# wayneeseguin / sequel-model



**sequel-model** / lib / **sequel_model.rb**


wayneeseguin 4 years ago
Merge branch 'master' of git@github.com:wayneeseguin/sequel-model

**1** contributor

100644 325 lines (307 sloc) 10.699 kb



```
1  module Sequel
2    class Model
3      alias_method :model, :class
4    end
5  end
6
7  files = %w[
8    base hooks record schema relations
9    caching plugins validations relationships
10 ]
11 dir = File.join(File.dirname(__FILE__), "sequel_model")
12 files.each {|f| require(File.join(dir, f))}
13
14 module Sequel
15   # == Sequel Models
16   #
17   # Models in Sequel are based on the Active Record pattern described by Martin Fowler (http://www.martinfowler.com/eaaCatalog/activeRecord.html). A model class corr
18   #
19   # Model classes are defined as regular Ruby classes:
20   #
21   #   DB = Sequel('sqlite:/blog.db')
22   #   class Post < Sequel::Model
23   #     set_dataset DB[:posts]
24   #   end
25   #
26   # You can also use the shorthand form:
27   #
```

```
28  #   DB = Sequel('sqlite:/blog.db')
29  #   class Post < Sequel::Model
30  #   end
31  #
32  # === Model instances
33  #
34  # Model instance are identified by a primary key. By default, Sequel assumes the primary key column to be :id. The Model#[] method can be used to fetch records by t
35  #
36  #   post = Post[123]
37  #
38  # The Model#pk method is used to retrieve the record's primary key value:
39  #
40  #   post.pk #=> 123
41  #
42  # Sequel models allow you to use any column as a primary key, and even composite keys made from multiple columns:
43  #
44  #   class Post < Sequel::Model
45  #     set_primary_key [:category, :title]
46  #   end
47  #
48  #   post = Post['ruby', 'hello world']
49  #   post.pk #=> ['ruby', 'hello world']
50  #
51  # You can also define a model class that does not have a primary key, but then you lose the ability to update records.
52  #
53  # A model instance can also be fetched by specifying a condition:
54  #
55  #   post = Post[:title => 'hello world']
56  #   post = Post.find {:stamp < 10.days.ago}
57  #
58  # === Iterating over records
59  #
60  # A model class lets you iterate over specific records by acting as a proxy to the underlying dataset. This means that you can use the entire Dataset API to create
61  #
62  #   Post.filter(:category => 'ruby').each {|post| p post}
63  #
64  # You can also manipulate the records in the dataset:
65  #
66  #   Post.filter {:stamp < 7.days.ago}.delete
67  #   Post.filter {:title =~ /ruby/}.update(:category => 'ruby')
68  #
69  # === Accessing record values
70  #
71  # A model instances stores its values as a hash:
72  #
73  #   post.values #=> {:id => 123, :category => 'ruby', :title => 'hello world'}
74  #
75  # You can read the record values as object attributes:
76  #
77  #   post.id #=> 123
78  #   post.title #=> 'hello world'
79  #
80  # You can also change record values:
81  #
82  #   post.title = 'hey there'
83  #   post.save
84  #
85  # Another way to change values by using the #set method:
86  #
87  #   post.set(:title => 'hey there')
88  #
89  # === Creating new records
90  #
91  # New records can be created by calling Model.create:
92  #
93  #   post = Post.create(:title => 'hello world')
94  #
95  # Another way is to construct a new instance and save it:
96  #
97  #   post = Post.new
98  #   post.title = 'hello world'
99  #   post.save
100 #
101 # You can also supply a block to Model.new and Model.create:
102 #
103 #   post = Post.create {|p| p.title = 'hello world'}
104 #
105 #   post = Post.new do |p|
106 #     p.title = 'hello world'
107 #     p.save
108 #   end
109 #
110 # === Hooks
111 #
112 # You can execute custom code when creating, updating, or deleting records by using hooks. The before_create and after_create hooks wrap record creation. The before
113 #
114 # Hooks are defined by supplying a block:
115 #
116 #   class Post < Sequel::Model
117 #     after_create do
118 #       set(:created_at => Time.now)
119 #     end
```

```
120 #
121 #     after_destroy do
122 #       author.update_post_count
123 #     end
124 #   end
125 #
126 # === Deleting records
127 #
128 # You can delete individual records by calling #delete or #destroy. The only difference between the two methods is that #destroy invokes before_destroy and after_d
129 #
130 #   post.delete #=> bypasses hooks
131 #   post.destroy #=> runs hooks
132 #
133 # Records can also be deleted en-masse by invoking Model.delete and Model.destroy. As stated above, you can specify filters for the deleted records:
134 #
135 #   Post.filter(:category => 32).delete #=> bypasses hooks
136 #   Post.filter(:category => 32).destroy #=> runs hooks
137 #
138 # Please note that if Model.destroy is called, each record is deleted separately, but Model.delete deletes all relevant records with a single SQL statement.
139 #
140 # === Associations
141 #
142 # The most straightforward way to define an association in a Sequel model is as a regular instance method:
143 #
144 #   class Post < Sequel::Model
145 #     def author; Author[author_id]; end
146 #   end
147 #
148 #   class Author < Sequel::Model
149 #     def posts; Post.filter(:author_id => pk); end
150 #   end
151 #
152 # Sequel also provides two macros to assist with common types of associations. The one_to_one macro is roughly equivalent to ActiveRecord?'s belongs_to macro. It d
153 #
154 #   class Post < Sequel::Model
155 #     one_to_one :author, :from => Author
156 #   end
157 #
158 #   post = Post.create(:name => 'hi!')
