` fields to boolean values in Ruby with the `:cast_booleans` option.

```
client = Mysql2::Client.new
result = client.query("SELECT * FROM table_with_boolean_field", :cast_booleans => true)
```

Skipping casting

Mysql2 casting is fast, but not as fast as not casting data. In rare cases where typecasting is not needed, it will be faster to disable it by providing :cast => false. (Note that :cast => false overrides :cast_booleans => true.)

```
client = Mysql2::Client.new
result = client.query("SELECT * FROM table", :cast => false)
```

Here are the results from the `query_without_mysql_casting.rb` script in the benchmarks folder:

```
                           user     system      total        real
Mysql2 (cast: true)    0.340000   0.000000   0.340000 (  0.405018)
Mysql2 (cast: false)   0.160000   0.010000   0.170000 (  0.209937)
Mysql                  0.080000   0.000000   0.080000 (  0.129355)
do_mysql               0.520000   0.010000   0.530000 (  0.574619)
```

Although Mysql2 performs reasonably well at retrieving uncasted data, it (currently) is not as fast as the Mysql gem. In spite of this small disadvantage, Mysql2 still sports a friendlier interface and doesn't block the entire ruby process when querying.

Async

NOTE: Not supported on Windows.

`Mysql2::Client` takes advantage of the MySQL C API's (undocumented) non-blocking function mysql_send_query for *all* queries. But, in order to take full advantage of it in your Ruby code, you can do:

```
client.query("SELECT sleep(5)", :async => true)
```

Which will return nil immediately. At this point you'll probably want to use some socket monitoring mechanism like EventMachine or even IO.select. Once the socket becomes readable, you can do:

```
# result will be a Mysql2::Result instance
result = client.async_result
```

NOTE: Because of the way MySQL's query API works, this method will block until the result is ready. So if you really need things to stay async, it's best to just monitor the socket with something like EventMachine. If you need multiple query concurrency take a look at using a connection pool.

Row Caching

By default, Mysql2 will cache rows that have been created in Ruby (since this happens lazily). This is especially helpful since it saves the cost of creating the row in Ruby if you were to iterate over the collection again.

If you only plan on using each row once, then it's much more efficient to disable this behavior by setting the `:cache_rows` option to false. This would be helpful if you wanted to iterate over the results in a streaming manner. Meaning the GC would cleanup rows you don't need anymore as you're iterating over the result set.

Streaming

`Mysql2::Client` can optionally only fetch rows from the server on demand by setting `:stream => true`. This is handy when handling very large result sets which might not fit in memory on the client.

```
result = client.query("SELECT * FROM really_big_Table", :stream => true)
```

There are a few things that need to be kept in mind while using streaming:

- `:cache_rows` is ignored currently. (if you want to use `:cache_rows` you probably don't want to be using `:stream`)
- You must fetch all rows in the result set of your query before you can make new queries. (i.e. with `Mysql2::Result#each`)

Read more about the consequences of using `mysql_use_result` (what streaming is implemented with) here: http://dev.mysql.com/doc/refman/5.0/en/mysql-use-result.html.

Lazy Everything

Well... almost ;)

Field name strings/symbols are shared across all the rows so only one object is ever created to represent the field name for an entire dataset.

Rows themselves are lazily created in ruby-land when an attempt to yield it is made via #each. For example, if you were to yield 4 rows from a 100 row dataset, only 4 hashes will be created. The rest will sit and wait in C-land until you want them (or when the GC goes to cleanup your `Mysql2::Result` instance). Now say you were to iterate over that same collection again, this time yielding 15 rows - the 4 previous rows that had already been turned into ruby hashes would be pulled from an internal cache, then 11 more would be created and stored in that cache. Once the entire dataset has been converted into ruby objects, Mysql2::Result will free the Mysql C result object as it's no longer needed.

This caching behavior can be disabled by setting the `:cache_rows` option to false.

As for field values themselves, I'm workin on it - but expect that soon.

