[Overview](#h.ocl3rjzdnt04)

[Lesson 4.3: Seeding and Populating Databases](#h.sw1f5g8kvabx)

[Gem of the Day](#h.zn00kjo0nyc)

[Seeding Our Database](#h.hdcdppm2sjsl)

[Populating Development Database](#h.kmdbog3qniqs)

# Overview

1. Rails
   1. Rails console, DB console
   2. More gems
2. Data
   1. Generating models
   2. Data validations
   3. What are the elements of an SQL database (tables, rows, columns/fields)
   4. Basic SQL commands and CRUD

# Lesson 4.3: Seeding and Populating Databases

Roll call.

Answer any homework questions.

## Gem of the Day

Show and discuss what they are:

* Faker <https://github.com/stympy/faker>
* Populator <https://github.com/ryanb/populator>

## Seeding Our Database

Rails has a 'seeds' feature that should be used for seeding a database with **initial** data. If you know you will always need the same data, for example a list of states, you can seed them so that whenever a collaborator starts working on your app or when you push to production, you will have that data pre-populated.

It's a really simple feature: just fill up db/seeds.rb with some Ruby code, and run rake db:seed:

|  |
| --- |
| 5.times do |i|  Product.create(name: "Product ##{i}", description: "A product.")  end |

Let’s do this for our Music DB app so that genres will automatically be populated any time we do a db setup. This is convenient if we have multiple collaborators but also for setting up production.

Open up db/seeds.rb. Who remembers how we created new content using Rails console and ActiveRecord? Add all your genres now:

Genre.create(name: "pop")

Genre.create(name: "country")

Genre.create(name: "hip hop")

Genre.create(name: "rock")

Genre.create(name: "jazz")

Genre.create(name: "blues")

Save your file.

Before we can seed our database, we need to drop it and then set it up again. You can do this in two separate commands:

rake db:drop

rake db:setup

Or, you can do it in one command with:

rake db:reset

To see all your rake command options, type rake -T. (do this)

Go ahead and drop your database and set it up again.

## Populating Development Database

Use faker and populator to add data to development database for Doggie Daycare.

Faker::Name - scope resolution operator. Name lives inside of Faker. It’s a courtesy code so that you don’t accidentally collide with the Faker classes or subclasses. If you know you won’t have any collisions, then you can write “include faker” (the module) and then you can just call without the Faker:: prefix

In gemfile, add faker and populator to development only. Bundle.

Don’t do gem install faker. That installs at a system level. This is bad because different projects might be written on different versions and could break. Also, if you push to production to Heroku, for example, the production environment would not know to use that gem.

In lib/tasks, create a new file called populate.rake, then inside create this empty rake task:

namespace :db do

desc "Clear the DB and fill with excellent sample data"

task :populate => :environment do

end

end

Save and run rake -T - notice that your task is already created!

Example? <https://github.com/siakaramalegos/tts_resources/blob/master/lib/tasks/populate.rake>

Homework: seed another database, design model for Group project, install ImageMagick

# Lesson 4.4: Rails and Database Review Using Group Project

Roll call.

Answer any homework questions.

## Gem of the Day

Paperclip <https://github.com/thoughtbot/paperclip>

## Rails Review: Doggy Daycare

### Planning

First, let’s draw out what we want to generate for our app. Basically, we want an app that helps a business manage it’s doggy daycare while also allowing customers to order pet products online. We might add more features as we go along.

Trello is a great tool for managing app development. We can dump in all of our customer’s feature requests into a board called “icebox”. When we start working on a feature, we can move it to “wip”. When we are done, we move it to “complete”. We could even create a staging board for the next feature. The idea is that sometimes a customer will have feature ideas that sound great, but priorities change over time.

Today, we are going to focus on generating a cleaner model as well as adding dog photos and updating views. We will save building the merchant and user models for when we work on those features. Let’s decide what we want to build today - (draw these tables and fields on the board)

|  |  |  |
| --- | --- | --- |
| Dogs | Owners | Breeds |
| dog name  last name of owner (can pull in through owner\_id)  owner (owner\_id)  breed  date of birth  vet name  vet phone  currently at daycare? (boolean)  photo (avatar, later today) | first name  last name  primary phone  secondary phone  street address  city  state  zip  emergency contact name  emergency contact number | breed |

What steps do we need to take in order to re-create our app (get them to provide this list)?

rails new

cd into it

git/github

gems, bundle

scaffolds and model - generate and relationships

seed Breeds

generate pages

set root url

starter generator theme and navbar

fix forms and views to show actual data, not ID’s

### Implementing What We Already Know

Let’s do a Rails and database review by creating a clean copy of doggy daycare from scratch.

