Assignment 2: Riding Rails

CS390P: Web Application Architecture

John Samson

October 14th, 2018

Abstract

The focus of this paper will be expanding and upgrading the simple web application we completed in our previous assignment. Here we will examine how Ruby on Rails can create a *join table* to allow two databases to create a *many-to-many* relationship between the Students and Sections database tables of our web application. In addition, we'll begin investigating ways of *validating* the contents of our various input fields whenever database transactions are made from the server side of our web application. We'll add functionality to our pages, starting with a simple Search function. We'll end our work here by testing our web application; first we'll get to know Rails' built-in testing framework, Minitest. Then we'll perform *system* or *integration testing*; using the Selenium Framework and Chrome Browser we can automatically enter fields and click buttons on our website to test specific functionality from a *user's* perspective.

Introduction

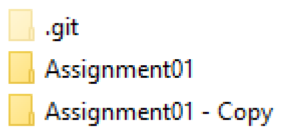
Previously we created a simple university website that manages registration information. We created a simple *one-to-many* relationship between Courses and Sections and added a simple dropdown menu to select a Course when creating a new Section. Today we will add a *many-to-many* relationship between Sections and Students, a new Rails scaffold we will create. We'll look into validating each input field for things like data type, string length, and numerical range. We'll create a more robust application, that displays appropriate Student and Section data in each *show* view and add a simple Search function to search each database table. We'll close our work here with an exploration of both unit and system/integration testing basics, using (respectively) the built-in Minitest framework with Rails and installing the Capybara framework and Selenium browser automation tool for Google Chrome.

Prerequisites

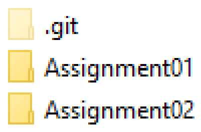
The following web application was created on Windows 10, 64-bit Professional edition, using the RailsInstaller Ruby on Rails package installer, available here: http://railsinstaller.org/en. For a more detailed setup tutorial for both Ruby on Rails and Git please see the previous paper in this series. Necessary components are otherwise listed individually below:

• Ruby language + Devkit version 2.3.3p222:	https://rubyinstaller.org/downloads/
• Rails version 5.1.6:	https://rubyonrails.org/
• Bundler version 1.16.5:	https://bundler.io/
• SQLite3 version 3.8.7.2:	https://www.sqlite.org/download.html
• Git version 2.18.0.windows.1:	https://git-scm.com/download/win

Once you have all the necessary components installed, let's create a new project, building from our previous work, and make a quick initial commit to our Git repository. First let's create a copy of our first Assignment, and rename it 'Assignment02':



.git
Assignment01
Assignment01 - Copy



.git
Assignment01
Assignment02

- Now navigate to the root of your repository folder, and add the new Assignment02 folder and all files in it:

```
C:\CS390P-Web-Application-Architecture>git add .
```

- Now git commit:

```
C:\CS390P-Web-Application-Architecture>git commit -am "New Project Assignment02"
```

- and push:

```
C:\CS390P-Web-Application-Architecture>git push origin master
```

Now we're ready to begin altering our work from our previous assignment. There is quite a steep learning curve from the simple test case of our previous example to a much more dynamic, fully-functioning website, a website we will both *validate* and *test* for desired behaviors. Let's roll up our sleeves and find out just how much we *don't know* about web design with Ruby on Rails.

Step 1: Add a many-to-many relationship between Students and Sections

Any good university is nothing without its Students, so let's add that to our database by creating a new scaffold, called 'Student', with a single string field 'name':

```
C:\CS390P-Web-Application-Architecture>rails generate scaffold Student name:string
```

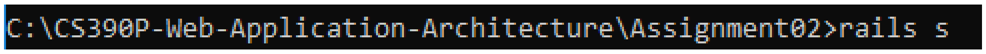
- Likewise, we can save time by abbreviating 'rails generate' to simply 'rails g':

```
C:\CS390P-Web-Application-Architecture\Assignment02>rails g scaffold Student name:string
```

- Make sure to migrate the database, as we've made changes when we created a new Scaffold:

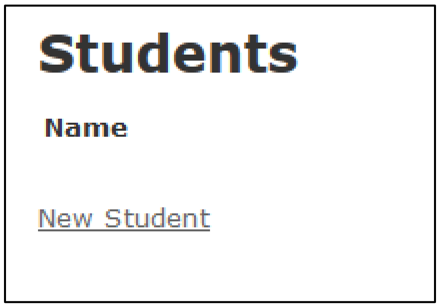
```
C:\CS390P-Web-Application-Architecture\Assignment02>rails db:migrate
```

- Now let’s run the server to make sure our new scaffold was created properly:

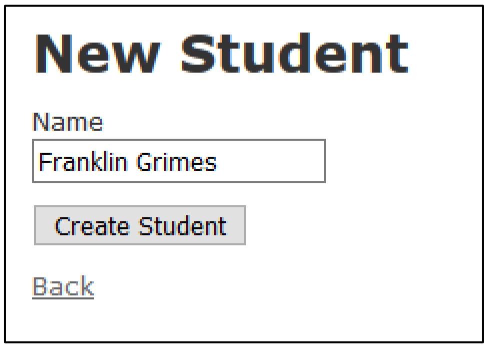


```
C:\CS390P-Web-Application-Architecture\Assignment02>rails s
```

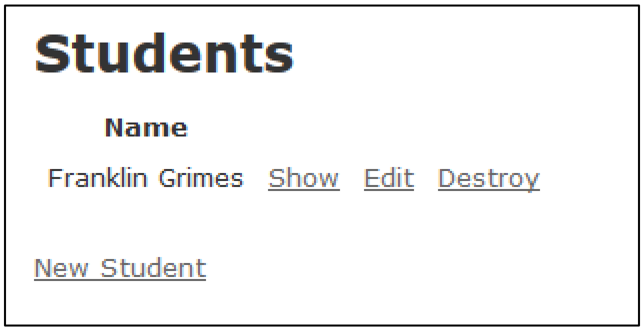
- Navigating to localhost:3000/students shows our new scaffold:



- Create a new Student:



- We can see we successfully created a new Student:



Now that we have a scaffold created for both Students and Sections, we can begin to create a *many-to-many* relationship between them. Upon some research we have found 3 distinct methods for creating this relationship, detailed below (Stack Overflow, 2018):

- Option 1: use `has_and_belongs_to_many` to create references between Students and Sections without using a join table or additional data model.
- Option 2: use Rails to create a simple join table with either of the following commands (this will create a Model and Controller, but no View):

‘`rails generate model SectionsStudents section:references student:references`’ *or*

‘`rails generate migration CreateJoinTableSectionsStudents`’

- Option 3: Create a new scaffold that references both Students and Sections, using the '`has_many, through:`' relationship. This will create a new data model (called 'Enrollments'), that will hold our join table to create the *many-to-many* relationship between the two, and a new view for them. First create a new scaffold:

```
C:\CS390P-Web-Application-Architecture\Assignment02>rails g scaffold Enrollment section:references student:references
```

```
rails g scaffold Enrollment section:references student:references
```

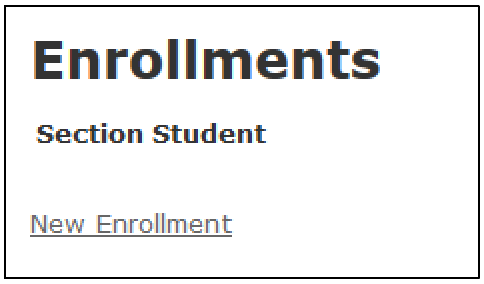
- Don't forget to create a new database migration:

```
C:\CS390P-Web-Application-Architecture\Assignment02>rails db:migrate
```

- Now run the server and test that Enrollments was created properly:

```
C:\CS390P-Web-Application-Architecture\Assignment02>rails s
```

- Enrollments:

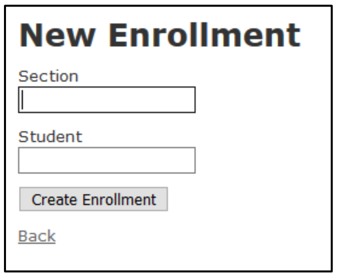


Enrollments

Section Student

New Enrollment

- New Enrollment:



New Enrollment

Section

Student

Create Enrollment

Back

- Now that we've created appropriate scaffolds, it would be a great idea to commit and push to Git:

```
C:\CS390P-Web-Application-Architecture>git add .
```

```
C:\CS390P-Web-Application-Architecture>git commit -am "Created Student and Enrollment scaffolds"
```

```
C:\CS390P-Web-Application-Architecture>git push origin master
```

Create the many-to-many Relationship:

- Opening our '`Assignment02\app\models\enrollment.rb`' shows us we the scaffold created our Enrollment model properly:

```
class Enrollment < ApplicationRecord
  belongs_to :section
  belongs_to :student
end
```

- We need to update our models to reflect a many-to-many relationship between Sections and Students. Edit the ‘`Assignment02\app\models\student.rb`’ include the following:

```ruby
class Student < ApplicationRecord
  has_many :enrollments
  has_many :sections, through: :enrollments
end
```

- Likewise edit ‘`Assignment02\app\models\section.rb`'

```ruby
class Section < ApplicationRecord
  belongs_to :course
  has_many :enrollments
  has_many :students, through: enrollments
end
```

- Make sure Course still *has_many* Sections (Note: we’ll worry ‘dependent: :destroy’ in a later section):

```ruby
class Course < ApplicationRecord
  has_many :sections, dependent: :destroy
end
```

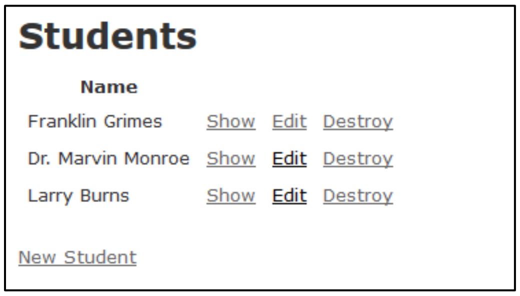
- Now we need to change the text input field to a dropdown menu. Edit the following ‘`Assignment02\app\views\enrollments\_form.html.erb`’ file:

```erb
<div class="field">
  <%= form.label :student_id %>
  <%= form.text_field :student_id, id: :enrollment_student_id %>
</div>
```

```erb
<div class="field">
  <%= form.label :student_id %>
  <%= form.collection_select :student_id, Student.all, :id, :name %>
</div>
```

Now that we’ve created both scaffolds, our many-to-many relationship, and dropdown menu for Students in Enrollments, let’s create some new Students to test everything out:

- Create new Students:

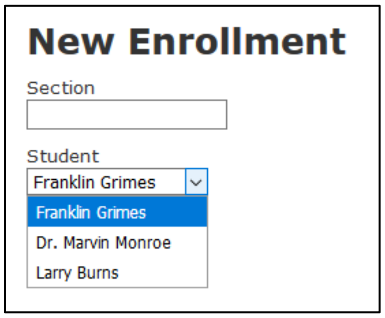


Students

Name			
Franklin Grimes	Show	Edit	Destroy
Dr. Marvin Monroe	Show	Edit	Destroy
Larry Burns	Show	Edit	Destroy

New Student

- Dropdown menu in new Enrollment:



New Enrollment

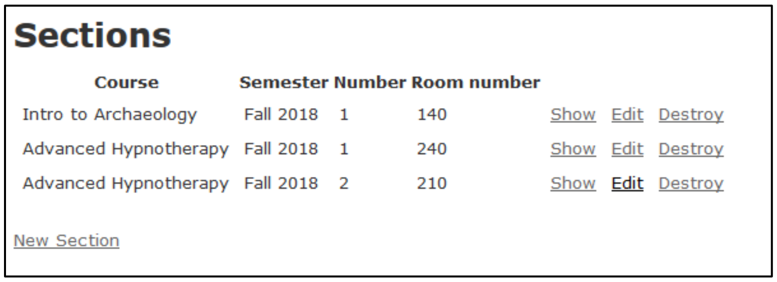
Section

Student

Franklin Grimes

We can see we've created our desired dropdown for Students in Enrollments, let's see about doing the same thing for Sections.

