Rapid Application Development

COSC2675 2017 Week 7

Dr. Andy Song
andy.song@rmit.edu.au

Last Week

- Action Pack
- Model
- Controller
- DB migration
- Working with a simple web form
- Scaffolding
- REST
- RESTful resources (with different formats)
- RESTful methods

Multiple Tables

- Working with multiple tables is made easy by Rails.
- Every model maps to a table.
- The relationships between tables are managed as relationships between models.
- Scaffolding will be helpful. However there are quite some coding need to done manually.
- The task will be much more difficult if you are retrofitting an existing database (not generated by Rails) rather than creating a fresh new database in Rails.

A Students Application

- Create a new application:

```
$ rails new students
$ cd students
$ rails generate scaffold student given_name:string
middle_name:string family_name:string date_of_birth:date
grade_point_average:decimal start_date:date

$ rails generate scaffold award name:string year:integer
student_id:integer
```

Two models have been created. Rails doesn't know they are connected yet.

Model **award** has a `student_id` field.

A Students Application

- Open `app/models/student.rb` and add

```
# a student can have many awards
has_many :awards
```

- Open `app/models/award.rb` and add

```
# every award is linked to a student, through student_id
belongs_to :students
```

- `has_many` and `belongs_to` are just two methods
- They specify the association between models
- Similar methods include:

```
has_one
has_and_belongs_to_many
```


Association methods

- Details of `has_many` and `belongs_to` can be seen in the Rails API

http://api.rubyonrails.org/classes/ActiveRecord/Associations/ClassMethods.html#method-i-has_many
http://api.rubyonrails.org/classes/ActiveRecord/Associations/ClassMethods.html#method-i-belongs_to

- They are class methods from

`ActiveRecord::Associations::ClassMethods`

- Their method signatures are:

```
has_many(name, scope = nil, options = {}, &extension)
belongs_to(name, scope = nil, options = {})
```

Association methods

```
has_many :friends, -> { where(author_id: 1) },
                    through: :societies, source: :member, do
  def find_or_create_by_name(name)
    first_name, last_name = name.split(" ", 2)
    find_or_create_by(first_name: first_name, last_name: last_name)
  end
end

belongs_to :firm, foreign_key: "client_of"
belongs_to :person, primary_key: "name", foreign_key: "person_name"
belongs_to :author, class_name: "Person", foreign_key: "author_id"
belongs_to :project, -> { readonly }
belongs_to :attachable, polymorphic: true
belongs_to :valid_coupon, ->(o) { where "discounts > ?", o.payments_count },
                          class_name: "Coupon", foreign_key: "coupon_id"
```


Run the application

- Through these association methods Rails knows the connection between models.
- Rails does not add automatic checking or validation to ensure that the relationships would work.
- E.g. does not require a valid student ID for every award.
- Although that can achieved using scope, option
- Now you can start the Students Application

```
$ rails db:migrate
$ rails server [ –p $PORT -b $IP // not needed on local ]
```

- Create new students at `http://…../students/new`

Adding new records

New Student

Given name
Tom

Middle name

Family name
Smith

Date of birth
2012 January 1

Grade point average
23

Start date
2016 March 1

Create Student

Back

How about create an award at
`http://…../awards/new`

New Award

Name

Year

Student
-3 Friday

Create Award

Back

Add an Award

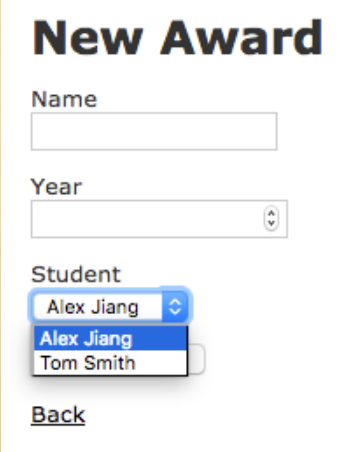
- A select field sounds like a good idea to solve the problem.
- Edit the student field in partial `app/views/awards/_form.html.erb`
- Remove ~~`<%= f.number_field :student_id %>`~~ and change to

```
< div class =" field" >
<%= f.label  :student_id %> < br/ >
<%= f.select :student_id,
     Student.all.order(:family_name, :given_name).collect
     {|s| [( s.given_name + " " + s.family_name), s.id]}%>
</div >
```

- `select` method creates the form, `all` method lists all student record, `order` method sorts all records by family names then give names, `collect` similar to `each` is another array method to run the block on each element but returns the new array.

Adding a new award

Now view `http://…../awards/new`




```
<div class="field">
<label for="award_student_id">
Student</label>

<select name="award[student_id]"
id="award_student_id">

<option value="2">Alex Jiang
</option>

<option value="1">Tom Smith
</option>

</select>
</div>
```

Assume we have created Alex and Tom beforehand.

