Week 5 Lecture 14

Applied

Helpful Resources

- http://api.rubyonrails.org/classes/ActiveRecord/Base.html
- http://rubyonrails.org/screencasts/rails3/active-relation-active-model
- https://github.com/brynary/arel
- http://railscasts.com/episodes/215-advanced-queries-in-rails-3
- http://api.rubyonrails.org/files/activerecord/README_rdoc.html

What's in this lecture?

- Rails Models and their relationships
- ActiveRecord

The Basics

- Model relationships flow naturally:
  - Photo **has one** PhotoType
  - User **belongs to** the Admin group
  - Account **has many** transactions

Rails and Relationships

- Rails ships with an Object Relational Mapper (ORM) called ActiveRecord
- Other ORMs can be used (DataMapper)
- an ORM allows you to map your business objects to database tables

Rails and Relationships

- Creating relationships require two modifications:
  - In the database structure (schema)
  - In the models themselves

What kind of modifications?

Structure

- Within migrations:
  - Ensure that tables don't have column name conflicts
  - Create proper foreign/primary key layout

Models

- Most database-backed models in Rails inherit from ActiveRecord::Base
- This inheritance allows your model to be flexibly defined by your database table
- Gives special access to methods:
  - has_one
  - belongs_to
  - has_many

What it looks like:

Models

```
class PhotoFamily < ActiveRecord::Base

  set_primary_key :photo_family_id

  has_many :attribs
  has_many :nodes, :through => :attribs
  has_many :pictures
```

has_one

- Defines a singular relationship from parent model to child:

```
has_one :note_book
```

- Used when needing:
  - possession
  - singular hierarchy

has_one

```
class Writer < ActiveRecord::Base
  has_one :note_book
end
```

```
class NoteBook < ActiveRecord::Base
  belongs_to :writer
end
```

belongs_to

- Used in correspondence with the has_* methods
- Designates flow of the model association
- Implemented on the possessed model

has_many

- Defines a multiple hierarchical relationship

```
has_many :books
```

- Important:
  - Pluralization needed for has_many
  - Singular needed for has_one

has_many

```
class BookShelf < ActiveRecord::Base
  has_many :books
end
```

```
class Book < ActiveRecord::Base
  belongs_to :book_shelf
end
```

has_and_belongs_to_many

- Typically used with:
  - roles
  - tags
- Example:
  You want a user to belong to many different groups, and groups can have many different users

has_many :through

- For complex has_and_belongs_to_many
- We want to relate A and C through B
- Example:
  - teacher has many students through classes
  - When her classes change, her students change

Quiz

- We have an app where users can have friends who are other users. What should the table setup look like?

Using ARel

- ARel allows for chaining to create:

  Node.where(:hidden=>false).limit(10)

- => "SELECT `nodes`.* FROM `nodes` WHERE `nodes`.`hidden` = 0 LIMIT 10"

Words of Wisdom

- If you find yourself with mostly many-to-many relationships, your database structure probably needs rethinking.

Exercises

- Take your most mature Rails application and write out database diagram of relationships
- Given the task of creating a banking app with Accounts, Owners, and Transactions: write out the schema for the app's database