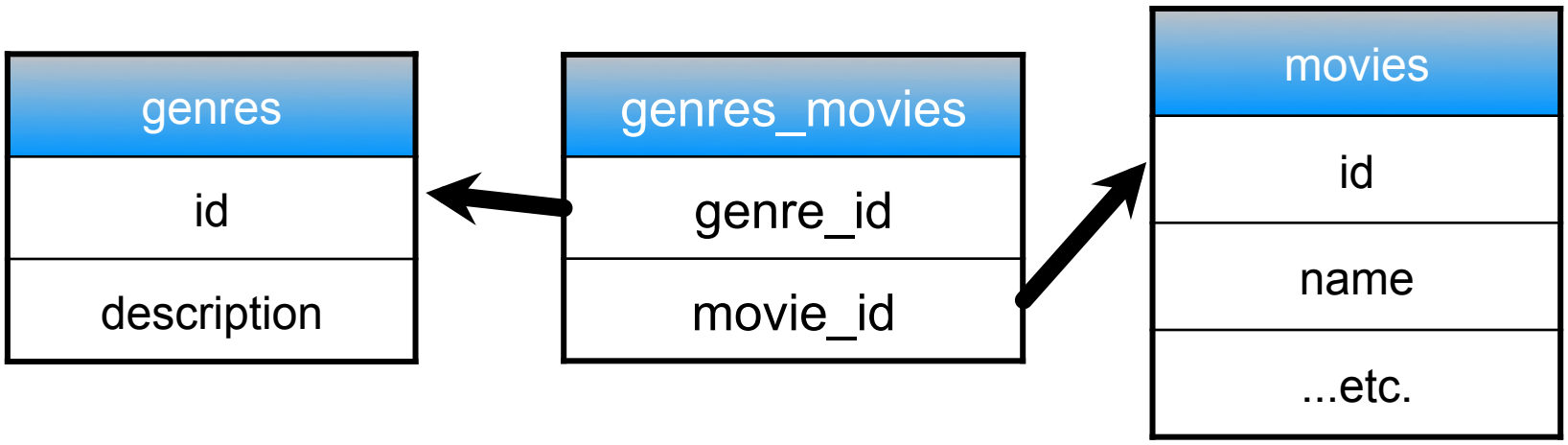
A shortcut: has and belongs to many (habtm)

- join tables express a relationship **between** existing model tables using FKs
- Join table has *no primary key*
- because there's *no object* being represented!

http://pastebin.com/tTVGtNLx

```
movie has_and_belongs_to_many :genres
genre has_and_belongs_to_many :movies
@movie.genres << Genre.find_by_name('scifi')
```



Rules of Thumb

- if you can conceive of things as different real-world objects, they should probably be distinct models linked through an association
- if you don't need to represent *any* other aspect of a M-M relationship, use habtm
- otherwise, use has_many :through

*

*

HABTM Naming Conventions

- M-M relationship naming convention:

  if a Bar

  has_and_belongs_to_many :foos

  then a Foo

  has_and_belongs_to_many :bars

  and the database table is the **plural** AR names in **alphabetical** order

  bars_foos

*

We want to model students having appointments with faculty members. Our model would include which relationships:

- Faculty has-many appointments, Student has-many appointments
- Faculty HABTM Students, Students HABTM Faculty
- Faculty belongs-to appointment, Student belongs-to appointment
- Faculty has-many appointments, through Students

RESTful Routes for Associations *(ELLS §7.5)*

Armando Fox

Nested RESTful Routes

- in config/ routes.rb:resources :moviesbecomesresources :movies do
  resources :reviews
end
-
- Nested Route: access reviews by going ”through” a movie

Nested RESTful Routes

Helper method	RESTful Route and action	
movie_reviews_path(m)	GET /movies/:movie_id/reviews	index
movie_review_path(m)	POST /movies/:movie_id/reviews	create
new_movie_review_path(m)	GET /movies/:movie_id/reviews/new	new
edit_movie_review_path(m,r)	GET /movies/:movie_id/reviews/:id/edit	edit
movie_review_path(m,r)	GET /movies/:movie_id/reviews/:id	show
movie_review_path(m,r)	PUT /movies/:movie_id/reviews/:id	update
movie_review_path(m,r)	DELETE /movies/:movie_id/reviews/:id	destroy

available as params[:movie_id]
available as params[:id]

ReviewsController#create

```
# POST /movies/1/reviews
# POST /movies/1/reviews.xml
def create
  # movie_id because of nested route
  @movie = Movie.find(params[:movie_id])

  # build sets the movie_id foreign key automatically
  @review =
     @movie.reviews.build(params[:review])

  if @review.save
    flash[:notice] = 'Review successfully created.'
    redirect_to(movie_reviews_path(@movie))
  else
    render :action => 'new'
  end
end
```

ReviewsController#new

```
# GET /movies/1/reviews/new
def new
  # movie_id because of nested route
  @movie = Movie.find(params[:movie_id])
  # new sets movie_id foreign key automatically
  @review ||=  @movie.reviews.new
  @review = @review || @movie.reviews.new
end
```

Views

```
%h1 Edit

= form_tag movie_review_path(@movie,@review),
   :method => :put  do |f|

  ...Will f create form fields for a Movie or a Review?

= f.submit "Update Info"

= link_to 'All reviews for this movie',
      movie_reviews_path(@movie)
```

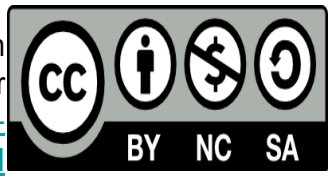
- Remember, these are for *convenience.* Invariant is: review when created or edited must be associated with a movie.