- First let's create a few Sections for testing purposes:



Sections

Course	Semester	Number	Room number			
Intro to Archaeology	Fall 2018	1	140	Show	Edit	Destroy
Advanced Hypnotherapy	Fall 2018	1	240	Show	Edit	Destroy
Advanced Hypnotherapy	Fall 2018	2	210	Show	Edit	Destroy

New Section

- Open the '`Assignment02\app\views\enrollments\_form.html.erb`' file, and change the following to add a dropdown menu for Sections in Enrollments:

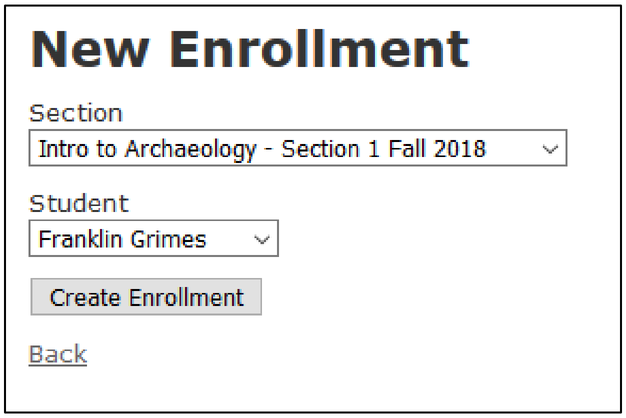
```
<div class="field">
  <%= form.label :section_id %>
  <%= form.text_field :section_id, id: :enrollment_section_id %>
</div>
```

```
<div class="field">
  <%= form.label :section_id %>
  <%= form.collection_select :section_id, Section.all, :id, :name %>
</div>
```

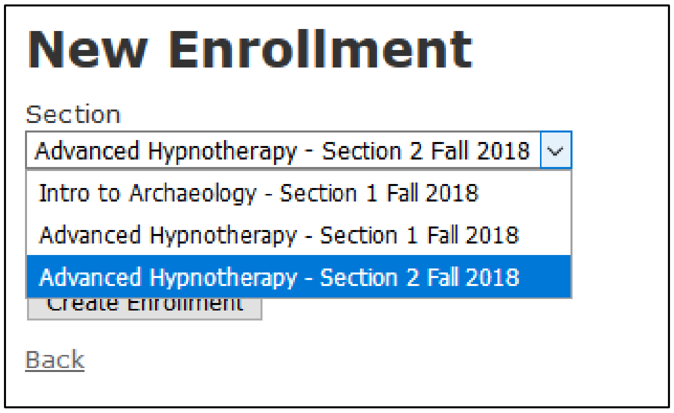
- We need to ensure that Section has a string called 'name', that displays the appropriate information for the dropdown menu in Enrollments.
- Edit the '`Application02\app\models\section.rb`' file to include a new method definition *name*:

```
class Section < ApplicationRecord
  belongs_to :course
  has_many :enrollments
  has_many :students, through: enrollments
  def name
    "#{course.name} #{" - Section "} #{number} #{" - "} #{semester}"
  end
end
```

- Navigate to the New Enrollment page to see the dropdown menu we created:

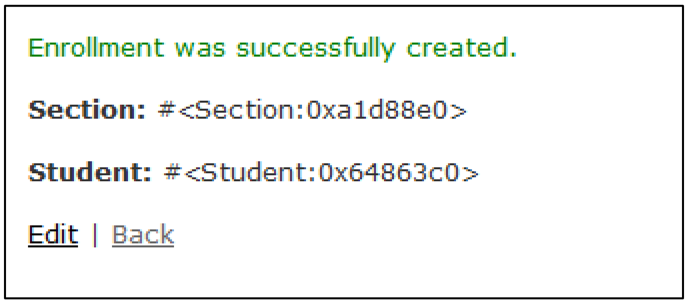


- We can see our dropdown is working properly:



New Enrollment
Section
Advanced Hypnotherapy - Section 2 Fall 2018
Intro to Archaeology - Section 1 Fall 2018
Advanced Hypnotherapy - Section 1 Fall 2018
Advanced Hypnotherapy - Section 2 Fall 2018
Create Enrollment
Back

- Here we created an Enrollment successfully, but something doesn't quite look right:



```
Enrollment was successfully created.

Section: #<Section:0xa1d88e0>

Student: #<Student:0x64863c0>

Edit | Back
```

- Let's investigate the '`Application02\app\views\enrollments\show.html.erb`' view file to find the problem.

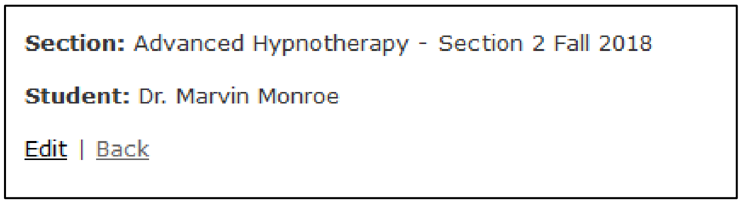
- Change this:

```
<p>
  <strong>Section:</strong>
  <%= @enrollment.section %>
</p>

<p>
  <strong>Student:</strong>
  <%= @enrollment.student %>
</p>
```

- To this:

```
<p>
  <strong>Section:</strong>
  <%= @enrollment.section.name %>
</p>

<p>
  <strong>Student:</strong>
  <%= @enrollment.student.name %>
</p>
```

- Now we're displaying the appropriate information for creating a new Section properly:

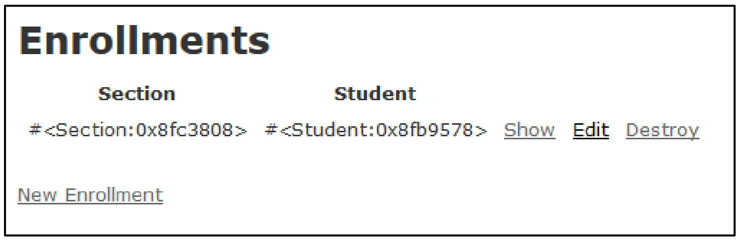


Section: Advanced Hypnotherapy - Section 2 Fall 2018

Student: Dr. Marvin Monroe

Edit | Back

- Likewise, we have a similar small issue in our Enrollment view:



Enrollments

Section	Student			
#<Section:0x8fc3808>	#<Student:0x8fb9578>	Show	Edit	Destroy

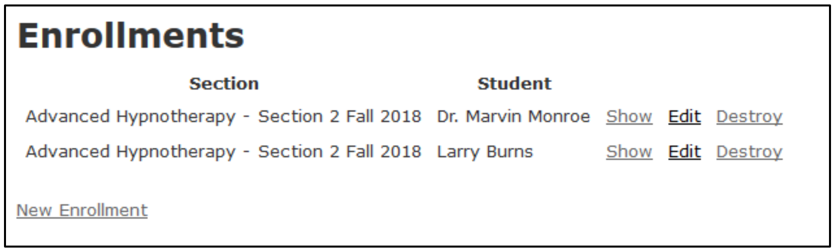
New Enrollment

- Edit 'app\views\enrollments\index.html.erb'

```
<td><%= enrollment.section %></td>
<td><%= enrollment.student %></td>
```

```
<td><%= enrollment.section.name %></td>
<td><%= enrollment.student.name %></td>
```

- Enrollments is much more readable now:



Enrollments

Section	Student			
Advanced Hypnotherapy - Section 2 Fall 2018	Dr. Marvin Monroe	Show	Edit	Destroy
Advanced Hypnotherapy - Section 2 Fall 2018	Larry Burns	Show	Edit	Destroy

New Enrollment

We can see that we have a few different options when creating a many-to-many relationship using Rails. In some cases, it makes sense to create a view for a named join table, in this case it serves us well. In other cases, it may be more appropriate to keep such information hidden from the user. We'll learn more on how to address specific columns of a database table soon. For now, let's simply marvel at our incredibly cool dropdown menus.

Step 2: Validation

An important aspect of creating secure websites is *validating* that input data meets certain specification. While most browsers offer built-in input validation, we want to protect every database transaction from any invalid input, or worse, the potential for malicious attacks such as SQL Query injection (Ruby on Rails Guides, 2018). We cannot rely on any browser to do this for us, it must be implemented in the model on the server side.

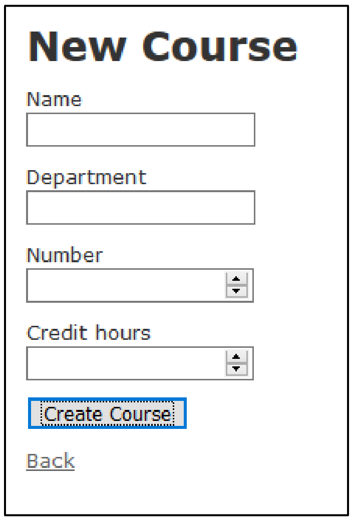
- First let's create a basic form of null protection. Edit the model file '`Application02\app\models\course.rb`':

```ruby
class Course < ApplicationRecord
  has_many :sections, dependent: :destroy
end
```

- Add lines to *validate* the various input fields, checking to ensure no empty input occurs:

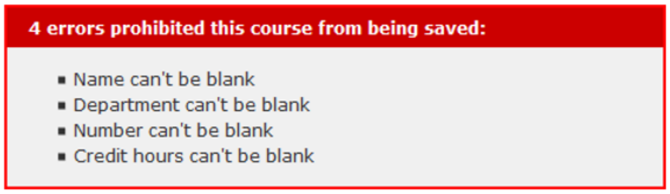
```ruby
class Course < ApplicationRecord
  has_many :sections, dependent: :destroy
  validates :name, presence: true
  validates :department, presence: true
  validates :number, presence: true
  validates :credit_hours, presence: true
end
```

- If we try to create a new Course with no data, we can see we are given some error messages:



New Course

Name
Department
Number
Credit hours



4 errors prohibited this course from being saved:

- Name can't be blank
- Department can't be blank
- Number can't be blank
- Credit hours can't be blank

- We can add more robust features, including checking for length, input type, or numerical range (Wintermeyer, 2012). We can also specify an error message:

```
length: { is: 3}
```

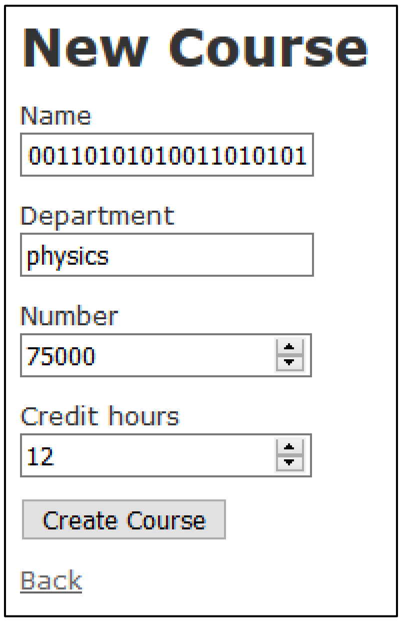
```
format: { with: /\A[a-zA-Z a-zA-Z]+\z/, message: "only allows letters" },
```

```
numericality: { greater_than: 1000,
                less_than_or_equal_to: 5000,
                message: "must be between 1000 and 5000" },
```

- A more thorough series of validations for the Course model:

```
class Course < ApplicationRecord
  has_many :sections, dependent: :destroy
  validates :name,          presence: true,
                            format: { with: /\A[a-zA-Z a-zA-Z]+\z/, message: "only allows letters" },
                            length: { maximum: 50, message: "must be less than 50 characters in length" }
  validates :department,    presence: true,
                            format: { with: /\A[A-Z]/, message: "only allows capital letters" },
                            length: { is: 3 }
  validates :number,        presence: true,
                            format: { with: /\0[0-9]/, message: "only allows numbers" },
                            numericality: { greater_than: 1000,
                                            less_than_or_equal_to: 5000,
                                            message: "must be between 1000 and 5000" },
                            length: { is: 4 }
  validates :credit_hours,  presence: true,
                            format: { with: /\0[0-9]/, message: "only allows numbers" },
                            numericality: { greater_than: 0,
                                            less_than_or_equal_to: 6,
                                            message: "must be between 1 and 6" },
                            length: { is: 1 }
end
```