Improve the view

- The form in previous slide submit the first student as "`1`"
- To replace with a proper name, update both `app/views/awards/show.html.erb` and `app/views/awards/index.html.erb`
- Replace ~~`<%= @award.student_id %>`~~ to `<%= @award.student.given_name %> <%= @award.student.family_name %>`

  and ~~`<%= award.student_id %>`~~ to `<%= award.student.given_name %> <%= award.student.family_name %>`
- Where does this `student` method in `award` come from?
- Thanks the `belongs_to` method. Now we have access to `given_name` and `family_name` methods.

Further improve the view

`[@]award.student.given_name, [@]award.student.family_name`

were repeated in the previous slides. Remember DRY?

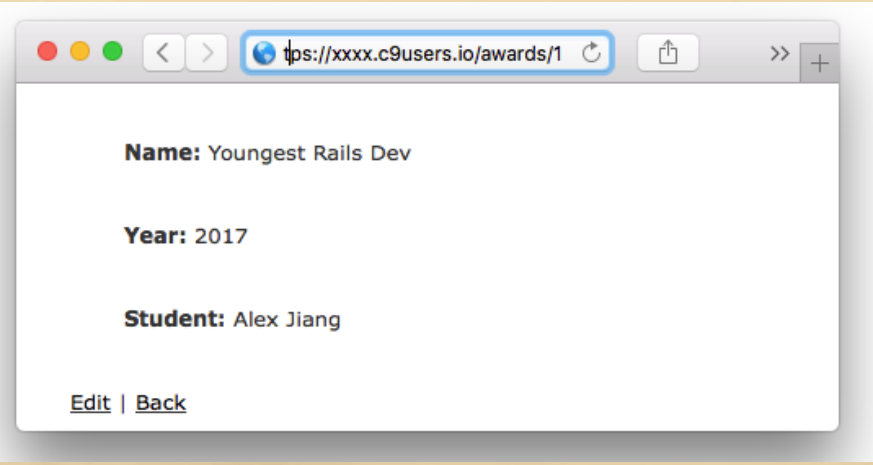
- Let define a `name` method in `app/models/student.rb`

```
def name
  given_name + " " + family_name
end
```

- Use the new version `<%= @award.student.name %>` in `app/views/awards/show.html.erb`, `<%= award.student.name %>` in `app/views/awards/index.html.erb`
- and `<%= f.select :student_id, Student.all.collect {|s| [s.name, s.id]} %>` in `app/views/awards/_form.html.erb`
- The `name` method is often called an *attribute* on the model. If you need to assign values to it, then define `name=` method.

An award record shows student

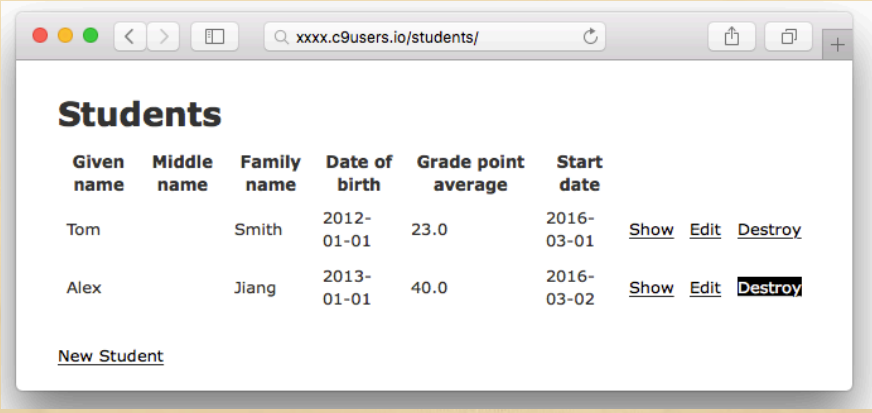
Now view a record as `http://…../awards/1`



The index action should also show names `http://…../awards`

What if a record is deleted

Let's delete Alex from the previous example



Oops! That messed up the awards page `http://…../awards`

You can disable the `name` method in awards's index, delete Alex's award, put back the `name` method. Then the application will be back to normal.

Connecting Students to Awards

- Orphaned record: when Alex was deleted, his award still remains.
- Rails did not notice the deletion.
- We need to keep `award` records in sync with `student` records.
- It is actually easy. Just update `app/models/student.rb`

```
has_many :awards
```

```
has_many :awards, dependent: :destroy
```

- Once a student record is deleted, then all the awards records of that student will also disappear.
- Warning: this deletion does not ask for confirmation.