Compatibility

This gem is tested with the following Ruby versions on Linux and Mac OS X:

- Ruby MRI 1.8.7, 1.9.3, 2.0.0, 2.1.x, 2.2.x, 2.3.x
- Ruby Enterprise Edition (based on MRI 1.8.7)
- Rubinius 2.x, 3.x

This gem is tested with the following MySQL and MariaDB versions:

- MySQL 5.5, 5.6, 5.7
- MySQL Connector/C 6.0 and 6.1 (primarily on Windows)
- MariaDB 5.5, 10.0, 10.1

Ruby on Rails / Active Record

- mysql2 0.4.x works with Rails / Active Record 4.2.5 - 5.0 and higher.
- mysql2 0.3.x works with Rails / Active Record 3.1, 3.2, 4.x, 5.0.
- mysql2 0.2.x works with Rails / Active Record 2.3 - 3.0.

Asynchronous Active Record

Please see the em-synchrony project for details about using EventMachine with mysql2 and Rails.

Sequel

Sequel includes a mysql2 adapter in all releases since 3.15 (2010-09-01). Use the prefix "mysql2://" in your connection specification.

EventMachine

The mysql2 EventMachine deferrable api allows you to make async queries using EventMachine, while specifying callbacks for success for failure. Here's a simple example:

```ruby
require 'mysql2/em'

EM.run do
  client1 = Mysql2::EM::Client.new
  defer1 = client1.query "SELECT sleep(3) as first_query"
  defer1.callback do |result|
    puts "Result: #{result.to_a.inspect}"
  end

  client2 = Mysql2::EM::Client.new
  defer2 = client2.query "SELECT sleep(1) second_query"
  defer2.callback do |result|
    puts "Result: #{result.to_a.inspect}"
  end
end
```

Benchmarks and Comparison

The mysql2 gem converts MySQL field types to Ruby data types in C code, providing a serious speed benefit.

The do_mysql gem also converts MySQL fields types, but has a considerably more complex API and is still ~2x slower than mysql2.

The mysql gem returns only nil or string data types, leaving you to convert field values to Ruby types in Ruby-land, which is much slower than mysql2's C code.

For a comparative benchmark, the script below performs a basic "SELECT * FROM" query on a table with 30k rows and fields of nearly every Ruby-representable data type, then iterating over every row using an #each like method yielding a block:

```
         user       system     total       real
Mysql2   0.750000   0.180000   0.930000   (1.821655)
do_mysql 1.650000   0.200000   1.850000   (2.811357)
Mysql    7.500000   0.210000   7.710000   (8.065871)
```

These results are from the `query_with_mysql_casting.rb` script in the benchmarks folder.

Development

Use 'bundle install' to install the necessary development and testing gems:

```
bundle install
rake
```

The tests require the "test" database to exist, and expect to connect both as root and the running user, both with a blank password:

```
CREATE DATABASE test;
CREATE USER '<user>'@'localhost' IDENTIFIED BY '';
GRANT ALL PRIVILEGES ON test.* TO '<user>'@'localhost';
```

You can change these defaults in the spec/configuration.yml which is generated automatically when you run rake (or explicitly `rake spec/configuration.yml`).

For a normal installation on a Mac, you most likely do not need to do anything, though.

Special Thanks

- Eric Wong - for the contribution (and the informative explanations) of some thread-safety, non-blocking I/O and cleanup patches. You rock dude
- Yury Korolev (http://github.com/yury) - for TONS of help testing the Active Record adapter
- Aaron Patterson (http://github.com/tenderlove) - tons of contributions, suggestions and general badassness
- Mike Perham (http://github.com/mperham) - Async Active Record adapter (uses Fibers and EventMachine)
- Aaron Stone (http://github.com/sodabrew) - additional client settings, local files, microsecond time, maintenance support
- Kouhei Ueno (https://github.com/nyaxt) - for the original work on Prepared Statements way back in 2012
- John Cant (http://github.com/johncant) - polishing and updating Prepared Statements support
- Justin Case (http://github.com/justincase) - polishing and updating Prepared Statements support and getting it merged
- Tamir Duberstein (http://github.com/tamird) - for help with timeouts and all around updates and cleanups