- Once again, we receive many errors when we attempt to create a new Course with invalid input:



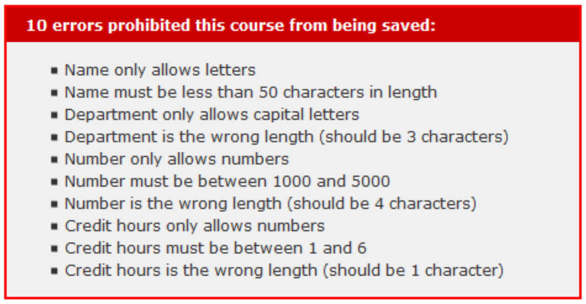
New Course

Name: 00110101010011010101

Department: physics

Number: 75000

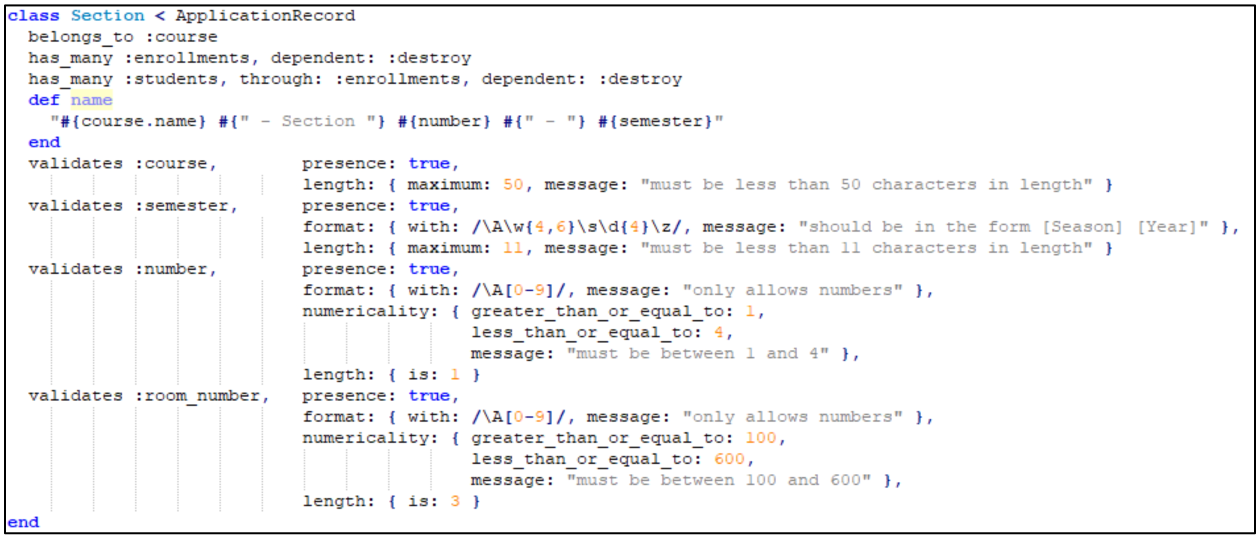
Credit hours: 12



10 errors prohibited this course from being saved:

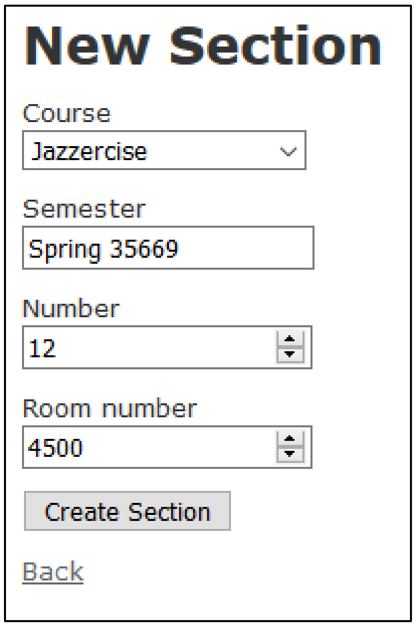
- Name only allows letters
- Name must be less than 50 characters in length
- Department only allows capital letters
- Department is the wrong length (should be 3 characters)
- Number only allows numbers
- Number must be between 1000 and 5000
- Number is the wrong length (should be 4 characters)
- Credit hours only allows numbers
- Credit hours must be between 1 and 6
- Credit hours is the wrong length (should be 1 character)

- We can also write fairly robust validations for Sections, this time using more detailed *regular expressions* to specify exactly how we want our 'semester' input to look (Castello, 2018).



```ruby
class Section < ApplicationRecord
  belongs_to :course
  has_many :enrollments, dependent: :destroy
  has_many :students, through: :enrollments, dependent: :destroy
  def name
    "#{course.name} #{" - Section "} #{number} #{" - "} #{semester}"
  end
  validates :course,          presence: true,
                              length: { maximum: 50, message: "must be less than 50 characters in length" }
  validates :semester,        presence: true,
                              format: { with: /\A\w{4,6}\s\d{4}\z/, message: "should be in the form [Season] [Year]" },
                              length: { maximum: 11, message: "must be less than 11 characters in length" }
  validates :number,          presence: true,
                              format: { with: /\A[0-9]/, message: "only allows numbers" },
                              numericality: { greater_than_or_equal_to: 1,
                                              less_than_or_equal_to: 4,
                                              message: "must be between 1 and 4" },
                              length: { is: 1 }
  validates :room_number,     presence: true,
                              format: { with: /\A[0-9]/, message: "only allows numbers" },
                              numericality: { greater_than_or_equal_to: 100,
                                              less_than_or_equal_to: 600,
                                              message: "must be between 100 and 600" },
                              length: { is: 3 }
end
```

- If we attempt to create a clearly 'bad' Section in the database, we get the desired error messages:



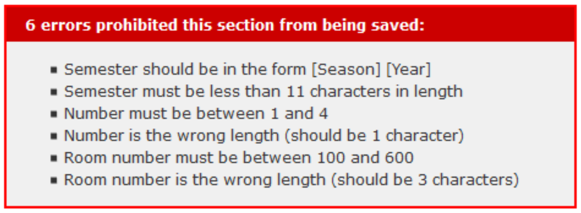
New Section

Course: Jazzercise

Semester: Spring 35669

Number: 12

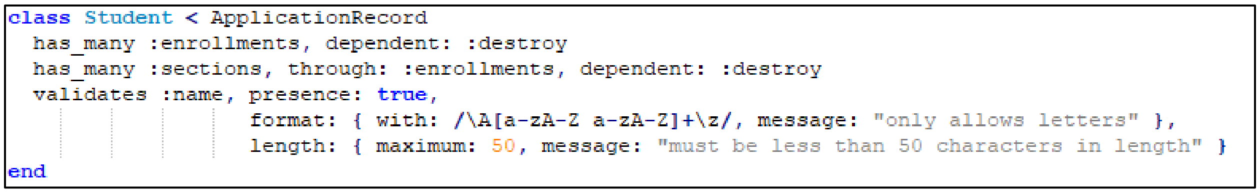
Room number: 4500



6 errors prohibited this section from being saved:

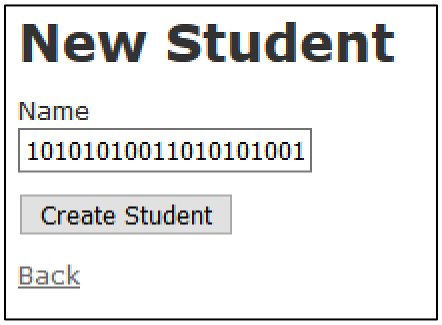
- Semester should be in the form [Season] [Year]
- Semester must be less than 11 characters in length
- Number must be between 1 and 4
- Number is the wrong length (should be 1 character)
- Room number must be between 100 and 600
- Room number is the wrong length (should be 3 characters)

- Students has much simpler validations we can perform:

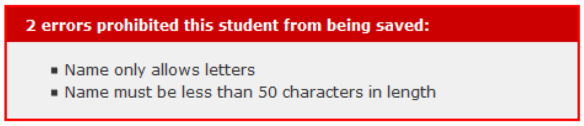


```ruby
class Student < ApplicationRecord
  has_many :enrollments, dependent: :destroy
  has_many :sections, through: :enrollments, dependent: :destroy
  validates :name, presence: true,
                   format: { with: /\A[a-zA-Z a-zA-Z]+\z/, message: "only allows letters" },
                   length: { maximum: 50, message: "must be less than 50 characters in length" }
end
```

- Once again, we validate for undesired input:



```
10101010011010101001
```



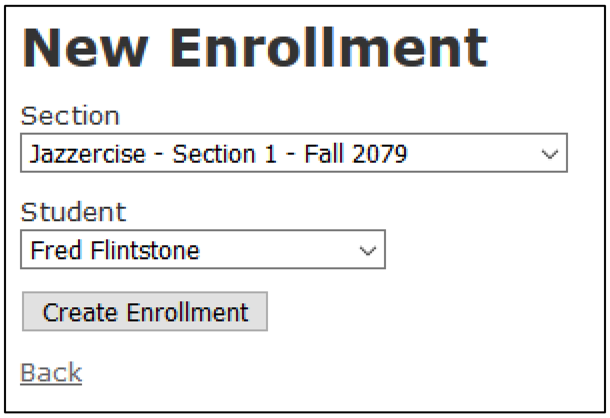
2 errors prohibited this student from being saved:

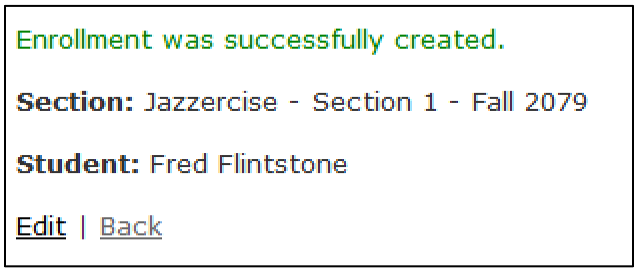
- Name only allows letters
- Name must be less than 50 characters in length

- Enrollments is a bit more complicated, so we leave more thorough validations for them to the reader as an advanced exercise. Here we'll simply validate against null input and length:

```
class Enrollment < ApplicationRecord
  belongs_to :section
  belongs_to :student
  validates :section,   presence: true,
                        length: { maximum: 50, message: "must be less than 50 characters in length" }
  validates :student,   presence: true,
                        length: { maximum: 50, message: "must be less than 50 characters in length" }
end
```

- Expectedly we receive no errors:





Enrollment was successfully created.

Section: Jazzercise - Section 1 - Fall 2079

Student: Fred Flintstone

Edit | Back

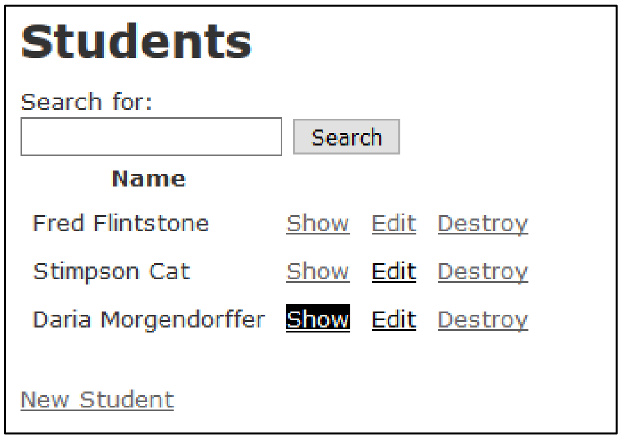
Step 3: Update Views and Check foreign keys

Now that we've created our many-to-many relationship and validated our input fields, let's confirm that our relationship is working as desired, and add some relevant information to our Sections and Students views.

