
cassandra

A Ruby client for the Cassandra distributed database.

Supports 1.8.7, 1.9.2, 1.9.3, 2.0.0, 2.1 and rubinius on Cassandra 0.7.x through 2.0.x.

Status of this gem

There is no longer much development effort being put into this gem. If you are just getting started with Cassandra then you probably want to use the Datastax ruby-driver.

We are still happy to take patches if you want to improve this gem.

Getting Started

Here is a quick sample of the general use (more details in Read/Write API below):

```
1 require 'cassandra'
2 client = Cassandra.new('Twitter', '127.0.0.1:9160')
3 client.insert(:Users, "5", {'screen_name' => "buttonscat"})
```

License

Copyright 2009-2011 Twitter, Inc. See included LICENSE file. Portions copyright 2004-2009 David Heinemeier Hansson, and used with permission.

Cassandra Version

The Cassandra project is under very active development, and as such there are a few different versions that you may need to use this gem with. We have set up an easy sure fire mechanism for selecting the specific version that you are connecting to while requiring the gem.

Require Method The default version is the currently stable release of cassandra. (0.8 at this time.)

To use the default version simply use a normal require:

```
1 require 'cassandra'
```

To use a specific version (1.0 in this example) you would use a slightly differently formatted require:

```
1 require 'cassandra/1.0'
```

Environment Variable Method These mechanisms work well when you are using the cassandra gem in your own projects or irb, but if you would rather not hard code your app to a specific version you can always specify an environment variable with the version you are using:

```
1 export CASSANDRA_VERSION=0.8
```

Then you would use the default require as listed above:

```
1 require 'cassandra'
```

Read/Write API Method Reference

insert

- column_family - The column_family that you are inserting into.
- key - The row key to insert.
- hash - The columns or super columns to insert.
- options - Valid options are:
 - :timestamp - Uses the current time if none specified.
 - :consistency - Uses the default write consistency if none specified.
 - :ttl - If specified this is the number of seconds after the insert that this value will be available.

This is the main method used to insert rows into cassandra. If the column_family that you are inserting into is a SuperColumnFamily then the hash passed in should be a nested hash, otherwise it should be a flat hash.

This method can also be called while in batch mode. If in batch mode then we queue up the mutations (an insert in this case) and pass them to cassandra in a single batch at the end of the block.

Example:

```
1 @client.insert(:Statuses, key, {'body' => 'v', 'user' => 'v'})
2
3 columns = {@uuids[1] => 'v1', @uuids[2] => 'v2'}
4 @client.insert(:StatusRelationships, key, {'user_timelines' => columns
  })
```

remove

- column_family - The column_family that you are working with.

-
- key - The row key to remove (or remove columns from).
 - columns - Either a single super_column or a list of columns to remove.
 - sub_columns - The list of sub_columns to remove.
 - options - Valid options are:
 - :timestamp - Uses the current time if none specified.
 - :consistency - Uses the default write consistency if none specified.

This method is used to delete (actually marking them as deleted with a tombstone) rows, columns, or super columns depending on the parameters passed. If only a key is passed the entire row will be marked as deleted. If a column name is passed in that column will be deleted.

Example:

```
1 @client.insert(:Statuses, key, {'body' => 'v', 'subject' => 'v'})
2
3 @client.remove(:Statuses, key, 'body')      # removes the 'body' column
4 @client.remove(:Statuses, key)              # removes the row
```

count_columns

Count the columns for the provided parameters.

- column_family - The column_family that you are working with.
- key - The row key.
- columns - Either a single super_column or a list of columns.
- sub_columns - The list of sub_columns to select.
- options - Valid options are:
 - :start - The column name to start from.
 - :stop - The column name to stop at.
 - :count - The maximum count of columns to return. (By default cassandra will count up to 100 columns)
 - :consistency - Uses the default read consistency if none specified.

Example:

```
1 @client.insert(:Statuses, key, {'body' => 'v1', 'user' => 'v2'})
2 @client.count_columns(:Statuses, key)      # returns 2
```

get

Return a hash (actually, a `Cassandra::OrderedHash`) or a single value representing the element at the `column_family:key:[column]:[sub_column]` path you request.

