CS3157: Advanced Programming

Lecture #12
Dec 6
Shlomo Hershkop
shlomo@cs.columbia.edu
Announcement

- Next Monday last class
  - Will do review
  - Please let me know if there is anything in particular you want to see reviewed
    - Email
    - Feedback today
Outline

- Ruby on rails
- Step by step examples today, please pay attention and keep up
- Ask if we do something you don’t understand
Background

- We covered ruby on Monday
  - Went kind of fast
  - Would not have learned it if you didn’t know perl
  - Although wont test you, should still read it over and understand at least how to read ruby basic code

- Lets take it to the next level
Install

- You need to install the frameworks

- Gems
  - Install interface to rubyforge.org
  - gem list --local
  - gem install rails --include-dependencies
Create a test directory
   - C:\test

Change there
   - cd c:\test

Create framework
   - Rails ap3157
     - Will see bunch of stuff

cd ap3157
ruby script\server
test

- Should be able to see something here on port 3000
- http://127.0.0.1:3000/

- This is the webbrick server
- Written completely in ruby
- Excellent for testing

- Real world:
  - Apache
  - Lighttpd
  - others
forward

- cd into the project directory
  - c:\test\ap3157
- ruby script/generate controller Greeting

- Will setup initial files we need

- Ok so let's explain what we are looking at
Explanation

http://www.some.com/shopping_cart/total/45
Setup

- Each web server running ruby has one dispatcher
- List of available controllers to handle requests
- Default action is index
  - Like in html, default file will be index.htm or index.html
So now go to

http://127.0.0.1:3000/greeting

What is the error ??
Fix?

- Let's define the index action in the greeting controller

- In the controller greeting class:

```ruby
def index
  render :text => '<h1>Welcome to your first Rails application</h1>'
end
```
Side note

- Excellent docs in
Want to separate view from controller

So let's create a view

ruby script/generate controller Greeting index

Answer n
Change `greet_controller`

Update the class and then reload:

class GreetingController < ApplicationController
  def index
    end
end
Now add a variable to the controller

```ruby
@welcome_message = "Welcome to your first Rails application"
```

Render it in the view rhtml file:

```html
<h1><%= @welcome_message %></h1>
```
<%o= %o> tags

- `<%= %>`
  - Can contain either
  - Script-lets
  - Ruby Expressions
Try

- **Add to greeting**

```ruby
class GreetingController < ApplicationController
  def index
    @age=8
    @table={"headings" => ["addend", "addend", "sum"],
            "body" => [[1, 1, 2], [1, 2, 3], [1, 3, 4]]
    }
  end
end
```
<h1>A simple table</h1>

<table>
  <tr>
    <% @table["headings"].each do |head| %>
    <td>
      <b><%= head %></b>
    </td>
    <% end %>
  </tr>
  <% @table["body"].each do |row| %>
    <tr>
      <% row.each do |col| %>
      <td>
        <%= col %>
      </td>
      <% end %>
    </tr>
    <% end %>
  </tr>
</table>
Active record

- Create an idea of something in the database
- Has predefined function which can be used
- Don’t need to worry about underlying tech
Goal

- Say we want to build a system which will let us share recipes or photos online
Databases

- One of the powers to Ruby is the ability to interact with databases

- Lets go over some of the basics
Database system

- DBMS Database management system:
  - Software package to manage a database
  - Data independence and efficient access.
  - Reduced application development time.
  - Data integrity and security.
  - Uniform data administration.
  - Concurrent access, recovery from crashes.
Data Model

- A data model is a collection of concepts for describing data.
- A schema is a description of a particular collection of data, using a given data model.
- The relational model of data is the most widely used model today.
  - Main concept: relation, basically a table with rows and columns.
  - Every relation has a schema, which describes the columns, or fields.
University Database

- Conceptual schema:
  - Students(sid: string, name: string, login: string, age: integer, gpa: real)
  - Courses(cid: string, cname: string, credits: integer)
  - Enrolled(sid: string, cid: string, grade: string)

- Physical schema:
  - Relations stored as unordered files.
  - Index on first column of Students.

- External Schema (View):
  - Course_info(cid: string, enrollment: integer)
Constraints

- Some rules on how certain parts of the data can be stored

- Kind of data for each type

- ‘uniqueness’ of certain data items
  - Keys
  - Foreign keys

- Transaction constraints
index

- Specific parts of the data we are interested in gaining quicker access to
  - Specialized data structures
    - Usually B+ trees
  - overhead
Other points

- Concurrency
- Atomic action
- Log system
- Roll back
- Integrity guarantees

- Design issues
Side note

- Ruby allows you to define undefined functions

class Talker
  def method_missing(method)
    if method.to_s =~ /say_/
      puts $'
    end
  end
end
Active records are ruby classes which allows you to interact with the DB

It will grab undefined functions as table columns if possible

Plus other built in stuff:

- `Photo.find_by_filename("balboapark.jpg").destroy`
Recipe website

- Lets use Ruby-Rails to create a website to share recipes.
- Display a list of all recipes.
- Create new recipes and edit existing recipes.
- Assign a recipe to a category (like "dessert" or "soup").
Getting started

- Create a test directory and cd there
- rails cookbook
  - This will create the framework for our project
  - Will step through specific parts, but very powerful and very easy to use
  - Encouraged to go online and explore full potential after your finals
next

- Lets get started with webbrick
- cd into cookbook directory
- Ruby script\server

- To check:
  - http://127.0.0.1:3000/
Apps subdirectory

- Controllers
  - where Rails looks to find controller classes. A controller handles a web request from the user.

- Views
  - holds the display templates to fill in with data from our application, convert to HTML, and return to the user's browser.

