CS 683 Emerging Technologies
Fall Semester, 2006
Doc 21 Rails 5 Many-To-Many, Validation, Transactions, Filters
Nov 7, 2006

References

Agile Web Development with Rails 2nd Ed Bl.16 October 25, Thomas & Hanson, The Pragmatic Bookshelf, PDF

Rails API, http://api.rubyonrails.org/

Many-to-Many

http://api.rubyonrails.org/classes/ActiveRecord/Associations/ClassMethods.html

Many-To-Many

books

id	title	date	publisher_id
1	Agile Web Development with Rails	2005-11-17	1
2	Programming Ruby	2005-01-02	1
3	Web Component Development with Zope 3	2005-05-01	2

publishers

id	name
1	The Pragmatic Bookshelf
2	Springer

authors_books

book_id	author_id
1	1
1	2
2	1
3	3

authors

id	first_name	last_name
1	Dave	Thomas
2	David	Hansson
3	Philipp	Weiterhausen

Migrations to Create Tables

```
def self.up
  create_table :books, :options => '' do |t|
    t.column :title, :string
    t.column :date, :date
  end
  add_column :books, :publisher_id, :integer
  add_index :books, :publisher_id
  create_table :publishers do |t|
    t.column :name, :string
  end
  create_table :authors do |t|
    t.column :first_name, :string
    t.column :last_name, :string
  end
   create_table :authors_books do |t|
    t.column :book_id, :integer
    t.column :author_id, :integer
   end
end
```

Sample Test

```
class BookTest < Test::Unit::TestCase
  fixtures :books
  fixtures :authors_books
  fixtures :authors
  fixtures :publishers

  def test_Read
    ruby = Book.find_by_title('Agile Web Development with Rails')
    assert_equal(ruby.date, Date.new(2005,8,10))
    assert_equal(ruby.publisher.name, 'The Pragmatic Bookshelf')
    authors = ruby.authors
    assert_equal(2, ruby.authors.length )
    thomas = Author.find_by_last_name('Thomas')
    assert(authors.include?(thomas))
  end
end
```

Added Methods

```
class Author < ActiveRecord::Base
  has_and_belongs_to_many :books
end
```

Methods Added to Author

```
books(force_reload=false)
books<<aBook
books.push_with_attributes(book1, ...)
books.delete(book1, book2,...)
books=objects
books.clear
books.empty?
books.size
books.find(id)
```

Model Validation & Callbacks

Basic Validate Methods

```
class Example < ActiveRecord::Base
    def validate
        #Called on every save on an object
        if somethingBad()
            errors.add(:foo, 'does not have correct value')
    end

    def validate_on_create
        #called on creation
    end

    def validate_on_update
        #called on update
    end
end
```

Validation Helpers

```
class Account < ActiveRecord::Base
      valisates_numericality_of :balance, :only_integer => true

    def withdraw(amount)
      balance_and_save(-amount)
    end
    def deposit(amount)
      balance_and_save(amount)
    end

    def balance_and_save(amount)
        self.balance += amount
        save!
    end

end
```

Active Record Callbacks

```
class Order < ActiveRecord::Base
    def before_save
        @payment_due = Time.now + 30.days
    end
end
```

Callback Methods

New Record - model.save()

before_validation
before_validation_on_create
after_validation
after_validation_on_create
before_save
before_create
INSERT into database
after_create
after_save

Existing Record - model.save()

before_validation
before_validation_on_update
after_validation
after_validation_on_update
before_save
before_update
UPDATE into database
after_update
after_save

