Rails: Associations and Validation

Lecture 17
Schemas, Migrations, Models

- **migrations**
  - `schema.rb`
  - `db:migrate`

- **models**
  - `database.yml`
  - `db:create`

- **db**: `db:schema:load`
Recall: Migrations

class CreatePosts < ActiveRecord::Migration
  def change
    create_table :posts do |t|
      t.string :name
      t.string :title
      t.text :content
      t.timestamps
    end
  end
end
class Post < ActiveRecord::Base
  attr_accessible :content, :name, :title
end
Generating Code: rails generate

- Notice: Two blobs of Ruby code need to be in sync
  - Migration (creates table and columns)
    - db/migrate/xxx_create_students.rb
  - Model (matching name and attributes)
    - app/models/student.rb

- Single point of control: Generate both automatically
  - `$ rails generate model Student`  
    - `fname:string lname:string buckid:integer`
  - Use model name (singular) and attributes
  - Note: this does not generate the schema.rb (use rake)

- Migrations for table edits can also be generated
  - `$ rails g migration AddNickNameToStudent`  
    - `nick:string`
  - Name is meaningful! (starts with add or remove)
  - Creates a migration that changes students table
class CreateStudents < ActiveRecord::Migration
  def change
    create_table :students do |t|
      t.string :fname
      t.string :lname
      t.integer :buckid
      t.timestamps
    end
  end
end

class Student < ActiveRecord::Base
  attr_accessible :buckid, :fname, :lname
end
$ rails new demo #creates directory
$ cd demo
$ rails generate model Student
name:string buckid:integer
$ rails console
> Student.class  #lots available!
> Student.find :all #error, no table
> s = Student.new #will this work?
Demo with rails console

$ rails new demo #creates directory
$ cd demo
$ rails generate model Student
name:string buckid:integer
$ rails console
> Student.methods #lots available!
> Student.find :all #error, no table
> s = Student.new #error, no table
$ rake db:migrate #creates schema.rb
$ rails console
> Student.all #=> []
Working With Models

```ruby
> s = Student.new
> s2 = Student.new name: "Jo"
> s3 = Student.new name: "Xi",
    buckid: 23
> Student.all #=> [] still
> s.save
> Student.all #=>[<id: 1, ...>]
> s.name = "Mary"
> s.save
```
## Associations (1:N Relationship)

### students

<table>
<thead>
<tr>
<th>id (key)</th>
<th>buckid (integer)</th>
<th>team_id (foreign key)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>22352022</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>334432</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>34822039</td>
<td>6</td>
</tr>
</tbody>
</table>

### teams

<table>
<thead>
<tr>
<th>id (key)</th>
<th>name (string)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Wicked Wicky</td>
</tr>
<tr>
<td>2</td>
<td>The Happy Crew</td>
</tr>
<tr>
<td>6</td>
<td>No Names</td>
</tr>
</tbody>
</table>
Invariants

- A student belongs to exactly 1 team
  - Weaker: A student belongs to at most 1 team
- Driven by representation: Either invariant is supported by one column in students
- Maintaining stronger invariant
  - Students can only be added with team_id set to something valid
  - Deleting a team deletes member students!
- Maintaining weaker invariant
  - Students can be added with null team_id
  - Deleting a team null-ifies members' team_id
Rails Migration and Models

```ruby
class AddTeamForeignKeys < ActiveRecord::Migration
  def change
    add_column :students, :team_id, :integer
    add_index :students, :team_id  # for quick load
  end
end

class Student < ActiveRecord::Base
  belongs_to :team  # note singular form
  # creates :team method
end

class Team < ActiveRecord::Base
  has_many :students  # note plural form
end
```
Association Method

- **Belongs_to** creates method for accessing owner
  ```ruby
  @student = Student.find 1 #=> 22352022
  @student.team #=> 'Wicked Wicky'
  @student.team.name = 'Evil Wicky'
  ```

- **Has_many** creates method for accessing members
  ```ruby
  @team = Team.find 1
  @team.students #=> array of students
  @team.students.first
  @team.students.size
  @team.students.destroy_all
  @team.students.any? { |s| ... }
  ```
Asymmetry in Writes to Assoc.