- First let's add a simple loop to our Student 'show' view to display all Sections that Student is enrolled in. Edit the Student view file '`\app\views\students\show.html.erb`' and add the following block:

```
<p>
  <strong>Sections:</strong>
  <ol>
  <% @student.sections.each do |section| %>
    <li><%= section.name %></li>
  <% end %>
  </ol>
</p>
```

- When we *show* a desired Student, we can see the Sections they're currently enrolled in:



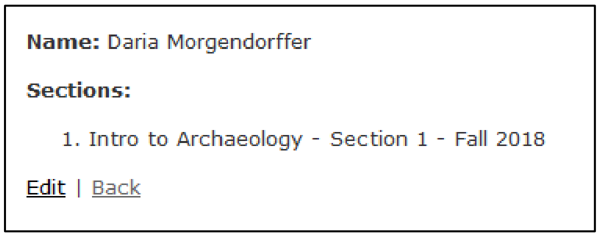
Students

Search for:

Search

Name			
Fred Flintstone	Show	Edit	Destroy
Stimpson Cat	Show	Edit	Destroy
Daria Morgendorffer	Show	Edit	Destroy

New Student



Name: Daria Morgendorffer

Sections:

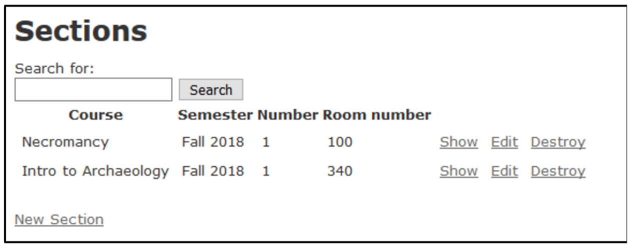
1. Intro to Archaeology - Section 1 - Fall 2018

Edit | Back

- Now let's likewise show each enrolled Student in each Section. Edit the Section view file '`\app\views\sections\show.html.erb`' and add the following block:

```
<p>
  <strong>Students:</strong>
  <ol>
  <% @section.students.each do |student| %>
    <li><%= student.name %></li>
  <% end %>
  </ol>
</p>
```

- We can see the appropriate data in the Show view for a Section:

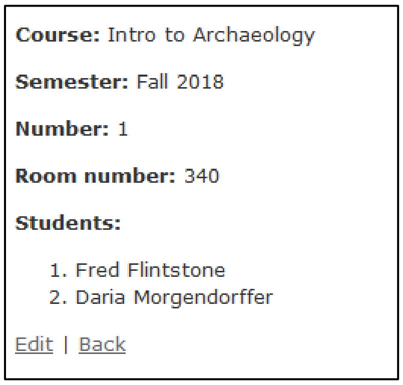


Sections

Search for: [ ] Search

Course	Semester	Number	Room number			
Necromancy	Fall 2018	1	100	Show	Edit	Destroy
Intro to Archaeology	Fall 2018	1	340	Show	Edit	Destroy

New Section



Course: Intro to Archaeology

Semester: Fall 2018

Number: 1

Room number: 340

Students:

1. Fred Flintstone
2. Daria Morgendorffer

Edit | Back

Step 4: Add a search in each index function

Now that we've setup our many-to-many relationship and validated user input, let's expand our website to include a simple Search function. We experienced quite a learning curve here, so we will take care to explain each step in detail, to save the poor reader from recreating our own mistakes.

The general idea is that we need to create a parameter to be passed from the view to the controller for a given page. If a parameter is present, then the controller will then execute some functionality to limit the rows returned by the index. To gain a better understanding of SQLite3 database operations, and to troubleshoot implementing our Search bar, we recommend a third-party database browser tool, DB Browser for SQLite, available here https://sqlitebrowser.org/. This is an excellent tool for database newcomers, as we will be able to directly see the contents of our database, as well as confirm that database transactions are behaving as we want them to.

Add a search to Students index:

- First edit the '`app\views\students\index.html.erb`' file to include the following:

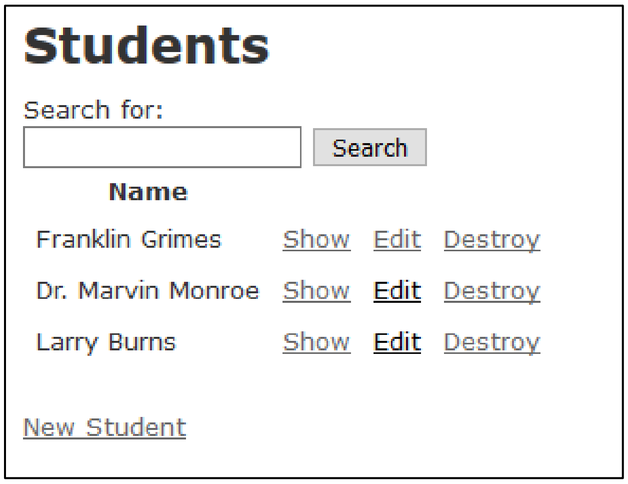
```
<h1>Students</h1>

<%= form_tag(students_path, method: :get) do %>
  <%= label_tag(:query, "Search for:") %>
  <%= text_field_tag(:query) %>
  <%= submit_tag("Search") %>
<% end %>

<table>
```

- Next, edit the ‘`app\controllers\students_controller.rb`’ file to add search functionality:

```
# GET /students
# GET /students.json
def index
  @students = Student.all
end
```

```
# GET /students
# GET /students.json
def index
  if params[:query]
    @students = Student.where("name like ?", "%#{params[:query]}%")
  else
    @students = Student.all
  end
end
```

- Now we have a search bar to search all students:



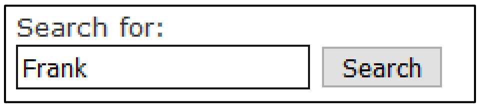
Students

Search for:

Search

Name			
Franklin Grimes	Show	Edit	Destroy
Dr. Marvin Monroe	Show	Edit	Destroy
Larry Burns	Show	Edit	Destroy

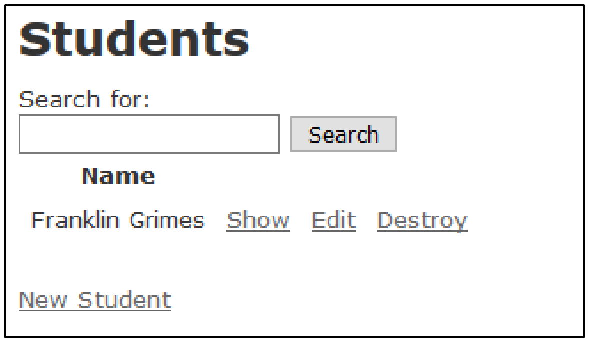
New Student



Search for:

Frank

Search



Students

Search for:

Search

Name			
Franklin Grimes	Show	Edit	Destroy

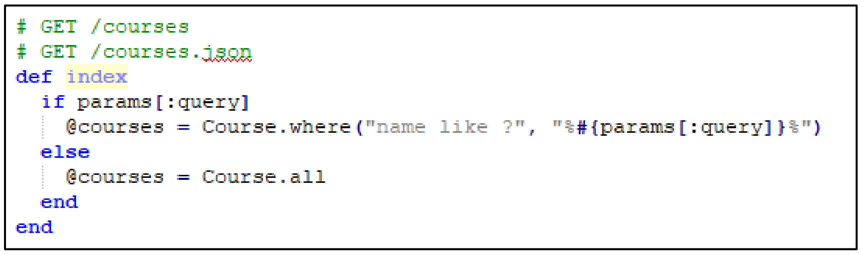
New Student

<u>Add a search bar to Courses index:</u>

- Implementing a Search function for Courses is just as straightforward. First create a parameter and text input field in the view file ‘`\app\views\courses\index.html.erb`’:

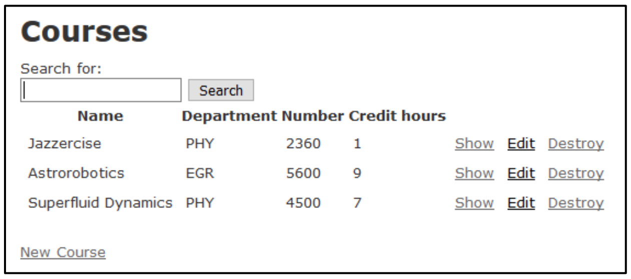
```
<%= form_tag(courses_path, method: :get) do %>
  <%= label_tag(:query, "Search for:") %>
  <%= text_field_tag(:query) %>
  <%= submit_tag("Search") %>
<% end %>
```

- Then edit the ‘index’ method in ‘`app\controllers\courses_controller.rb`’ to the following:



```ruby
# GET /courses
# GET /courses.json
def index
  if params[:query]
    @courses = Course.where("name like ?", "%#{params[:query]}%")
  else
    @courses = Course.all
  end
end
```

- We can see our search bar now:
- And search for courses by name:

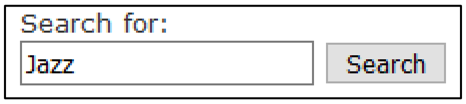


Courses

Search for:

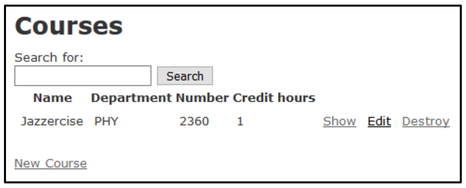
Name	Department	Number	Credit hours			
Jazzercise	PHY	2360	1	Show	Edit	Destroy
Astrorobotics	EGR	5600	9	Show	Edit	Destroy
Superfluid Dynamics	PHY	4500	7	Show	Edit	Destroy

New Course



Search for:

Jazz



Courses

Search for:

Name	Department	Number	Credit hours			
Jazzercise	PHY	2360	1	Show	Edit	Destroy

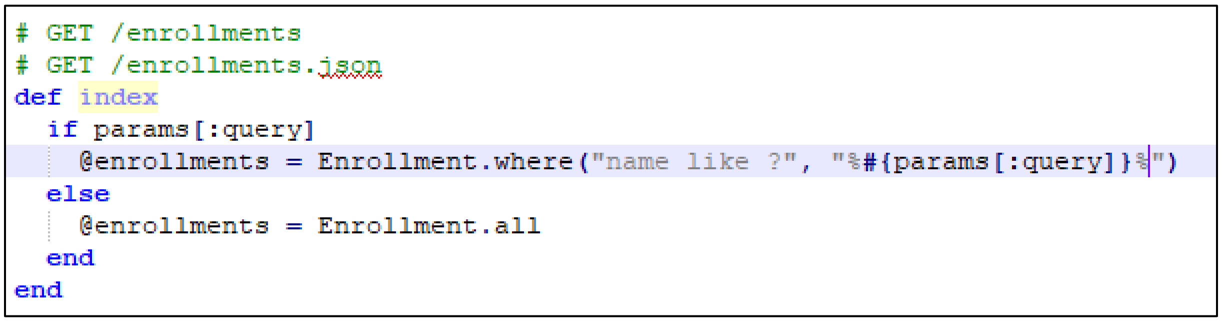
New Course

Add a search bar to Enrollments index

- Once again let’s start by editing the ‘`\app\views\enrollments\index.html.erb`’ file to include our search function and ‘:query’ parameter:

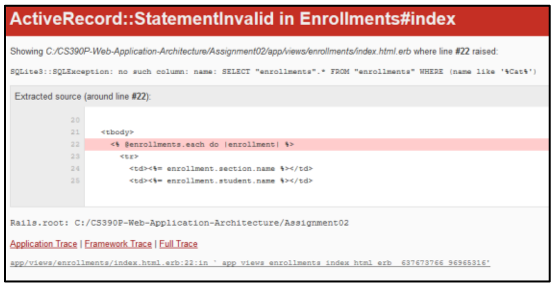
```erb
<%= form_tag(enrollments_path, method: :get) do %>
  <%= label_tag(:query, "Search for:") %>
  <%= text_field_tag(:query) %>
  <%= submit_tag("Search") %>
<% end %>
```

- Again, we edit the index in the ‘`app\controllers\enrollments_controller.rb`’ enrollments controller file:



```ruby
# GET /enrollments
# GET /enrollments.json
def index
  if params[:query]
    @enrollments = Enrollment.where("name like ?", "%#{params[:query]}%")
  else
    @enrollments = Enrollment.all
  end
end
```

- Testing our Search in Enrollments (likewise with Sections) returns the following error:



ActiveRecord::StatementInvalid in Enrollments#index

Showing *C:/CS390P-Web-Application-Architecture/Assignment02/app/views/enrollments/index.html.erb* where line **#22** raised:

```
SQLite3::SQLException: no such column: name: SELECT "enrollments".* FROM "enrollments" WHERE (name like '%Cat%')
```