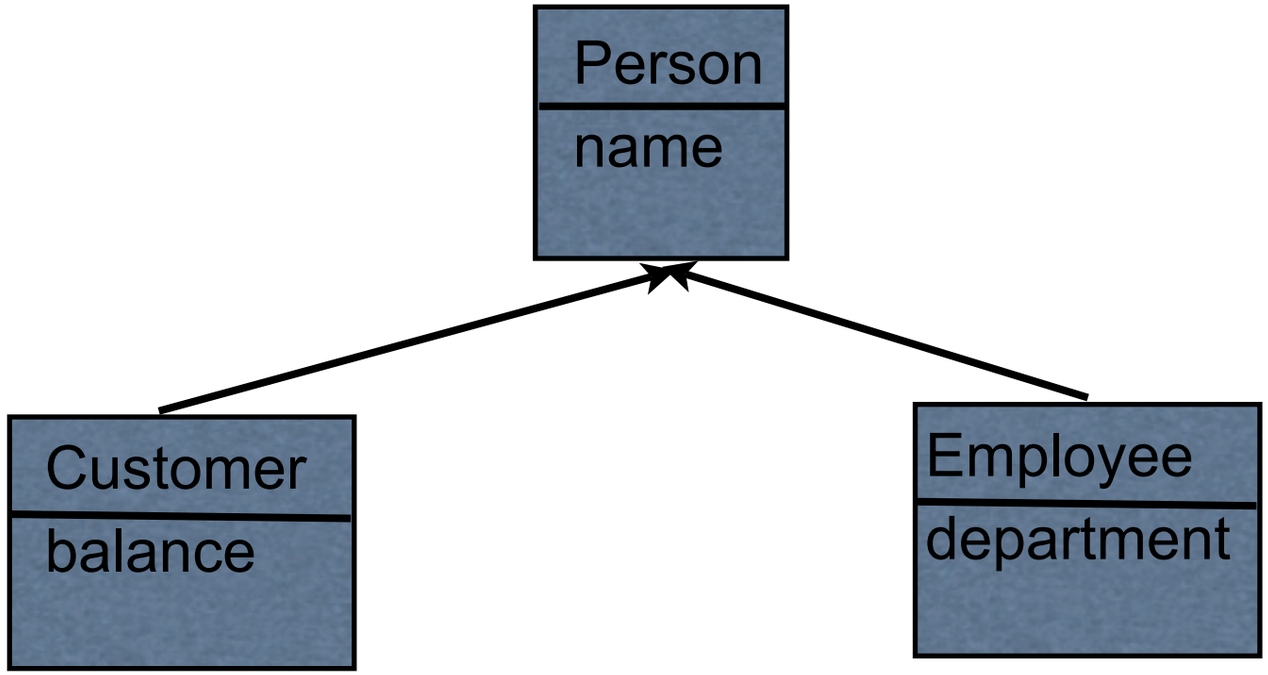
model.destroy()
before_destroy
DELETE from database
after_destroy

Inheritance and Tables

Active Record Supports

Single Table Inheritance
Polymorphic Associations

Single Table Inheritance



The Single Table

```
class CreatePeople < ActiveRecord::Migration
  def self.up
    create_table :people do |t|
      t.column :type, :string        #Used by Rails
      t.column :name, :string        #Person
      t.column :balance, :integer    #Customer
      t.column :department, :string #Employee
    end
  end

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

Classes

```
class Person < ActiveRecord::Base
end


class Customer < Person
end


class Employee < Person
end
```

Sample Usage

```
class PersonTest < Test::Unit::TestCase
  def setup
    Customer.create(:name => "Ernie", :balance => 55)
    Employee.create(:name => "Bert", :department => 'Sales')
  end

  def test_SingleTableInheritance
    customer = Person.find_by_name("Ernie")
    assert_equal(Customer, customer.class)
    assert_equal(55, customer.balance)

    customer = Person.find_by_name("Bert")
    assert_equal(Employee, customer.class)
    assert_equal('Sales', customer.department)
  end

  def teardown
    Customer.find(:all).each {|x| x.destroy}
    Employee.find(:all).each {|x| x.destroy}
  end
end
```

Polymorphic Association

A foreign key can reference multiple tables

catalog_entries

id	name	resource_id	resource_type
1	foo	1	Article
2	bar	1	Image

articles

id	content
1	XXX

images

id	content
2	YYY

Table Definitions

