ActiveRecord

THIS SOUNDS WEIRD

ActiveRecord is an object-relational mapping layer that is responsible for interoperability between the application and the database and for abstracting data.

O Active Record é uma camada de mapeamento objeto-relacional (object-relational mapping layer), responsável pela interoperabilidade entre a aplicação e o banco de dados e pela abstração dos dados.

(wikipedia)

New options for assocations: :validate

A new option for associations has been added to Rails. If you include the `:validate => false` option on an association, `ActiveRecord` will save the data of the parent object without validating the associated objects. For example:

```
class AuditLog < ActiveRecord::Base
  belongs_to :developer, :validate => false
end

log = AuditLog.create(:developer_id => 0 , :message => "")
log.developer = Developer.new

puts log.developer.valid?
# => false

puts log.valid?
# => true

puts log.save
# => true
```

Note that even though the association is not valid, the object **log** gets saved.

The default value is `false`, that is, all validations in `belongs_to` associations will be disconnected by default and if you want to connect them you need to use `:validate => true`.

A new way of specifying conditions with a Hash

When performing database queries, sometimes you need to use the `:joins` option, either to improve application performance or when you need to retrieve information that depends on results from more than one table.

For example, if you wanted to retrieve all users who bought red items, you could do something like this:

```
User.all :joins => :items, :conditions => ["items.color = ?", 'red']
```

This syntax is a little painful, since you need to include the name of the table (**items** in this case) inside a **string**. The code seems weird.

In Rails 2.2 there is a new way of doing this, using a hash key to identify the table:

```
User.all :joins => :items, :conditions => {
  :age => 10,
  :items => { :color => 'red' }
}

# another example that perhaps makes the code a little clearer
User.all :joins => :items, :conditions => {
  :users => { :age => 10 },
  :items => { :color => 'red' }
}
```

In my opinion, the code is a lot clearer this way, especially when you need to provide conditions for many fields from various tables.

Just keep in mind that the key you use is the (plural) name of the table or an alias, if you have specified one in the query.

New :from option for calculation methods in ActiveRecord

A new option has been included in the calculation methods of `ActiveRecord` (`count`, `sum`, `average`, `minimum` and `maximum`).

When you use the `:from` option, you can override the name of the table in the query that `ActiveRecord` generates, which doesn't seem very useful at first glance, but one interesting thing that this allows you to do is force MySQL to use a specific index for the calculation.

Check out some examples:

```
# Forcing MySQL to use an index for a calculation
Edge.count :all, :from => 'edges USE INDEX(unique_edge_index)',
           :conditions => 'sink_id < 5')

# Doing the calculation in a different table from the associated class
Company.count :all, :from => 'accounts'
```

The merge_conditions method in ActiveRecord is now public

The `merge_conditions` method in `ActiveRecord` is now a public method, which means that it will be available in all your **Models**.

This method does exactly what its name says—it allows you provide many different **conditions** in your parameters that get combined into a single condition.

```
class Post < ActiveRecord::Base
end

a = { :author => 'Carlos Brando' }
b = [ 'title = ?', 'Edge Rails' ]

Post.merge_conditions(a, b)
# => "(\"posts\".\"author\" = 'Carlos Brando') AND (title = 'Edge Rails')"
```

Note that the **conditions** always get combined with **AND**.

Defining how validates_length_of should function

The `validates_length_of` is one of many validation methods in `ActiveRecord`. This particular method is for making sure that the value stored in a given database column has a certain length. It lets you specify a maximum length, a minimum length, an exact length, or even a range of lengths.

"Length," however, is relative. When we say "length" these days we are talking about the number of characters in the text. But imagine a case where you have a form field in which the limit is not defined by the number of characters, but by the number of words, such as, "Write a paragraph with at least 100 words." (I think a better example would be "Submit a comment with no more than 100 words.") Imagine a page where the user must submit an essay, for example.

Before Rails 2.2, our only option would be to create a custom validation method, but now you can personalize `validates_length_of` using the `:tokenizer` option. The following example resolves the aforementioned problem:

```
validates_length_of :essay,
                    :minimum => 100,
                    :too_short => "Sua redação deve ter no mínimo %d palavras."),
                    :tokenizer => lambda {|str| str.scan(/\w+/) }
```

This is just one example of what you can do with this new option. In addition to this, you can use it to count only the number of digits, the times a certain word was used, etc.

ActiveSupport

Active Support is a collection of useful classes and default libraries extensions which were considered useful for Ruby on Rails Applications. (wikipedia)

ActiveResource

ActiveResource is a layer responsible by the client side implementation of RESTful systems. Through ActiveResource is possible to consume RESTful services by using objects that work like a proxy for remote services.

ActionPack

Comprises ActionView (visualization generation for end user, like HTML, XML, JavaScript, and etc) and ActionController (business flow control) (adapted from wikipedia)

ActionController

ActionController is the layer responsible by receiving web requests and taking decisions of what is going to be run and rendered or to redirect the request to another action. An Action is defined as public methods within controllers which are automatically available through routes.

ActionView

ActionView is the layer responsible by the generation of the viewable interface visible to users through conversion of ERB templates.

Railties

Rake Tasks, Plugins and Scripts

Prototype and script.aculo.us

Ruby 1.9

Debug

Bugs and Fixes

Additional Information

CHANGELOG