Extracted source (around line **#22**):

```
20
21    <tbody>
22      <% @enrollments.each do |enrollment| %>
23        <tr>
24          <td><%= enrollment.section.name %></td>
25          <td><%= enrollment.student.name %></td>
```

```
Rails.root: C:/CS390P-Web-Application-Architecture/Assignment02
```

Application Trace | Framework Trace | Full Trace

```
app/views/enrollments/index.html.erb:22:in `_app_views_enrollments_index_html_erb__637673766_96965316'
```

- If we look at the database schema ‘`\db\schema.rb`’ file we can clearly see we are missing these fields for both. We can generate a simple migration to add a string field ‘name’ for both with the following commands:

```
C:\CS390P-Web-Application-Architecture\Assignment02>rails g migration add_name_to_enrollments name:string
```

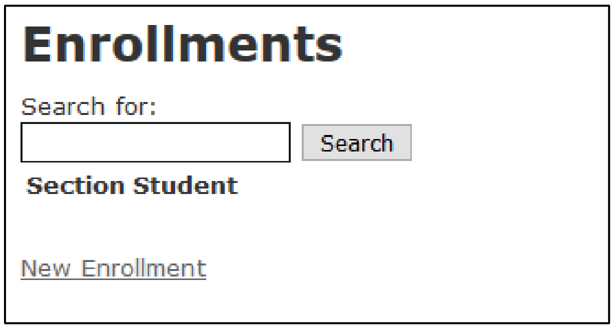
```
C:\CS390P-Web-Application-Architecture\Assignment02>rails g migration add_name_to_sections name:string
```

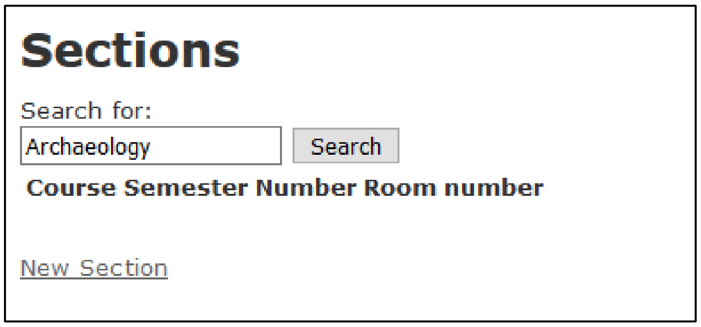
- Alternatively, we could have simply created a name field for both when we created the scaffolds:

```
C:\CS390P-Web-Application-Architecture\Assignment02>rails g scaffold Section course:references semester:string number
:integer room_number:integer name:string
```

```
C:\CS390P-Web-Application-Architecture\Assignment02>rails g scaffold Enrollments section:references student:references
name:string --force-plural
```

- While we have fixed our error (for now), both Enrollments and Sections yield no results.



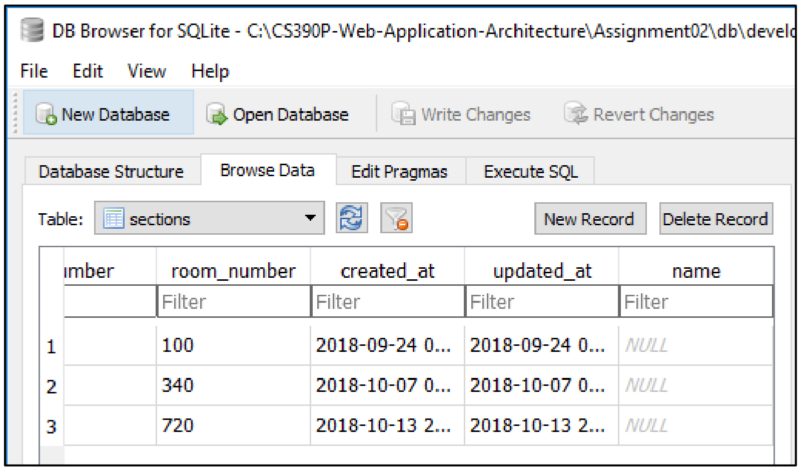


This is a very serious problem for our Search bar, one that confounded us for quite some time. We decided to seek out third-party help and satisfy our need to look directly inside the SQLite3 database.

Troubleshooting Tool: DB Browser for SQLite

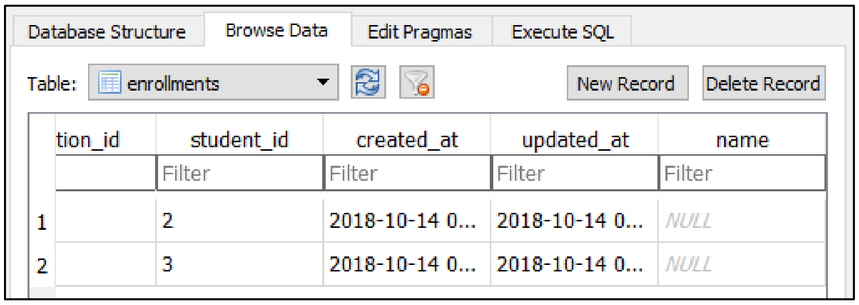
Once again, we will recommend DB Browser for SQLite, available here: https://sqlitebrowser.org/. Download an appropriate version for your system and architecture, install the file and click on 'Open Database' to navigate to the '`\db\development.sqlite3`' database file to begin exploring the database. Note that Rails and DB Browser for SQLite may not play nicely, so it may be idea to close DB Brower before testing out our web application after making changes.

- DB Browser for SQLite:



	ımber	room_number	created_at	updated_at	name
		Filter	Filter	Filter	Filter
1		100	2018-09-24 0...	2018-09-24 0...	NULL
2		340	2018-10-07 0...	2018-10-07 0...	NULL
3		720	2018-10-13 2...	2018-10-13 2...	NULL

- Here we can *see* that all of our Section entries in the database have *NULL* in their name field. Likewise, Enrollments are woefully unpopulated:

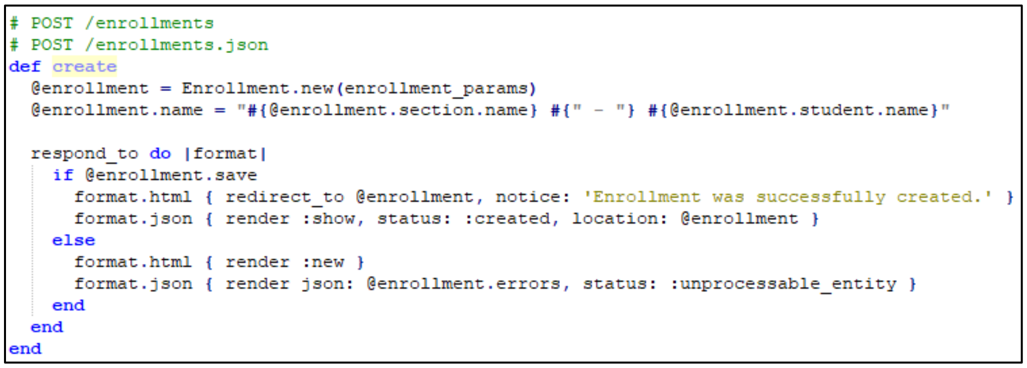


	tion_id	student_id	created_at	updated_at	name
		Filter	Filter	Filter	Filter
1		2	2018-10-14 0...	2018-10-14 0...	NULL
2		3	2018-10-14 0...	2018-10-14 0...	NULL

- Again, we'll edit the controller file '`app\controllers\enrollments_controller.rb`' to add the following line in 'def create'. This will properly fill the 'name' field of the Enrollments database table:

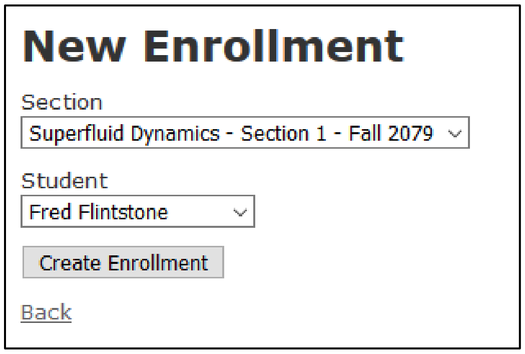
```
@enrollment.name = "#{@enrollment.section.name} #{" - "} #{@enrollment.student.name}"
```

- Updated enrollments controller:



```ruby
# POST /enrollments
# POST /enrollments.json
def create
  @enrollment = Enrollment.new(enrollment_params)
  @enrollment.name = "#{@enrollment.section.name} #{" - "} #{@enrollment.student.name}"

  respond_to do |format|
    if @enrollment.save
      format.html { redirect_to @enrollment, notice: 'Enrollment was successfully created.' }
      format.json { render :show, status: :created, location: @enrollment }
    else
      format.html { render :new }
      format.json { render json: @enrollment.errors, status: :unprocessable_entity }
    end
  end
end
```

- If we create a new enrollment…



New Enrollment

Section
Superfluid Dynamics - Section 1 - Fall 2079

Student
Fred Flintstone

Create Enrollment

Back

- …we can see that we updated the ‘name’ field of the database table Enrollments properly.

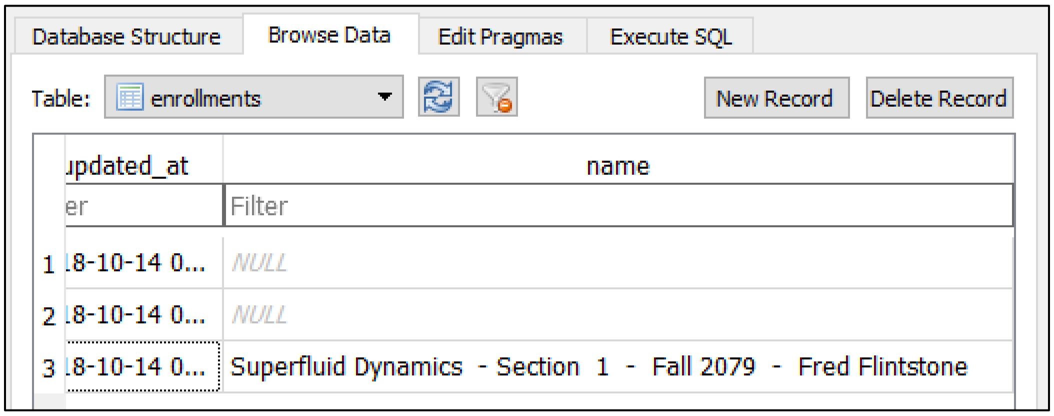
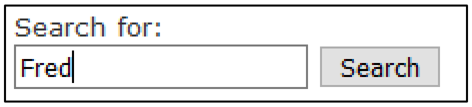
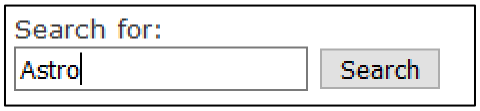


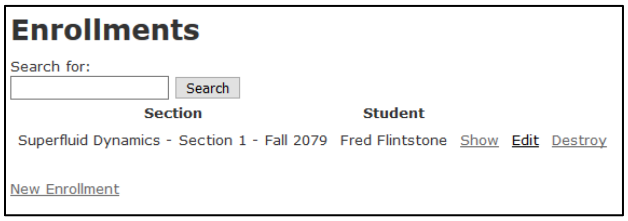
Table: enrollments

	ıpdated_at	name
	er	Filter
1	.8-10-14 0...	NULL
2	.8-10-14 0...	NULL
3	.8-10-14 0...	Superfluid Dynamics - Section 1 - Fall 2079 - Fred Flintstone

- We can now search by Student name:
- Or by Section name:


Search for:
Fred
Search

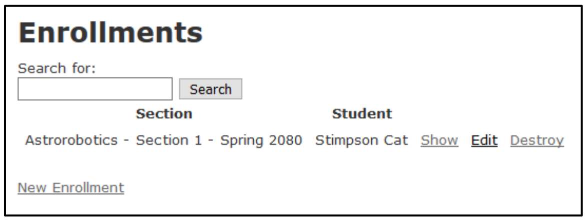

Search for:
Astro
Search


Enrollments

Search for:
Search

Section	Student			
Superfluid Dynamics - Section 1 - Fall 2079	Fred Flintstone	Show	Edit	Destroy

