**Cloud Application Development - Lab 5**

Active Record and Associations: One to Many Association An association allows to specify a logical relationship between two different entities. In databases, an association specifies a relationship between two tables. In relational databases the associations are supported via the **foreign keys**. A foreign key is a column (or multiple columns) in a table, whose values reference the **primary key** of another table to form associations between the entries of the two tables.

There are multiple types of associations between tables:

* One to one
* One to many
* Many to many

Rails supports the following types of associations:

* has\_one
* has\_many
* belongs\_to
* has\_and\_belongs\_to\_many

In today’s web application, we will create a simple web application which demonstrates the encoding of one to many association. We create a simple movie application which allows user to post reviews. The application has two models: **Movie** and **Review**. Each Movie can have zero or many Reviews.

Create a simple Rails application with two models

* Movie
* Review

The fields of the Movie model are title, rating and release. The field of the Review model is details.

1. Create the Rails application called movieApp.

* rails new movieApp

2. Check that the application can be run, namely start the Rails server. In another terminal start the web server to run the application. Please recall that we work with two terminals, in one terminal we run the server, and in another terminal we type different commands to extend the application. Recall that before we start the web server, we must first navigate to the root/ parent folder of the Rails application.

At the terminal type the following commands:

* Navigate to the root of your application
  + cd movieApp

• Start a web server to run the application

* rails server --binding 0.0.0.0
* rails server

• Access the application via the browser

* In the browser, enter the application’s URL http://localhost:3000

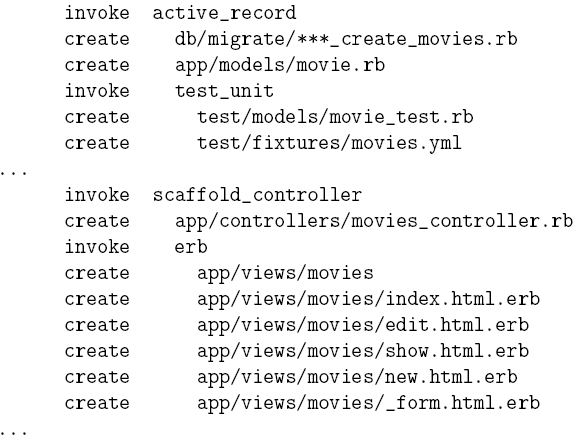
3. Create a scaffold for Movie.

Note1: Type at the terminal each command on one single line (i.e. press Enter key only at the end of a command) even though in this tutorial a command might be written across multiple lines.

Note2: do not leave blank spaces between the name of a field/column and its datatype!

Note3: In general, type all the commands provided in the tutorials, rather than copying and pasting them.

* **rails generate scaffold Movie title:string rating:string release:string**
* The above command generates, among other files, the following files



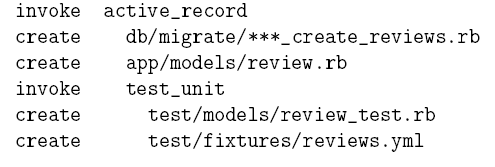
4. Create the database and the movies table

* rake db:migrate

5. Run the application. Q: How can we list all the movies from the movies table? A: In browser, append to the URL the /movies. Let’s create a new movie.

