Rails: Models

Lecture 18
Rails Overview
Architecture: Desktop App

- User Interface
- Application
- Data

Graphical events (mouse moves, button pushed)
Processing, Calculating
Persistence, Transactions, Triggers
Model-View-Controller Pattern

- **Model**
  - The data (*i.e.* state)
  - Methods for accessing and modifying state

- **View**
  - Renders contents of model for user
  - When model changes, view must be updated

- **Controller**
  - Translates user actions (*i.e.* interactions with view) into operations on the model
  - Example user actions: button clicks, menu selections
Basic Interactions in MVC

- **Input**: "user action"
- **Controller**: "change data" → "new state" → "change display"
- **View**: Output
- **Model**: Input
Basic Web App Skeleton: 3-Tier

User Interface

http

HTML, CSS, Javascript

Application

SQL

Data

http, HTML, CSS, Javascript

php, Java

Apache Tomcat, IIS7

MySQL, Microsoft SQL Server, SQLite
MVC in a Basic Web Application

- **Model**
  - Database (table with rows)
  - Classes that wrap database operations (class with instances)

- **View**
  - HTML (+ CSS, JavaScript) files rendered by client's browser
  - Skeleton files used by server to generate these HTML files

- **Controller**
  - Receives HTTP requests via web server
  - Orchestrates activity (model and view)
MVC with Rails
MVC with Rails
Directory Structure of Rails

depot/
   ...../app
      ............/controllers
      ............/helpers
      ............/models
      ............/views
         .............../layouts
   ...../components
   ...../config
   ...../db
   ...../doc
   ...../lib
   ...../log
   ...../public
   ...../script
   ...../test
   ...../tmp
   ...../vendor
       ....../README
       ....../Rakefile
"Convention Over Configuration"

- Use naming & location conventions to wire components together *implicitly*
- Explicit routing too, based on *names* and pattern matching
- Contrast with:
  - Configuration files (e.g., XML)
  - Configuration code (e.g., Swing register listener)
  - Configuration tools (e.g., IDEs to connect GUI widgets to code snippets)
Wiring Parts Together in Rails

- **Example: Event → Controller wiring**
  - HTTP GET request for URL `/say/hello` gets routed to controller:
    - Class called `SayController`
    - File `say_controller.rb` in `app/controllers`
    - Method `hello`

- **Example: Controller → View wiring**
  - HTTP response formed from:
    - File `app/views/say/hello.html.erb`

- **Example: Model → Database wiring**
  - Class `Order` maps to database table "orders"
  - Attributes of `Order` map to `columns` of table
  - Instances of `Order` map to a `rows` of table
Models in Rails Architecture

Ruby on Rails
Web Applications

Browser or client

Requests

Web Server

Forwards

Dispatcher

Loads

Controller

CRUDs
Responds

Active Record

Queries
Data or Errors

Database

MySQL, PostgreSQL or Oracle

Invokes FastCGI, mod_ruby or CGI processor

Apache, WeBrick or Lighttpd

HTTP, RSS, ATOM or SOAP

XHTML, CSS, JS & images, XML

XML response

Responds
Displays

Action View

Renders

Action WebServices

Delegates
Delivers

ActionMailer
Models in Rails Architecture
Mapping Tables to Objects

- General strategy for OO languages
  - Table in database -- a class
  - Table columns -- attributes of the class
  - Table rows -- instances of class (objects)

- Application works with database using ordinary language syntax
  - Class methods for finding row(s) in table

- Example: Java POJOs, Rails models
Database Tables

- A database is a collection of tables
  - Naming convention: Table names plural
- Each table has a list of columns
- Each column has a name and a type
- A table has a list of rows

<table>
<thead>
<tr>
<th>students</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>fname</strong> (string)</td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td>Marco</td>
</tr>
<tr>
<td>Primo</td>
</tr>
<tr>
<td>Cher</td>
</tr>
</tbody>
</table>
Models

☐ Programmatic way for application to interact with database
  ■ Model = a Ruby class
  ■ Extend ApplicationRecord
  ■ Found in app/models

☐ Each class corresponds to a table
  ■ Note: Models are singular (tables are plural)
  ■ Includes attributes corresponding to columns implicitly

```ruby
class Post < ApplicationRecord
  # attr_accessible :name,:title,:content
end
```
Class Methods for Models