```
create_table :catalog_entries, :force => true do |t|
    t.column :name, :string
    t.column :resource_id, :integer
    t.column :resource_type, :string
end

create_table :articles, :force => true do |t|
    t.column :content, :text
end

create_table :images, :force => true do |t|
    t.column :content, :binary
end
```

Class Definitions

```
class CatalogEntry < ActiveRecord::Base
    belongs_to :resource, :polymorphic => true
end

class Article < ActiveRecord::Base
    has_one :catalog_entry, :as => :resource
end

class Sound < ActiveRecord::Base
    has_one :catalog_entry, :as => :resource
end

class Image < ActiveRecord::Base
    has_one :catalog_entry, :as => :resource
end
```

Sample Usage

```
a = Article.new(:content => "This is my new article")
c = CatalogEntry.new(:name => 'Article One', :acquired_at => Time.now)
c.resource = a
c.save!
```

Transactions

http://api.rubyonrails.org/classes/ActiveRecord/Transactions/ClassMethods.html

Simple Bank Account

accounts

id	number	balance
1	a	100
2	b	200

```
class Account < ActiveRecord::Base
  def withdraw(amount)
    balance_and_save(-amount)
  end
  def deposit(amount)
     balance_and_save(amount)
  end

  def balance_and_save(amount)
      self.balance += amount
      save!
  end

  def validate
    errors.add(:balance, "is negative") if balance < 0
  end
end
```

Successful Transaction

test/fixtures/accounts.yml

```
a_account:
  id: 1
  number: a
  balance: 100.0
b_account:
  id: 2
  number: b
  balance: 200.0
```

test/unit/account_test.rb

```
class BookTest < Test::Unit::TestCase
  fixtures :accounts

  def setup
    @a = Account.find_by_number('a')
    @b = Account.find_by_number('b')
  end

  def test_valid_transaction
    Account.transaction(@a, @b) do
      @a.deposit(10)
      @b.withdraw(10)
    end
    assert_equal 110.0 ,@a.balance
    assert_equal 190.0 ,@b.balance
    @a.reload
    assert_equal 110.0 ,@a.balance
  end
```

RolledBack Transaction

test/fixtures/accounts.yml

```
a_account:
  id: 1
  number: a
  balance: 100.0
b_account:
  id: 2
  number: b
  balance: 200.0
```

test/unit/account_test.rb

```
def test_invalid_transaction
  assert_raise(ActiveRecord::RecordInvalid) {
    Account.transaction(@a, @b) do
      @a.deposit(300)
      @b.withdraw(300)
    end
  }

  assert_equal 100.0 ,@a.balance
  assert_equal 200.0 ,@b.balance
end
```

How does this happen?

test/fixtures/accounts.yml

```
a_account:
  id: 1
  number: a
  balance: 100.0
b_account:
  id: 2
  number: b
  balance: 200.0
```

```
def test_invalid_transaction
    assert_raise(ActiveRecord::RecordInvalid) {
      Account.transaction(@a, @b) do
        @a.deposit(300)
        @b.withdraw(300)
      end
    }

    assert_equal 100 ,@a.balance
    assert_equal 200 ,@b.balance
    @a.reload
    @b.reload
    assert_equal 400 ,@a.balance
    assert_equal 200 ,@b.balance
end
```

Local Verses Remote Values

accounts

id	number	balance
1	a	100
2	b	200

object a

id: 1

number: a

balance: 100

@a.deposit(300)

Multiple Ruby threads can have copies of same data

Database and Ruby thread data can become inconsistent

The longer Ruby code holds a value the higher the chance of inconsistencies

Cookies & Sessions

http://api.rubyonrails.org/classes/ActionController/Cookies.html

http://api.rubyonrails.org/classes/ActionController/SessionManagement/ClassMethods.html

Cookies

Cookie Options

:domain

browser sends cookie only to pages in the given domain

:expires

When the cookie is no longer valid

:path

browser sends cookie only to pages whose leading part of the request path matches given path

:secure

browser only sends cookie if request uses https

:value

value of the cookie, must be a string