- Models
  - holds the classes that model and wrap the data stored in our application's database. In most frameworks, this part of the application can grow pretty messy, tedious, verbose, and error-prone. Rails makes it dead simple!

- Helpers
  - holds any helper classes used to assist the model, view, and controller classes. This helps to keep the model, view, and controller code small, focused, and uncluttered.
- `ruby script\generate controller Mytest`

- Read the output

- What is happening ??
next

- Open the mytest controller and add an index action

- How to trigger other events?
Start with dbase

- Lets manually setup the database now

- Log in to the mysql server running on your machine
1. mysql –u root –p

2. create database cookbook;

3. use cookbook;

4. create table recipes (id INT unsigned NOT NULL auto_increment, title varchar(255), instructions text, PRIMARY KEY(id)) engine=InnoDB;

5. describe recipes;
Back in Ruby

- Edit cookbook/config/database.yml

  development:
    adapter: mysql
    database: cookbook
    username: <your userid>
    password: <your password>
    host: localhost
ruby script\generate model Recipe
  This is the active record
ruby script\generate controller Recipe

Edit recipe_controller.rb
  scaffold :recipe

http://127.0.0.1:3000/recipe/new
Is everyone ok so far?
Back to mysql

Use coobook;

We can add to a table by issuing an alter command

```
alter table recipes add column description varchar(255);
alter table recipes add column date DATE;
```

And now...reload the new event
Add some recipes

127.0.0.0/recipe/list

Try to add and list recipes
  - They can be made up
scaffold :recipe

- Creates the the actions
  - list
  - show
  - edit
  - delete

- It also created default view templates for each of these actions.

- Which means we can now customize them
Edit : recipe_controller.rb

def list
  end

Now reload...what do you see ??
Lets create the template

- \cookbook\app\views\recipe

- create a file named list.rhtml
<body>
<h1>Online Cookbook - All Recipes</h1>
<table border="1">
<tr>
<td width="80%"><p align="center"><i><b>Recipe</b></i></td>
<td width="20%"><p align="center"><i><b>Date</b></i></td>
</tr>
<% @recipes.each do |recipe| %>
<tr>
<td><%= link_to recipe.title, :action => "show", :id => recipe.id %></td>
<td><%= recipe.date %></td>
</tr>
<% end %>
</table>
<p><%= link_to "Create new recipe", :action => "new" %></p>
</body>
</html>
Now go back and edit the list function

```ruby
def list
  @recipes = Recipe.find_all
end
```
Refresh

- 127.0.0.1:3000/recipe/list
So how would we add categories ??
Db side

- First let's create a database for it

```sql
mysql -u root -p
use cookbook;
create table categories (id INT unsigned NOT NULL auto_increment, name varchar(50), PRIMARY KEY(ID)) engine=InnoDB;
```
Ruby side shortcut

- Next slide shows how to do it manually
- Here is a huge shortcut
- `ruby script\generate scaffold Category`

- `http://127.0.0.1:3000/category/new`
Ruby side

- `ruby script\generate controller Category`
- `ruby script\generate model Category`

- **Edit** `cookbook\app\controllers\category_controller.rb`

- **Add scaffolding**
  - `scaffold :category`

- **Now browse**
  - `http://127.0.0.1:3000/category/new`
- Ok so we can create categories
- So how to link the two together?
- Let's extend the recipe table by adding a column called `category_id` field to the recipe table as an `int(6)`

- Any guesses how it's done?
- mysql –u root –p
- use cookbook
- alter table recipe add column category_id int(6);
next

- cookbook\app\models\recipe.rb and
- Add the following:
  - belongs_to :category

- cookbook\app\models\category.rb
- Add the following:
  - has_many :recipes
Now

- So
- if I have a recipe object in `@recipe`,
- I can find its category name with: `@recipe.category.name`.
- and if I have a category object in `@category`, I can fetch a collection of all recipes in that category using the code `@category.recipes`.
cookbook\app\controllers\recipe_controller.rb

def list
    @recipes = Recipe.find_all
end

def edit
    @recipe = Recipe.find(@params["id"]) 
    @categories = Category.find_all
end
Now need to include template for editing recipes to support this

cookbook\app\views\recipe\edit.rhtml
<form action="../update/<%= @recipe.id %>"> method="POST">
<input id="recipe_id" name="recipe[id]" size="30"
 type="hidden" value="<%= @recipe.id %>">
<p><b>Title</b><br>
<input id="recipe_title" name="recipe[title]" size="30"
 type="text" value="<%= @recipe.title %>">
</p>
<p><b>Description</b><br>
<input id="recipe_description" name="recipe[description]"
 size="30" type="text" value="<%= @recipe.description %>">
</p>
<p><b>Category:</b><br>
<select name="recipe[category_id]">
<% @categories.each do |category| %>
<option value="<%= category.id %>
<%= ' selected' if category.id == @recipe.category_id %>>
<%= category.name %>
</option>
<% end %>
</select></p>
<p><b>Instructions</b><br>
<textarea cols="40" id="recipe_instructions"
 name="recipe[instructions]"
 rows="20" wrap="virtual">
<%= @recipe.instructions %>
</textarea> </p>
<input type="submit" value="Update" />
</form>

<a href="/recipe/show/<%= @recipe.id %>">Show</a> | 
<a href="/recipe/list">Back</a>
Bunch of other things, won’t have time to cover

Type:
  - Rake stats

Let change delete
<%= link_to "(delete)", {:action => "delete", :id => recipe.id}, :confirm => "Really delete #{recipe.title}?" %>
More info

- http://www.onlamp.com/pub/a/onlamp/2005/03/03/rails.html
next

- Please look over the labs and homeworks

- Prepare for Monday review