6. Create a model manually (i.e. we do not use a scaffold) for Review

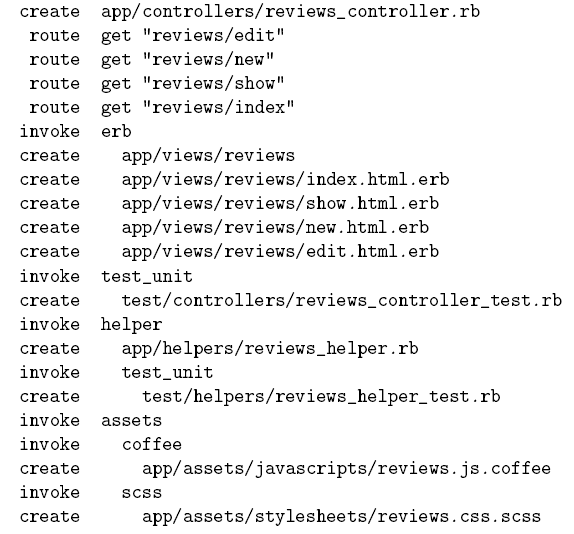
* rails generate model Review details:string movie:references
* "movie:references" creates a foreign key column named movie\_id in reviews table which reference the movies table
* Note the naming convention for the foreign key is < singularized\_table\_name > \_id e.g. movie\_id
* The files generated are



7. Generate the controller and views for Reviews

* rails generate controller Reviews index show new edit
* The above command creates the Reviews controller and 4 actions:

index, show, new and edit, and their corresponding views. Note that both the actions and the views are empty! (In contrast with the same elements corresponding to the movies.)



8. Create the reviews table

* rake db:migrate

9. Let’s modify the movie.rb model. Open the app/models/movie.rb file.

class Movie < ActiveRecord::Base

end

We update the model so that a Movie can have many Reviews (i.e. a one to many relationship/ association) by adding the reference “has\_many :reviews” and have specified that the reviews associated with the movie will be deleted when the movie is deleted.

class Movie < ApplicationRecord

has\_many :reviews, :dependent => :destroy

end

10. Update the review.rb model in app/models/review.rb file to contain the following

class Review < ApplicationRecord

end

11. Next, let’s update the routes.rb file to reflect the above relationship.

Open config/routes.rb file and replace the line

resources :movies

with

resources :movies do

resources :reviews

end

12. Apply the migrations to update the changes to our models

* rake db:migrate

13. Modify the Reviews Controller: Open the *app/controllers/reviews\_controller.rb*

file and add iteratively from the following content the method(s) corresponding to one of the CRUD operations to the *reviews\_controller.rb* file. For each method added to the controller, add the corresponding view from item 13 in the list below. Let’s start adding the body of the index method first.

# GET /movies/1/reviews

def index

# For URL like /movies/1/reviews

# Get the movie with id=1

@movie = Movie.find(params[:movie\_id])

# Access all reviews for that movie

@reviews = @movie.reviews

end

# GET /movies/1/reviews/2

def show

@movie = Movie.find(params[:movie\_id])

# For URL like /movies/1/reviews/2

# Find an review in movies 1 that has id=2

@review = @movie.reviews.find(params[:id])

end

# GET /movies/1/reviews/new

def new

@movie = Movie.find(params[:movie\_id])

# Associate an review object with movie 1

@review = @movie.reviews.build

end

# POST /movies/1/reviews

def create

@movie = Movie.find(params[:movie\_id])

# For URL like /movies/1/reviews

# Populate an review associate with movie 1 with form data

# Movie will be associated with the review

# @review = @movie.reviews.build(params.require(:review).permit!)

@review = @movie.reviews.build(params.require(:review).permit(:details))

if @review.save

# Save the review successfully

redirect\_to movie\_review\_url(@movie, @review)

else

render :action => "new"

end

end

# GET /movies/1/reviews/2/edit

def edit

@movie = Movie.find(params[:movie\_id])

# For URL like /movies/1/reviews/2/edit

# Get review id=2 for movie 1

@review = @movie.reviews.find(params[:id])

end

# PUT /movies/1/reviews/2

def update

@movie = Movie.find(params[:movie\_id])

@review = Review.find(params[:id])

if @review.update(params.require(:review).permit(:details))

# Save the review successfully

redirect\_to movie\_review\_url(@movie, @review)

else

render :action => "edit"

end

end

# DELETE /movies/1/reviews/2

def destroy

@movie = Movie.find(params[:movie\_id])

@review = Review.find(params[:id])