- Add a student to a team's association: student automatically saved (assuming team is stored in database)
  \[
  \begin{align*}
  & t = \text{Team.find } 1 \\
  & t.\text{students } \Rightarrow [ ] \\
  & t.\text{students } \ll \text{Student.new } \# \text{gets an id} \\
  & t.\text{students } \Rightarrow [ #<\text{Student id: 1, ...}> ]
  \end{align*}
  \]

- Set a student's association to a team: student is not automatically saved
  \[
  \begin{align*}
  & s = \text{Student.find } 1 \\
  & s.\text{team } = \text{my_team} \\
  & s.\text{reload } \Rightarrow s's \text{ team is unchanged}
  \end{align*}
  \]
Modifiers for `belongs_to`

class Student < ActiveRecord::Base
  belongs_to :team,
    :touch => :membership_updated
  belongs_to :project_group,
    :class_name => 'Team'
  belongs_to :major,
    :foreign_key => 'OSU_maj_code'
end
Modifiers for has_many

class Team < ActiveRecord::Base

  has_many :students,

    :limit => 5,

    :dependent => :destroy

end
More Relationships

- **1:1 (one-to-one)**
  - Use `belongs_to` with `has_one`
    - `Has_one` is just `has_many` with limit 1
  - Same asymmetry in writing exists

- **N:M (many-to-many)**
  - A third, intermediary table is used with 2 columns (for foreign keys from two tables)
  - In rails, use `has_many :through` association
Validations

- Invariants on the data in a table
  - Every student has a (non-null) buckid
  - Buckids are unique
  - Team names are less than 30 characters
  - Usernames match a regular expression

- To maintain invariant:
  - Must be true initially
  - Must be satisfied by each insertion

- These "validations" are in the model
  - A model instance can be checked
  - Invalid objects can not be saved

```ruby
student = Student.new { :lname => 'Vee' } 
student.valid? #=> false (no buckid) 
student.save #=> false
```
Rails Implementation

- Model objects have an errors attribute
  - Errors is a hash

- Failing a validity check adds an item to the errors hash
  - Empty hash corresponds to valid object
  - Each attribute is a key in the errors hash, plus there's an overall key, :base

```
s.errors[:buckid] = "is not a number"
```

- The valid? method does the following:
  - Empties errors hash
  - Runs validations
  - Returns errors.empty?
Example

class Post < ActiveRecord::Base
  attr_accessible :content, :name, :title

  validates :name, :presence => true
  validates :title, :presence => true,
    :length =>
      { :minimum => 5, :maximum => 50 } 

end
Validates Method in Model

- **Uniqueness**
  
  :uniqueness => true
  :uniqueness => {:message => 'Username taken'}

- **Non-nullness (not the same as being true!)**
  
  :presence => {:message => 'Title needed'}

- **Truth of a boolean field**
  
  :acceptance => {:message => 'Accept the terms'}

- **Matching a regular expression**
  
  :format => {:with => /.*/, :message =>...}
  :format => /[A-Za-z0-9]+/ 

- **Being a number**
  
  :numericality => {:only_integer => true}

- **Having a length**
  
  :length => {:minimum => 5}
Alternative: Declarative Style

- Special methods for each flavor of validation

  validates_uniqueness_of :username
  validates_presence_of :password
  validates_acceptance_of :terms
  validates_format_of :name, :with =&gt; /.*/
  validates_numericality_of :buckid, :only_integer =&gt; true
Summary

- **Associations**
  - 1:N (or 1:1) relationships via foreign keys
  - Rails methods `belongs_to, has_many`
  - Create association attributes, which can be read and written
  - Asymmetry in writing owner vs member

- **Validations**
  - Invariants checked before saving
  - Errors hash contains list of problems
  - Declarative style for common case checks
  - Custom validity checkers possible too
Rails: Routes
Configuration

- Need to map an HTTP request (ie verb, URL, parameters) to an application action (ie a method in a Ruby class)
  - Framework invokes that method, passing in parameters from request
- Produces an HTTP response, typically HTML sent back to client's browser
- These mappings are called *routes*
- Defined in config/routes.rb
  - Ruby code, but highly stylized (another DSL)
  - Checked top to bottom for first match
REST

- REpresentational State Transfer
- An architectural style for developing web applications, mapping request types to HTTP
- Resources: bundles of (server-side) state
- Resources are identified by URLs
- The protocol is stateless
- Small set of verbs, with fixed semantics (e.g., idempotence)
  - GET, POST, PUT, DELETE
- Small set of request categories (CRUD)
  - Create, read, update, delete
Resources

- A resource could be a *collection* of items
  - Example: a set of students
  - Corresponds to a table

- A resource could be an individual *member*
  - Example: a single student
  - Corresponds to a row in a table

- In REST, resources have URLs
  - A collection has its own URL
    http://quickrosters.com/students
  - Each member element has its own URL
    http://quickrosters.com/students/42
Read Collection: GET