New Enrollment


Enrollments

Search for:
Search

Section	Student			
Astrorobotics - Section 1 - Spring 2080	Stimpson Cat	Show	Edit	Destroy

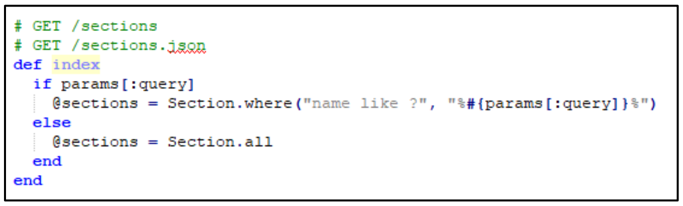
New Enrollment

<u>Add a search bar to Sections index</u>

- We should have this process down to a science by now. Edit the Section index view file '`\app\views\sections\index.html.erb`' to create a new text input field so we can pass our ':query' parameter to the controller for indexing:

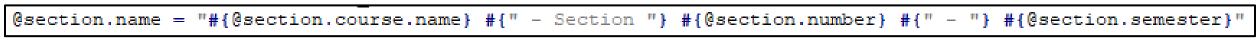
```
<%= form_tag(sections_path, method: :get) do %>
  <%= label_tag(:query, "Search for:") %>
  <%= text_field_tag(:query) %>
  <%= submit_tag("Search") %>
<% end %>
```

- Likewise add the search function to the index in the Sections controller '`app\controllers\sections_controller.rb`':

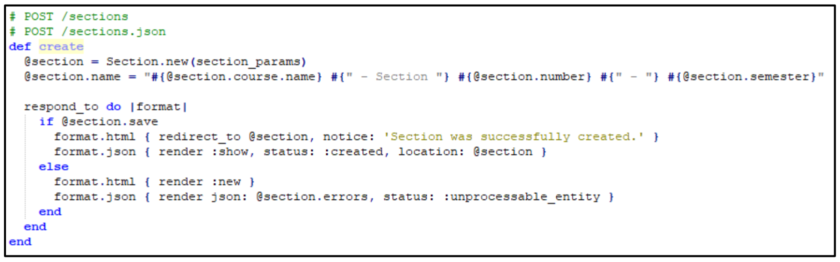


```
# GET /sections
# GET /sections.json
def index
  if params[:query]
    @sections = Section.where("name like ?", "%#{params[:query]}%")
  else
    @sections = Section.all
  end
end
```

- Once again we create a new string to populate the 'name' field of a new Section when it gets added to the database:

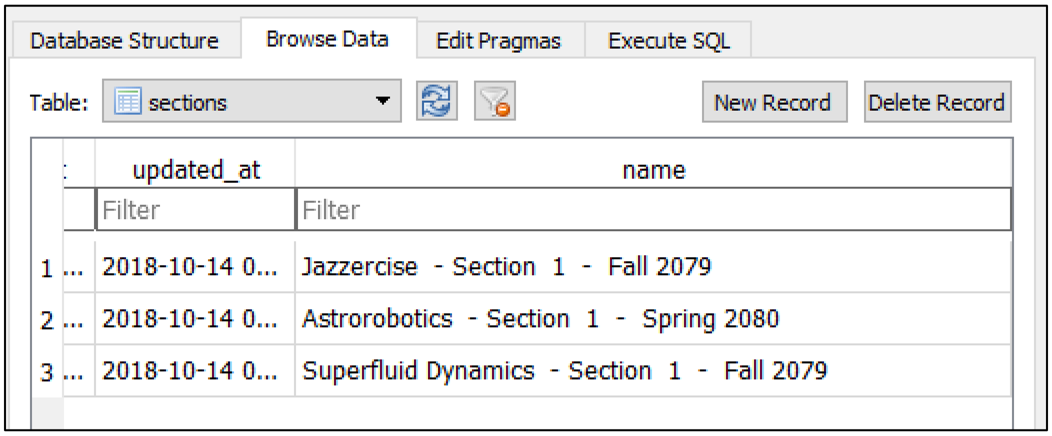


```
@section.name = "#{@section.course.name} #{" - Section "} #{@section.number} #{" - "} #{@section.semester}"
```



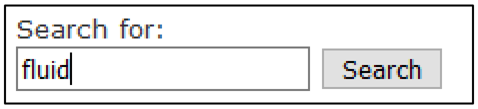
```
# POST /sections
# POST /sections.json
def create
  @section = Section.new(section_params)
  @section.name = "#{@section.course.name} #{" - Section "} #{@section.number} #{" - "} #{@section.semester}"

  respond_to do |format|
    if @section.save
      format.html { redirect_to @section, notice: 'Section was successfully created.' }
      format.json { render :show, status: :created, location: @section }
    else
      format.html { render :new }
      format.json { render json: @section.errors, status: :unprocessable_entity }
    end
  end
end
```

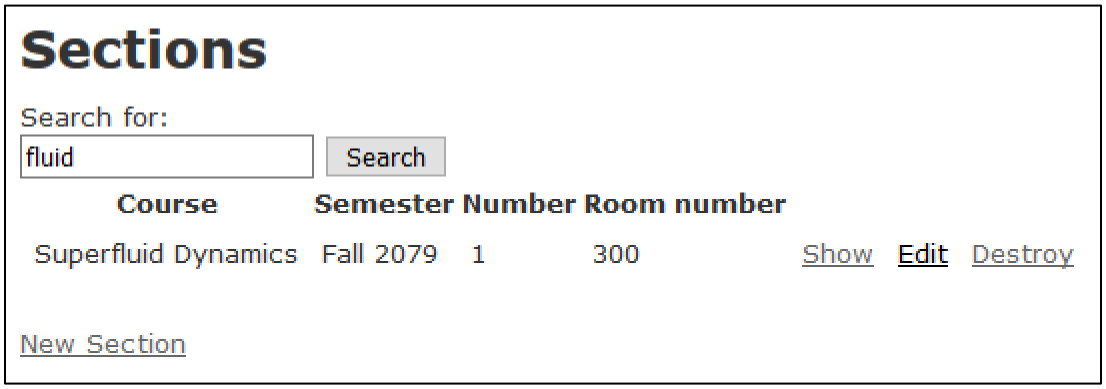
- When we create new Sections, we can see they properly have a ‘name’ field in the database:



		updated_at	name
		Filter	Filter
1	...	2018-10-14 0...	Jazzercise - Section 1 - Fall 2079
2	...	2018-10-14 0...	Astrorobotics - Section 1 - Spring 2080
3	...	2018-10-14 0...	Superfluid Dynamics - Section 1 - Fall 2079

- Likewise, our Search function now works for Sections:





Sections

Course	Semester	Number	Room number			
Superfluid Dynamics	Fall 2079	1	300	Show	Edit	Destroy

New Section

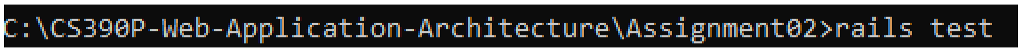
We’ve successfully created a simple Search function for our web application. Through no small amount of time spent troubleshooting, we’ve learned a great deal about how database entries get created and updated. We’ve learned more about not just how to create references in a relational database, but how to *use them*. It should be noted that in the previous example, we are only able to search Sections via course name. The following line will allow for searching by Semester. More robust functionality such as searching by room number will be left as an advanced exercise for the reader.

```
@sections = Section.where("name like ?", "%#{params[:query]}%").where("semester like ?", "%#{params[:query]}%")
```

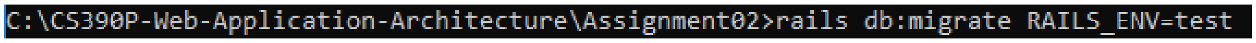
Step 5: Unit Testing

One important feature of the agile design process is Test-Driven Development (TDD). TDD emphasizes a modular approach to design, in which individual tests are written to specific application functions, to which a singular function can be easily refactored or rewritten as needed. Rails comes included with the built-in Minitest framework for unit and system testing. Rails creates and pre-configures some very basic unit test files when a new project is created. Let's begin looking at the ways Rails can test our web applications by simply running the included Minitest tests (Ruby on Rails Guides, 2018).

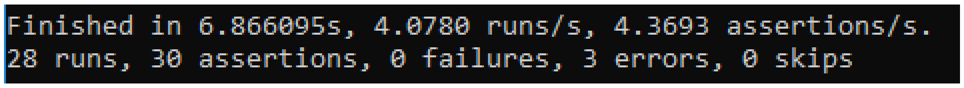
- Enter '`rails test`' into the terminal to run all of the pre-made tests generated by Rails:



```
C:\CS390P-Web-Application-Architecture\Assignment02>rails test
```

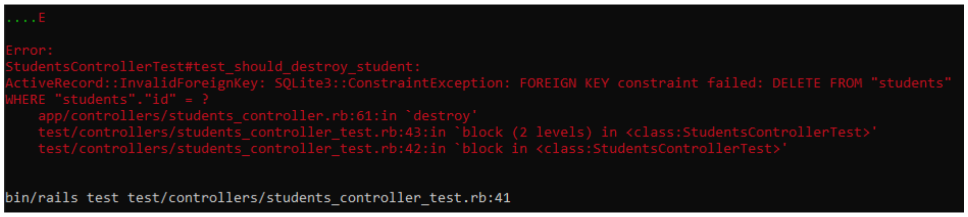


```
C:\CS390P-Web-Application-Architecture\Assignment02>rails db:migrate RAILS_ENV=test
```

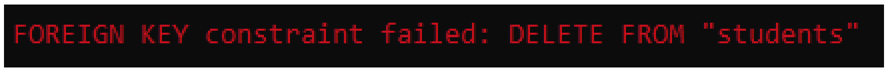


```
Finished in 6.866095s, 4.0780 runs/s, 4.3693 assertions/s.
28 runs, 30 assertions, 0 failures, 3 errors, 0 skips
```

- We see a few failed tests already:

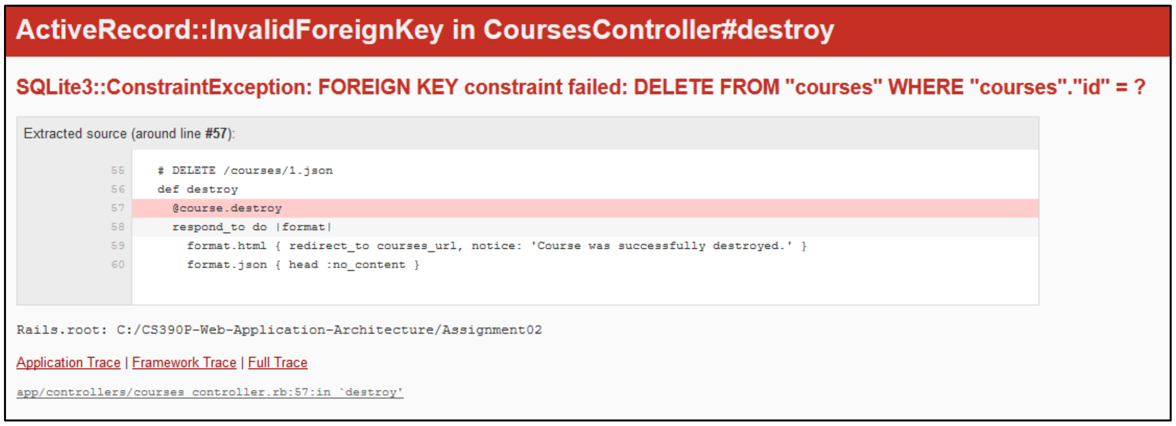


```
....E

Error:
StudentsControllerTest#test_should_destroy_student:
ActiveRecord::InvalidForeignKey: SQLite3::ConstraintException: FOREIGN KEY constraint failed: DELETE FROM "students"
WHERE "students"."id" = ?
    app/controllers/students_controller.rb:61:in `destroy'
    test/controllers/students_controller_test.rb:43:in `block (2 levels) in <class:StudentsControllerTest>'
    test/controllers/students_controller_test.rb:42:in `block in <class:StudentsControllerTest>'


bin/rails test test/controllers/students_controller_test.rb:41
```



```
FOREIGN KEY constraint failed: DELETE FROM "students"
```

- You may have already noticed that 'Destroy' results in an error:



ActiveRecord::InvalidForeignKey in CoursesController#destroy

SQLite3::ConstraintException: FOREIGN KEY constraint failed: DELETE FROM "courses" WHERE "courses"."id" = ?

