

Lesson 13:

Model associations

Objectives

- Understand data model associations
- Implement one-to-many model association
- Understand Rails migrations
- Implement many-to-many model association

How does Rails work with related and unrelated data?

A database table is like a spreadsheet.

Rails interacts with each table through a model.

If data in two tables is unrelated, write the code you need for each model.

How does Rails work with related and unrelated data?

But, what if the data in two separate tables is related somehow? For example:

one movie has *many* reviews

one director has *many* movies

many movies have *many* film festivals

one movie has *one* production studio

When should data be separated?

Should movies and reviews be separate models?

What is relational database design?

A database table defines a set of columns of specified data types for one part of an app.

Tables may have *relationships* with other tables.

What is relational database design?

How tables and relationships should be defined is guided by data normalization standards:

- *one to many* relationships use *primary to foreign key* relationships
- *many to many* relationships use *a join table*

A primary key is a *unique* identifier for each record in a table, often simply called “id”

What is relational database design?

A *foreign key* is a column in a table that holds the primary key value from another related table.

A Rails model represents a record, its table, and its relationships models may have associations with other models.

Rails provides assistance in defining associations between models database design knowledge is helpful with Rails and, for some things, required:

http://en.wikipedia.org/wiki/Relational_database

How do I create a one to many relationship tables?

The “many” table needs a foreign key column to hold a reference to the related “one” table:

one movie *has many* reviews

each review *belongs to* one movie

How do I create a one to many relationship tables?

The review table needs a *foreign key* for Movie

```
rails generate scaffold Review author:string review:text  
score:integer movie_id:integer
```

The Movie table does not need a foreign key for Review

```
rails generate scaffold Movie title:string synopsis:text  
release:date
```

How do I define a one to many relationship in the models?

The many reviews each *belong to* one movie
and each movie *has many* reviews.

```
class Review < ActiveRecord::Base
  belongs_to :movie
end
```

```
class Movie < ActiveRecord::Base
  has_many :reviews
end
```

How do I display available “ones” when creating a “many” record?

Models support a find method for accessing its records

One way to display movies from a Review form would be to find all records using the Movie model and display them within a Select control

```
<%=  
  f.select :movie_id,  
    Movie.all.map { |movie| [movie.title, movie.id] }  
%>
```

Exercise:

Creating & using
two models in one form

How do I determine what records have been created?

Structured Query Language (SQL) is the primary way to interact with a database

<http://en.wikipedia.org/wiki/Sql>

SELECT queries retrieve database records use this general format:

```
SELECT [column names or *] FROM [table name] WHERE [these conditions are true];
```

How do I determine what records have been created?

All records in a Movies table could be retrieved using:

```
SELECT * FROM Movies;
```

Rails supports a database console to run SQL queries against the current application:

```
rails dbconsole
```

How do I destroy related “many” records when I destroy their “one”?

If a movie record is destroyed, you might want to destroy the related reviews should be destroyed, otherwise could be orphaned in the database.

The `has_many` command enable destruction of dependently related records.

```
class Movie < ActiveRecord::Base
  has_many :reviews, dependent: :destroy
end
```

Exercise:
**Listing & destroying
related records**

How do I create a route to display a list of related records?

An additional member route can be added to the :movies resource in config/routes.rb

