ActiveRecord Basics

Sang Shin
Michèle Garoche
www.javapassion.com
"Learn with Passion!"

Topics

- What is Active Record?
- Active Record Object Creation
- Find operation
- Dynamic Attribute-based Finders
- Validation
- Migration
- Callbacks
- Exception Handling

What is Active Record?

What is Active Record?

- Active Record is a Ruby library that provides mapping between business objects and database tables
  - Accessing, saving, creating, updating operations in your Rails code are performed by Active Record
- It‘s an implementation of the object-relational mapping (ORM) pattern by the same name as described by Martin Fowler:
  - "An object that wraps a row in a database table or view, encapsulates the database access, and adds domain logic on that data."
- Contains domain logic

Major Features

- Automated mapping between classes and tables, attributes and columns.
- Callbacks as methods on the entire lifecycle (instantiation, saving, destroying, validating, etc).
  - Observers for the entire lifecycle
- Transaction support on both a database and object level.
- Reflections on columns, associations, and aggregations
- Database abstraction through simple adapters

Major Features (Continued)

- Associations between objects controlled by simple meta-programming macros.
- Aggregations of value objects controlled by simple meta-programming macros.
- Inheritance hierarchies

Automatic Mapping between Classes and Tables

Mapping between Classes and Tables

- Class definition below is automatically mapped to the table named “products”

```
class Product < ActiveRecord::Base
end
```

- Schema of “products” table

```
CREATE TABLE products (
  id int(11) NOT NULL auto_increment,
  name varchar(255),
  PRIMARY KEY  (id)
 );
```

Active Record Definition & Domain Logic

- Active Record typically contain domain logic

```
class Product < ActiveRecord::Base
    def my_business_method
        # Whatever business logic
    end
end
```

Attributes

- Active Record objects don‘t specify their attributes directly, but rather infer them from the table definition with which they‘re linked.
  - That is why you don't find attribute definitions in the Active Record class
- You can have getter and setter methods of attributes, however

ActiveRecord Object Creation, Update, Delete

ActiveRecord Objects Can Be Created in 3 Different Ways

- Constructor parameters in a hash

```
user = User.new(:name => "David",
                :occupation => "Artist")
```

- Use block initialization

```
user = User.new do |u|
  u.name = "David"
  u.occupation = "Code Artist"
end
```

- Create a bare object and then set attributes

```
user = User.new
user.name = "David"
user.occupation = "Code Artist"
```

ActiveRecord Objects Can Be Saved

- Use *save* instance method - a row is added to the table

```
user = User.new(:name => "David",
                :occupation => "Artist")
user.save
```

ActiveRecord Find Operation

find Method

- *find* is a class method
  - In the same way, *new* and *create* are class methods
  - You use it with a ActiveRecord class (not with an object instance)

    Product.find(..)

Find Operation

- Find operates with four different retrieval approaches:
  - Find by id - This can either be a specific id (1), a list of ids (1, 5, 6), or an array of ids ([5, 6, 10]). If no record can be found for all of the listed ids, then *RecordNotFound* will be raised.
  - Find first - This will return the first record matched by the options used
  - Find last - This will return the last record matched by the options used.
  - Find all - This will return all the records matched by the options used. If no records are found, an empty array is returned.

Find Criteria

- *:conditions* - An SQL fragment like "administrator = 1" or ["user_name = ?", username]
- *:order*
- *:group* - An attribute name by which the result should be grouped. Uses the GROUP BY SQL-clause.
- *:limit* - An integer determining the limit on the number of rows that should be returned.
- *:offset* - An integer determining the offset from where the rows should be fetched.

Find Criteria (Continued)

- *:joins* - Either an SQL fragment for additional joins like "LEFT JOIN comments ON comments.post_id = id" (rarely needed) or named associations in the same form used for the :include option, which will perform an INNER JOIN on the associated table(s).
- *:include* - Names associations that should be loaded alongside. The symbols named refer to already defined associations.
- *:select* - By default, this is "*" as in "SELECT * FROM", but can be changed if you, for example, want to do a join but not include the joined columns.

Find Criteria (Continued)

- *:from* - By default, this is the table name of the class, but can be changed to an alternate table name (or even the name of a database view).
- *:readonly* - Mark the returned records read-only so they cannot be saved or updated.
- *:lock* - An SQL fragment like "FOR UPDATE" or "LOCK IN SHARE MODE". :lock => true gives connection‘s default exclusive lock, usually "FOR UPDATE".

