This repository | Search | Pull requests | Issues | Gist

appdevspring16 / **msm_associations**

Watch 3 | Star 0 | Fork 1

Code | Issues 0 | Pull requests 40 | Wiki | Pulse | Graphs

No description or website provided.

20 commits | **46** branches | **0** releases | **1** contributor

Branch: **master** | New pull request | Create new file | Upload files | Find file | Clone or download

raghubetina Update README.md — Latest commit `d3dca34` 5 days ago

.bundle	Update gems	8 days ago
app	Windows fixes	9 months ago
bin	rails new	9 months ago
config	Remove hirb	8 days ago
db	rails new	9 months ago
lib	Adds seed data	9 months ago
log	rails new	9 months ago
public	rails new	9 months ago
spec	Add test config	8 days ago
vendor/assets	rails new	9 months ago
.gitignore	Add test config	8 days ago
.rspec	Add test config	8 days ago
Gemfile	Update gems	8 days ago
Gemfile.lock	Update gems	8 days ago
README.md	Update README.md	5 days ago
Rakefile	rails new	9 months ago
circle.yml	Add test config	8 days ago
config.ru	rails new	9 months ago

README.md

Must See Movies Associations

In this project, we'll practice associating rows from different tables to one another.

Our goal will be to build something that works like this target. (Don't worry about styling -- focus on functionality only. Also, ignore the pagination links at the bottom of the characters index -- but think about how you would go about it if you had to.)

There is a Getting Started video on Canvas.

Setup

1. Clone and open the code.
2. Add the starter_generators gem.
3. `bundle install`
4. Generate the Director resource:

```
rails generate starter:resource director name:string dob:string bio:text image_url:string
```

5. `rake db:migrate`
6. Start the server and navigate to http://localhost:3000/directors; verify that the CRUD resource boilerplate was generated properly.
7. Quickly add a few rows to the directors table:

```
rake db:seed:directors
```

Two important notes about `rails console`

1. Sometimes when the output of a command is very long, `rails console` is going to paginate it for you. You will have a `:` prompt when this is true, and you can hit `Return` to scroll through line by line, or `Space` to scroll through page by page.

   To get back to the regular prompt so that you can enter your next command, just hit `q`.

2. If you are in `rails console` and then make a change to a model (for example, you add a validation or fix a syntax error), then, annoyingly, **you have to `exit` and then relaunch `rails console`** to pick up the new logic.

Solution

Once you've struggled for a while, it's okay to peek at one possible solution.

Associating Directors and Movies

Can X have many of Y? Can Y have many of X?

- Can a director have many movies? Yes
- Can a movie have many directors? No (in this app, anyway)

Therefore, we have a one (director) to many (movies) relationship.

Whenever you have a One-to-Many relationship, the way to keep track of it in the database is to **add a column to the Many to keep track of which One it belongs to**.

In this case, we want to add a column to the movies table to keep track of which director each movie belongs to.

We could add a column called "director_name", but that's not very reliable because there could be two directors with the same name, or a name could change.

Instead, we usually store the ID number of the row in the other table that we want to associate to, which is guaranteed by the database never to change or be duplicated. We can always use the ID number to look up the rest of the details.

So, let's now generate the Movie resource with all of the columns it needs:

```
rails generate starter:resource movie title:string year:integer duration:integer description:text image_url:string di
```

The `director_id` column is intended to hold the `id` of a row from over in the directors table. Such columns are called **foreign key columns**.

Execute the newly generated instructions to add the movies table:

```
rake db:migrate
```

Quickly add a few rows to the movies table:

```
rake db:seed:movies
```

Validations

Let's add the following validation rules to guard our tables against bogus rows sneaking in. Refer to the official RailsGuide on Validations.

```
Movie:
 - director_id: must be present
 - title: must be present; must be unique in combination with year
 - year: must be integer between 1870 and 2050
 - duration: must be integer between 0 and 2764800
 - description: no rules
 - image_url: no rules

Director:
 - name: must be present; must be unique in combination with dob
 - dob: no rules
 - bio: no rules
 - image_url: no rules
```

Querying practice

In `rails console`, answer the following questions. Refer to your CRUD with Ruby cheatsheet, and/or the offical RailsGuide on ActiveRecord querying.

For each question, see if you can craft a single Ruby expression that returns the final answer when entered into `rails console`.

