Ch 14 notes - Following users

My tutorial notes… be brief, H3 for each section, bold terms etc, code is italics

In chapter 14… "add a social layer to allow users to follow (and unfollow) other users"

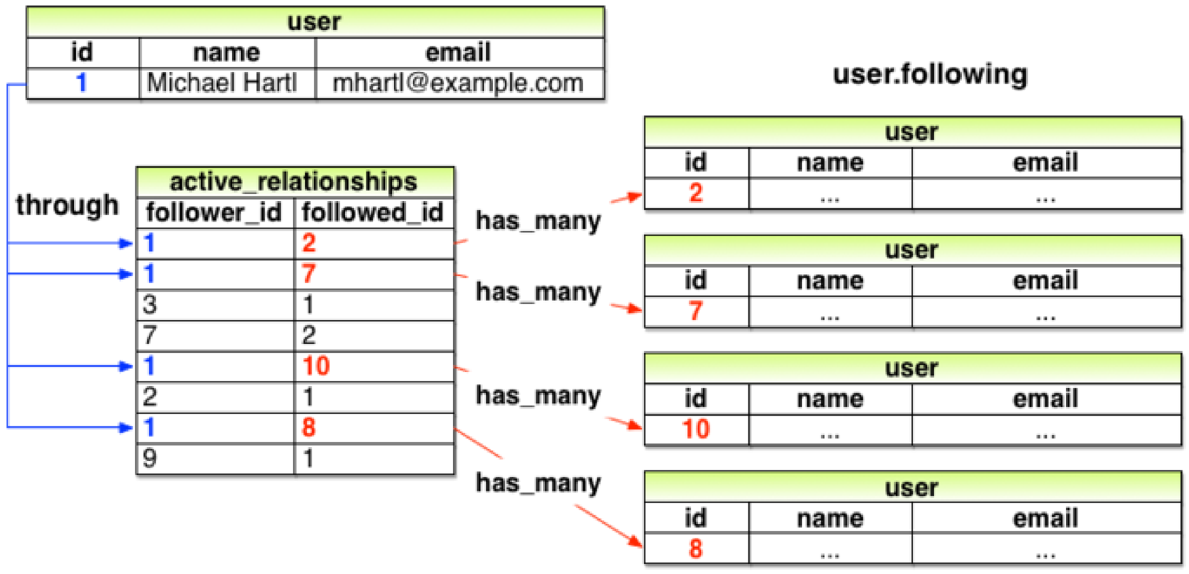
- ➢ The tutorial: www.railstutorial.org/
- ➢ Hartl's reference version of sample_app: bitbucket.org/railstutorial/sample_app_4th_ed
- ➢ My sample_app: sampleapp694.herokuapp.com/

"This final chapter contains some of the most challenging material in the tutorial, including some **Ruby/SQL trickery** to make the status feed."
Impacted screens include: 1) current user profile, 2) all users, 3) other user profile and follow button, 4) other user profile with unfollow button and followers count, and 5) Home page with status feed and following count.

I didn't do the chapter, I just read through it and highlighted the important parts. enjoy… yow, bill

14.1 The Relationship model

The many-to-many relationship between following and followed users needs a new construct, a separate "active relationships" table.



follower_id	followed_id
1	2
1	7
3	1
7	2
1	10
2	1
1	8
9	1

Add relationships to user data model. Following is the active_relationship and followers are passive_relationship. (compare to micropost 1-to-many construct):

```
class User < ApplicationRecord
  has_many :microposts, dependent: :destroy
  has_many :active_relationships,  class_name:  "Relationship",
                                   foreign_key: "follower_id",
                                   dependent:   :destroy
  has_many :passive_relationships, class_name:  "Relationship",
                                   foreign_key: "followed_id",
                                   dependent:   :destroy
  has_many :following, through: :active_relationships,  source: :followed
  has_many :followers, through: :passive_relationships, source: :follower
```