The dependent option

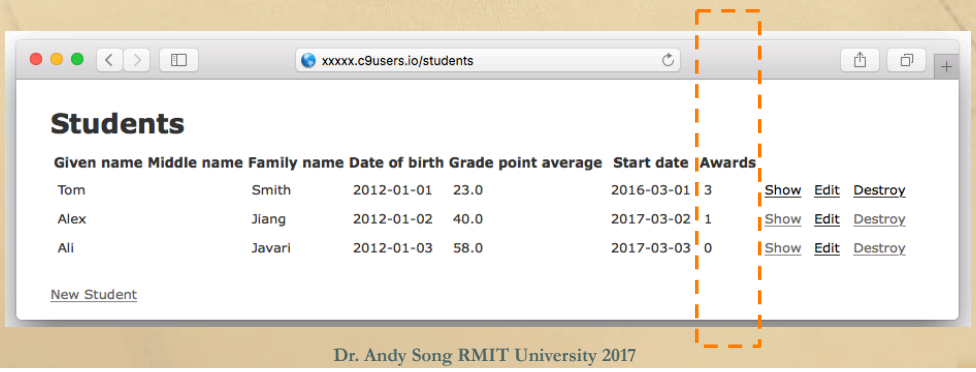
- Rails API Doc explains

  " Controls what happens to the associated objects when their owner is destroyed. Note that these are implemented as callbacks, and Rails executes callbacks in order. Therefore, other similar callbacks may affect the :dependent behavior, and the :dependent behavior may affect other callbacks."

- **Options for dependent are :**
  - `:destroy`
  - `:delete_all`
  - `:nullify`
  - `:restrict_with_exception`
  - `:restrict_with_error`
  - If using with the `:through` option, the association on the join model must be a `belong_to`, and the records which get deleted are the join records, rather than the associated records.

Counting Awards

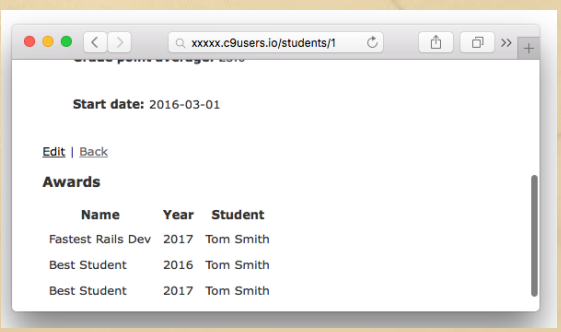
- How to add a new column for students to show the total number of awards that each student has?
- Obviously we need to change the view of index `app/views/students/index.html.erb`
- Add `<th>Awards</th>` in table head `<thead></thead>` after Start date
- Add `<td><%= student.awards.count %></td>` also after `start_date`



List the awards for each student

- Add a table in `app/views/students/show.html.erb`

```
<h3> Awards </h3 >
<table>
 <tr>
  <th>Name</th>
  <th>Year</th>
  <th>Student</th>
 </tr>
<% @student.awards.each do | award | %>
 <tr>
  <td> <%= award.name %> </td>
  <td> <%= award.year %> </td>
  <td> <%= award.student.name %>
  </td>
 </tr>
<% end %>
</table>
```



Nested Resources

- Students and awards aren't really parallel, not like students and courses. That dependency can be better modeled.
- Manually change `config/routes.rb`

```
resources :awards
resources :students
resources :students do
  resources :awards
end
```

- The students page `http://…../students` is OK. However the awards page `http://…../awards` is no longer working.
- Accessing the awards page needs to go through students like `http://…../students/1/awards/2`

Implementing nested awards

- Changing the controller is a bit complicated.
- Mainly to add `student` as context in `AwardsController`.

~~before_action :set_award, only: [:show, :edit, :update, :destroy]~~

```
before_action :get_student
before_action :set_award, only: [:show, :edit, :update, :destroy]
```

Method `index`:

~~@awards = Award.all~~
```
@awards = @student.awards
```

Method `new`:

~~@award = Award.new(award_params)~~
```
@award = @student.awards.build( award_params)
```

Method `create`:

~~@award = Award.new(award_params)~~
```
@award = @student.awards.build( award_params)
```
~~format.html { redirect_to @award, notice: 'Award..'}~~
```
format.html { redirect_to student_awards_url(@student),
```


Implementing nest awards…

Method `update`:

~~if @award.update(award_params)~~
```
@award = @student.awards.build( award_params)
```
~~format.html { redirect_to @award, notice: 'Award..'}~~
```
format.html { redirect_to student_awards_url(@student),
```

Method `delete`:

~~format.html { redirect_to @award, notice: 'Award..'}~~
```
format.html { redirect_to student_awards_url(@student),
```

Private methods:

Method `set_award`:

~~@award = Award.find(params[:id])~~
```
@award = @student.awards.find(params[:id])
```

Add method `get_student`

```
def get_student
   @student = Student.find(params[:student_id])
end
```

Change the views for awards

- Update `app/views/awards/index.html.erb`

```
<h1>Awards</h1>
<table>
<h1> Awards for <%= @student.name %> </h1>
<% if !@student.awards.empty? %> <table>

   <%# the whole table body here .... then the else %>
<% else %>
   <p> <%= @student.given_name %> has no awards yet. </p>
<% end %>
```