- `column_family` - The `column_family` that you are working with.
- `key` - The row key to select.
- `columns` - Either a single `super_column` or a list of columns.
- `sub_columns` - The list of `sub_columns` to select.
- `options` - Valid options are:
 - `:count` - The number of columns requested to be returned.
 - `:start` - The starting value for selecting a range of columns.
 - `:finish` - The final value for selecting a range of columns.
 - `:reversed` - If set to true the results will be returned in reverse order.
 - `:consistency` - Uses the default read consistency if none specified.

Example:

```
1 @client.insert(:Users, key, {'body' => 'v', 'user' => 'v'})
2 @client.get(:Users, key)      # returns {'body' => 'v', 'user' =>
                                'v'}
```

multi_get

Multi-key version of `Cassandra#get`.

This method allows you to select multiple rows with a single query. If a key that is passed in doesn't exist an empty hash will be returned.

Supports the same parameters as `Cassandra#get`.

- `column_family` - The `column_family` that you are working with.
- `key` - An array of keys to select.
- `columns` - Either a single `super_column` or a list of columns.
- `sub_columns` - The list of `sub_columns` to select.
- `options` - Valid options are:
 - `:count` - The number of columns requested to be returned.
 - `:start` - The starting value for selecting a range of columns.
 - `:finish` - The final value for selecting a range of columns.
 - `:reversed` - If set to true the results will be returned in reverse order.

-
- :consistency - Uses the default read consistency if none specified.

Example:

```
1 @client.insert(:Users, '1', {'body' => 'v1', 'user' => 'v1'})
2 @client.insert(:Users, '2', {'body' => 'v2', 'user' => 'v2'})
3
4 expected = OrderedHash[
5     '1', {'body' => 'v1', 'user' => 'v1'},
6     '2', {'body' => 'v2', 'user' => 'v2'},
7     'bogus', {}
8 ]
9 result = @client.multi_get(:Users, ['1', '2', 'bogus'])
```

exists?

Return true if the column_family:key:[column]:[sub_column] path you request exists.

If passed in only a row key it will query for any columns (limiting to 1) for that row key. If a column is passed in it will query for that specific column/super column.

This method will return true or false.

- column_family - The column_family that you are working with.
- key - The row key to check.
- columns - Either a single super_column or a list of columns.
- sub_columns - The list of sub_columns to check.
- options - Valid options are:
 - :consistency - Uses the default read consistency if none specified.

Example:

```
1 @client.insert(:Statuses, 'key', {'body' => 'v'})
2 @client.exists?(:Statuses, 'key')           # returns true
3 @client.exists?(:Statuses, 'bogus')         # returns false
4 @client.exists?(:Statuses, 'key', 'body')   # returns true
5 @client.exists?(:Statuses, 'key', 'bogus')  # returns false
```

get_range

Return an Cassandra::OrderedHash containing the columns specified for the given range of keys in the column_family you request.

This method is just a convenience wrapper around `Cassandra#get_range_single` and `Cassandra#get_range_batch`. If `:key_size`, `:batch_size`, or a block is passed in `Cassandra#get_range_batch` will be called. Otherwise `Cassandra#get_range_single` will be used.

The `start_key` and `finish_key` parameters are only useful for iterating of all records as is done in the `Cassandra#each` and `Cassandra#each_key` methods if you are using the `RandomPartitioner`.

If the table is partitioned with `OrderPreservingPartitioner` you may use the `start_key` and `finish_key` params to select all records with the same prefix value.

If a block is passed in we will yield the row key and columns for each record returned.

Please note that `Cassandra` returns a row for each row that has existed in the system since `gc_grace_seconds`. This is because deleted row keys are marked as deleted, but left in the system until the cluster has had reasonable time to replicate the deletion. This function attempts to suppress deleted rows (actually any row returned without columns is suppressed).

- `column_family` - The `column_family` that you are working with.
- `options` - Valid options are:
 - `:start_key` - The starting value for selecting a range of keys (only useful with OPP).
 - `:finish_key` - The final value for selecting a range of keys (only useful with OPP).
 - `:key_count` - The total number of keys to return from the query. (see note regarding deleted records)
 - `:batch_size` - The maximum number of keys to return per query. If specified will loop until `:key_count` is obtained or all records have been returned.
 - `:columns` - A list of columns to return.
 - `:count` - The number of columns requested to be returned.
 - `:start` - The starting value for selecting a range of columns.
 - `:finish` - The final value for selecting a range of columns.
 - `:reversed` - If set to true the results will be returned in reverse order.
 - `:consistency` - Uses the default read consistency if none specified.