And here's the relationship model:

```
class Relationship < ApplicationRecord
  belongs_to :follower, class_name: "User"
  belongs_to :followed, class_name: "User"
```

Multiple key index to create relationship…. here it's follower and followed:

```
@relationship = Relationship.new(follower_id: users(:michael).id,
                                 followed_id: users(:archer).id)
```

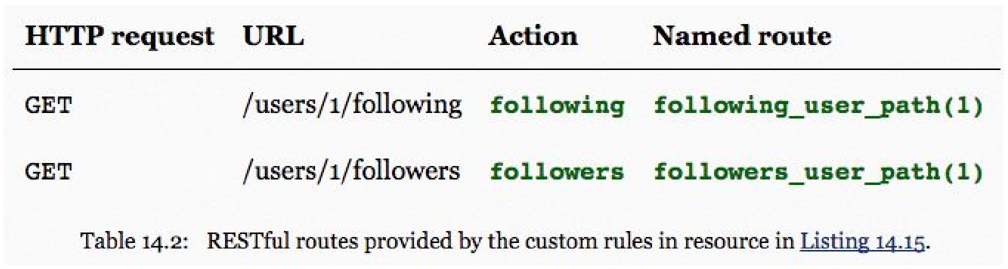
Use **has_many :through**: a user has-many following through relationships.

```
has_many :following, through: :active_relationships, source: :followed
```

14.2 A web interface for following users

Some fancy new stuff in routes.rb to get follower and following URL's:

```
resources :users do
    member do
      get :following, :followers
    end
  end
```



HTTP request	URL	Action	Named route
GET	/users/1/following	following	following_user_path(1)
GET	/users/1/followers	followers	followers_user_path(1)

Table 14.2: RESTful routes provided by the custom rules in resource in Listing 14.15.

Interesting User controller code… these methods drive the follower and following pages:

```
def following
    @title = "Following"
    @user  = User.find(params[:id])
    @users = @user.following.paginate(page: params[:page])
    render 'show_follow'
  end

  def followers
    @title = "Followers"
    @user  = User.find(params[:id])
    @users = @user.followers.paginate(page: params[:page])
    render 'show_follow'
  end
```

Ajax: “Because adding Ajax to web forms is a common practice, Rails makes Ajax easy to implement.” Change **form_for** to **form_for ..., remote: true**

In HTML this “sets the variable data-remote="true" inside the form tag, which tells Rails to allow the form to be handled by JavaScript.”

More complexity here that I’ll bypass… but a good starting point if we do any Ajax.

14.3 The status feed

Important: Use the **where method** for an easy SQL query:

```
Micropost.where("user_id = ?", id)
Micropost.where("user_id IN (?) OR user_id = ?", following_ids, id)
```

Arbitrary SQL commands (SELECT, etc) can be included in a where method call. I do **not** expect us to need this however. If you do this, please show me/others.

14.4 Conclusion

Hartl's lists 10 options for further learning, including his own venture, The Learn Enough Society.

What we learned:

- ➔ Rails' **has_many :through** allows modeling of complicated data relationships.
- ➔ The **has_many method** takes several optional arguments, including the object class name and the foreign key.
- ➔ Using **has_many** and **has_many :through** with properly chosen class names and foreign keys, we can model both active (following) and passive (being followed) relationships.
- ➔ Rails routing supports **nested routes**.
- ➔ The **where method** is a flexible and powerful way to create database queries.
- ➔ Rails supports issuing lower-level **SQL queries** if needed.

Don't forget that Hartl's reference code for sample_app is here:

- ➢ bitbucket.org/railstutorial/sample_app_4th_ed

Done.
thanks… yow, bill

THE
RUBY ON RAILS TUTORIAL
LEARN WEB DEVELOPMENT WITH RAILS
MICHAEL HARTL
RUBY ON RAILS TUTORIAL (RAILS 5)
Learn Web Development with Rails
Michael Hartl
BOOK INFO
CONTACT AUTHOR