- Inside the table:

```
<td><%= link_to 'Show', award%></td>
<td><%= link_to 'Show', [@student, award] %></td>

<td><%= link_to 'Edit', edit_award_path(award) %></td>
<td><%= link_to 'Edit', edit_student_award_path(@student,
award) %> </td>
```

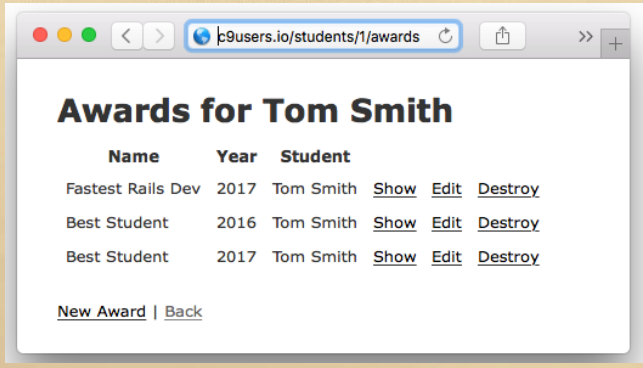

Change the views for awards …

- Inside the table in `app/views/awards/index.html.erb`

```
<td><%= link_to 'Destroy', award,                method: :delete,
<td><%= link_to 'Destroy', [@student, award], .............
```

- After the table:

```
<%= link_to 'New Award', new_award_path %>
<%= link_to 'New Award', new_student_award_path(@student) %> |
<%= link_to 'Back', @student %>
```

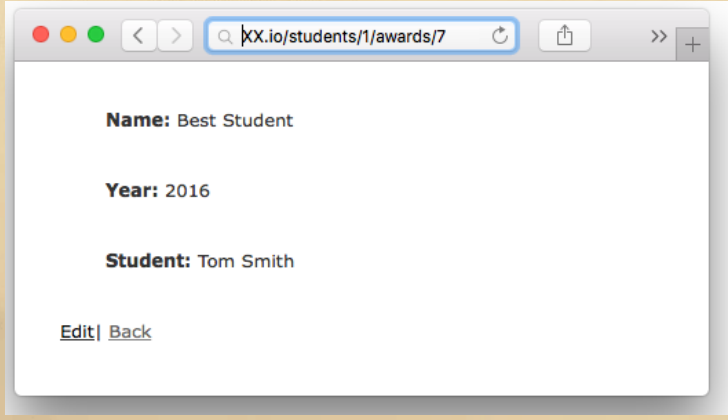


More views to change (show)

- Also need to change `app/views/awards/show.html.erb`

```
<%= link_to 'Edit', edit_award_path(@award) %>
| <%= link_to 'Back', awards_path %>
```

```
<%= link_to 'Edit', edit_student_award_path(@student, @award) %>
| <%= link_to 'Back', student_awards_path(@student) %>
```



More views to change (new)

- Also need to change `app/views/awards/new.html.erb`

```
<h1>New Award</h1>
<%= render 'form', award: @award %>
<%= link_to 'Back', awards_path %>
```

```
<h1> New Award for <%= @student.name %> </h1>
<%= render 'form', award: @award %>
<%= link_to 'Back', student_awards_path(@student) %>
```

- Change the first line of partial `app/views/awards/_form.html.erb`

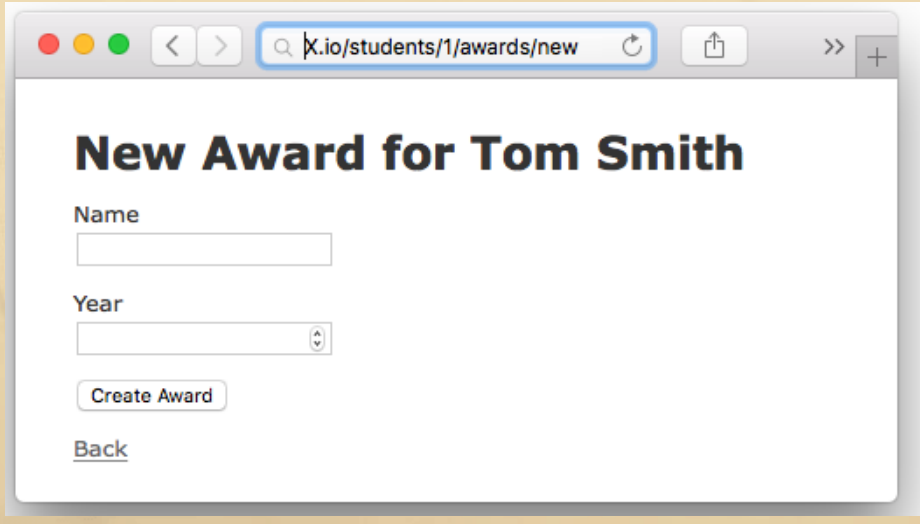
```
<%= form_for(award) do |f| %>
```

```
<%= form_for([@student, award]) do |f| %>
```

- Delete the student selector from the form

More views to change (new) ...