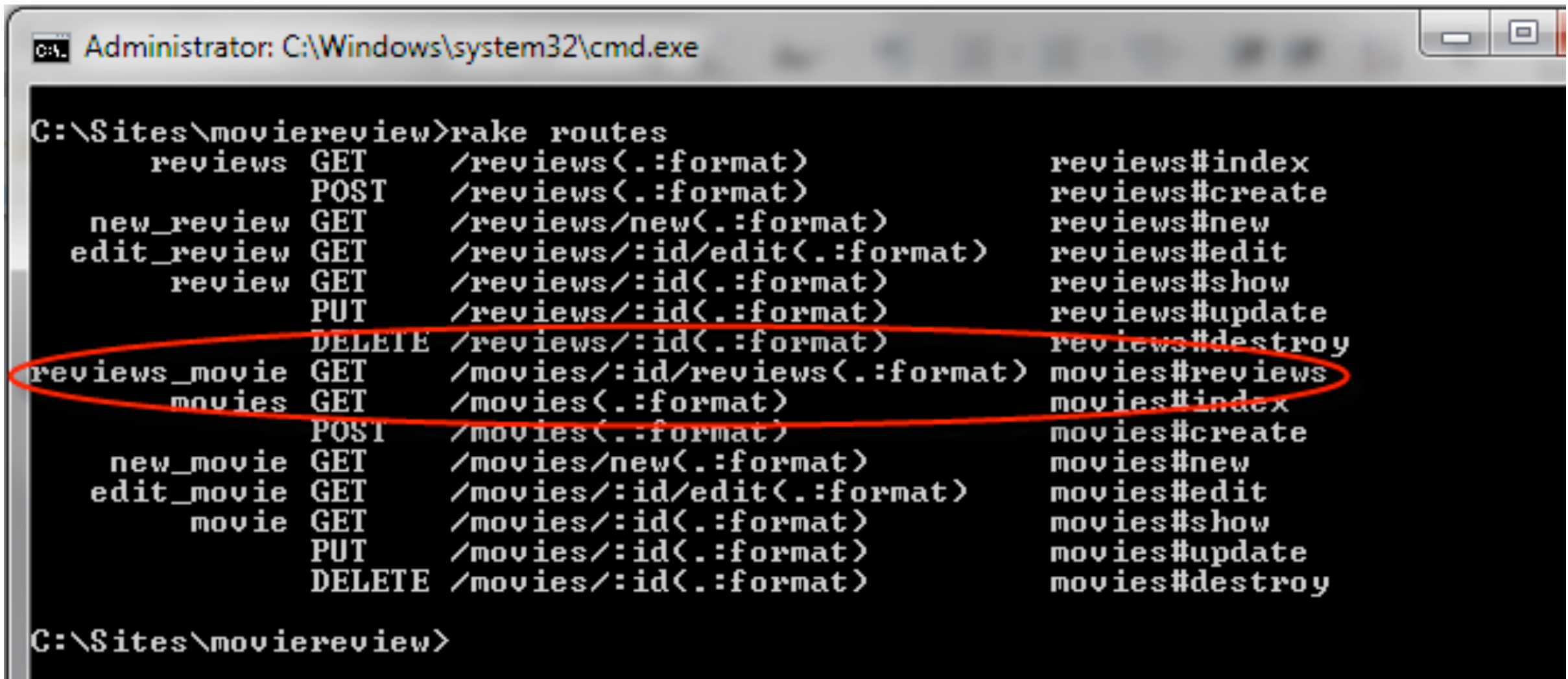
```
# comment out existing resources routing
# resources :movies

# add member to extend :movies resources routing
resources :movies do
  member do
    get 'reviews'
  end
end
```

The new route will use a reviews action in movies_controller.rb

How do I create a route to display a list of related records?

The new route will use a reviews action in movies_controller.rb



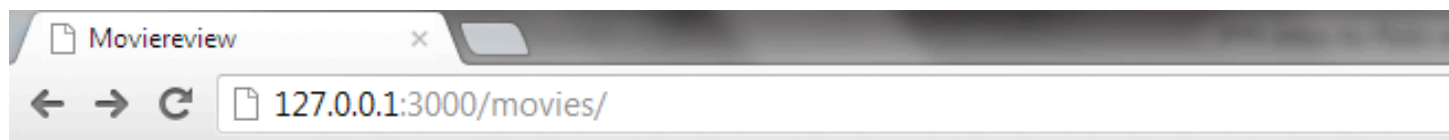
```
Administrator: C:\Windows\system32\cmd.exe

C:\Sites\moviereview>rake routes
      reviews GET    /reviews(.:format)      reviews#index
      reviews POST   /reviews(.:format)      reviews#create
  new_review GET    /reviews/new(.:format)  reviews#new
edit_review GET    /reviews/:id/edit(.:format) reviews#edit
   review GET    /reviews/:id(.:format)  reviews#show
      review PUT    /reviews/:id(.:format)  reviews#update
      review DELETE /reviews/:id(.:format)  reviews#destroy
reviews_movie GET    /movies/:id/reviews(.:format) movies#reviews
      movies GET    /movies(.:format)       movies#index
      movies POST   /movies(.:format)       movies#create
  new_movie GET    /movies/new(.:format)   movies#new
edit_movie GET    /movies/:id/edit(.:format) movies#edit
   movie GET    /movies/:id(.:format)   movies#show
      movie PUT    /movies/:id(.:format)   movies#update
      movie DELETE /movies/:id(.:format)   movies#destroy

C:\Sites\moviereview>
```

How do I create a link to this new route?

