CS169.1x Lecture 6: Basic Rails

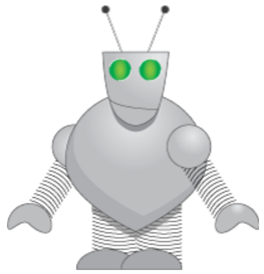
Fall 2012

The Database is Golden

- Contains valuable customer data—don’t want to test your app on that!
- Rails solution: development, production and test *environments* each have own DB
  - Different DB types appropriate for each
- How to make *changes* to DB, since will have to repeat changes on production DB?
- Rails solution: *migration*—script describing changes, portable across DB types

Migration Advantages

- Can identify each migration, and know which one(s) applied and when
  - Many migrations can be created to be *reversible*
- Can manage with version control
- *Automated == reliably repeatable*
  - Compare: use Bundler vs. manually install libraries/gems



- Theme: *don't do it—automate it*
  - *specify* what to do, create tools to automate

Meet a Code Generator

rails generate migration CreateMovies

http://pastebin.com/VYwbc5fq

- Note, this just *creates* the migration. We haven't *applied* it.
- Apply migration to development: **rake db:migrate**
- Apply migration to production: **heroku rake db:migrate**
- Applying migration also records in DB itself which migrations have been applied

Rails Cookery #1

- Augmenting app functionality ==
adding models, views, controller actions

To *add a new model* to a Rails app:

- (or change/add attributes of an existing model)

1. Create a migration describing the changes: `rails generate migration` (gives you boilerplate)
2. Apply the migration: `rake db:migrate`
3. If new model, create model file `app/models/`*model*`.rb`

- Update test DB schema: `rake db:test:prepare`

Based on what you've seen of Rails, what kind of object is *likely* being yielded in the migration code:

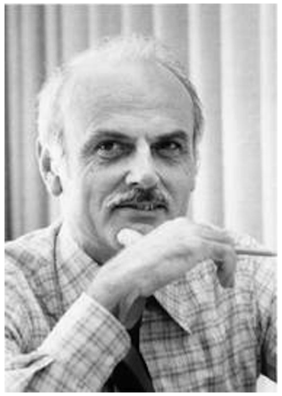
```
def up
  create_table 'movies' do |t|
    t.datetime 'release_date' ...
  end
end
```

- ☐ An object representing a database
- ☐ An object representing an instance of a model
- ☐ An object representing a table
- ☐ Come on, it could be anything

*

CRUD in SQL

"Ted" Codd



- Structured Query Language (SQL) is the query language used by RDBMS's
- Rails *generates* SQL statements at runtime, based on your Ruby code
- 4 basic operations on a table row: **C**reate, **R**ead, **U**pdate attributes, **D**elete

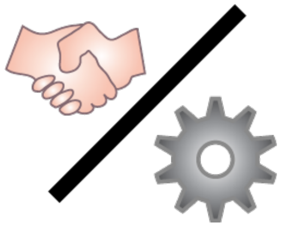
```
INSERT INTO users
(username, email, birthdate) VALUES ("fox",
"Fox@cs.berkeley.edu", "1968-05-12"),
     "patterson", "pattrsn@cs.berkeley.edu", "????")
SELECT * FROM usersWHERE (birthdate BETWEEN "1987-01-01" AND
"2000-01-01")
UPDATE users SET email = "armandofox@gmail.com"WHERE
username="fox"
DELETE FROM users WHERE id=1
```

The Ruby side of a model

- Subclassing from ActiveRecord::Base
- "connects" a model to the database
- provides CRUD operations on the model

http://pastebin.com/ruu5y0D8

- Database table name derived from model's name: **M**ovie→**m**ovie**s**



- Database table column names are getters & setters for model attributes
- *Observe: the getters and setters <u>do not</u> simply modify instance variables!*

Creating: new ≠ save

- Must call save or save! on an AR model instance to actually save changes to DB
  - '!' version is “dangerous”: throws exception if operation fails
  - create just combines new and save
- Once created, object acquires a primary key (id column in every AR model table)
  - if x.id is nil or x.new_record? is true, x has never been saved
- These behaviors inherited from ActiveRecord:: Base—not true of Ruby objects in general

Read: finding things in DB

- class method where selects objects based on attributes

```
Movie.where("rating='PG'")
Movie.where('release_date < :cutoff and
   rating = :rating',
   :rating => 'PG', :cutoff => 1.year.ago)
Movie.where("rating=#{rating}") # BAD IDEA!
```