- Create a new instance with `new`
  
  ```ruby
  p1 = Post.new
  p2 = Post.new author: 'Xi', title: 'Hola'
  ```

  **Warning:** this only creates the model (object) it does *not* modify the database.

- Create instance *and* add it to database
  
  ```ruby
  p3 = Post.create author: 'Zippy'
  ```

- Retrieve particular row(s) from table
  
  ```ruby
  p = Post.find 4  # search by id
  p = Post.find_by author: 'Xi'
  s = Student.find_by buckid: 543333
  blog = Post.all
  post = Post.first
  post = Post.last
  ```
Instance Methods for Models

☐ To save a model (object) as a row in the database

\[ p = \text{Post}.\text{new} \text{ author: 'Xi'} \]
\[ p.\text{save} \# \text{commits change to database} \]

☐ Read/write attributes like an ordinary Ruby class

\[ p = \text{Post}.\text{find_by} \text{ author: 'Xi'} \]
\[ p.\text{title} \#=> \text{nil} \]
\[ p.\text{title} = \text{'A Successful Project'} \]
\[ p.\text{save} \# \text{don't forget to save!} \]

☐ To delete a row from the table

\[ p.\text{destroy} \# \text{no save needed} \]
Directory Structure of Rails

depot/
   ....../app
      ........../controllers
      ........../helpers
      ........../models
      ........../views
      .............../layouts
   ....../config
   ....../db
   ....../lib
   ....../log
   ....../public
   ....../storage
   ....../test
   ....../tmp
   ....../vendor
      ....Gemfile
      ....package.json
      ....README.md
      ....Rakefile
A Bit of Configuration

- Which database to use?
  - SQLite is the easiest (no setup!)
  - MySQL has better performance
  - PostgreSQL favored for Heroku deployment

- Different environments: development, test, production
  - Default (for rake command) is development

- See config/database.yml
  ```yaml
  default: &default
      adapter: sqlite3
      pool: <%= ENV.fetch("RAILS_MAX_THREADS") {5} %>
      timeout: 5000
  
  development:
    <<: *default
    database: db/development.sqlite3
  ```
# Database Column Types

<table>
<thead>
<tr>
<th></th>
<th>SQLite</th>
<th>Postgresql</th>
<th>MySQL</th>
</tr>
</thead>
<tbody>
<tr>
<td>blob</td>
<td>blob</td>
<td>bytea</td>
<td>blob</td>
</tr>
<tr>
<td>boolean</td>
<td>boolean</td>
<td>tinyint(1)</td>
<td>tinyint</td>
</tr>
<tr>
<td>date</td>
<td>date</td>
<td>date</td>
<td>date</td>
</tr>
<tr>
<td>datetime</td>
<td>timestamp</td>
<td>datetime</td>
<td></td>
</tr>
<tr>
<td>decimal</td>
<td>decimal</td>
<td>decimal</td>
<td></td>
</tr>
<tr>
<td>float</td>
<td>float</td>
<td>float</td>
<td></td>
</tr>
<tr>
<td>integer</td>
<td>integer</td>
<td>int(11)</td>
<td></td>
</tr>
<tr>
<td>varchar(255)</td>
<td>varchar</td>
<td>varchar</td>
<td>varchar</td>
</tr>
<tr>
<td>text</td>
<td>text</td>
<td>text</td>
<td></td>
</tr>
<tr>
<td>datetime</td>
<td>time</td>
<td>time</td>
<td></td>
</tr>
<tr>
<td>datetime</td>
<td>timestamp</td>
<td>datetime</td>
<td></td>
</tr>
</tbody>
</table>
Table Constraints

- Invariants on table entries beyond type information
  - “lname is not null”
  - “buckid is unique”

- Often useful to have a unique identifier for each row (a primary key)
  - Easy: Include an extra (integer) column
  - Database responsible for assigning this value every time a row is added
  - No way to change this value after creation
## Primary Key With Autoincrement

### students

<table>
<thead>
<tr>
<th>id (key)</th>
<th>fname (string)</th>
<th>lname (string)</th>
<th>buckid (integer)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Marco</td>
<td>Pantani</td>
<td>22352022</td>
</tr>
<tr>
<td>3</td>
<td>Primo</td>
<td>Carnera</td>
<td>334432</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>Cher</td>
<td>34822039</td>
</tr>
</tbody>
</table>
Linking Tables