- Create is viewable at `https://…students/1/awards/new`



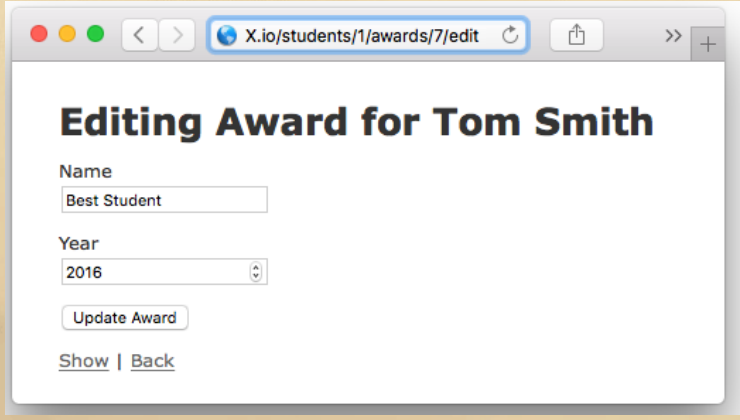
More views to change (edit)

- Change `app/views/awards/edit.html.erb`

```
~~<h1>Editing Award</h1>~~
<h1> Editing Award for <%= @student.name %> </h1>

~~<%= link_to 'Show', @award %> |~~
~~<%= link_to 'Back', awards_path %>~~

<%= link_to 'Show', [@student, @award] %> |
<%= link_to 'Back', student_awards_path(@student) %>
```



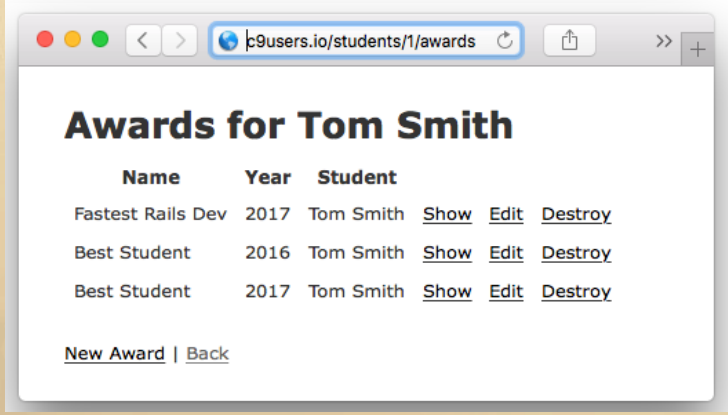
Connecting the student views

- Change `app/views/students/show.html.erb`

```
<%= link_to 'Edit', edit_student_path(@student) %> |
<%= link_to 'Back', students_path %>
```

-

```
<%= link_to 'Edit', edit_student_path(@student) %> |
<%= link_to 'Awards', student_awards_path(@student) %> |
<%= link_to 'Back', students_path %>
```



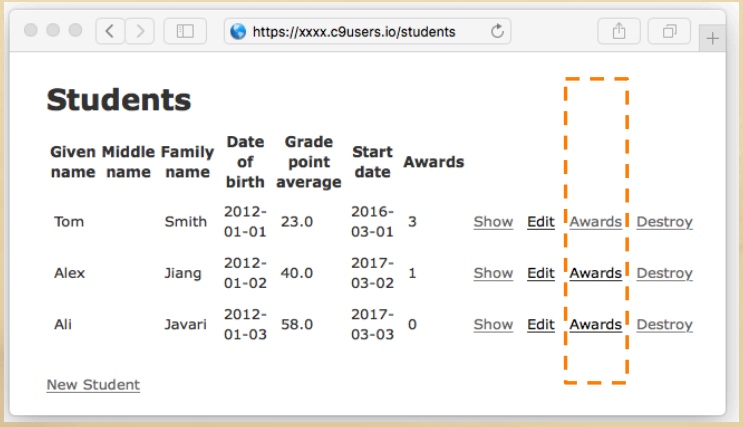
Connecting the student views…

- Change `app/views/students/index.html.erb`

```
<th colspan="3"></th>
<th colspan="4"></th>
```

- Add behind the edit link

```
<td><%= link_to 'Awards', student_awards_path(student) %></td>
```



Is Nesting Worth It?

- It was a lot of work!!
- However it is the "right" approach in Rails.
- It makes the `has_many/belongs_to relationship` explicit on every level, not just in the model
- Routing and controllers are updated so both the web interface and the RESTful web services interface work in nested way.
- Whether to use nesting depends on the application. For the award example, general users may want easy navigation so nesting is better. For an admin, direct interface might be better.
- If you are going to nest resources, do it early.
- Nested resources may require additional interfaces.

