Making business beautiful.

Cheatsheet ✻ Validations

Class convenience methods

◆validates_acceptance_of

Encapsulates the pattern of wanting to validate the acceptance of a terms of service check box (or similar agreement).

```
  validates_acceptance_of :terms_of_service
  validates_acceptance_of :eula, :message => "must be abided"
```

If the database column does not exist, the `terms_of_service` attribute is entirely virtual. This check is performed only if `terms_of_service` is not `nil` and by default on `save`.

`:accept`	Specifies value that is considered accepted. The default value is a string `"1"`, which makes it easy to relate to an HTML checkbox. This should be set to `true` if you are validating a database column, since the attribute is typecast from `"1"` to `true` before validation.

◆validates_associated

Validates whether the associated object or objects are all valid themselves. Works with any kind of association.

```
  has_many :pages
  belongs_to :library
  validates_associated :pages, :library
```

Warning! If, after the above definition, you then wrote...

```
  belongs_to :book
  validates_associated :book
```

...this would specify a circular dependency and cause infinite recursion.

This validation will not fail if the association hasn't been assigned. If you want to ensure that the association is both present and guaranteed to be valid, you also need to use `validates_presence_of`

◆validates_confirmation_of

Encapsulates the pattern of wanting to validate a password or email address field with a confirmation. Example:

Model

```
class Person < ActiveRecord::Base
  validates_confirmation_of :user_name, :password
  validates_confirmation_of :email_address, :message => "should
match confirmation"
end
```

View

```
 <%= password_field "person", "password" %>
 <%= password_field "person", "password_confirmation" %>
```

The added `password_confirmation` attribute is virtual: it exists only as an in-memory attribute for validating the password. To achieve this, the validation adds accessors to the model for the confirmation attribute. This check is performed only if `password_confirmation` is not `nil`, and by default only on `save`. To require confirmation, make sure to add a presence check for the confirmation attribute:

```
  validates_presence_of :password_confirmation, :if => :password_changed
```

◆validates_each

Validates each attribute against a block.

```
  validates_each :first_name, :last_name do |record, attr, value|
    record.errors.add attr, 'starts with z.' if value[0] == ?z
  end
```

◆validates_exclusion_of

Validates that the value of the specified attribute is not in a particular enumerable object.

```
  validates_exclusion_of :username, :in => %w( admin superuser ), :message
=> "You don't belong here"
  validates_exclusion_of :age, :in => 30..60, :message => "This site is
only for under 30 and over 60"
  validates_exclusion_of :format, :in => %w( mov avi ), :message => "exten-
sion %s is not allowed"
```

`:in`	An enumerable object of items that the value shouldn't be part of

◆validates_format_of

Validates whether the value of the specified attribute is of the correct form by matching it against the regular expression provided.

```
  validates_format_of :email, :with => /\A([^@\s]+)@((?:[-a-z0-9]+\.)+[a-z]
{2,})\Z/i, :on => :create
```

Note: use `\A` and `\Z` to match the start and end of the string, `^` and `$` match the start/end of a line.

A regular expression must be provided or else an exception will be raised.

`:with`	The regular expression used to validate the format with

◆validates_inclusion_of

Validates whether the value of the specified attribute is available in a particular enumerable object.

```
  validates_inclusion_of :gender, :in => %w( m f ), :message => "woah! what
are you then!??!!"
  validates_inclusion_of :age, :in => 0..99
  validates_inclusion_of :format, :in => %w( jpg gif png ), :message =>
"extension %s is not included in the list"
```

`:in`	An enumerable object of items that the value shouldn't be part of

◆validates_length_of

Validates that the specified attribute matches the length restrictions supplied. Only one option can be used at a time:

```
  validates_length_of :first_name, :maximum => 30
  validates_length_of :last_name, :maximum => 30, :message=>"less than %d
if you don't mind"
  validates_length_of :fax, :in => 7..32, :allow_nil => true
  validates_length_of :phone, :in => 7..32, :allow_blank => true
  validates_length_of :user_name, :within => 6..20, :too_long => "pick a
shorter name", :too_short => "pick a longer name"
  validates_length_of :fav_bra_size, :minimum => 1, :too_short => "please
enter at least %d character"
```

Common options

`:message`	a custom error message
`:on`	specifies when this validation is active (default is `:save`, other options `:create, :update`)
`:allow_nil`	skip validation if attribute is `nil` (default is `false`). Notice that for `fixnum` and `float` columns empty strings are converted to `nil`
`:allow_blank`	if set to `true`, skips this validation if the attribute is blank (default is `false`)
`:if`	specifies a method, `proc` or `"string"` to call to determine if the validation should occur (e.g. `:if => :allow_validation`, or `:if => Proc.new { |user| user.signup_step > 2 }`). The method, `proc` or string should return or evaluate to a `true` or `false` value.
`:unless`	specifies a method, `proc` or `"string"` to call to determine if the validation should not occur (e.g. `:unless => :skip_validation`, or `:unless => Proc.new { |user| user.signup_step <= 2 }`). The method, `proc` or string should return or evaluate to a `true` or `false` value.

```
  validates_length_of :smurf_leader, :is => 4, :message => "papa is
spelled with %d characters... don't play me."
  validates_length_of :essay, :minimum => 100, :too_short => "Your essay
must be at least %d words."), :tokenizer => lambda {|str| str.scan(/\
w+/) }
```