GET /students

Request

Listing students

Fname Lname  Buckid
Marco Pantani 22352022  Show Edit Destroy
Primo Camera 334432  Show Edit Destroy
Cher 34822039  Show Edit Destroy

New Student
Read Collection: GET

GET /students

Request

Listing students
Fname Lname    Buckid
Marco Pantani  22352022 Show Edit Destroy
Primo Camera   334432    Show Edit Destroy
Cher           34822039 Show Edit Destroy

New Student
<h1>Listing students</h1>
<table>
  <tr>
    <th>Fname</th>
    <th>Lname</th>
    <th>Buckid</th>
    <th></th>
    <th></th>
    <th></th>
  </tr>
  ...  
  <tr>
    <td>Primo</td>
    <td>Carnera</td>
    <td>334432</td>
    <td><a href="/students/3">Show</a></td>
    <td><a href="/students/3/edit">Edit</a></td>
    <td><a href="/students/3" data-confirm="Are you sure?" data-method="delete" rel="nofollow">Destroy</a></td>
  </tr>
  ...  
</table>

<a href="/students/new">New Student</a>
Read Member: GET

GET /students/3

Request
## Minimal Set of Routes

<table>
<thead>
<tr>
<th></th>
<th>Collection /students</th>
<th>Member /students/42</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>List all members</td>
<td>Show info about a member</td>
</tr>
<tr>
<td>PUT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>POST</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DELETE</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Minimal Set of Routes

<table>
<thead>
<tr>
<th>Collection /students</th>
<th>Member /students/42</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>List all members</td>
</tr>
<tr>
<td></td>
<td>Show info about a member</td>
</tr>
<tr>
<td>PUT</td>
<td>Update member</td>
</tr>
<tr>
<td>POST</td>
<td>Create a new member</td>
</tr>
<tr>
<td>DELETE</td>
<td>Delete this member</td>
</tr>
</tbody>
</table>

- **Implications**
  - You can't delete the collection
  - No idempotent operations on collection
Typical Workflow: Delete

- How does one issue a DELETE on particular member (eg, /students/42)?

  **Protocol:**
  - GET the collection to see the list
  - Click a button next to one item in the list to issue a DELETE for that member

  **Alternative:**
  - GET the member to see the details
  - Click a button to issue a DELETE for that member
GET List, DELETE Member

GET /students

Listing students

Fname Lname  Buckid
Marco  Pantani  22352022  Show Edit Destroy
Primo  Camera  334432  Show Edit Destroy
Cher  34822039  Show Edit Destroy

Delete /students/4

New Student
Typical Workflow: Create

- How does one issue a POST on collection?
  - GET a (blank) form
  - Fill in fields of form
  - Click a button to submit, resulting in the POST

- That first GET is a new route
  - GET on the collection
  - But instead of a list of members, the result is a form to be filled in and submitted
GET Blank Form, POST the Form

GET "a blank form"

POST /students
lname: ...etc
# Standard Set of Routes

<table>
<thead>
<tr>
<th><strong>Collection</strong> /students</th>
<th><strong>Member</strong> /students/42</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>GET</strong></td>
<td></td>
</tr>
<tr>
<td>1. List all members</td>
<td>• Show info about a member</td>
</tr>
<tr>
<td>2. Form for entering a new member's data</td>
<td></td>
</tr>
<tr>
<td><strong>PUT</strong></td>
<td>Update member</td>
</tr>
<tr>
<td><strong>POST</strong></td>
<td>Create a new member</td>
</tr>
<tr>
<td><strong>DELETE</strong></td>
<td>Delete this member</td>
</tr>
</tbody>
</table>
Typical Workflow: Update

- How does one issue a PUT on a member?
  - GET a (populated) form
  - Edit the fields of the form
  - Click a button to send, resulting in the PUT

- That first GET is a new route
  - GET on a member
  - But instead of a display of information about that member, the result is a populated form to modify and submit
GET Filled Form, PUT the Form

GET "a populated form"

PUT /students/4
lname: ...etc
# Standard Set of Routes

<table>
<thead>
<tr>
<th>Collection</th>
<th>Member</th>
</tr>
</thead>
<tbody>
<tr>
<td>/students</td>
<td>/students/42</td>
</tr>
<tr>
<td><strong>GET</strong></td>
<td></td>
</tr>
<tr>
<td>1. List all members</td>
<td>• Show info about a member</td>
</tr>
<tr>
<td>2. Form for entering a new member's data</td>
<td>• Form for editing an existing member's data</td>
</tr>
<tr>
<td><strong>PUT</strong></td>
<td>Update member</td>
</tr>
<tr>
<td><strong>POST</strong></td>
<td>Create a new member</td>
</tr>
<tr>
<td><strong>DELETE</strong></td>
<td>Delete this member</td>
</tr>
</tbody>
</table>
Rails Resource-Based Routes