Many-to-Many

- What about the relationship between students and courses?
- Students and courses do not belong to each other.
- So there is no need for nested resources.
- However there a lot of connections between them.
- Firstly we need deal with the models.
- Then update the controllers.
- Then the views.
- Rails provides a very good foundation.
- But we still need to add a lot.
- Warning: don't name a table "classes" otherwise you might experience many Rails disasters because of name conflicts.

Create Tables

- Creating a course table is not enough. We need additional table that joins courses and students.

```
$ rails generate scaffold course name:string
```

- The join table requires a few stable. Firstly create a migration

```
$ rails generate migration CreateCoursesStudents
```

It created a file

```
db/migrate/[time_stamp]_create_courses_students.rb
```

```
class CreateCoursesStudents < ActiveRecord::Migration[5.0]
  def change
    create_table :courses_students do |t|
  end
end
```


Coding the migration

- We have reached the boundaries of code auto-generation.
- Need to write the migration

```
class CreateCoursesStudents < ActiveRecord:: Migration
  def change
    create_table :courses_students, id: false do |t|
      t.integer :course_id, null: false
      t.integer :student_id, null: false
    end
  end
end
```

- Rails naming conventions is the key here.
- The table name is the combination of two joining models in alphabetical order.
- The fields within the table are id values for each of the models

Coding the migration…

- `create_table` method is a ActiveRecord method. See API
  http://api.rubyonrails.org/classes/ActiveRecord/ConnectionAdapters/SchemaStatements.html - method-i-create_table

- **create_table**(table_name, comment: nil, **options)

```
create_table(:books) do |t|
   t.column :name, :string, limit: 80
end

create_table(:books) do |t| # with shorthand
  t.string :name, limit: 80
end

create_table(:books)
add_column(:books, :name, :string, {limit: 80})
```

- The option hash include keys like :id, :primary_key, :temporary, :force, :as

  :id Whether to automatically add a primary key column. Defaults to true. Join tables for ActiveRecord::Base.has_and_belongs_to_many should set it to false.

Index in join table

- Rails use id value for tables to improve performance, so data can be processed rapidly.

- The id value is automatically indexed.

- To use your own index, you can add this method in migration:

```
add_index :courses_students, [:course_id, :student_id], unique: true
```

- `add_index` method's API documentation can be read here:
  http://api.rubyonrails.org/classes/ActiveRecord/ConnectionAdapters/SchemaStatements.html#method-i-add_index

  add_index(table_name, column_name, options = {})

- Now run the migration `$ rails db:migrate`

Connecting the Models

- Add the many-to-many relationship to `app/models/student.rb`

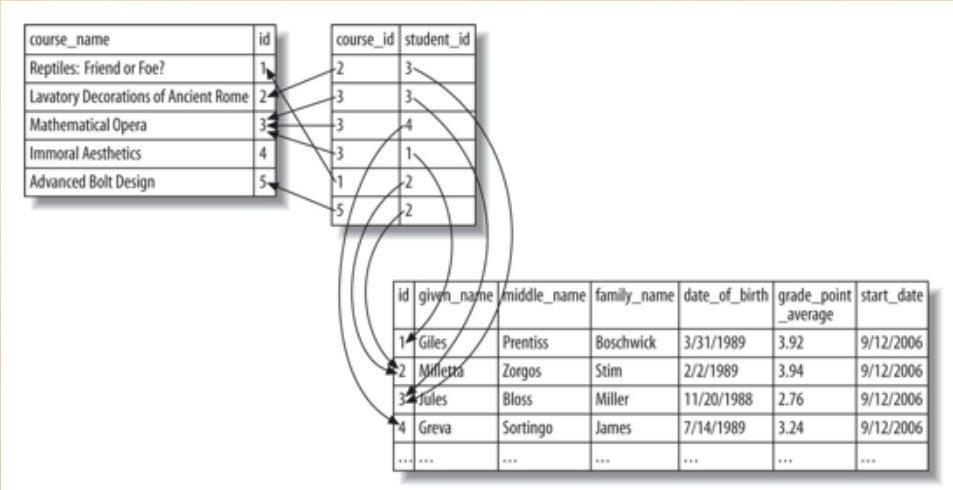
```
has_and_belongs_to_many :courses
```

- Add the many-to-many relationship to `app/models/course.rb`

```
has_and_belongs_to_many :students
```

- That's all for establishing the connection
- Thanks to Rails naming conventions.
- Now table `courses_students` can keep track of the connections.
- Note: `has_and_belongs_to_many` can be controversial; some developers may prefer `has_many :through` relationship, by creating an intermediate table.

Connecting the Models …



(Learning Rails 5, Mark Locklear Eric Gruber)

Methods which may be useful

- `app/models/student.rb` to check whether a student is enrolled?

```
def enrolled_in?(course)
  self.courses.include?(course)
end


def unenrolled_courses
  Course.all - self.courses
end
```

- Remember the minus – operator operating on arrays?