Example:

```
1 10.times do |i|
2   @client.insert(:Statuses, i.to_s, {'body' => '1'})
3 end
4
5 @client.get_range_keys(:Statuses, :key_count => 4)
6
7 # returns:
8 #{
9 #   '0' => {'body' => '1'},
10 #   '1' => {'body' => '1'},
```

```
11 # '2' => {'body' => '1'},
12 # '3' => {'body' => '1'}
13 #}
```

count_range

Return an Array containing all of the keys within a given range.

This method just calls `Cassandra#get_range` and returns the row keys for the records returned.

See `Cassandra#get_range` for options.

get_range_keys

Return an Array containing all of the keys within a given range.

This method just calls `Cassandra#get_range` and returns the row keys for the records returned.

See `Cassandra#get_range` for options.

each_key

Iterate through each key within the given range parameters. This function can be used to iterate over each key in the given column family.

This method just calls `Cassandra#get_range` and yields each row key.

See `Cassandra#get_range` for options.

Example: `10.times do |i| @client.insert(:Statuses, k + i.to_s, {"body-#{i.to_s}" => 'v'}) end`

```
1 @client.each_key(:Statuses) do |key|
2   print key
3 end
4
5 # returns 0123456789
```

each

Iterate through each row within the given column_family.

This method just calls `Cassandra#get_range` and yields the key and columns.

See `Cassandra#get_range` for options.

get_index_slices

This method is used to query a secondary index with a set of provided search parameters

Please note that you can either specify a `CassandraThrift::IndexClause` or an array of hashes with the format as below.

- `column_family` - The Column Family this operation will be run on.
- `index_clause` - This can either be a `CassandraThrift::IndexClause` or an array of hashes with the following keys:
 - `:column_name` - Column to be compared
 - `:value` - Value to compare against
 - `:comparison` - Type of comparison to do.
- `options`
 - `:key_count` - Set maximum number of rows to return. (Only works if `CassandraThrift::IndexClause` is not passed in.)
 - `:key_start` - Set starting row key for search. (Only works if `CassandraThrift::IndexClause` is not passed in.)
 - `:consistency`

Example:

```
1 @client.create_index('Twitter', 'Statuses', 'x', 'LongType')
2
3 @client.insert(:Statuses, 'row1', { 'x' => [0,10].pack("NN") })
4
5 (2..10).to_a.each do |i|
6   @twitter.insert(:Statuses, 'row' + i.to_s, { 'x' => [0,20].pack("NN")
7     , 'non_indexed' => [i].pack('N*') })
8 end
9
10 @client.insert(:Statuses, 'row11', { 'x' => [0,30].pack("NN") })
11
12 expressions = [{:column_name => 'x', :value => [0,20].pack("NN"), :
13   comparison => "=="}]
14
15 # verify multiples will be returned
16 @client.get_indexed_slices(:Statuses, expressions).length #
17   returns 9
18
19 # verify that GT and LT queries perform properly
20 expressions = [
21   { :column_name => 'x',
22     :value => [0,20].pack("NN"),
23     :comparison => "=="},
```

```
21         { :column_name => 'non_indexed',
22           :value => [5].pack("N*"),
23           :comparison => ">" }
24     ]
25
26 @client.get_indexed_slices(:Statuses, expressions).length #
    returns 5
```

batch

Takes a block where all the mutations (inserts and deletions) inside it are queued, and at the end of the block are passed to cassandra in a single batch.

If you don't want to send all the mutations inside the block in a big single batch, you can use the `:queue_size` option to send smaller batches. If the queue is not empty at the end of the block, the remaining mutations are sent.

- options
 - `:consistency` - Override the consistency level from individual mutations.
 - `:queue_size` - Maximum number of mutations to send at once.

Example:

```
1 @client.batch do
2   @client.insert(:Statuses, 'k1', {'body' => 'v1'})
3   @client.insert(:Statuses, 'k2', {'body' => 'v2'})
4   @client.remove(:Statuses, 'k3')
5 end
```

Reporting Problems

The Github issue tracker is [here](#). If you have problems with this library or Cassandra itself, please use the [cassandra-user](#) mailing list.