159 #   post.author = Author[:name => 'Sharon']
160 #
161 # The one_to_many macro is roughly equivalent to ActiveRecord's has_many macro:
162 #
163 #   class Author < Sequel::Model
164 #     one_to_many :posts, :from => Post, :key => :author_id
165 #   end
166 #
167 # You will have noticed that in some cases the association macros are actually more verbose than hand-coding instance methods. The one_to_one and one_to_many macro
168 #
169 # === Caching model instances with memcached
170 #
171 # Sequel models can be cached using memcached based on their primary keys. The use of memcached can significantly reduce database load by keeping model instances i
172 #
173 #   require 'memcache'
174 #   CACHE = MemCache.new 'localhost:11211', :namespace => 'blog'
175 #
176 #   class Author < Sequel::Model
177 #     set_cache CACHE, :ttl => 3600
178 #   end
179 #
180 #   Author[333] # database hit
181 #   Author[333] # cache hit
182 #
183 # === Extending the underlying dataset
184 #
185 # The obvious way to add table-wide logic is to define class methods to the model class definition. That way you can define subsets of the underlying dataset, chan
186 #
187 #   class Post < Sequel::Model
188 #     def self.old_posts
189 #       filter {:stamp < 30.days.ago}
190 #     end
191 #
192 #     def self.clean_old_posts
193 #       old_posts.delete
194 #     end
195 #   end
196 #
197 # You can also implement table-wide logic by defining methods on the dataset:
198 #
199 #   class Post < Sequel::Model
200 #     def dataset.old_posts
201 #       filter {:stamp < 30.days.ago}
202 #     end
203 #
204 #     def dataset.clean_old_posts
205 #       old_posts.delete
206 #     end
207 #   end
208 #
209 # This is the recommended way of implementing table-wide operations, and allows you to have access to your model API from filtered datasets as well:
210 #
211 #   Post.filter(:category => 'ruby').clean_old_posts
```

```
212 #
213 # Sequel models also provide a short hand notation for filters:
214 #
215 #   class Post < Sequel::Model
216 #     subset(:old_posts) {:stamp < 30.days.ago}
217 #     subset :invisible, :visible => false
218 #   end
219 #
220 # === Defining the underlying schema
221 #
222 # Model classes can also be used as a place to define your table schema and control it. The schema DSL is exactly the same provided by Sequel::Schema::Generator:
223 #
224 #   class Post < Sequel::Model
225 #     set_schema do
226 #       primary_key :id
227 #       text :title
228 #       text :category
229 #       foreign_key :author_id, :table => :authors
230 #     end
231 #   end
232 #
233 # You can then create the underlying table, drop it, or recreate it:
234 #
235 #   Post.table_exists?
236 #   Post.create_table
237 #   Post.drop_table
238 #   Post.create_table! # drops the table if it exists and then recreates it
239 #
240
241
242 class Model
243
244   # Returns a string representation of the model instance including
245   # the class name and values.
246   def inspect
247     "#<%s @values=%s>" % [self.class.name, @values.inspect]
248   end
249
250   # Defines a method that returns a filtered dataset.
251   def self.subset(name, *args, &block)
252     dataset.meta_def(name) {filter(*args, &block)}
253   end
254
255   # Finds a single record according to the supplied filter, e.g.:
256   #
257   #   Ticket.find :author => 'Sharon' # => record
258   #   Ticket.find {:price == 17} # => Dataset
259   #
260   def self.find(*args, &block)
261     dataset.filter(*args, &block).first
262   end
263
264   # TODO: doc
265   def self.[](*args)
266     args = args.first if (args.size == 1)
267     if args === true || args === false
268       raise Error::InvalidFilter, "Did you mean to supply a hash?"
269     end
270     dataset[(Hash === args) ? args : primary_key_hash(args)]
271   end
272
273   # TODO: doc
274   def self.fetch(*args)
275     db.fetch(*args).set_model(self)
276   end
277
278   # Like find but invokes create with given conditions when record does not
279   # exists.
280   def self.find_or_create(cond)
281     find(cond) || create(cond)
282   end
283
284   ##############################################################################
285
286   # Deletes all records in the model's table.
287   def self.delete_all
288     dataset.delete
289   end
290
291   # Like delete_all, but invokes before_destroy and after_destroy hooks if used.
292   def self.destroy_all
293     dataset.destroy
294   end
295
296   def self.is_dataset_magic_method?(m)
297     method_name = m.to_s
298     Sequel::Dataset::MAGIC_METHODS.each_key do |r|
299       return true if method_name =~ r
300     end
301     false
302   end
303
```

```
304    def self.method_missing(m, *args, &block) #:nodoc:
305      Thread.exclusive do
306        if dataset.respond_to?(m) || is_dataset_magic_method?(m)
307          instance_eval("def #{m}(*args, &block); dataset.#{m}(*args, &block); end")
308        end
309      end
310      respond_to?(m) ? send(m, *args, &block) : super(m, *args)
311    end
312
313    # TODO: Comprehensive description goes here!
314    def self.join(*args)
315      table_name = dataset.opts[:from].first
316      dataset.join(*args).select(table_name.to_sym.ALL)
317    end
318
319    # Returns an array containing all of the models records.
320    def self.all
321      dataset.all
322    end
323  end
324end
```



## 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

## Looking for the GitHub logo?

- **GitHub Logo**

Download

- **The Octocat**

 Download