```
class CookieController < ApplicationController
  def setting_cookies
    cookies[:name] = 'Sam'
    cookies[:time] = Time.now.to_s
    cookies[:special] = { :value => "good customer",
                          :expires => 30.days.from_now,
                          :path => "/cat"}
    redirect_to :action => "reading_cookies"
  end

  def reading_cookies
    cookie_time = cookies[:time]
    foo = cookies[:foo]
    render(:text => "Time = #{cookie_time}, foo = #{foo}")
  end
end
```

Sessions

Adding data to session

```
session[:foo] = 'bar'
```

Accessing session data

```
data = session[:foo]
```

Session id stored in cookie

Session data stored on server

- PStore
- ActiveRecord
- MemoryStore
- FileStore

Can store serializable objects

Clearing old session data an issue

Session Example

```
class Cart
  attr_reader :items
  attr_reader :total

  def initialize
    empty!
  end

  def add_product(name, price)
    @items << name
    @total += price
  end

  def empty!
    @items = []
    @total = 0.0
  end
end
```

```
class StoreController < ApplicationController
  model :cart
  before_filter  :find_cart

  def add_to_cart
    product = Product.find(params[:id])
    @cart.add_product(product.name, product.price)
  end

  private
  def find_cart
    @cart = (session[:cart] ||= Cart.new)
end
```

Example modified from Agile Web Development with Rails

Controller Filters & Verification

http://api.rubyonrails.org/classes/ActionController/Verification/ClassMethods.html

Types of Filters

Before
- Called before any action

After
- Called after each action

Around
- Both before and after an action

Filters can be
- method
- blocks

http://api.rubyonrails.org/classes/ActionController/Filters/ClassMethods.html

Before Filter Block

```
class AdminController < ApplicationController
  before_filter do |controller|
    logger.info("Performing #{controller.action_name}")
  end

  def index
    list
    render :action => 'list'
  end
end
```

Output in log/development.log
Already contains useful log data

Example modified from Agile Web Development with Rails

Around Filter with Methods

app/controllers/admin_controller.rb

```
class AdminController < ApplicationController

  around_filter TimingFilter.new

  def index
    list
    render :action => 'list'
  end
```

app/models/timing_filter.rb

```
class TimingFilter
  def before(controller)
    @started = Time.now
  end

  def after(controller)
    elapsed = Time.now - @started
    action = controller.action_name
    controller.logger.info(
              "#{action} took #{elapsed} seconds to run")
  end
end
```

Before Filter

```
class AdminController < ApplicationController

  before_filter :authorize, :except => :login

  def index
    list
    render :action => 'list'
  end

  def authorize
    unless session[:user]
      redirect_to(:action => "login")
  end
end
```

```
def login
  if request.get?
    session[:user] = nil
  else
    if  params[:user][:name] == 'fred'
      session[:user] = 'fred'
      redirect_to(:action => "index")
    else
      flash[:notice] = "Invalid user"
    end
  end
end
```

```
<h1>Login</h1>

<%= start_form_tag :action => 'login' %>
  <p>Username
  <%= text_field 'user', 'name'  %></p>
  <%= submit_tag "Login" %>
<%= end_form_tag %>
```

Verification

Shortcut for filters

Verifies that condition is true before running

```
class GlobalController < ActionController::Base
  # prevent the #update_settings action from being invoked unless
  # the 'admin_privileges' request parameter exists.
  verify :params => "admin_privileges",
                :only => :update_post,
                :redirect_to => { :action => "settings" }
```

Rails Overview