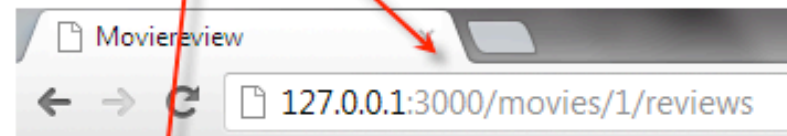
```
<%= link_to movie.reviews.count,  
             reviews_movie_path(movie) %>
```



Listing movies

Title	Synopsis	Release	Reviews	
Battleship Potemkin	Russian ship riot.	1924-12-24	2	Show Edit Destroy
Beau Brummel	Broken hearts parade.	1913-02-19	1	Show Edit Destroy
Voyage to the Moon	Rockets away!	1902-10-04	1	Show Edit Destroy

[New Movie](#) [Reviews](#)



Listing reviews

Author	Review	Score	Movie	
Fred	Rave!	10	1	Show Edit Destroy
Ginger	Pan!	2	1	Show Edit Destroy

[New Review](#)

How do I implement an action to gather and display a filtered review list?

A new reviews action in `movies_controller.rb` can

- find the movie specified by the `id` passed in the URI
- retrieve this movie's reviews collection for use in the view
- render these results using the `reviews/index` view

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

  respond_to do |format|
    format.html { render 'reviews/index' }
    format.json { render json: @reviews }
  end
end
```

Exercise:

**Drilling down from
a summary to a detail view**

How can I centrally manage site wide navigation?

Add a `link_to` for each model in a new `app/views/_navigation.html.erb` partial:

```
<p>
  <%= link_to 'Movies', movies_path %>
  <%= link_to 'Reviews', reviews_path %>
  <%= link_to 'Festivals', festivals_path %>
</p>
```

How can I centrally manage site wide navigation?

Render this partial in the overall application layout in `app/views/layouts/application.html.erb`

```
...  
<body>  
  <%= render 'navigation' %>  
  <%= yield %>  
...
```

Exercise:
Adding site-wide
navigation

How do i create a migration and what does this mean?

Migrations are classes enabling you to modify your database in staged, controllable, and reversible ways.

When Rails generates a migration, a class is created and registered in an internal database to track its use.

How do i create a migration and what does this mean?

```
rails generate migration CreateFestivalsMovies
```

This creates db/migrate/[timestamp]_create_festivals_movies.rb

```
class CreateFestivalsMovies < ActiveRecord::Migration
  def up
  end

  def down
  end
end
```

What is the basic anatomy of a migration?

Migrations may expose three basic methods:

up	executes when the migration is run
down	executes when the migration is rolled back
change	executes both when the migration is run and (in reverse) when rolled back

What is the basic anatomy of a migration?

Databases can be modified in numerous ways within a migration

add_column	add_index	change_column
change_table	create_table	drop_table
remove_column	remove_index	rename_column

Migrations support many data types, and translate them for the underlying database

:binary	:boolean	:date
:datetime	:decimal	:float
:integer	:primary_key	:string
:text	:time	:timestamp

How might an up/down migration work?

```
class CreateFestivalsMovies < ActiveRecord::Migration
  def up
    create_table :festivals_movies, id: false do |t|
      t.integer :festival_id
      t.integer :movie_id
    end
    add_index :festivals_movies,
      [:festival_id, :movie_id], unique: true
  end