If we also have moviegoer has_many reviews,
can we use moviegoer_review_path() as a helper?

- Yes, it should work as-is because of convention over configuration
- Yes, but we must declare reviews as a nested resource of moviegoers in routes.rb
- No, because there can be only one RESTful route to any particular resource
- No, because having more than one through-association involving Reviews would lead to ambiguity

DRYing Out Queries with Reusable Scopes *(ELLS §7.6)*

Armando Fox



"Customizing" associations with declarative scopes

- Movies appropriate for kids?
- Movies with at least N reviews?
- Movies with at least average review of N?
- Movies recently reviewed?
- *Combinations of these?*

Scopes Can Be "Stacked"

Movie.for_kids.with_good_reviews(3)

Movie.with_many_fans.recently_reviewed

- Scopes are evaluated lazily!

http://pastebin.com/BW40LAHX

```
1 # in controller:
2 def good_movies_for_kids
3  @m = Movie.for_kids.with_good_reviews(3)
4 end
5 # in view:
6  - @m.each do |movie|
7    %p= pretty_print(movie)
```

Where does the database query happen?

- Line 3
- Line 6
- Line 7
- Depends on argument of with_good_reviews (i.e. whether there will be anything to return)

Associations wrap-up *(ELLS §7.7-7.9)*

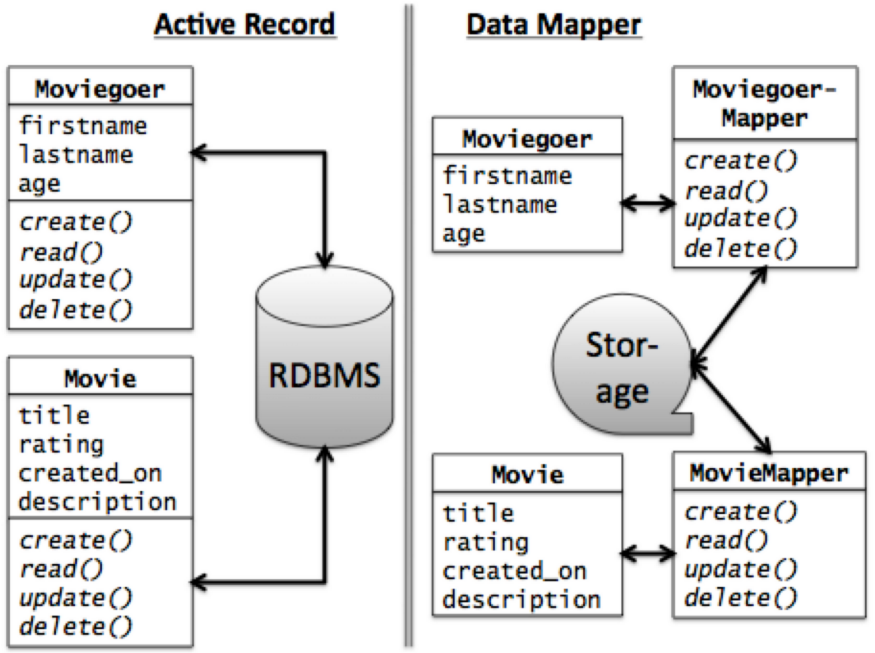
Armando Fox

Associations wrap-up

- Associations are part of *application architecture*
- provides high-level, reusable association constructs that manipulate RDBMS foreign keys
- Mix-ins allow Associations mechanisms to work with any ActiveRecord subclass
- *Proxy methods* provide Enumerable-like behaviors
- A many-fold association quacks like an Enumerable
- Proxy methods are an example of a *design pattern*
- Nested routes help you maintain associations RESTfully—but they're optional, and not magic

Elaboration: DataMapper

- Data Mapper associates separate *mapper* with each model
- Idea: keep mapping *independent* of particular data store used => works with more types of databases
- Used by Google AppEngine
- Con: can't exploit RDBMS features to simplify complex queries & relationships



*

Referential Integrity

- What if we delete a movie with reviews?
- movie_id field of those reviews then refers to *nonexistent* primary key
- another reason primary keys are *never* recycled
- Various possibilities depending on app...
- delete those reviews? has_many :reviews, :dependent => :destroy
- make reviews "orphaned"? (no owner)

```
has_many :reviews, :dependent => :nullify
```

- Can also use *lifecycle callbacks* to do other things (eg, merging)

Testing Referential Integrity

```
it "should nuke reviews when movie deleted" do
@movie = @movie.create!(...)
  @review = @movie.reviews.create!(...)
  review_id = @review.id
  @movie.destroy

    lambda { Review.find(review_id) }.should
      raise_error(ActiveRecord::RecordNotFound)

end
```

Advanced Topics

- Single-Table Inheritance (STI) & Polymorphic Associations
- Self-referential has_many :through
- Many declarative options on manipulating associations (like validations)
- To learn (much) more:
- http://guides.rubyonrails.org/association_basics.html
- *The Rails Way*, Chapter 9