1. In what year was the oldest movie in our list released?
2. In what year was the most recent movie in our list released?
3. What is the duration of the shortest movie in our list?
4. What is the longest movie in our list?
5. How many movies in our list have the word 'godfather' in their titles?
6. Who directed *Life Is Beautiful*?
7. How many movies in our list were directed by Francis Ford Coppola?
8. What is the most recent movie in our list directed by Francis Ford Coppola?

Improving the generated boilerplate views

1. Currently, on the movies index page and a movie's show page, the code that the generator wrote for you is showing users raw director ID numbers. This is bad. Replace the id number with the name of the director.
2. On the new and edit movie pages, let's give our users a dropdown box to select a director, rather than having to type in a valid ID number. Let's use the `select_tag` view helper method to make this slightly easier than writing the raw HTML `<select>` and `<option>` tags by hand:

```
<%= select_tag(:director_id, options_from_collection_for_select(Director.all, :id, :name, @movie.director_id), :class
```

1. Let's also add a link to the new director form in case the director doesn't exist yet.
2. On a director's show page, display a count of how many movies belong to that director.
3. On a director's show page, display a list of the movies that belong to that director.
4. At the bottom of the list of movies, write a form to add a new movie directly to that director (without having to go to http://localhost:3000/movies/new). You can start by copying over the boilerplate new movie form, and then modify it to pre-populate the `director_id` input with the correct value. Finally, switch the `type` of the input to "hidden".

The above are all extremely common steps that you will want to go through for almost every One-to-Many that you ever build.

Associating Movies and Actors

Let's now add Actors to our application. Our end goal is to show a cast on each movie's show page, and a filmography on each actor's show page.

```
rails generate starter:resource actor name:string dob:string bio:text image_url:string
```

`rake db:migrate` and navigate to http://localhost:3000/actors and verify that the CRUD resource boilerplate was generated properly.

Then, quickly add a few rows:

```
rake db:seed:actors
```

Can X have many of Y? Can Y have many of X?

Ask yourself the standard two questions:

- Can a movie be associated to many actors? Yes
- Can an actor be associated to many movies? Yes

So, we know we have a Many-to-Many on our hands.

If it had been a One-to-Many, we would simply have added a `movie_id` column to actors, or a `actor_id` column to movies. But this won't work because that limits you to connecting to only one.

The trick we'll use instead is to create a whole new table to keep track of the individual connection between each movie/actor pair.

Each row in the new table, or **join table**, will have both an `actor_id` and a `movie_id` , and will represent one actor appearing in one movie (e.g., Christian Bale in The Dark Knight).

If at all possible, try to think of a good, descriptive, real-world name for the join table. What is the connection between Christian Bale and The Dark Knight called in the real world?

How about "Role"? Or "Gig"? Or "Casting"? Or "Character"?

Let's go with "Character", and while we are at it, let's add a column to save the name of the character too (e.g., "Bruce Wayne").

Add the Character CRUD resource to our application:

```
rails generate starter:resource character movie_id:integer actor_id:integer name:string
```

`rake db:migrate` and navigate to http://localhost:3000/characters and verify that the CRUD resource boilerplate was generated properly. Then, add a few rows:

```
rake db:seed:characters
```

(This might take a minute.)

Validations

Whenever you add a model, you should immediately try to put in your validations to prevent bogus rows from sneaking in to your table. Let's go with the following in our new models:

```
Character:
 - movie_id: must be present
 - actor_id: must be present
 - name: no rules

Actor:
 - name: must be present; must be unique in combination with dob
 - dob: no rules
 - bio: no rules
 - image_url: no rules
```

Every Many-to-Many is just two One-to-Manies

We now have two foreign keys in the characters table. That means, essentially, **we've broken the many-to-many between Movies and Actors down into two one-to-manies**. A character belongs to a movie, a movie has many characters. A character belongs to an actor, an actor has many characters.

So, we should first go through the steps we went through above when we were setting up the one-to-many between directors and movies:

1. Currently, on the characters index page and a character's show page, the code that the generator wrote for you is showing users raw movie ID numbers. This is bad. Replace the id number with the title of the movie.
2. On the new and edit character pages, let's give our users a dropdown box to select a movie, rather than having to type in a valid ID number. Let's use the `select_tag` view helper method to make this slightly easier than writing the raw HTML `<select>` and `<option>` tags by hand:

```
<%= select_tag(:movie_id, options_from_collection_for_select(Movie.all, :id, :title, @character.movie_id), :class =>
```

1. Let's also add a link to the new movie form in case the movie doesn't exist yet.
2. On a movie's show page, display a count of how many characters belong to that movie.
3. On a movie's show page, display a list of the characters that belong to that movie.
4. At the bottom of the list of characters, write a form to add a new character directly to that movie (without having to go to http://localhost:3000/characters/new). You can start by copying over the boilerplate new character form, and then modify it to pre-populate the `movie_id` input with the correct value. Finally, switch the `type` of the input to "hidden".