- Can be chained together efficiently

```
kiddie = Movie.where("rating='G'")

old_kids_films =
 kiddie.where "release_date < ?", 30.years.ago
```

Read: find_*

•find by id:Movie.find(3) *#exception if not found*Movie.find_by_id(3) *# nil if not found*

•dynamic attribute-based finders using method_missing:Movie.find_all_by_rating('PG')
Movie.find_by_rating('PG')
Movie.find_by_rating**!**('PG')

Update: 2 ways

•Modify attributes, then save object

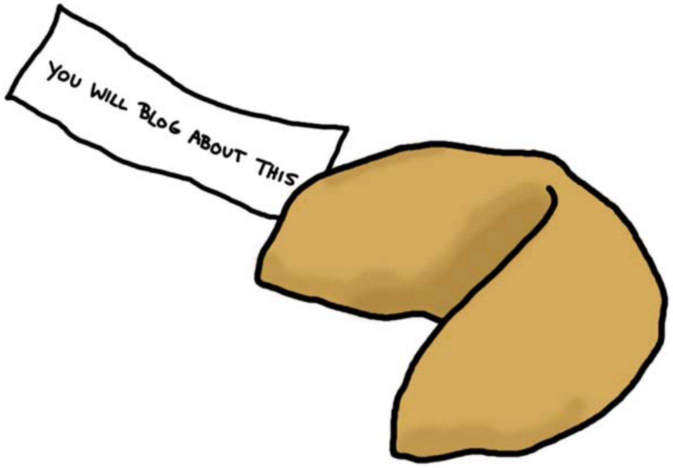
```
m=Movie.find_by_title('The Help')
m.release_date='2011-Aug-10'
m.save!
```

•Update attributes on existing object

```
Movie.find_by_title('The Help').
  update_attributes!(
     :release_date => '2011-Aug-10'
  )
```

•Transactional: either all attributes are updated, or none are

Assume table fortune_cookies has column fortune_text
Which of these instance methods ofFortuneCookie < ActiveRecord::Base
will **not** return a silly fortune (if any)?



☐
```
def silly_fortune_1
  @fortune_text + 'in bed'
end
```

☐
```
def silly_fortune_2
  self.fortune_text + 'in bed'
end
```

☐
```
def silly_fortune_3
  fortune_text + 'in bed'
end
```

☐ They will all return a silly fortune

*

Rails Cookery #2

•To *add a new action* to a Rails app

1. Create *route* in **config/routes.rb** if needed
2. Add the *action* (method) in the appropriate **app/controllers/*_controller.rb**
3. Ensure there is something for the action to *render* in **app/views/*model*/*action*.html.haml**

4. We'll do Show action & view (book walks through Index action & view)

MVC responsibilities

- *Model:* methods to get/manipulate data

Movie.where(...), Movie.find(...)

- *Controller:* get data from Model, make available to View

```
def show
  @movie = Movie.find(params[:id])
end
```

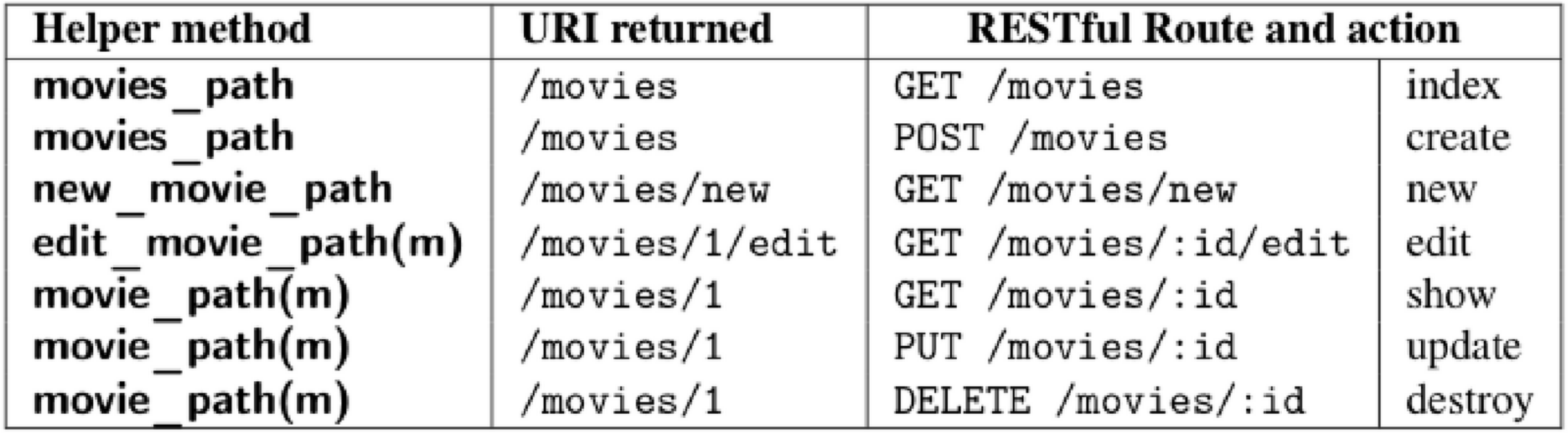
Instance variables set in Controller available in View