`:minimum`	The minimum size of the attribute
`:maximum`	The maximum size of the attribute
`:is`	The exact size of the attribute
`:within`	A range specifying the minimum and maximum size of the attribute
`:in`	Alias for `:within`
`:too_long`	The error message if the attribute goes over the maximum (default is: `"is too long (maximum is %d characters)"`)
`:too_short`	The error message if the attribute goes under the minimum (default is: `"is too short (min is %d characters)"`)
`:wrong_length`	The error message if using the `:is` method and the attribute is the wrong size (default is: `"is the wrong length (should be %d characters)"`)
`:tokenizer`	Specifies how to split up the attribute string. (e.g. `:tokenizer => lambda {|str| str.scan(/\w+/)}` to count words as in above example.) Defaults to `lambda{ |value| value.split(//) }` which counts individual characters

◆validates_numericality_of

Validates whether the value of the specified attribute is numeric by trying to convert it to a float with `Kernel.Float` (if `only_integer` is `false`) or applying it to the regular expression `/\A[+\-]?\d+\Z/` (if `only_integer` is set to `true`).

```
  validates_numericality_of :age, :greater_than => 18
```

`:only_integer`	Specifies whether the value has to be an integer, e.g. an integral value (default is `false`)
`:greater_than`	Specifies the value must be greater than the supplied value
`:greater_than_or_equal_to`	Specifies the value must be greater than or equal the supplied value
`:equal_to`	Specifies the value must be equal to the supplied value
`:less_than`	Specifies the value must be less than the supplied value
`:less_than_or_equal_to`	Specifies the value must be less than or equal the supplied value
`:odd`	Specifies the value must be an odd number
`:even`	Specifies the value must be an even number

◆validates_presence_of

Validates that the specified attributes are not blank (as defined by `blank?`). Happens by default on `save`.

```
  validates_presence_of :first_name
```

The `first_name` attribute must be in the object and it cannot be blank.

If you want to validate the presence of a boolean field (where the real values are `true` and `false`), you will want to use `validates_inclusion_of :field_name, :in => [true, false]` This is due to the way `blank?` handles boolean values.

◆validates_size_of

Alias for `validates_length_of`

◆validates_uniqueness_of

Validates whether the value of the specified attributes are unique across the system. Useful for making sure that only one user can be named "davidhh".

```
  validates_uniqueness_of :user_name
```

It can also validate whether the value of the specified attributes are unique based on multiple scope parameters. For example, making sure that a teacher can only be on the schedule once per semester for a particular class.

```
  validates_uniqueness_of :teacher_id, :scope => [:semester_id, :class_
id]
```

When the record is created, a check is performed to make sure that no record exists in the database with the given value for the specified attribute (that maps to a column). When the record is updated, the same check is made but disregarding the record itself.

Because this check is performed outside the database there is still a chance that duplicate values will be inserted in two parallel transactions. To guarantee against this you should create a unique index on the field. See `add_index` for more information.

`:scope`	One or more columns by which to limit the scope of the uniqueness constraint
`:case_sensitive`	Looks for an exact match. Ignored by non-text columns (`false` by default)

SOME RIGHTS RESERVED

This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivs 2.0 License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-nd/2.0/uk

Low level validations

You can create your own validations by implementing the `validate` method (or the variations, `validate_on_create` and `validate_on_update`) in your model. The `validate` method will be called by ActiveRecord before every save or update, whilst the `validate_on_create` and `validate_on_update` methods will only be called when saving new records or updating existing records respectively. Most validation methods simply do two things:

- Inspect the attributes of an object and check whether their values pass one or more conditions (such as making sure a *name* attribute is not empty and/or that it matches a certain regular expression)
- If an attribute's value does not pass, add an error message to it

Errors

Each ActiveRecord object has its own `errors` object which stores a list of errors related to that object and is accessible via the `errors` method on the object.

The `errors` object itself has a number of methods which enable you to both add to the list of errors, or inspect the object for errors which have already been added. For example, to add an error to the name attribute of an object, we would use the add method...

```
  object.errors.add(:name, "shouldn't be empty!")
```

...and to retrieve errors on the :name attribute we could read from the errors object...

```
  object.errors.on(:name)    # => "shouldn't be empty!"
```

You will find our cheatsheet **ActiveRecord Validation Errors** a useful resource for manipulating the `errors` object.

Here are some examples of implementing low-level, roll-your-own validation methods...

◆Example

```
class Person < ActiveRecord::Base
  protected
    def validate
      errors.add_on_empty %w( first_name last_name )
      errors.add("phone_number", "has invalid format") unless phone_number
=~ /[0-9]*/
    end

    def validate_on_create # is only run the first time a new object is
saved
      unless valid_discount?(membership_discount)
        errors.add("membership_discount", "has expired")
      end
    end

    def validate_on_update
      errors.add_to_base("No changes have occurred") if unchanged_attri-
butes?
    end
end

person = Person.new("first_name" => "David", "phone_number" => "what?")
person.save                         # => false (and doesn't do the save)
person.errors.empty?                # => false
person.errors.count                 # => 2
person.errors.on "last_name"        # => "can't be empty"
person.errors.on "phone_number"     # => "has invalid format"
person.errors.each_full { |msg| puts msg }
                                    # => "Last name can't be empty\n" +
                                         "Phone number has invalid format"

person.attributes = { "last_name" => "Heinemeier", "phone_number" => "555-
555" }
person.save # => true (and person is now saved in the database)
```

SOME RIGHTS RESERVED
This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivs 2.0 License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-nd/2.0/uk