References

Agile Web Development with Rails, Thomas & Hanson, The Pragmatic Bookshelf, 2005

Active Record Documentation, http://ar.rubyonrails.com/
Transactions

Simple Bank Account

<table>
<thead>
<tr>
<th>id</th>
<th>number</th>
<th>balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a</td>
<td>100</td>
</tr>
<tr>
<td>2</td>
<td>b</td>
<td>200</td>
</tr>
</tbody>
</table>

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
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
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

a_account:
id: 1
number: a
balance: 100.0

b_account:
id: 2
number: b
balance: 200.0
def test_no_local_rollback
  assert_raise(ActiveRecord::RecordInvalid) {
    Account.transaction do
      @a.deposit(300)
      @b.withdraw(300)
    end
  }
  assert_equal 400.0, @a.balance
  assert_equal -100.0, @b.balance
end

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 100.0, @a.balance
  assert_equal 200.0, @b.balance
  @a.reload
  @b.reload
  assert_equal 400.0, @a.balance
  assert_equal 200.0, @b.balance
end

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
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
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

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
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
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
end

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

Example modified from Agile Web Development with Rails
Filters

Before
Called before any action

After
Called after each action

Around
Both before and after an action

Filters can be
method
blocks

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

```ruby
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
```

```ruby
class AdminController < ApplicationController
  around_filter TimingFilter.new

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

Example modified from Agile Web Development with Rails
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
end

views/admin/login.rhtml

Example modified from Agile Web Development with Rails