

PREVIEW ONLY...GET THE FULL COPY FOR US\$9 AT PEEPCODE.COM

PeepCode  
press

\$9

# Rails 2.1

*New features for your applications*

by Ryan Daigle

 **Factorial**

PREVIEW ONLY...GET THE FULL COPY FOR US\$9 AT PEEPCODE.COM

CONTENTS

4 **ActiveRecord**

- 4 Validations
- 6 Query Caching
- 9 Sexy Migrations
- 11 Foxy Fixtures
- 13 Many to Many Associations
- 14 Polymorphic Relationships
- 15 Other Tidbits
- 16 Enhanced Migrations
- 18 Change Table
- 19 Named Scope
- 23 Dirty Objects & Partial Updates
- 26 Timezone Support

33 **ActionController & ActionView**

- 33 Asset Servers
- 36 Asset Caching
- 39 Default Cookie-based Sessions
- 40 Partial Layouts

- 44 RESTful Routing Updates
- 47 rescue\_from Exception Handlers
- 49 HTTP Authentication
- 51 Convenient Access to Helpers

53 **Miscellaneous**

- 53 De-Cruft Your Environment File
- 54 Collection Fixtures
- 55 Rake Tasks
- 59 script/generate Update
- 60 Debugging
- 64 Gem Dependencies
- 66 Git for the Rails Source
- 70 Git for Plugins

72 **Deprecated Features**

- 72 Deprecations from Previous Versions
- 75 Pagination Removed
- 76 acts\_as Plugins
- 77 Database Adapters Removed
- 78 smtp\_settings

## INTRODUCTION

Those who saw David Heinemeier Hansson's (<http://www.loudthinking.com>) keynote (<http://www.bestechvideos.com/2007/10/02/david-heinemeier-hansson-keynote-at-railsconf-2007>) at RailsConf 2007 know that Rails 2.0 doesn't contain waves of extraordinary new features and paradigm shifts. Instead, it focuses on further supporting RESTful web architectures, some small feature maturations and the removal of libraries and features that have been determined not to be essential to Rails.

I've been covering new Rails features as they've been introduced to Rails in the What's New in Edge Rails (<http://ryandaigle.com>) series of articles for several releases now. Most major features have been discussed there, but some have not and some have even changed slightly since the initial feature commit. My hope is that this mini-book will provide developers with a timely handbook that will clearly outline new features and how to use them.

---

The Rails 2.1 features released in June of 2008 are also covered here – as a complimentary update of the original Rails2 mini-book.

---

Ryan Daigle, *Founder*, yFactorial, LLC (<http://yfactorial.com>)

# ActiveRecord

## CHAPTER 1

ActiveRecord is the biggest part of Rails, so it's no surprise that it has received some enhancements for Rails 2.0.

## Validations

### Numerical Validation Becomes Useful

The `validates_numericality_of` method sounds like a really useful validation, but has failed me more often than not. I expect that no longer to be the case with the new options supported in Rails 2.

Here's a quick run-down of these options:

```
validations/numericality.rb
validates_numericality_of :salary,
                          :greater_than => 39999

validates_numericality_of :salary,
                          :greater_than_or_equal_to => 40000

validates_numericality_of :ten,
                          :equal_to => 10

validates_numericality_of :bonus,
                          :less_than => 5000

validates_numericality_of :bonus,
                          :less_than_or_equal_to => 4999

validates_numericality_of :prime,
                          :odd => true

validates_numericality_of :squared,
```

```
:even => true
```

You can even stack the conditions (just make sure they're not in conflict with each other):

```
validates_numericality_of :salary,  
                           :greater_than_or_equal_to => 8000,  
                           :even => true
```

## :allow\_blank

Have you ever used the `:allow_nil` option to skip validation if the particular attribute in question is nil?

```
ar/post.rb  
class Post < ActiveRecord::Base  
  validates_numericality_of :rating, :allow_nil => true  
end  
  
Post.new(:rating => nil).valid?  
#=> true
```

This is useful for optional attributes or for those attributes whose existence is already being validated with `validates_presence_of`. But, for optional attributes that come in from your standard web request, the value is often set to be the empty string (since the form field was submitted with an empty value). The only way to deal with that case used to be by crafting your own `:if` proc:

```
validations/if_proc.rb  
class Post < ActiveRecord::Base  
  
  validates_numericality_of :rating,  
    :if => Proc.new { |post| not post.rating.blank? }  
  
end
```

Now that pattern has been factored out into the `:allow_blank` option, which will skip validation if the attribute is nil, empty or a whitespace string.

```
validations/blank.rb
class Post < ActiveRecord::Base

  validates_numericality_of :rating, :allow_blank => true

end
```

---

The default value for `:allow_blank` is `false`, as it is for the existing `:allow_nil`.

---

## Query Caching

Prior to Rails 2, ActiveRecord didn't have a built-in caching mechanism. You were forced to roll your own instance variable cache like:

```
@cached_all ||= find(:all)
```

or bust out some big guns like memcached (<http://www.danga.com/memcached>).

Starting with Rails 2, ActiveRecord has been enabled with a very simple, but effective, query cache. It's automatically enabled, so you can just sit back and enjoy the speed-up.

How it works is that all select statement results are cached, keyed by the statement itself, so subsequent identical statements don't need to hit the database. The cache is cleared anytime an INSERT,

### CACHE SCOPE

The query cache is a short-lived beast. Because Rails is single-threaded, the cache only lives as long as its host model instance lives. In most cases this is only the scope of the incoming HTTP request. As a result, identical select statements across different requests won't be able to take advantage of the query cache.

UPDATE or DELETE operation is executed even if it doesn't affect the results of cached select statements. This is not a robust, handle every case solution. However, it is a low impact and easy way to provide caching that won't get in your way.