Controllers

- Controllers for many-to-many are simpler than nested resources. However we may need some supporting methods.
- `app/controllers/courses_controller.rb` to add after `destroy`

```
# GET /courses/1/roll
def roll
  @course = Course.find(params[:id])
end                                          # What is this for?
```

- `app/controllers/students_controller.rb`

```
# GET /students/1/courses
def courses
  @student = Student.find(params[:id])
  @courses = @student.courses
end                                          # What is this for?
```

Add course method

```
# POST /students/1/course_add?course_id=2
def course_add
  #Convert ids from routing to objects
  @student = Student.find(params[:id])
  @course = Course.find(params[:course])
  unless @student.enrolled_in?(@course)
    #add course to list using << operator
    @student.courses << @course
    flash[:notice] = 'Student was successfully enrolled'
  else
     flash[:error] = 'Student was already enrolled'
  end
  redirect_to action: "courses", id: @student
end
```


Remove course method

```
def course_remove
  #Convert ids from routing to object
  @student = Student.find(params[:id])

  #get list of courses to remove from query string
  course_ids = params[:courses]
  if course_ids.any?
    course_ids.each do |course_id|
      course = Course.find(course_id)
      if @student.enrolled_in?(course)
        logger.info "Removing student from course #{course.id}"
        @student.courses.delete(course)
        flash[:notice] = 'Course was successfully deleted'
      end
    end
  end
  redirect_to action: "courses", id: @student
end
```

Routing

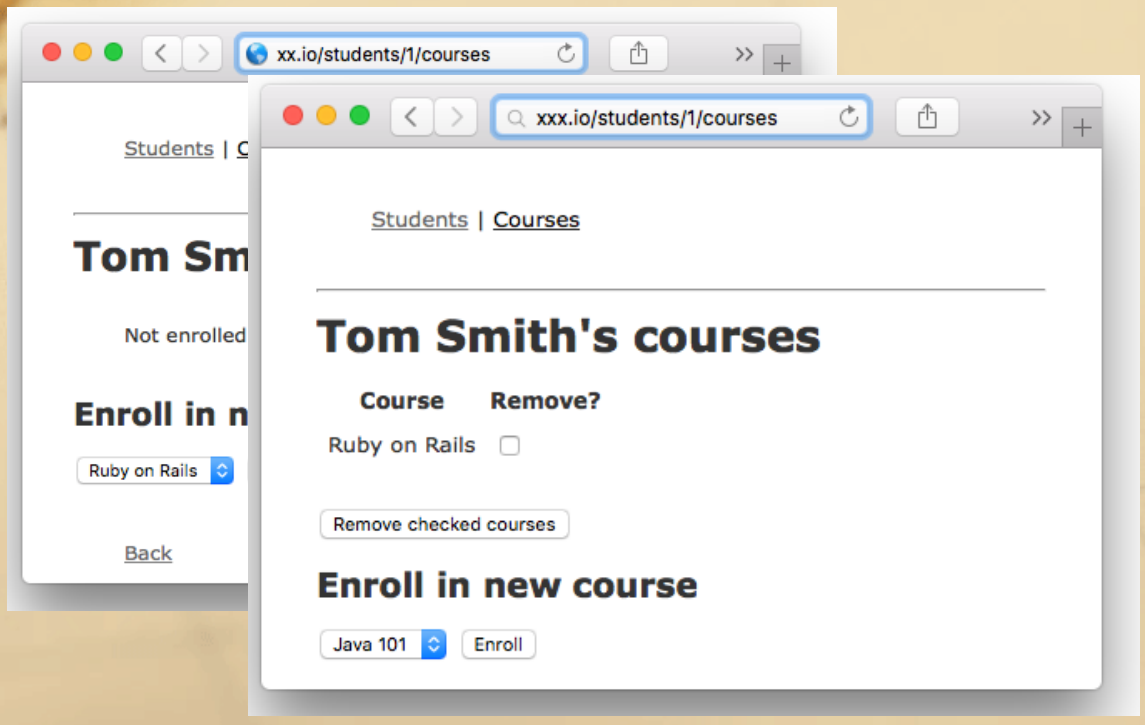
Update the routing: `config/routes.rb`

```
resources :courses
resources :students do
resources :awards
end

resources :courses do
 member do
   get :roll
 end
end

resources :students do
  resources :awards
  member do
    get  :courses
    post :course_add
    post :course_remove
  end
end
```


Now the Views



Making the Views

- Create `app/views/application/_navigation.html.erb`
- Write the navigation partial

```
<p> <%= link_to "Students", students_path %> |
    <%= link_to "Courses", courses_path %> </p>
<hr>
```

- Add this partial to `app/views/layouts/application.html.erb`

```
<body>
   <%= render 'navigation' %>
   <%= yield %>
```

