bmabey / database_cleaner

Strategies for cleaning databases in Ruby. Can be used to ensure a clean state for testing. — Read more

http://gemcutter.org/gems/database_cleaner

multi_orms_connections

rework_ar_connections

unspecified

vagrant

- Files
- Commits
- Branches 7
- Tags 21
- Downloads 0

Latest commit to the **master** branch

releases v0.7.1

commit 4da2ea308c

bmabey authored about 7 hours ago

database_cleaner /

name	age	history message
examples	January 17, 2011	Add support for default strategy for each ORM [sikachu]
features	January 17, 2011	Add support for default strategy for each ORM [sikachu]
lib	January 09, 2012	Fixed Rails 3.2 superclass mismatch error [ddemaree]
spec	December 16, 2011	Don't clean it if nothing can be cleaned. [dskim]
.gitignore	June 15, 2010	finish merge, all specs, all features pass [JonRowe]
.rvmrc	November 13, 2011	adds rvmrc [bmabey]
Gemfile	March 30, 2011	added basic specs [netskin]
Gemfile.lock	March 30, 2011	added basic specs [netskin]
History.txt	about 7 hours ago	adds credit [bmabey]
LICENSE	March 05, 2009	added a readme and bumped the version [bmabey]
README.textile	December 20, 2011	updates README to include new sequel adapter info [bmabey]
Rakefile	June 21, 2011	updates rake task to delete swap files [bmabey]
TODO	June 03, 2010	update TODO [JonRowe]
VERSION.yml	about 7 hours ago	releases v0.7.1 [bmabey]
cucumber.yml	March 04, 2009	cucumber feature and example app done. Got the AR transaction strateg… [bmabey]
database_cleaner.gemspec	about 7 hours ago	releases v0.7.1 [bmabey]

README.textile

Database Cleaner

Database Cleaner is a set of strategies for cleaning your database in Ruby. The original use case was to ensure a clean state during tests. Each strategy is a small amount of code but is code that is usually needed in any ruby app that is testing with a database.

ActiveRecord, DataMapper, Sequel, MongoMapper, Mongoid, and CouchPotato are supported.

Here is an overview of the strategies supported for each library:

ORM	Truncation	Transaction	Deletion
ActiveRecord	Yes	**Yes**	Yes
DataMapper	Yes	**Yes**	No
CouchPotato	**Yes**	No	No
MongoMapper	**Yes**	No	No
Sequel	**Yes**	Yes	No

(Default strategy for each library is denoted in bold)

The ActiveRecord `:deletion` strategy is useful for when the `:truncation` strategy causes locks (as reported by some Oracle DB users). The `:deletion` option has been reported to be faster than `:truncation` in some cases as well. In general, the best approach is to use `:transaction` since it is the fastest.

Database Cleaner also includes a `null` strategy (that does no cleaning at all) which can be used with any ORM library. You can also explicitly use it by setting your strategy to `nil`.

How to use

```
require 'database_cleaner'
DatabaseCleaner.strategy = :truncation

# then, whenever you need to clean the DB
DatabaseCleaner.clean
```

With the :truncation strategy you can also pass in options, for example:

```
DatabaseCleaner.strategy = :truncation, {:only => %w[widgets dogs some_other_table]}

DatabaseCleaner.strategy = :truncation, {:except => %w[widgets]}
```

(I should point out the truncation strategy will never truncate your schema_migrations table.)

Some strategies require that you call DatabaseCleaner.start before calling clean (for example the :transaction one needs to know to open up a transaction). So you would have:

```
require 'database_cleaner'
DatabaseCleaner.strategy = :transaction

DatabaseCleaner.start # usually this is called in setup of a test
dirty_the_db
DatabaseCleaner.clean # cleanup of the test
```

At times you may want to do a single clean with one strategy. For example, you may want to start the process by truncating all the tables, but then use the faster transaction strategy the remaining time. To accomplish this you can say:

```
require 'database_cleaner'
DatabaseCleaner.clean_with :truncation
DatabaseCleaner.strategy = :transaction
# then make the DatabaseCleaner.start and DatabaseCleaner.clean calls appropriately
```

RSpec Example

```
RSpec.configure do |config|

  config.before(:suite) do
    DatabaseCleaner.strategy = :transaction
    DatabaseCleaner.clean_with(:truncation)
  end

  config.before(:each) do
    DatabaseCleaner.start
  end

  config.after(:each) do
    DatabaseCleaner.clean
  end

end
```

Cucumber Example

Add this to your features/support/env.rb file:

```
begin
  require 'database_cleaner'
  require 'database_cleaner/cucumber'
  DatabaseCleaner.strategy = :truncation
rescue NameError
  raise "You need to add database_cleaner to your Gemfile (in the :test group) if you wish to use it."
end
```

A good idea is to create the before and after hooks to use the DatabaseCleaner.start and DatabaseCleaner.clean methods.