Extracted source (around line #57):

```
55    # DELETE /courses/1.json
56    def destroy
57      @course.destroy
58      respond_to do |format|
59        format.html { redirect_to courses_url, notice: 'Course was successfully destroyed.' }
60        format.json { head :no_content }
```

```
Rails.root: C:/CS390P-Web-Application-Architecture/Assignment02
```

Application Trace | Framework Trace | Full Trace

```
app/controllers/courses_controller.rb:57:in `destroy'
```

- We get an error here because we're trying to delete a Course without allowing the database to delete the related Sections as well. We can fix this by adding the phrase 'dependent: :destroy' to '`\app\models\course.rb`'. We need to do this for any model with a *has_many* relationship.

```ruby
class Course < ApplicationRecord
  has_many :sections, dependent: :destroy
end
```

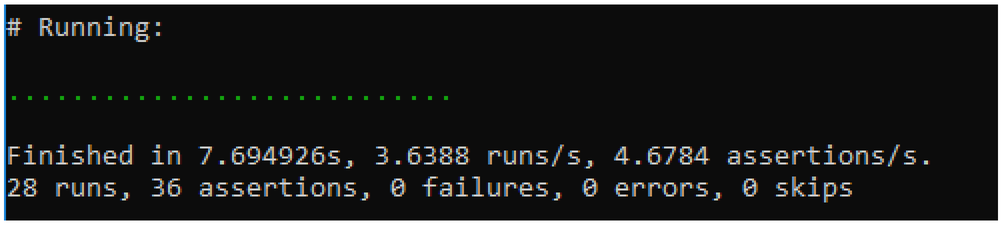
- Likewise, we should add 'dependent: :destroy' to Student, Section, and Enrollment models:

```ruby
class Student < ApplicationRecord
  has_many :enrollments, dependent: :destroy
  has_many :sections, through: :enrollments, dependent: :destroy
end
```

```ruby
class Section < ApplicationRecord
  belongs_to :course
  has_many :enrollments, dependent: :destroy
  has_many :students, through: enrollments, dependent: :destroy
  def name
    "#{course.name} #{" - Section "} #{number} #{" - "} #{semester}"
  end
end
```

```ruby
class Course < ApplicationRecord
  has_many :sections, dependent: :destroy
end
```

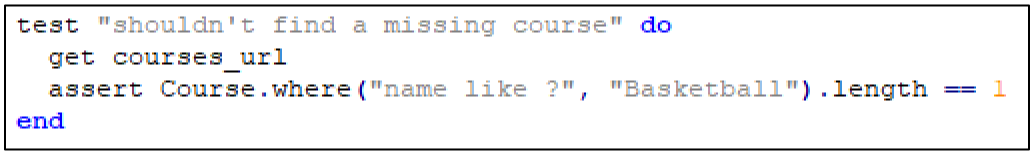
- Running 'rails test' once more shows our basic unit tests pass.



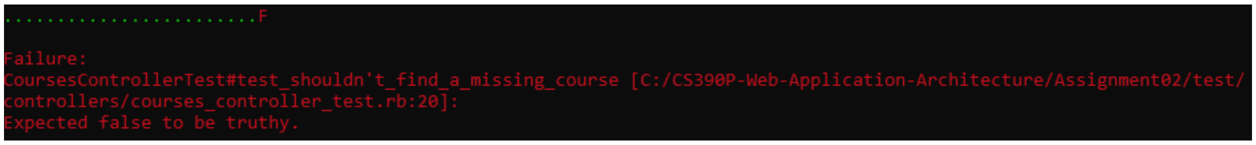
```
# Running:

............................

Finished in 7.694926s, 3.6388 runs/s, 4.6784 assertions/s.
28 runs, 36 assertions, 0 failures, 0 errors, 0 skips
```

We can see that running tests using Rails built-in Minitest framework is a snap. We encourage the reader to explore the '`\test`' folder in their Rails project, there you'll find pre-made tests for controllers, models and other files to get you started. Let's write a simple test to make sure we don't return anything from a bad database query.

- Open the '\test\controllers\courses_controller_test.rb' controller test file and add the following test. First, we write a failing test, and assert that we get '1' of something that shouldn't be in the database at all.

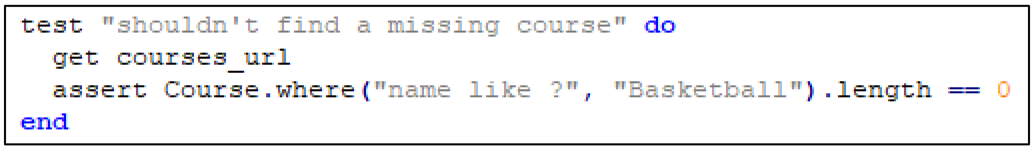


```
test "shouldn't find a missing course" do
  get courses_url
  assert Course.where("name like ?", "Basketball").length == 1
end
```

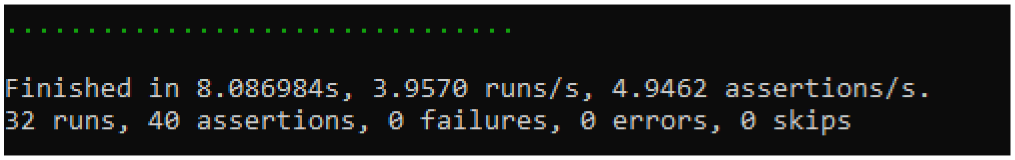


```
........................F

Failure:
CoursesControllerTest#test_shouldn't_find_a_missing_course [C:/CS390P-Web-Application-Architecture/Assignment02/test/
controllers/courses_controller_test.rb:20]:
Expected false to be truthy.
```

- This is standard Test-Driven Development (TDD) practice; (1) Write a failing test, (2) Write just enough code to fix that test, (3) Repeat. Now let's expect that we don't return anything from a course that's not part of the database:



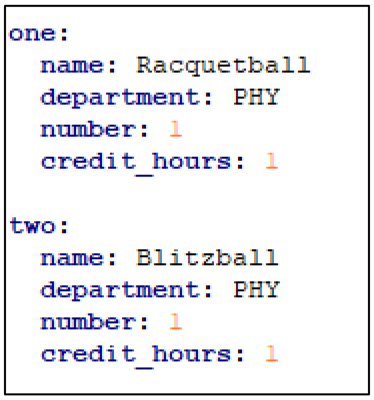
```
test "shouldn't find a missing course" do
  get courses_url
  assert Course.where("name like ?", "Basketball").length == 0
end
```



```
................................

Finished in 8.086984s, 3.9570 runs/s, 4.9462 assertions/s.
32 runs, 40 assertions, 0 failures, 0 errors, 0 skips
```

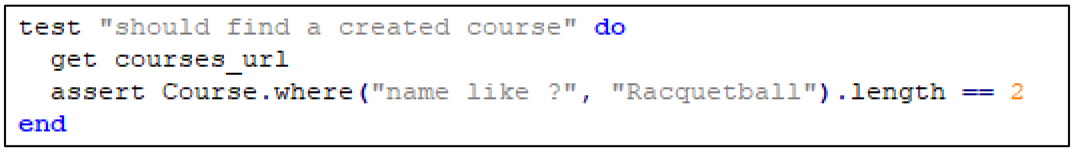
Before we begin testing that can retrieve an already created Course from the database, first let's explore Minitest and its use of *test fixtures*. Test fixtures are simple files that create and populate a small database for the purposes of testing.

- Edit the '`\test\fixtures\courses.yml`' file to get a feel for test fixtures:



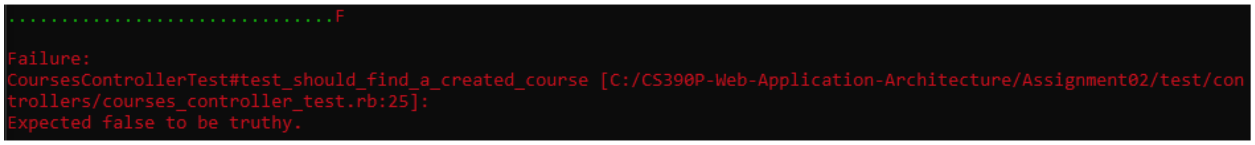
```
one:
  name: Racquetball
  department: PHY
  number: 1
  credit_hours: 1

two:
  name: Blitzball
  department: PHY
  number: 1
  credit_hours: 1
```

- Now let's write another 'obviously bad' test, this time to test the retrieval of a specific Course from the database. Once again edit '`\test\controllers\courses_controller_test.rb`':



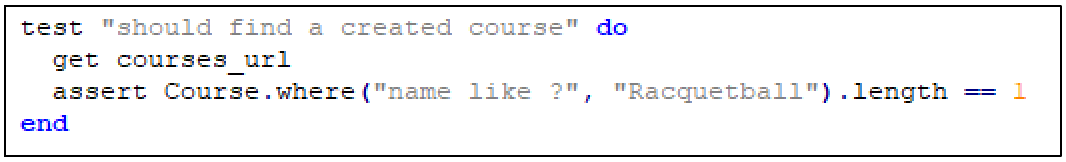
```
test "should find a created course" do
  get courses_url
  assert Course.where("name like ?", "Racquetball").length == 2
end
```

- We can see that our test failed, as expected.

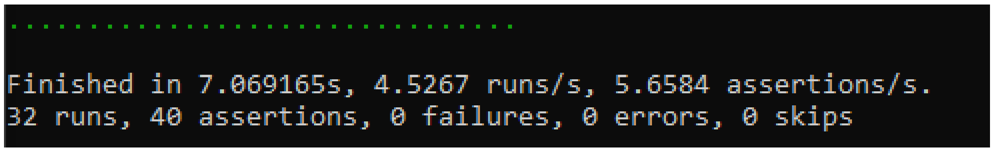


```
.............................F

Failure:
CoursesControllerTest#test_should_find_a_created_course [C:/CS390P-Web-Application-Architecture/Assignment02/test/con
trollers/courses_controller_test.rb:25]:
Expected false to be truthy.
```

- A simple fix should get our test to pass:



```
test "should find a created course" do
  get courses_url
  assert Course.where("name like ?", "Racquetball").length == 1
end
```



```
................................

Finished in 7.069165s, 4.5267 runs/s, 5.6584 assertions/s.
32 runs, 40 assertions, 0 failures, 0 errors, 0 skips
```

We've wrote some simple tests to demonstrate the Minitest framework with Rails for unit testing. We can continue to write similar simple tests for our remaining classes. Our goal should be to gain a complete understanding of the test framework to write robust, thorough tests for our web applications. For now, let us finish our study here and look at one more kind of testing, and turn our attention to , and turn our attention to *system* or *integration* testing.

Step 6: System Testing

Whereas unit tests can test out individual functions of our applications, we may at some point want more *holistic* tests of our website. For this we will look to *system* or *integration* testing. We can use tools such as the Selenium browser automation tool to fully automate the navigation of our page, including clicking on buttons and links, and filling in text fields. This is much closer to a 'live' test to see how all of our applications functions are working in an actual web browser with Google Chrome.