Absent other info, Rails will look for app/views/movies/show.html.haml

- *View:* display data, allow user interaction
- Show details of a movie (description, rating)
- But…
- What else can user do from this page?
- How does user get to this page?

http://pastebin.com/kZCB3uNj

How we got here: URI helpers



Helper method	URI returned	RESTful Route and action	
movies_path	/movies	GET /movies	index
movies_path	/movies	POST /movies	create
new_movie_path	/movies/new	GET /movies/new	new
edit_movie_path(m)	/movies/1/edit	GET /movies/:id/edit	edit
movie_path(m)	/movies/1	GET /movies/:id	show
movie_path(m)	/movies/1	PUT /movies/:id	update
movie_path(m)	/movies/1	DELETE /movies/:id	destroy

index.
html.
haml

```
link_to movie_path(3)
```

↓

```
<a href="/movies/3">...</a>
```

↓

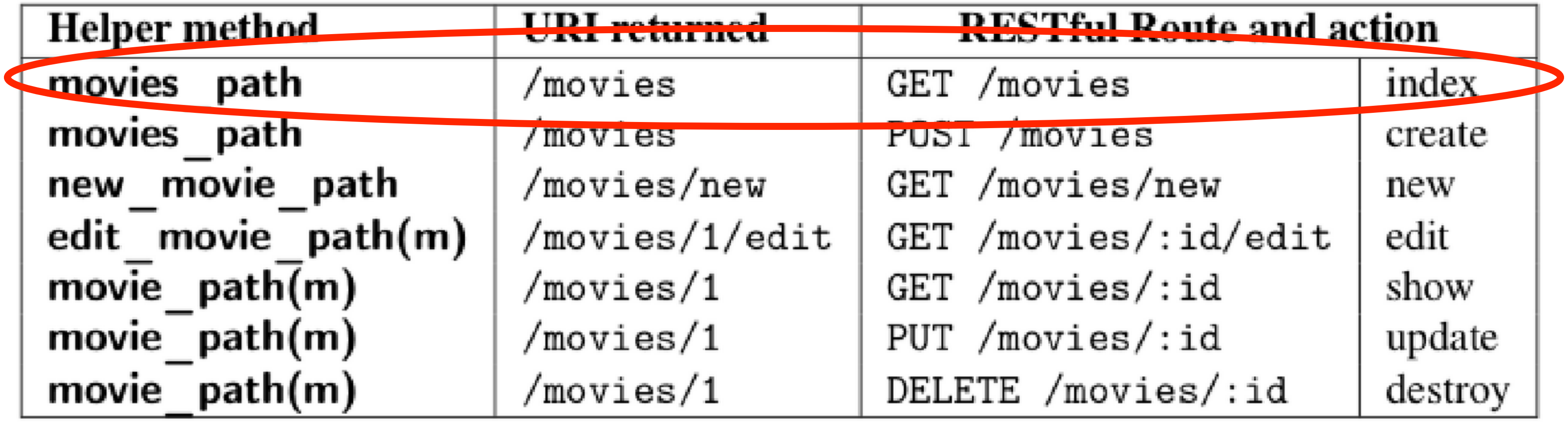
GET /movies/:id
{:action=>"show", :controller=>"movies"}
params[:id]←3

```
def show
 @movie =
   Movie.find(params[:id])
end
```

What else can we do?