Inside features/support/hooks.rb:

```
Before do
  DatabaseCleaner.start
end

After do |scenario|
  DatabaseCleaner.clean
end
```

This should cover the basics of tear down between scenarios and keeping your database clean.
For more examples see the section “Why?”

Common Errors

In rare cases DatabaseCleaner will encounter errors that it will log. By default it uses STDOUT set to the ERROR level but you can configure this to use whatever Logger you desire. Here’s an example of using the Rails.logger in env.rb:

```
  DatabaseCleaner.logger = Rails.logger
```

If you are using Postgres and have foreign key constraints, the truncation strategy will cause a lot of extra noise to appear on STDERR (in
the form of “NOTICE truncate cascades” messages). To silence these warnings set the following log level in your postgresql.conf file:

```
  client_min_messages = warning
```

How to use with multiple ORM’s

Sometimes you need to use multiple ORMs in your application. You can use DatabaseCleaner to clean multiple ORMs, and multiple connections for those ORMs.

```
#How to specify particular orms
DatabaseCleaner[:active_record].strategy = :transaction
DatabaseCleaner[:mongo_mapper].strategy = :truncation

#How to specify particular connections
DatabaseCleaner[:active_record,{:connection => :two}]
```

Usage beyond that remains the same with DatabaseCleaner.start calling any setup on the different configured connections, and DatabaseCleaner.clean executing afterwards.

Configuration options

ORM	How to access	Notes
Active Record	DatabaseCleaner[:active_record]	Connection specified as :symbol keys, loaded from config/database.yml
Data Mapper	DatabaseCleaner[:data_mapper]	Connection specified as :symbol keys, loaded via Datamapper repositories
Mongo Mapper	DatabaseCleaner[:mongo_mapper]	Multiple connections not yet supported
Mongoid	DatabaseCleaner[:mongoid]	Multiple connections not yet supported
Couch Potato	DatabaseCleaner[:couch_potato]	Multiple connections not yet supported
Sequel	DatabaseCleaner[:sequel]	?

Why?

One of my motivations for writing this library was to have an easy way to turn on what Rails calls “transactional_fixtures” in my non-rails ActiveRecord projects. For example, Cucumber ships with a Rails world that will wrap each scenario in a transaction. This is great, but what if you are using ActiveRecord in a non-rails project? You used to have to copy-and-paste the needed code, but with DatabaseCleaner you can now say:

```
#env.rb
require 'database_cleaner'
require 'database_cleaner/cucumber'
DatabaseCleaner.strategy = :transaction
```

Now lets say you are running your features and it requires that another process be involved (i.e. Selenium running against your app’s server.) You can simply change your strategy type:

```
#env.rb
require 'database_cleaner'
require 'database_cleaner/cucumber'
DatabaseCleaner.strategy = :truncation
```

You can have the best of both worlds and use the best one for the job:

```
#env.rb

require ‘database_cleaner’

require ‘database_cleaner/cucumber’

DatabaseCleaner.strategy = (ENV[‘SELENIUM’] == ‘true’) ? :truncation : :transaction
```

COPYRIGHT

Markdown Cheat Sheet

Format Text

Headers

```
# This is an <h1> tag
## This is an <h2> tag
###### This is an <h6> tag
```

Text styles

```
*This text will be italic*
_This will also be italic_
**This text will be bold**
__This will also be bold__

*You **can** combine them*
```

Lists

Unordered

```
* Item 1
* Item 2
  * Item 2a
  * Item 2b
```

Ordered

```
1. Item 1
2. Item 2
3. Item 3
   * Item 3a
   * Item 3b
```

Miscellaneous

Images

```
![GitHub Logo](/images/logo.png)
Format: ![Alt Text](url)
```

Links

```
http://github.com - automatic!
[GitHub](http://github.com)
```

Blockquotes

```
As Kanye West said:

> We're living the future so
> the present is our past.
```

Code Examples in Markdown

Syntax highlighting with GFM

```
```javascript
function fancyAlert(arg) {
  if(arg) {
    $.facebox({div:'#foo'})
  }
}
```
```

Or, indent your code 4 spaces

```
Here is a Python code example
without syntax highlighting:

    def foo:
      if not bar:
        return true
```

Inline code for comments

```
I think you should use an
`<addr>` element here instead.
```

Something went wrong with that request. Please try again. Dismiss