@review.destroy

respond\_to do |format|

format.html { redirect\_to movie\_reviews\_path(@movie) }

format.xml { head :ok }

end

end

14. Add the content for the Views

(a) Add to the file app/views/reviews/index.html.erb the following content

<h1>Reviews in <%= @movie.title %></h1>

<table>

<tr>

<th>Review details</th>

</tr>

<% for review in @reviews %>

<tr>

<td><%= review.details %></td>

<td><%= link\_to 'Show', movie\_review\_path(@movie, review) %></td>

<td><%= link\_to 'Edit', edit\_movie\_review\_path(@movie, review) %></td>

<td><%= button\_to "Destroy this movie", movie\_review\_path(@movie, review), method: :delete %></td>

</tr>

<% end %>

</table>

<br />

<%= link\_to 'New review', new\_movie\_review\_path(@movie) %>

<%= link\_to 'Back to Movie', @movie %>

(b) Add to the file app/views/reviews/show.html.erb the next content

<h1>Review in <%= @movie.title %></h1>

<p>

<b>Review details:</b>

<%= @review.details %>

</p>

<%= link\_to 'Edit', edit\_movie\_review\_path(@movie, @review) %> |

<%= link\_to 'Back', movie\_reviews\_path(@movie) %>

(c) Append to the file app/views/reviews/new.html.erb the content below. Note that the new.html.erb form uses the form template.

<h1>New review</h1>

<%= render 'form' %>

<%= link\_to 'Back', movie\_reviews\_path(@movie) %>

(d) Customize the app/views/reviews/edit.html.erb with the content below. Note that the edit.html.erb form uses the form template.

<h1>Editing review</h1>

<%= render 'form' %>

<%= link\_to 'Show', movie\_review\_path(@movie, @review) %> |

<%= link\_to 'Back', movie\_reviews\_path(@movie) %>

15. Create the form used by both the new.html.erb and edit.html.erb template views from the item 14 of this tutorial. Create the file named \_form.html.erb in the app/views/reviews/ with the following content

<%= form\_for([@movie, @review]) do |f| %>

<% if @review.errors.any? %>

<div id="error\_explanation">

<h2><%= pluralize(@review.errors.count, "error")%>

prohibited this review from being saved:</h2>

<ul>

<% @review.errors.full\_messages.each do |msg| %>

<li><%= msg %></li>

<% end %>

</ul>

</div>

<% end %>

<div class="field">

<p>

<%= f.label :Details %><br />

<%= f.text\_field :details %>

</p>

</div>

<div class="actions">

<%= f.submit %>

</div>

<% end %>

16. Customize app/views/movies/show.html.erb to allow a user to display all the reviews corresponding to a movie.

<p id="notice"><%= notice %></p>

<p>

<strong>Title:</strong>

<%= @movie.title %>

</p>

<p>

<strong>Rating:</strong>

<%= @movie.rating %>

</p>

<p>

<strong>Release Date:</strong>

<%= @movie.release %>

</p>

<!-- Display the reviews of a movie -->

<h2>Reviews</h2>

<% @movie.reviews.each do |review| %>

<p>

<b>Review details:</b>

<%= review.details %>

</p>

<% end %>

<%= link\_to 'Edit Movie', edit\_movie\_path(@movie) %> |

<%= link\_to 'Back', movies\_path %> |

<%= link\_to 'Show Reviews', movie\_reviews\_path(@movie) %>

<%= button\_to "Destroy this movie", @movie, method: :delete %>

17. Run the application. In browser, append to the URL the /movies.

Create a new movie and add several reviews to the movie. Repeat the process for several movies.

Upload your completed application here.

Useful resources

* Active Record http://api.rubyonrails.org/classes/ActiveRecord/Base.html#M002281
* Migration http://guides.rubyonrails.org/migrations.html
* Migration http://api.rubyonrails.org/classes/ActiveRecord/Migration.html
* SQLite http://www.sqlite.org/