

has_many :bugs, :through => :rails (/)

Active Record Query Interface 3.0
Published almost 2 years ago

I've been working on revamping the Active Record query interface for the last few weeks (while taking some time off in India from consulting work, before [joining 37signals](http://37signals.com/svn/posts/2068-pratik-naik-joins-37signals) (<http://37signals.com/svn/posts/2068-pratik-naik-joins-37signals>), building on top of [Emilio's](http://www.miloops.com) (<http://www.miloops.com>) GSOC project of integrating [ARel](http://github.com/rails/arel) (<http://github.com/rails/arel>) and ActiveRecord. So here's an overview of how things are going to work in Rails 3.

What's going to be deprecated in Rails 3.1 ?

These deprecations will be effective in Rails' 3.1 release (NOT Rails 3) and will be fully removed in Rails 3.2, though there will be an official plugin to continue supporting them. Consider this an advance warning as it involves changing a lot of code.

In short, passing *options* hash containing `:conditions`, `:include`, `:joins`, `:limit`, `:offset`, `:order`, `:select`, `:readonly`, `:group`, `:having`, `:from`, `:lock` to any of the *ActiveRecord* provided class methods, is now deprecated.

Going into details, currently ActiveRecord provides the following finder methods :

- `find(id_or_array_of_ids, options)`
- `find(:first, options)`
- `find(:all, options)`
- `first(options)`
- `all(options)`
- `update_all(updates, conditions, options)`

And the following calculation methods :

- `count(column, options)`
- `average(column, options)`
- `minimum(column, options)`
- `maximum(column, options)`
- `sum(column, options)`
- `calculate(operation, column, options)`

Starting with Rails 3, **supplying any option to the methods above will be deprecated**. Support for supplying options will be removed from Rails 3.2. Moreover, `find(:first)` and `find(:all)` (without any options) are also being deprecated in favour of `first` and `all`. A tiny little exception here is that *count()* will still accept a `:distinct` option.

The following shows a few example of the deprecated usages :

```
User.find(:all, :limit => 1)
User.find(:all)
User.find(:first)
User.first(:conditions => {:name => 'lifo'})
User.all(:joins => :items)
```

But the **following is NOT deprecated** :

```
User.find(1)
User.find(1,2,3)
User.find_by_name('lifo')
```

Additionally, supplying options hash to `named_scope` is also deprecated :

```
named_scope :red, :conditions => { :colour => 'red' }
named_scope :red, lambda { |colour| { :conditions => { :colour => colour } } }
```

Supplying options hash to `with_scope`, `with_exclusive_scope` and `default_scope` has also been deprecated :

```
with_scope(:find => { :conditions => { :name => 'lifo' } }) { ... }
with_exclusive_scope(:find => { :limit => 1 }) { ... }
default_scope :order => "id DESC"
```

Dynamic `scoped_by_` are also going to be deprecated :

```
red_items = Item.scoped_by_colour('red')
red_old_items = Item.scoped_by_colour_and_age('red', 2)
```

New API

ActiveRecord in Rails 3 will have the following new finder methods.

- `where (:conditions)`
- `having (:conditions)`
- `select`
- `group`
- `order`

- `limit`
- `offset`
- `joins`
- `includes (:include)`
- `lock`
- `readonly`
- `from`

¹ (**#fnr1**) Value in the bracket (if different) indicates the previous equivalent finder option.

Chainability

All of the above methods returns a *Relation*. Conceptually, a *relation* is very similar to [an anonymous named scope \(http://railscasts.com/episodes/112-anonymous-scopes\)](http://railscasts.com/episodes/112-anonymous-scopes). All these methods are defined on the *Relation* object as well, making it possible to chain them.

```
lifo = User.where(:name => 'lifo')
new_users = User.order('users.id DESC').limit(20).includes(:items)
```

You could also apply more finders to the existing relations :

```
cars = Car.where(:colour => 'black')
rich_ppls_cars = cars.order('cars.price DESC').limit(10)
```