- How about letting user return to movie list?
- RESTful URI helper to the rescue again:
- movies_path with no arguments links to Index action

=link_to 'Back to List', movies_path



Helper method	URI returned	RESTful Route and action	
movies_path	/movies	GET /movies	index
movies_path	/movies	POST /movies	create
new_movie_path	/movies/new	GET /movies/new	new
edit_movie_path(m)	/movies/1/edit	GET /movies/:id/edit	edit
movie_path(m)	/movies/1	GET /movies/:id	show
movie_path(m)	/movies/1	PUT /movies/:id	update
movie_path(m)	/movies/1	DELETE /movies/:id	destroy

A) A route consists of **both** a URI and an HTTP method

B) A route URI **must** be generated by Rails URI helpers

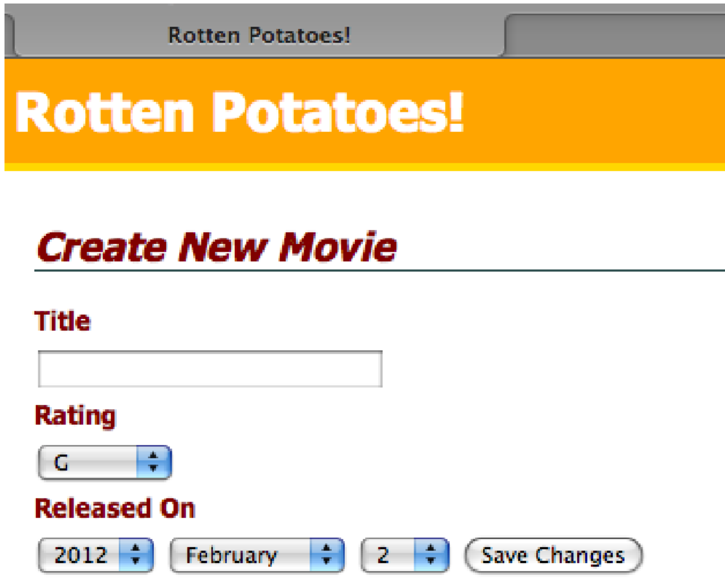
C) A route URI **may** be generated by Rails URI helpers

- ☐ Only (A) is true
- ☐ Only (C) is true
- ☐ Only (A) and (B) are true
- ☐ Only (A) and (C) are true

*

Dealing with forms

- Creating a resource usually takes 2 interactions
- new: Retrieve blank form
- create: Submit filled form
- How to generate/display?
- How to get values filled in by user?
- What to "return" (render)?



Rails Cookery #3

- To create a new submittable form:

1. Identify the action that gets the form itself
2. Identify the action that receives *submission*
3. Create routes, actions, views for each

- In form view, form element name attributes control how values will appear in params[]
- Helpers provided for many common elements

Creating the Form

- Anatomy of a form in HTML

http://pastebin.com/k8Y49EhE

- the *action* and *method* attributes (i.e., the route)
- only *named* form inputs will be submitted
- Generating the form in Rails
- often can use URI helper for *action,* since it's just the URI part of a route (still need *method*)
- *form field helpers* (see api.rubyonrails.org) generate conveniently-named form inputs

http://pastebin.com/3dGWsSq8

Redirection, the Flash and the Session *(ELLS §4.7)*

Armando Fox

What view should be rendered for create action?

- Idiom: *redirect* user to a more useful page.
  - e.g., list of movies, if create successful
  - e.g., New Movie form, if unsuccessful
- Redirect triggers a *whole new HTTP* request
  - How to inform user *why* they were redirected?
- Solution: flash[]—quacks like a hash that *persists until end of **next** request*
  - flash[:notice] conventionally for information
  - flash[:warning] conventionally for "*errors*"

Flash & Session

- session[]: like a hash that persists forever
- reset_session nukes the whole thing
- session.delete(:some_key), like a hash
- By default, cookies store *entire contents* of session & flash
- Alternative: store sessions in DB table (Google "rails session use database table")
- Another alternative: store sessions in a "NoSQL" storage system, like *memcached*

Ben Bitdiddle says: “You can put arbitrary objects (not just “simple” ones like ints and strings) into the session[].” What do you think?

- ☐ True—knock yourself out!
- ☐ True—but a bad idea!
- ☐ False, because you can’t put arbitrary objects into a hash
- ☐ False, because session[] isn’t really a hash, it just quacks like one