  def down
    drop_table :festivals_movies
  end
end
```

What are some options for running migrations?

The Rails rake tool enables several ways for working with migrations

`rake db:migrate`
call the up method of all migrations not yet run, in date order

`rake db:migrate VERSION=[timestamp]`
run all migrations forward or back to this one

`rake db:migrate:up VERSION=[timestamp]`
run the up method of this particular migration

`rake db:migrate:down VERSION=[timestamp]`
run the down method of this particular migration

What are some options for running migrations?

`rake db:rollback`
run the down method of the most recent migration

`rake db:rollback STEP=[number]`
rolls back this number of migrations from the current one

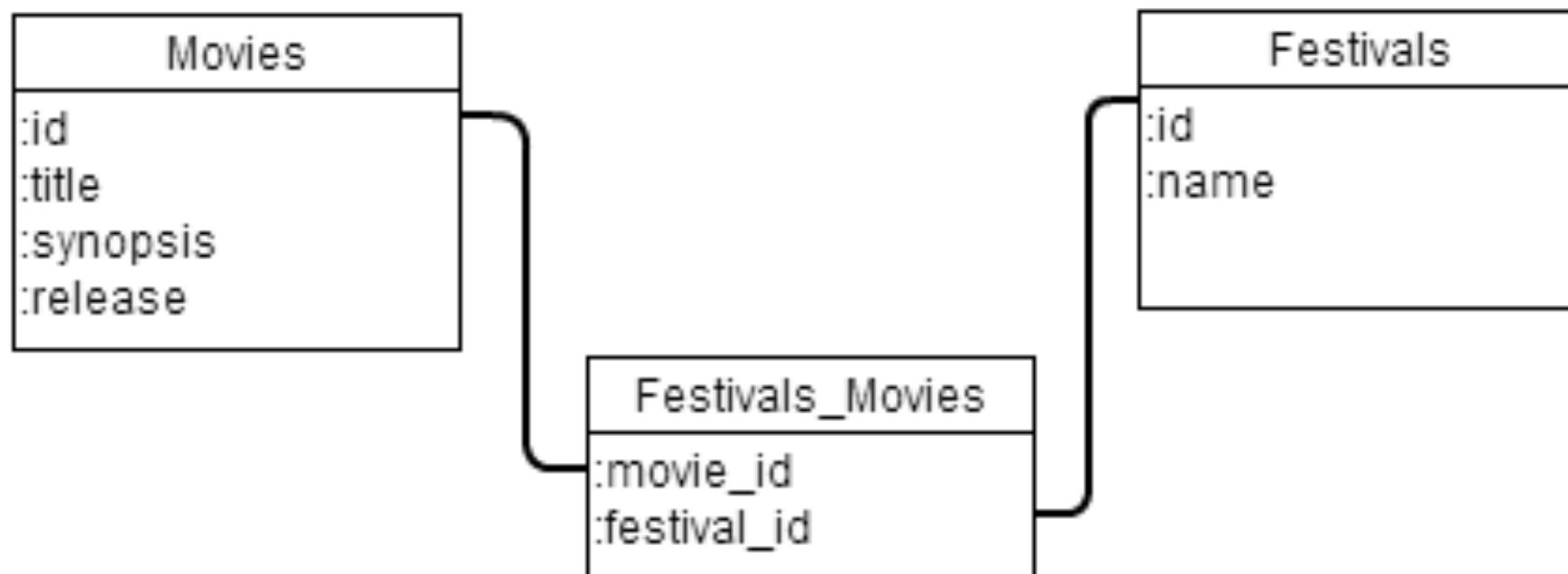
`rake db:migrate:redo STEP=[number]`
roll back then re-run this number of migrations

`rake db:reset`
drop the current database and recreate its schema

How do I create many to many relationships between tables?

Each record in a table is uniquely identified by its primary key (usually called id)

This relationship requires a junction (aka cross-reference) table



How do I create a junction table?

Supporting a many to many relationship in Rails requires creating a junction table

```
class CreateFestivalsMovies < ActiveRecord::Migration

  def up
    create_table :festivals_movies, id: false do |t|
      t.integer :festival_id
      t.integer :movie_id
    end

    add_index :festivals_movies, [:festival_id, :movie_id], unique: true
  end

  def down
    drop_table :festivals_movies
  end

end
```

How do I implement this association in my models?

Rails supports has and belongs to many (HABTM for short) through two similar commands and approaches

`has_and_belongs_to_many :festivals`

The `has_and_belongs_to_many` approach expects a plain junction table with no added custom data

`has_many :festivals, through: :festivals_movies`

The `has_many :through` approach supports a junction table with added custom data about the relationship

Exercise:
Enabling many to many
associations

Project

Project

Your project can about anything, but it must include:

- A user sytem that includes:
 - sign up
 - log in
 - encrypted password
 - a username
 - an avatar
- A one to many association *e.g. user has many tweets, blog posts*
- URLs are only available when logged in *e.g. profile, add new tweet*
- URLs with different content depending on user *e.g. tweet stream, reviews page*