Quacks like a Model

A relation quacks just like a model when it comes to the primary CRUD methods. You could call any of the following methods on a relation :

- `new(attributes)`
- `create(attributes)`
- `create!(attributes)`
- `find(id_or_array)`
- `destroy(id_or_array)`
- `destroy_all`
- `delete(id_or_array)`
- `delete_all`
- `update(ids, updates)`
- `update_all(updates)`
- `exists?`

So the following code examples work as expected :

```
red_items = Item.where(:colour => 'red')
red_items.find(1)
item = red_items.new
item.colour #=> 'red'
```

```
red_items.exists? #=> true
red_items.update_all :colour => 'black'
red_items.exists? #=> false
```

Note that calling any of the update or delete/destroy methods would reset the relation, i.e delete the cached records used for optimizing methods like `relation.size`.

Lazy Loading

As it might be clear from the examples above, relations are loaded lazily – i.e you call an enumerable method on them. This is very similar to how associations and named_scopes already work.

```
cars = Car.where(:colour => 'black') # No Query
cars.each {|c| puts c.name } # Fires "select * from cars where ..."
```

This is very useful along side fragment caching. So in your controller action, you could just do :

```
def index
  @recent_items = Item.limit(10).order('created_at DESC')
end
```

And in your view :

```
<% cache('recent_items') do %>
  <% @recent_items.each do |item| %>
    ...
  <% end %>
<% end %>
```

In the above example, `recent_items` are loaded on `@recent_items.each` call from the view. As the controller doesn't actually fire any query, fragment caching becomes more effective without requiring any special work arounds.

Force loading – all, first & last

For the times you don't need lazy loading, you could just call *all* on the relation :

```
cars = Car.where(:colour => 'black').all
```

It's important to note that *all* returns an *Array* and not a *Relation*. This is similar to how things work in Rails 2.3 with *namedscopes* and associations.

Similarly, *first* and *last* will always return an *ActiveRecord* object (or *nil*).

```
cars = Car.order('created_at ASC')
oldest_car = cars.first
newest_car = cars.last
```

named_scope → scopes

Using the method `named_scope` is deprecated in Rails 3.0. But the only change you'll need to make is to remove the `"named_"` part. Supplying finder options hash will be deprecated in Rails 3.1.

`named_scope` have now been renamed to just **scope** (<http://github.com/rails/rails/commit/d60bb0a9e4be2ac0a9de9a69041a4ddc2e0cc914>).

So a definition like :

```
class Item
  named_scope :red, :conditions => { :colour => 'red' }
  named_scope :since, lambda { |time| { :conditions => ["created_at > ?", time] } }
end
```

Now becomes :

```
class Item
  scope :red, :conditions => { :colour => 'red' }
  scope :since, lambda { |time| { :conditions => ["created_at > ?", time] } }
end
```

However, as using options hash is going to be deprecated in 3.1, you should write it using the new finder methods :

```
class Item
  scope :red, where(:colour => 'red')
  scope :since, lambda { |time| where("created_at > ?", time) }
end
```

Internally, named scopes are built on top of *Relation*, making it very easy to mix and match them with the finder methods :

```
red_items = Item.red
available_red_items = red_items.where("quantity > ?", 0)
old_red_items = Item.red.since(10.days.ago)
```

Model.scoped

If you want to build a complex relation/query, starting with a blank relation, *Model.scoped* is what you would use.

```
cars = Car.scoped
rich_ppls_cars = cars.order('cars.price DESC').limit(10)
white_cars = cars.where(:colour => 'red')
```

Speaking of internals, *ActiveRecord::Base* has the following delegations :

```
delegate :find, :first, :last, :all, :destroy, :destroy_all, :exists?, :delete, :delete_all, :update, :update_all, :to => :scoped
delegate :select, :group, :order, :limit, :joins, :where, :preload, :eager_load, :includes, :from, :lock, :readonly, :having, :to => :s
delegate :count, :average, :minimum, :maximum, :sum, :calculate, :to => :scoped
```