Do the same steps for the one-to-many relationship between actors and characters.

The last hop

Now that you have a list of character names on a movie's show page, replace the character name with the actor name -- voilà, we have a cast.

Now that you have a list of character names on a actor's show page, replace the character name with the movie title -- voilà, we have a filmography.

We have achieved the many-to-many relationship between movies and actors by adding a join table (characters) and then building two one-to-manies.

Association Helper Methods

Now that we have an understanding of how to establish one-to-manies and many-to-manies in our data tables (basically, you just have to put foreign keys in the right places), let's look at some convenience methods that Rails gives us for working with associations.

belongs_to

Let's say I have a movie in a variable `m`. It is annoying and error prone to, whenever I want the director associated with a movie, have to type

```
d = Director.find_by({ :id => m.director_id })
```

Wouldn't it be great if I could just type

```
d = m.director
```

and it would know how to go look up the corresponding row in the directors table based on the movie's `director_id` ?

Unfortunately, I can't, because `.director` isn't a method that `Movie` objects know how to perform -- it is undefined. (`Movie` objects know how to perform `.director_id` because we get a method for every column in the table.)

Fortunately, since domain modeling and associations are at the heart of every application's power, Rails makes it really easy to define such a method. Just go to the `Movie` model and add a line like this:

```
belongs_to :director, :class_name => "Director", :foreign_key => "director_id"
```

This line tells Rails:

- `:director` : Define a method called `.director` for all movie objects.
- `:class_name => "Director"` : When someone invokes `.director` on a movie, go fetch a result from the directors table.
- `:foreign_key => "director_id"` : Use the value in the `director_id` column of the movie to query the directors table for a row.

This is exactly what we were doing by hand with

```
Director.find_by({ :id => m.director_id })
```

but we can now use the shorthand of just

```
m.director
```

Even better, if you've named your method and foreign key column conventionally (exactly matching the name of the other table), you can use the super-shorthand version:

```
belongs_to :director
```

Neat!

has_many

Let's say I have a director in a variable `d` . It is annoying and error prone to, whenever I want the movies associated with the director, have to type

```
a = Movie.where({ :director_id => d.id })
```

Wouldn't it be great if I could just type

```
a = d.movies
```

and it would know how to go look up the corresponding rows in the movies table?

Unfortunately, I can't, because `.movies` isn't a method that `Director` objects know how to perform -- it is undefined.

Fortunately, since domain modeling and associations are at the heart of every application's power, Rails makes it really easy to define such a method. Just go to the `Director` model and add a line like this:

```
has_many :movies, :class_name => "Movie", :foreign_key => "director_id"
```

This line tells Rails:

- `:movies` : Define a method called `.movies` for all director objects.
- `:class_name => "Movie"` : When someone invokes `.movies` on a director, go fetch results from the movies table.
- `:foreign_key => "director_id"` : Search for the director's id in the `director_id` column of the movies table.

This is exactly what we were doing by hand with

```
Movie.where({ :director_id => d.id })
```

but we can now use the shorthand of just

```
d.movies
```

Even better, if you've named your method and foreign key column conventionally (exactly matching the name of the other table), you can use the super-shorthand version:

```
has_many :movies
```

Neat!

Clean-up

For each one-to-many relationship in our application (there are three: director-movies, movie-characters, actor-characters), add the `belongs_to` and `has_many` association helpers to the models.

Then, in all of your views, replace messy `.find_by(...)` and `.where(...)` s with clean `.director` , `.movies` , etc.

has_many/through

After you have established all of your one-to-many association helper methods, you can also add many-to-many helper methods:

```
class Movie < ActiveRecord::Base
  ...

  has_many :characters
  has_many :actors, :through => :characters
end
```

This will allow you to call `.actors` directly on any movie object, and it will walk through the characters table, assemble the collection of corresponding actors, and return it to you!

Similarly,

```
class Actor < ActiveRecord::Base
  ...

  has_many :characters
  has_many :movies, :through => :characters
end
```

You may or may not need these many-to-many helper methods in this project, but it's nice to know you can easily add them.