Examples: Find by id

```
Person.find(1)       # returns the object for ID = 1
Person.find(1, 2, 6) # returns an array for objects with IDs in (1, 2, 6)
Person.find([7, 17]) # returns an array for objects with IDs in (7, 17)
Person.find([1])     # returns an array for the object with ID = 1
Person.find(1, :conditions => "administrator = 1",
                    :order => "created_on DESC")
```

Examples: Find first

```
Person.find(:first) # returns the first object fetched
                    # by SELECT * FROM people
Person.find(:first, :conditions => [ "user_name = ?", user_name])
Person.find(:first, :order => "created_on DESC", :offset => 5)
```

Example: Find all

```
Person.find(:all) # returns an array of objects for all the rows
                  #fetched by SELECT * FROM people
Person.find(:all,
            :conditions => [ "category IN (?)", categories],
            :limit => 50)
Person.find(:all,
            :conditions => { :friends => ["Bob", "Steve", "Fred"] }
Person.find(:all, :offset => 10, :limit => 10)
Person.find(:all, :include => [ :account, :friends ])
Person.find(:all, :group => "category")
```

Find with lock

- Imagine two concurrent transactions: each will read person.visits == 2, add 1 to it, and save, resulting in two saves of person.visits = 3. By locking the row, the second transaction has to wait until the first is finished; we get the expected person.visits == 4.

```
Person.transaction do
    person = Person.find(1, :lock => true)
    person.visits += 1
    person.save!
end
```

Conditions

- Conditions can either be specified as a string, array, or hash representing the WHERE-part of an SQL statement.
  - The array form is to be used when the condition input is tainted and requires sanitization.
  - The string form can be used for statements that don‘t involve tainted data.
  - The hash form works much like the array form, except only equality and range is possible. Examples:

Examples: Conditions

- String form
  - > *User.find(:all, :conditions=>"hobby='swimming'", :order=>"hobby DESC, age")*
- Array form
  - > *User.find(:all, :conditions=>["hobby=? AND name=?", 'swimming', 'Tom']*
- Hash form
  - > *User.find(:all, :conditions=>{:hobby=>'swimming', :name=>'Tom'}, :order=>"hobby DESC, age")*

Dynamic Attribute-based Finders

Dynamic Attribute-based Finders

- Dynamic attribute-based finders are a cleaner way of getting (and/or creating) objects by simple queries without turning to SQL.
- They work by appending the name of an attribute to *find_by_* or *find_all_by_*, so you get finders like
  - *Person.find_by_user_name*
  - *Person.find_all_by_last_name*
  - *Payment.find_by_transaction_id*

Dynamic Find Operation

- So instead of writing *Person.find(:first, :conditions => ["user_name = ?", user_name]),* you just do *Person.find_by_user_name(user_name)*.
- And instead of writing *Person.find(:all, :conditions => ["last_name = ?", last_name]),* you just do *Person.find_all_by_last_name(last_name)*.