1. In your TTS folder (not in rails\_practice), run rails new doggy\_daycare (or whatever animal type you like). Make sure it is not inside another git repo as we will want to push this to Heroku later on.
2. cd into doggy\_daycare
3. Initialize a git repo, make your initial commit, create a repo on github, and push.
4. Add our gems, bundle, and commit:

gem 'thin'

gem 'starter\_generators'

gem 'awesome\_print'

gem 'annotate'

gem 'stamp'

group :development do

gem 'quiet\_assets'

gem 'faker'

gem 'populator'

end

1. Generate our scaffolds and model, double-check the migrations, migrate, and commit:  
   rails g scaffold Owner first\_name:string last\_name:string primary\_phone:string secondary\_phone:string address:string city:string state:string zip:string emerg\_name:string emerg\_phone:string  
     
   rails g model Breed breed:string  
     
   rails g scaffold Dog name:string owner:references breed:references dob:date vet:string vet\_phone:string in\_daycare:boolean
2. Finish setting the remaining relationships in your app/model files (has\_many).
3. Seed breeds with valid breed names. Then run rake db:seed. Commit.
4. Generate two pages - index and about. rails g controller Pages index about
5. Set the root to the pages index.
6. Now let’s dump in our starter generator theme. Run rails g starter:style flatly, or use another theme you like from Bootswatch.
7. Try to run rails s. It will error out because we don’t have routes for Breeds, but that’s ok. Let’s just manually take that code out of the navbar (in application.html.erb). While we are there, let’s make the About link on the navbar active - replace the anchor html tag with an erb link\_to method to the about page.  
   <%= link\_to "About", pages\_about\_url %>
8. Run rails server to check your pages, then commit your changes.
9. Now, let’s fix our forms and views to show actual owners and breeds, not ID’s. First, add at least one owner. Have you noticed how we have first and last names separate for owners? Well, that makes collection\_selects have a tiny obstacle. We can make it easier by creating a first\_last method in the Owner model:  
    def last\_first  
    "#{last\_name}, #{first\_name}"  
    end  
     
   While we are there, let’s add the table class on the index tables.
   1. form  
      f.collection\_select :owner\_id, Owner.all, :id, :last\_first, {prompt: "Select an owner"}   
      f.collection\_select :breed\_id, Breed.all, :id, :breed, {prompt: "Select a breed"}
   2. index and show  
      dog.owner.last\_name, or @dog.owner.first\_name + ' ' + @dog.owner.last\_name  
      dog.breed.breed
10. Add a dog to make sure the form, show, and index pages work. Commit your changes.

We will add validation and make our pages look prettier in the homework.

## Git Branch

(feature branches - create a new branch for paperclip)

## Setting up Paperclip

Paperclip is an awesome gem that handles file uploads and manipulates images. First, you must have ImageMagick installed. See the Lesson 4.3 homework for more info.

Take a look at the Paperclip gem documentation. The readme file is actually quite detailed. It can be overwhelming, but when you take it step-by-step it’s not too bad.

1. First, we need to let Paperclip have access to ImageMagick. See [here](https://github.com/thoughtbot/paperclip#image-processor). First, run which convert. Grab that path and put it in config/environments/development.rb:  
    # Telling Paperclip gem where to find ImageMagick  
    Paperclip.options[:command\_path] = "/usr/local/bin/"
2. Add gem ‘paperclip’, bundle, commit.  
    gem "paperclip", "~> 4.2"
3. Now we can generate our Paperclip stuff. Paperclip conveniently comes with an automatic migration generator to add an avatar to a table. Read more [here](https://github.com/thoughtbot/paperclip#migrations). Run:  
    rails g paperclip dog avatar
4. Check the migration then run rake db:migrate. Add :avatar to the Dog controller whitelist.
5. Now, let’s add some validations to Dog for avatar for safety.  
    has\_attached\_file :avatar, :styles => { :medium => "300x300>", :thumb => "100x100>" }, :default\_url => "/images/:style/missing.png"  
    validates\_attachment\_content\_type :avatar, :content\_type => /\Aimage\/.\*\Z/
6. Now we can add the file upload to our dog form:  
    <div class="field">  
    <%= f.label :avatar %><br>  
    <%= f.file\_field :avatar %>  
    </div>
7. And to show:

<p>

<%= image\_tag @dog.avatar.url %>

<%= image\_tag @dog.avatar.url(:medium) %>

<%= image\_tag @dog.avatar.url(:thumb) %>

</p>

Run your rails server, upload a file for a dog, and test everything out. So cool! If everything worked, **commit your changes**. Work through any errors during office hours and over the weekend.