- For a resource like :students, the action pack includes
  - 1 controller (StudentController)
  - 7 routes (each with a method in controller)
  - 4 Views (list of students, show 1 student, new, edit)

<table>
<thead>
<tr>
<th>HTTP Verb</th>
<th>URL</th>
<th>Resource</th>
<th>Method</th>
<th>Response (View)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>/students</td>
<td>Collection</td>
<td>index</td>
<td>list all</td>
</tr>
<tr>
<td>POST</td>
<td>/students</td>
<td>Collection</td>
<td>create</td>
<td>show one</td>
</tr>
<tr>
<td>GET</td>
<td>/students/new</td>
<td>Collection</td>
<td>new</td>
<td>blank form</td>
</tr>
<tr>
<td>GET</td>
<td>/students/3</td>
<td>Member</td>
<td>show</td>
<td>show one</td>
</tr>
<tr>
<td>GET</td>
<td>/students/3/edit</td>
<td>Member</td>
<td>edit</td>
<td>filled form</td>
</tr>
<tr>
<td>PUT</td>
<td>/students/3</td>
<td>Member</td>
<td>update</td>
<td>show one</td>
</tr>
<tr>
<td>DELETE</td>
<td>/students/3</td>
<td>Member</td>
<td>destroy</td>
<td>list all</td>
</tr>
</tbody>
</table>
Defining Resource-Based Routes

- In RosterTool app's config/routes.rb

```ruby
rosterTool::Application.routes.draw do
  resources :students
  resources :faculty
end
```
Customizing Routes

- To change which 7 routes are created
  ```ruby
  resources :students, :except => [:update, :destroy]
  resources :grades, :only => [:index, :show]
  ```

- To specify a particular controller
  ```ruby
  resources :students, :controller => 'ugrads'
  ```

- To rename certain actions
  ```ruby
  resources :students, :path_names =>
  { :create => 'enroll' }
  ```

- To add more routes to standard set
  - Add /students/:id/avatar (ie on member)
  - Add /students/search (ie on collection)
  ```ruby
  resources :students do
    get 'avatar', :on => :member
    get 'search', :on => :collection
  end
  ```
Segment Keys

- URL request has *parameters* for controller
  - Example: `products/42`
  - Pattern string: 'products/:id'
- Segment key gets value when route matches
- Controller gets a hash (called params) of segment keys and their values
  - Example: `params[:id]` is '42'
- Common case: Look up an item by id
  ```ruby
  class ProductsController < ApplicationController
    def show
      @product = Product.find(params[:id])
    end
  end
  ```
Recognition vs Generation

- Dual problems
  - Recognize a URL (request for an action)
  - Generate a URL (a hyperlink or redirect)

- Routes do both!

- Helper methods give relative paths
  - `photos_path` #=>/photos
  - `new_photo_path` #=>/photos/new
  - `edit_photo_path(:id)` #=>/photos/4/edit
  - `photo_path(:id)` #=>/photos/4

- And _url helpers give full URL, including hostname

- To see the full list (eg for debugging)
  $ rake routes
Root Route

- Commented out in default routes.rb
  
  ```ruby
  # root :to => "welcome#index"
  ```

- With no matching route, GET for http://example.com gets index.html from application's public directory

- To customize landing page, 2 choices:
  - Remove (or replace) public/index.html
  - Add root :to route to config, sending root request to a controller#action
Singleton Resources

- Declared with singular syntax
  ```ruby
  resource :system
  ```
- You get only 1 resource
  - Not both collection and member
  - Controller still plural (eg SystemsController)
- Only 6 standard routes
  - No index collection action to list members
  - POST /system -> create
  - GET /system/new -> new
  - GET /system/edit -> edit
  - GET /system/ -> show
  - PUT /system -> update
  - DELETE /system -> destroy
Summary

- **REST and CRUD**
  - Create, read, update, destroy
  - Map data to resources
  - Map actions to HTTP requests (verb + URL)

- **Routes**
  - Connect HTTP request to specific method in a controller class
  - Defined in `config/routes.rb`
  - Resource based, or match-based
  - Dual problem: recognition and generation