- First, let's edit '`\test\test_helper.rb`'. This file points to our various test fixtures, in addition to specifying test frameworks we'll be using. Make sure you have the following 'require' lines below:

```
# Add more helper methods to be used by all tests here...
require 'capybara/rails'
require 'capybara/minitest'
```

- Now let's run our system tests:

```
C:\CS390P-Web-Application-Architecture\Assignment02>rails test:system
```

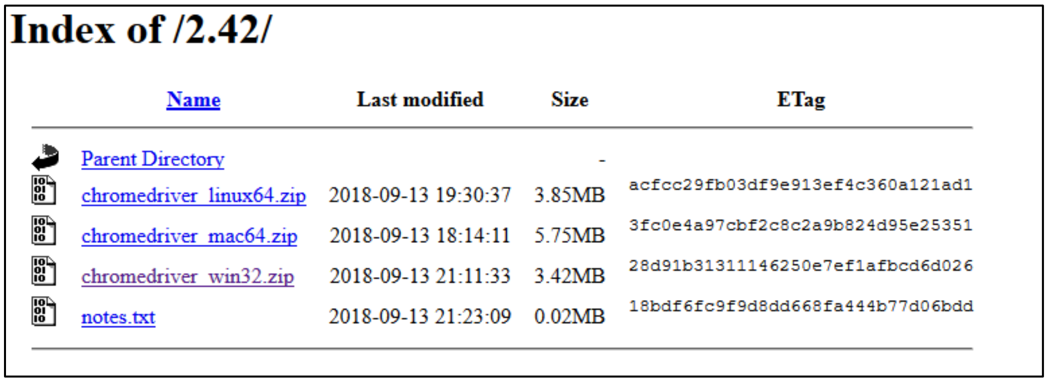
- Simply running '`rails test:system`' results in the following error message:

```
Unable to find chromedriver.
```

```
Please download the server from http://chromedriver.storage.googleapis.com/index.html
```

```
and place it somewhere on your PATH.
```

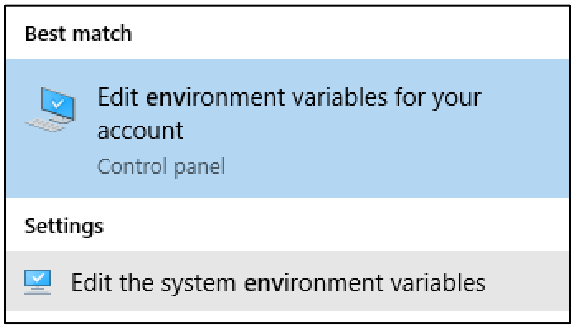
- It appears we're missing a key component, 'Chrome Driver'. This is the driver that allows for Selenium to do automation testing and manipulate the Chrome browser. Download Chrome Driver version 2.42 here: https://chromedriver.storage.googleapis.com/index.html?path=2.42/



Index of /2.42/

Name	Last modified	Size	ETag
Parent Directory		-	
chromedriver_linux64.zip	2018-09-13 19:30:37	3.85MB	acfcc29fb03df9e913ef4c360a121ad1
chromedriver_mac64.zip	2018-09-13 18:14:11	5.75MB	3fc0e4a97cbf2c8c2a9b824d95e25351
chromedriver_win32.zip	2018-09-13 21:11:33	3.42MB	28d91b31311146250e7ef1afbcd6d026
notes.txt	2018-09-13 21:23:09	0.02MB	18bdf6fc9f9d8dd668fa444b77d06bdd

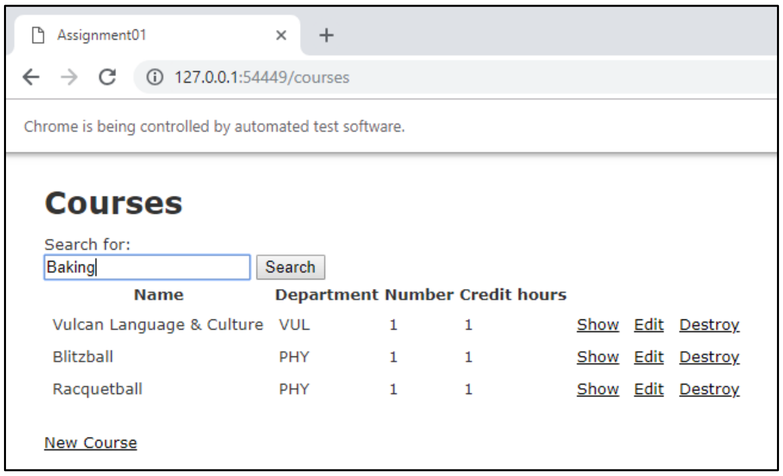
- Once you've downloaded the appropriate file for your operating system, extract the compressed folder and then run the '`chromedriver.exe`' application to install Chrome Driver for Selenium integration testing with Rails.
- Next, we need to edit the *system environment variables* in Windows 10 to specify the path containing Chrome Driver. In Windows 10 system search, enter 'environment', and then click on 'Edit the system environment variables. You should see the 'System Properties' window pop up. Click on 'Environment Variables'. Under 'System variables' click 'Path', then 'Edit'. Click 'New', then add the file path you saved chrome driver to.


Best match
Edit environment variables for your account
Control panel
Settings
Edit the system environment variables

- Let's write a simple system test to make sure Chrome Driver is working properly. Edit the '`\test\system\courses_test.rb`' system test file to include the following. This test will navigate to our Courses page, search Courses for a course called "Baking", then click on our Search button. We then use the *refute* keyword to show that we should not return any database entries when searching for "Baking", as it is not part of the text fixture we created.

```
test "course not in search" do
  visit courses_url
  fill_in "query", with: "Baking"
  click_on "Search"
  refute_selector "td"
end
```

- Now let's run '`rails test:system`' once again to test our new system test. You should see the Chrome browser pop up *very* briefly, hopefully long enough to see something like the following:

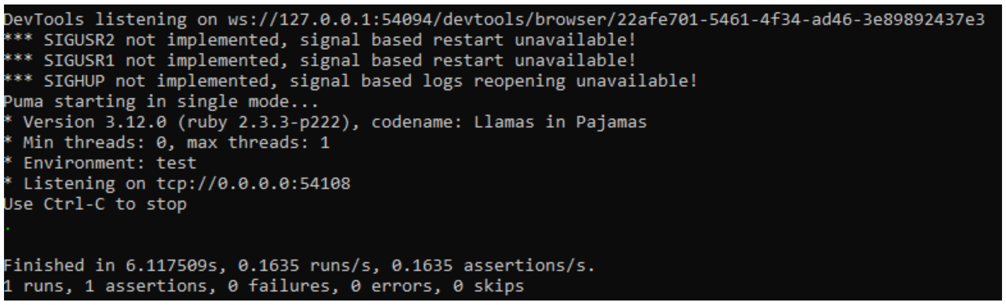


Courses

Search for:
Baking

Name	Department	Number	Credit hours			
Vulcan Language & Culture	VUL	1	1	Show	Edit	Destroy
Blitzball	PHY	1	1	Show	Edit	Destroy
Racquetball	PHY	1	1	Show	Edit	Destroy

- We can see that our simple system test passes:



```
DevTools listening on ws://127.0.0.1:54094/devtools/browser/22afe701-5461-4f34-ad46-3e89892437e3
*** SIGUSR2 not implemented, signal based restart unavailable!
*** SIGUSR1 not implemented, signal based restart unavailable!
*** SIGHUP not implemented, signal based logs reopening unavailable!
Puma starting in single mode...
* Version 3.12.0 (ruby 2.3.3-p222), codename: Llamas in Pajamas
* Min threads: 0, max threads: 1
* Environment: test
* Listening on tcp://0.0.0.0:54108
Use Ctrl-C to stop
.

Finished in 6.117509s, 0.1635 runs/s, 0.1635 assertions/s.
1 runs, 1 assertions, 0 failures, 0 errors, 0 skips
```

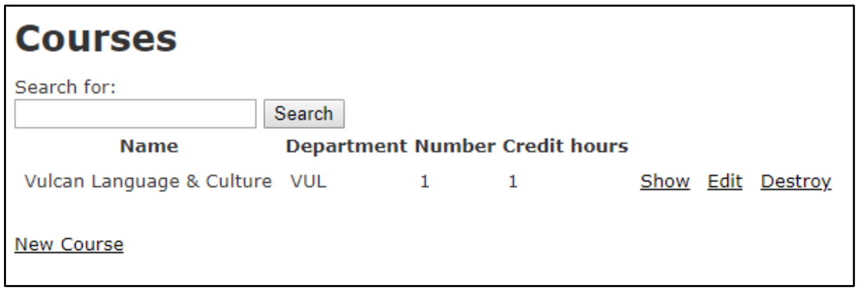
- Now let's write another test, this time to visit the Courses index page, and confirm that certain text is contained within specific HTML tags:

```
test "visiting the index" do
  visit courses_url
  assert_selector "h1", text: "Courses"
  assert_selector "th", text: "Name"
  assert_selector "td", text: "Racquetball"
end
```

- Finally, let's test our Search function. First, we'll visit the Courses index page, then search for "Vul", to hopefully return "Vulcan Language & Culture", we named in the test fixture:

```
test "course in search" do
  visit courses_url
  fill_in "query", with: "Vul"
  click_on "Search"
  assert_selector "td", text: "Vulcan Language & Culture"
end
```

- If you'd like to slow down the automated browsing process, adding a simple line like '`sleep 5`' will cause Selenium to *sleep* for 5 seconds after a specific test instruction. We can see that our test does indeed Search for "Vul" and return "Vulcan Language & Culture" from the database:



Courses

Search for:

Name	Department	Number	Credit hours			
Vulcan Language & Culture	VUL	1	1	Show	Edit	Destroy

New Course

Selenium is a powerful tool in our toolkit for testing. We should strive to test every aspect of our applications in a variety of different environments. Whenever we add new features and functions to our websites, we should increase our understanding of those functions by writing detailed tests.

Conclusion

Such a long way we've come in two short assignments. We learned everything about creating a web application here; we began with setting up our *many-to-many* relationship between database tables, explored input validation and Search functionality, and finished our work with two kinds of testing. While it is no small task to simply create a working web application, we emphasize that it is just as important to create one that is both well tested and secure. We hope you'll find creating web applications with Rails as exciting as we did; it's satisfying to see such direct feedback whenever we run our Rails server. While there are a great deal of moving parts in any web application, we hope you'll appreciate that despite this complexity Rails does indeed make things as easy as possible for fledgling web designers. It is a humbling experience finding out just how much there is still to learn.

Bibliography

Stack Overflow (2018). *Generate migration – create join table* [forum post]. Retrieved from https://stackoverflow.com/questions/17765249/generate-migration-create-join-table#

Ruby on Rails Guides (2018). *Active Record associations* [Website]. Retrieved from https://guides.rubyonrails.org/association_basics.html

Ruby on Rails Guides (2018). *Active Record validations* [Website]. Retrieved from https://guides.rubyonrails.org/active_record_validations.html

Wintermeyer, S. (2012). *ActiveRecord validation* [website]. Retrieved from http://www.xyzpub.com/en/ruby-on-rails/3.2/activerecord_validation.html

Castello, J. (2018). *Mastering Ruby regular expressions* [website]. Retrieved from http://www.rubyguides.com/2015/06/ruby-regex/

SQLite Browser (2018). *DB Browser for SQLite* [website]. Retrieved from https://sqlitebrowser.org/

Ruby on Rails Guides (2018). *Testing Rails applications* [Website]. Retrieved from https://guides.rubyonrails.org/testing.html

Instructor Feedback

What was too difficult, too easy?

Troubleshooting, and the wide variety of tasks for this Assignment, for myself and other students meant this Assignment “ballooned” into more work than was initially thought. New frameworks, new errors, time spent confused all contributed to this being a difficult assignment overall. I’ve had some difficulty finding an appropriate level of detail for a given task.

What would have made the learning experience better?

Difficulty aside, there wasn’t much that could have made the learning experience better. Perhaps splitting this assignment into two more appropriately sized assignments would have helped me in the long run.

What did you learn?

I learned much, much more than I expected to for this assignment. I learned the basics of adding fields to a Rails website, and how to create relationships in a relational database. I learned everything I know about input validation, and had a great time learning more about regular expressions for validation as well. I learned more about Rails testing frameworks, though TDD was not a new concept. It’s a bit hard summarize everything as the learning was so deep and broad in scope.

How did you learn it?

I seem to have got into some sort of routine in working with Rails. My basic process is as follows: (1) Run ‘rails s’, (2) Experiment with application functionality, (3) Write some code, (4) Restart the Rails server and see what changed. Though not too different from my usual trial-and-error process, there was simply a staggering amount of new content to learn this time. Having extra time to work on it increased my understanding drastically.