The above might give you a better insight on how ActiveRecord is doing things internally. Additionally, dynamic finder methods `find_by_name`, `find_all_by_name_and_colour` etc. are also delegated to *Relation*.

with_scope and with_exclusive_scope

`with_scope` and `with_exclusive_scope` are now implemented on top of *Relation* as well. Making it possible to use any relation with them :

```
with_scope(where(:name => 'lifo')) do
  ...
end
```

Or even use a named scope :

```
with_exclusive_scope(Item.red) do
  ...
end
```

That's all. Please open a [lighthouse ticket](https://rails.lighthouseapp.com/projects/8994-ruby-on-rails/overview) (<https://rails.lighthouseapp.com/projects/8994-ruby-on-rails/overview>) if you find a bug or have a patch for an improvement!

UPDATE 1 : Added information about deprecating `scoped_by_` dynamic methods.

UPDATE 2 : Added information about deprecating `default_scope` with finder options.

(#)

Add New Comment

Type your comment here.

Post as ...

Showing 20 comments

Sort by **Newest first**   [Subscribe by email \(#\)](#)  [Subscribe by RSS \(http://monkeytest.disqus.com/has_many_bugs_through_rails_78/latest.rss\)](http://monkeytest.disqus.com/has_many_bugs_through_rails_78/latest.rss)

<http://disqus.com/facebook-627405995/> **Sigidi Vezubuhle** (<http://www.facebook.com/sigidi>) **1 day ago** (#comment-370227179)

Thanks a lot, i was looking for the order('created_at DESC')



<http://disqus.com/guest/7c780f5d2e2a3ead9519c05296bc33b4/> **Explorer** **1 month ago** (#comment-336448324)

Thanks for the heads up.

<http://disqus.com/yahoo-UPOAZZ7UH5HC66VDXCAUSH6TEE/> **RealAmerican** (<http://pulse.yahoo.com/UPOAZZ7UH5HC66VDXCAUSH6TEE>) **2 months ago** (#comment-302608876)



I've spent hours of finding the right syntax for where(:condition)... thanks a lot for your article.

2 people liked this.  

<http://disqus.com/twitter-204939517/> **BaiYun Group** (<http://twitter.com/baiyungroup>) **3 months ago** (#comment-292518889)

Very helpful. I found the solution here where I worked in rails 3.1. I think the guide in below need to update with this as well.

<http://guides.rubyonrails.org/...> (http://guides.rubyonrails.org/active_record_querying.html)

1 person liked this.  



<http://disqus.com/facebook-1174283883/> **Adam Stankiewicz** (<http://www.facebook.com/adam.stankiewicz>) **3 months ago** (#comment-287973586)

Also use of :default_scope is highly discouraged (double post, had to edit to something clever :P)

<http://disqus.com/facebook-1174283883/> **Adam Stankiewicz** (<http://www.facebook.com/adam.stankiewicz>) **3 months ago** (#comment-287973562)

It seems that we should not use 'with_scope' and 'with_exclusive_scope' for a 'scoped' and 'unscoped' methods (the previous ones are protected). See: <https://github.com/rails/rails...> (<https://github.com/rails/rails/commit/bd1666ad1de88598ed6f04ceffb8488a77be4385>)

1 person liked this.  

<http://disqus.com/guest/81288118362a0a0fd558313ec12763bd/> **Paul Gibler** **3 months ago** (#comment-280611792)

I am not sure if this is an error or if I'm misunderstanding something, but under your chainability section, you have the following code:

```
lifo = User.where(name => 'lifo')
new_users = User.order('users.id http://users.id DESC').limit(20).includes(:items)
Shouldn't that be this?
```

```
lifo = User.where(name => 'lifo')
new_users = lifo.order('users.id http://users.id DESC').limit(20).includes(:items)
```

I'm very confused, let me know!

<http://disqus.com/guest/217432574e6a0cb056593495d887fd3c/> **Mike Bailey** **4 months ago** (#comment-243302718)

The following expression results in "select * from cars" to be run.

```
cars = Car.scoped
```

This surprised me given it's suggested we should use this to create a blank relation.