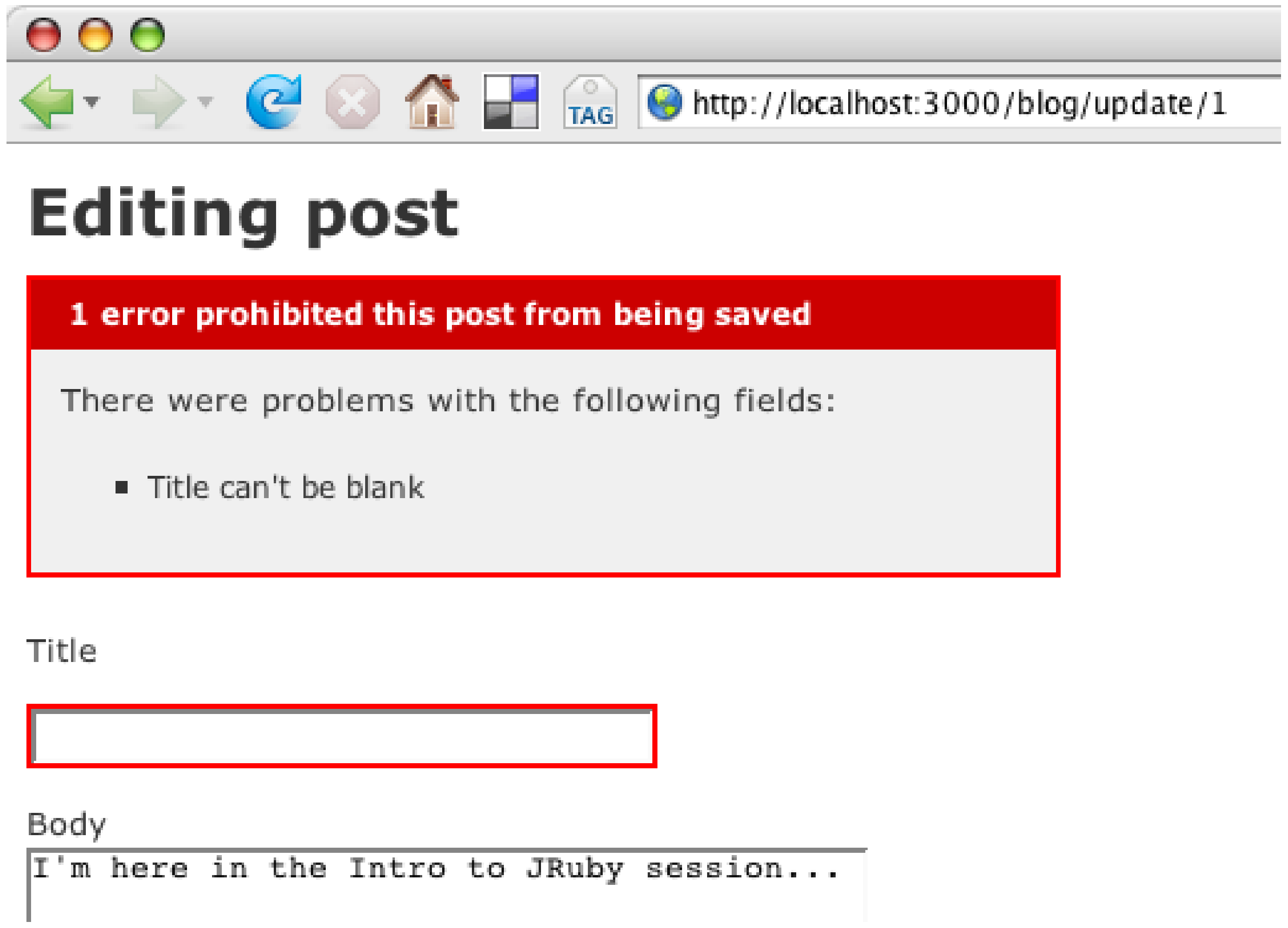
ActiveRecord Validation

ActiveRecord::Validations

- Validation methods

```
class User < ActiveRecord::Base
  validates_presence_of    :username, :level
  validates_uniqueness_of :username
  validates_length_of  :username, :maximum => 3,
   :allow_nil
  validates_numericality_of  :value, :on => :create
end
```

Validation Error


http://localhost:3000/blog/update/1

Editing post

1 error prohibited this post from being saved

There were problems with the following fields:

- Title can't be blank

Title

Body

```
I'm here in the Intro to JRuby session...
```

ActiveRecord::Validations

- Active Records implement validation by overwriting *Base#validate* (or the variations, *validate_on_create* and *validate_on_update*).

```
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
end
```

ActiveRecord Migration

ActiveRecord Migration

- Manage the evolution of a schema used
  - It’s a solution to the common problem of adding a field to make a new feature work in your local database, but being unsure of how to push that change to other developers and to the production server.
- You can describe the transformations in self-contained classes that can be checked into version control systems and executed against another database that might be one, two, or five versions behind.

Example: Migration

- Add a boolean flag called *ssl_enabled* to the *accounts* table and remove it again, if you're backing out of the migration.

```
class AddSsl < ActiveRecord::Migration
    def self.up
      add_column :accounts, :ssl_enabled, :boolean, :default => 1
    end

    def self.down
      remove_column :accounts, :ssl_enabled
    end
  end
```

Example: Migration

- First adds the system_settings table, then creates the very first row in it using the Active Record model that relies on the table.

```
class AddSystemSettings < ActiveRecord::Migration
 def self.up
  create_table :system_settings do |t|
   t.column :name,     :string
   t.column :label,    :string
   t.column :value,    :text
  end

  SystemSetting.create :name => "notice", :label => "Use notice?", :value => 1
 end
 def self.down
  drop_table :system_settings
 end
end
```

Available Transformations

- *create_table(name, options)*
- *drop_table(name)*
- *rename_table(old_name, new_name)*
- *add_column(table_name, column_name, type, options)*
- *rename_column(table_name, column_name, new_column_name)*
- *change_column(table_name, column_name, type, options)*
- *remove_column(table_name, column_name)*
- *add_index(table_name, column_names, index_type, index_name)*
- *remove_index(table_name, index_name)*

Callbacks

What is Callback?

- Callbacks are hooks into the lifecycle of an Active Record object that allow you to trigger logic before or after an alteration of the object state.
- This can be used to make sure that associated and dependent objects are deleted when destroy is called (by overwriting before_destroy) or to massage attributes before they‘re validated (by overwriting before_validation).

Lifecycle of an ActiveRecord Object

- (-) save
- (-) valid
- (1) before_validation
- (2) before_validation_on_create
- (-) validate
- (-) validate_on_create
- (3) after_validation
- (4) after_validation_on_create
- (5) before_save
- (6) before_create
- (-) create
- (7) after_create
- (8) after_save

Example: Callbacks in a Model

- The callback *before_validation_on_create* gets called on create.

```
class User < ActiveRecord::Base

  # Strip everything but alphabets
  def before_validation_on_create
    self.name = name.gsub(/[^A-Za-z]/, "") if attribute_present?
    ("name")
  end

  # More code
end
```

Exception Handling

Exception Handling

- Handle *RecordNotFound* Exception

```
begin
    User.find(2345)
rescue ActiveRecord::RecordNotFound
    outs “Not found!”
end
```

Thank you!

We do Instructor-led Codecamps!
http://www.javapassion.com/codecamps