Rail Demo 3: Associating Authors and Posts

1. Let's now assume that we want to have authors associated with the Posts. We can start by creating an Author entity w/scaffolding:

   ```
   rails g scaffold Author fname:string lname:string email:string thumbnail:string
   ```

2. Now that we know how to validate our models, let's go ahead and add some validations to our author model in models/author.rb. Please note that we should probably first write out our model requirements, than write the tests, then write the actual validates methods in our model. However, for demonstration purposes we'll go right to the model code:

   First, let's require fname, lname, and email all to be present:

   ```
   validates :fname, :lname, :email, presence: true
   ```

   Second, make sure a unique and properly formatted email address is provided:

   ```
   VALID_EMAIL_REGEX = /\A[\w+\-.]+@[a-z\d\-.]+\.[a-z]+\z/i
   validates :email, presence: true, length: {maximum: 105},
           uniqueness: { case_sensitive: false },
           format: { with: VALID_EMAIL_REGEX }
   ```

   Third, validate the thumbnail making sure the filename ends in .gif, jpg, or png.

   ```
   validates :thumbnail, allow_blank: true, format: {with:
   %r{\.(gif|jpg|png)\Z}i,
   message: 'must be a URL for GIF, JPG or PNG image.'}
   ```

3. Amazing stuff heh? Now let's go ahead and run the generated authors migration, run our server and examine our handiwork:

   ```
   rake db:migrate
   rails server
   ```

4. Now its time to formally associate authors with posts. First we need to create a new migration to update our database schema:

   ```
   rails g migration AddAuthorToPost author:references
   ```

5. Now we need to inform our models of this relationship. We do this by adding the following line to models/author.rb:

   ```
   has_many :posts
   ```

   and the following line to models/post.rb

```
belongs_to :author
```

6. Now go ahead and run the migration, and then examine the new author_id field in the posts table by examining the db/schema.rb.

   ```
   rake db:migrate
   ```

7. Now let's explore what we've accomplished in our rails console and note the magic that these minor tweaks has brought about!

   ```
   rails console

   p = Post.first
   ```

   Notice how the author_id on the post is nil! The column was added to the Post table when we ran the migration above, but the data has not yet been populated. Let’s manually do that! Let’s take a look at the authors that are available to us:

   ```
   authors = Author.all
   ```

   or better yet, let's just pick out the one with id = 2

   ```
   a = Author.find(2)
   ```

   now we can do an assignment like this:

   ```
   p.author = a
   ```

   but in order to save it we still have to commit to the database by calling the save method!

   ```
   p.save!
   ```

   now if we do a query for the first post you will see that the author_id has been updated to 2!

   ```
   p = Post.first
   ```

   notice that we can simply reference the author attribute to access the name of the author:

   ```
   puts "authored by #{p.author.fname} #{p.author.lname}"
   ```

   Similarly, we can get the array of all posts by a given author by simply asking for them:

   ```
   a = Author.find(2)
   ap = a.posts
   ```

8. Now, we need to update the views to handle the association between author and post. First let's go ahead and add a author selection drop down in the Post _form partial:

   ```
   <div class="field">
       <%= form.label :author_id %>
       <%= form.select :author_id,
           options_for_select(Author.all.collect {|a| ["#{a.lname},
           #{a.fname}", a.id]},
           selected: (@post.author ? @post.author.id : Author.first.id)),
           {} %>
   </div>
   ```

9. Next, we add the author info to the Post listing view in views/posts/index.html.erb. We can add a column heading in the first row of the table:

   ```
   <th>Author</th>
   ```

   and inside the @posts.each loop place this item as the first element of the row:

   ```
   <td><%= (post.author.try(:lname) || "NA") %></td>
   ```

   finally, in the views/posts/show.html.erb be sure to add markup to show the post's author:

   ```
   <p>
       <strong>Author:</strong>
       <%= (@post.author.try(:lname) || "NA")%>
   </p>
   ```

10. Almost done, now we need to update hte Post controller so it saves the association between post and author. First we need to allow the new author*id field in the form variables the controller allows by modifying this private method on controllers/post*controller.rb:

    ```
    def post_params
        params.require(:post).permit(:title, :article, :likes, :status, :author_id)
    end
    ```

    and now we wrap things up by adding additional logic in the create method to build out the association by placing the following immediately after creating the new Post instance:

    ```
    author = Author.find(post_params[:author_id])
    @post.build_author(:id  => author.id)
    ```