How do I create a relation without reading the whole table?

1 person liked this.  

<http://disqus.com/guest/7d96fe54de0fba939594d5c62c759f55/> **Ming** **4 months ago** (#comment-238803169)

Awesome article. Thank you for putting this together!

<http://disqus.com/guest> **Andrei Lupusoru** (<http://www.skillful.ro>) **7 months ago** (#comment-188765433)

[/e0589c444ba34829db9f26e0b1b1a909/](#)

good stuff..keep on doing it:)

Like Reply

<http://disqus.com/guest/83329afd0b2029dd65ec0c5161062e8/> **nick** 7 months ago (#comment-181857597)

Very good article. Thanks

1 person liked this. Like Reply

<http://disqus.com/twitter-75077983/> **Aji** (http://twitter.com/l___j) 8 months ago (#comment-160420391)

:D :D :D

1 person liked this. Like Reply

<http://disqus.com/guest/f8400f35b6cf33562c04afb65c1f0cf/> **Qw23ert** 9 months ago (#comment-138838636) *in reply to Allam Matsubara (comment-131152342)*

Thanks a lot. Its very helpful

1 person liked this. Like Reply

<http://disqus.com/guest/311631724e85a3fc4975e7a8cf167cdb/> **Allam Matsubara** 10 months ago (#comment-131152342)

Great stuff! It helped me a lot!

Thanks!

1 person liked this. Like Reply

<http://disqus.com/guest/9db0726cf833ae6b8a04878317388627/> **zoekmachine optimalisatie** (<http://www.kiwi-online.nl>) 10 months ago (#comment-123545519)

Very nice, you a talented man as I also read on the 37signals blog :)

Like Reply

<http://disqus.com/guest/d9c8f9fdb9724c3c737f0baa06c39b4/> **MinalJain** 11 months ago (#comment-111490136)

Very good stuff. It is nice to know about the RAILS 3 activerecord depricated and newly added methods. It definitely help me in future.

Thanks a lot !!

2 people liked this. Like Reply

<http://disqus.com/guest/0e80fb3e0b2b67abd42d8815434febf/> **Abc** 11 months ago (#comment-111489984)

Very good stuff. It is nice to know about the RAILS 3 activerecord depricated and newly added methods. It definitely help me in future.

Thanks a lot !!

1 person liked this. Like Reply

<http://disqus.com/guest/3b372a5f9676793240da77c9fde82db9/> **vdw** (<http://www.vdw-subsidiedesk.nl>) 12 months ago (#comment-104630224)

Thank you for the warning it will save me a lot of time in the future.

1 person liked this. Like Reply

<http://disqus.com/guest/b28d52a0438c3c539cb8740c067d69ac/> **Inchirieri masini Baneasa** (<http://www.inchirieri-masini-bucuresti.ro>) 12 months ago (#comment-104058289)

There are certainly a lot of details like that to take into consideration. That is a great point to bring up. I offer the thoughts above as general inspiration but clearly there are questions like the one you bring up where the most important thing will be working in honest good faith. I don't know if best practices have emerged around things like that, but I am sure that your job is clearly identified as a fair game.

6 people liked this. Like Reply

<http://disqus.com/tylerhunt/> **Tyler Hunt** (<http://tylerhunt.com/>) 1 year ago (#comment-99534241)

It seems to me that the `find_by_...` methods *should* be deprecated in favor of the ability to use `by_...` on a relation. For example, instead of `User.find_by_name('lifo')`, you could do `User.by_name('lifo').first`, and instead of `User.find_all_by_name('lifo')`, you could do `User.by_name('lifo').all`. This would also obviate the need for things like `find_by_...and_...`, allowing you to chain multiple `by_...` methods, like `User.by_name('lifo').by_email('lifo@example.com').first`.

14 people liked this. Like Reply

All rights reserved ©2011 Pratik Naik



<http://feeds.feedburner.com/monkeyonrails>