- Student views

  Add course to student list `app/views/students/index.html.erb`

  Table head, insert before Awards

```
<th> Courses </th>
```

  Table body, insert before Awards

```
<td> <%= student.courses.count %> </td>
```


More work on the views…

- Course view `app/views/courses/index.html.erb`

  Table head, insert after (course) name

```
<th> Enrolled </th>
```

  Table body, insert after course name

```
<td> <%= course.students.count %> </td>
```

- Students show `app/views/students/show.html.erb`

  Add the course enrollment info

```
<p> <strong> Courses: </strong >
  <% if !@student.courses.empty? %>
       <%= @student.courses.collect {|c|
          link_to( c.name, c)}.join(",").html_safe %>
  <% else %>
       Not enrolled on any courses yet.
  <% end %>
</p>
```

  Insert another link in between Edit and Awards

```
<%= link_to 'Courses', courses_student_path(@student) %> |
```

Courses in students views

- Create app/views/students/courses.html.erb

```
<h1> <%= @student.name %>'s courses </h1>

<% if @courses.length > 0 %>
  <%= form_tag( course_remove_student_path(@student)) do %>
  <table>
    <thead>
      <tr>
        <th> Course </th>
        <th> Remove? </th>
      </tr>
    </thead>

    <tbody>
    <% for course in @courses do %>
      <tr> <td> <%= course.name %> </td>
           <td> <%= check_box_tag "courses[]", course.id %> </td> </tr>
    <% end %>
   </tbody>
   </table > <br />
```


Courses in students views…

- In app/views/students/courses.html.erb

```
    <%= submit_tag 'Remove checked courses' %>
  <% end %>
<% else %>
  <p> Not enrolled in any courses yet. </p>
<% end %>

<h2> Enroll in new course </h2>
<% if @student.courses.count < Course.count then %>
  <%= form_tag( course_add_student_path(@student)) do %>
      <%= select_tag(:course,
                     options_from_collection_for_select
                    (@student.unenrolled_courses, :id, :name)) %>
      <%= submit_tag 'Enroll' %>
  <% end %>
<% else %>
  <p> <%= @student.name %> is enrolled in every course. </p>
<% end %>
<p> <%= link_to 'Back', @student %> </p>
```

Courses views…

- In `app/views/courses/show.html.erb`

  Insert another link in between Edit and Back

```
<%= link_to 'Roll', roll_course_path(@course) %> |
```

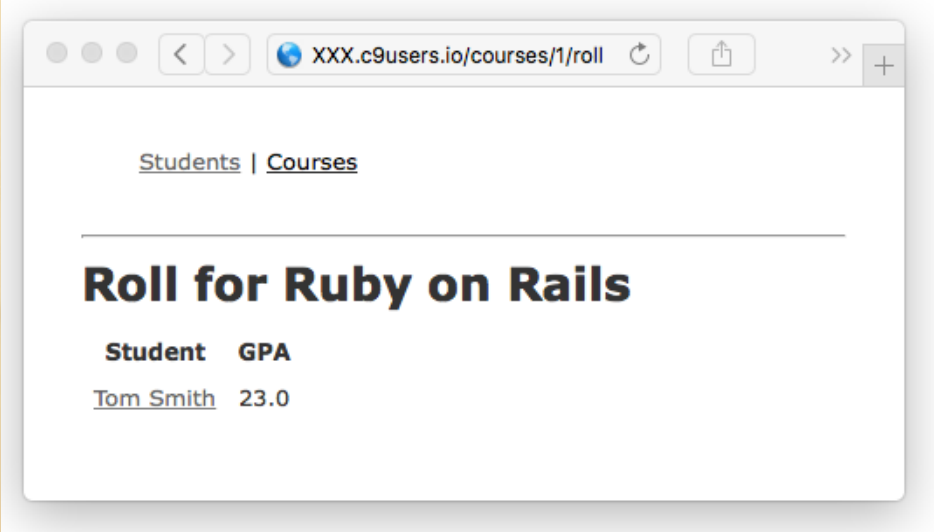
- Create `app/views/courses/roll.html.erb`

```
<h1> Roll for <%= @course.name %> </h1>

<% if @course.students.count > 0 %>
  <table>
    <thead> <tr> <th> Student </th>
                 <th> GP </th> </tr>
    </thead>
    <tbody> <% @course.students.each do |student| %>
        <tr> <td> <%= link_to student.name, student %> </td>
             <td> <%= student.grade_point_average %> </td>
        </tr>
      <% end %>
    </tbody >
   </table >
<% else %>
  <p> No students are enrolled. </p>
<% end %>
```


Viewing the Roll

- `http://... /courses/1/roll`



Summary

- Multiple tables
- Association methods
- Dependent tables
- Nested resources
- Implementing nested controllers and views
- Many-to-many
- Join tables
- Many-to-many controllers
- Many-to-many views