## Disable Caching

In the rare instance when you want to be sure you're going out and fetching the latest results from the database (maybe you have an external process that you know has affected your dataset outside the scope of the ActiveRecord model), you can disable the cache within an `uncached` block. Here's an easy way to provide a `find` method that will always bypass the cache:

```
railsapp/app/models/post.rb
class Post < ActiveRecord::Base

  class << self

    # Force to go out and get the latest from the db
    def force_find(*args)
      uncached { find(*args) }
    end

  end
end
```

Hits on the query cache are clearly denoted in the log files with the `CACHE` line prefix.

With this `Post` definition you could call `force_find` the same way you call `find`:

```
Post.force_find(:all,
               :limit => 5,
               :order => 'created_at DESC')
```

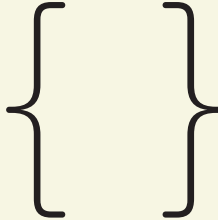
While the query cache is mostly a transparent feature, it's good to know what's happening under the covers of your ActiveRecord mod-

# THE QUERY CACHE

1 A query is created...

```
SELECT *  
FROM users  
WHERE id = 7
```

...the cache is empty...



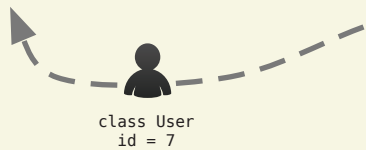
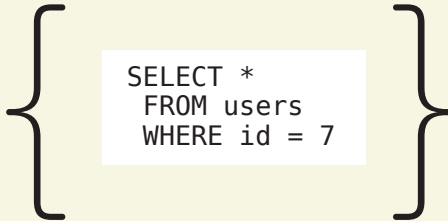
...so the database is searched and the record is cached.



2 Later during that request, the same query is created...

```
SELECT *  
FROM users  
WHERE id = 7
```

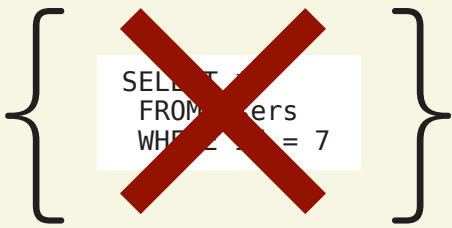
...it is found in the cache, so the User object is returned immediately.



3 Again during that same request, a record is updated...

```
UPDATE orders  
SET (amount = 100)  
WHERE id = 38273
```

...the entire cache is cleared.



els.

## Sexy Migrations

Migrations have their fans and their detractors. Me? Well, I like 'em. And I really like the Sexy Migrations plugin (<http://errtheblog.com/post/2381>) that let you condense your migrations and provide for some syntactical shortcuts. While not a direct port of the plugin, Rails 2 has taken most of the functionality of the plugin and made it part of Rails. Here's the lowdown.

## Condensed Column Declarations

Chances are many of your migrations have a lot of duplication between similarly defined columns:

```
railsapp/db/migrate/001_create_posts.rb
# The old way
class CreatePosts < ActiveRecord::Migration

  # Create a table holding blog posts
  create_table :posts do |t|

    t.column :user_id,      :integer, :null => false
    t.column :category_id, :integer, :null => false
    t.column :body,        :text

    # Standard auto-magic columns
    t.column :created_at,  :datetime
    t.column :updated_at, :datetime
  end

  def self.down
    drop_table :posts
  end
end
```

With the new sexy migrations syntax you can now declare the type of the column first, allowing you to declare multiple columns of the same type in a single line:

```
railsapp/db/migrate/001_create_posts.sexy.rb
class CreatePosts < ActiveRecord::Migration

  # Create a table holding blog posts
  create_table :posts do |t|
    t.integer :user_id, :category_id, :null => false
    t.text    :body
    t.timestamps
  end

  def self.down
    drop_table :posts
  end
end
```

The definition of the `user_id` and `category_id` columns can now be contained in the same line since they have the same definition. You can also get both auto-magic timestamp columns with a single `t.timestamps` call.

And instead of declaring the foreign key columns directly, as in this example, we can use the `references` method for a more intuitive syntax:

```
railsapp/db/migrate/001_create_posts.references.rb
class CreatePosts < ActiveRecord::Migration

  # Create a table holding blog posts
  create_table :posts do |t|
    t.references :user, :category, :null => false
    t.text      :body
    t.timestamps
  end

  def self.down
    drop_table :posts
  end
end
```

Rails2: New features for your applications

©2007 yFactorial, LLC

Every effort was made to provide accurate information in this document. However, neither yFactorial, LLC nor Topfunky Corporation shall have any liability for any errors in the code or descriptions presented in this book.

"Rails" and "Ruby on Rails" are trademarks of David Heinemeier Hansson.

This document is available for US\$9 at PeepCode.com (<http://peepcode.com>). Group discounts and site licenses can also be purchased by sending email to [peepcode@topfunky.com](mailto:peepcode@topfunky.com).

## OTHER PEEPCODE PRODUCTS

- RSpec (<http://peepcode.com/products/rspec-basics>) – A three part series on the popular behavior-driven development framework.
- Rails from Scratch (<http://peepcode.com>) – Learn Rails!
- RESTful Rails (<http://peepcode.com/products/restful-rails>) – Teaches the concepts of application design with REST.
- Subscription pack of 10 (<http://peepcode.com/products/subscription-pack-of-10>) – Save money! Buy 10 PeepCode credits.
- Javascript with Prototype (<http://peepcode.com/products/javascript-with-prototypejs>) – Code confidently with Javascript!
- Rails Code Review PDF (<http://peepcode.com/products/draft-rails-code-review-pdf>) – Common mistakes in Rails applications, and how to fix them.