- Different tables can be related to each other
  - “Each student has exactly 1 major”
  - “Each student can own 1 (or more) vehicles”
- Keys are used to encode this relationship
  - Include a column in table X containing keys from table Y (foreign keys)
  - For examples:
    - Students table includes a column identifying a student's major
    - Vehicles table includes a column identifying a (student) owner
- Association is an invariant between tables
# Association: Students & Vehicles

## Students

<table>
<thead>
<tr>
<th>id (key)</th>
<th>fname (string)</th>
<th>lname (string)</th>
<th>buckid (integer)</th>
<th>major (foreign key)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Marco</td>
<td>Pantani</td>
<td>22352022</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>Primo</td>
<td>Carnera</td>
<td>334432</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>Cher</td>
<td></td>
<td>34822039</td>
<td>3</td>
</tr>
</tbody>
</table>

## Vehicles

<table>
<thead>
<tr>
<th>id (key)</th>
<th>owner (foreign key)</th>
<th>license (string)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>K3F 443L</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>F8L 220J</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>GOHBUX</td>
</tr>
</tbody>
</table>
### Associations

<table>
<thead>
<tr>
<th>vehicles</th>
<th>students</th>
<th>programs</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>id (key)</strong></td>
<td><strong>owner (for. key)</strong></td>
<td><strong>id (key)</strong></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

**Diagram:**
- The diagram illustrates the relationships between vehicles, students, and programs.
- Each vehicle has an owner, and each student has a major.
- Programs are linked to students through major associations.
Schema

- Definition of table structure
  - Table name
  - Column names and types
  - Constraints

- Usually database manager-specific

- See db/schema.rb for Ruby-based schema description
  - Allows independence from particular DB manager
  - Schema is versioned by timestamp (really by migration...)

Example schema.rb

ActiveRecord::Schema.define(version: 2020_03_19_144259) do

  create_table "students", force: :cascade do |t|
    t.string  "fname"
    t.string  "lname"
    t.integer "buckid"
    t.datetime "created_at", null: false
    t.datetime "updated_at", null: false
  end

end
Q. Who writes schema.rb?

A. It is generated!

Golden rule: Never edit schema.rb directly

Instead, write a migration

A migration is Ruby code (a class) that represents a change in schema

- Create new tables (including column names and column types)
- Modify existing tables (adding/removing columns, or changing associations)
- Delete (“drop”) existing tables
Migration Classes

- See `db/migrate`
- Filename consists of
  - Timestamp (UTC) of creation
  - Class name (descriptive of delta)
  - Example: class `CreatePosts` in `20200319145307_create_posts.rb`

- Consequence: Migrations are run in a consistent order
  - Deltas do not commute, so order is important
- Class extends `ActiveRecord::Migration`
  - Contains method change
  - This method invoked by `rails db:migrate`
Example Migration Class

class CreatePosts < ActiveRecord::Migration
  def change
    create_table :posts do |t|
      t.string :name
      t.string :title
      t.text :content

      t.timestamps
    end
  end
end
# Result of Running This Migration

<table>
<thead>
<tr>
<th>:posts</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>:id (key)</td>
<td>:name (string)</td>
<td>:title (string)</td>
<td>:content (text)</td>
<td>:created_at (datetime)</td>
</tr>
</tbody>
</table>
Column Type Mappings

<table>
<thead>
<tr>
<th>Migration</th>
<th>Ruby</th>
<th>SQLite</th>
<th>Postgres</th>
<th>MySQL</th>
</tr>
</thead>
<tbody>
<tr>
<td>:binary</td>
<td>String</td>
<td>blob</td>
<td>bytea</td>
<td>blob</td>
</tr>
<tr>
<td>:boolean</td>
<td>Boolean</td>
<td>boolean</td>
<td>boolean</td>
<td>tinyint(1)</td>
</tr>
<tr>
<td>:date</td>
<td>Date</td>
<td>date</td>
<td>date</td>
<td>date</td>
</tr>
<tr>
<td>:datetime</td>
<td>Time</td>
<td>datetime</td>
<td>timestamp</td>
<td>datetime</td>
</tr>
<tr>
<td>:decimal</td>
<td>BigDecimal</td>
<td>decimal</td>
<td>decimal</td>
<td>decimal</td>
</tr>
<tr>
<td>:float</td>
<td>Float</td>
<td>float</td>
<td>float</td>
<td>float</td>
</tr>
<tr>
<td>:integer</td>
<td>Integer</td>
<td>integer</td>
<td>integer</td>
<td>int(11)</td>
</tr>
<tr>
<td>:string</td>
<td>String</td>
<td>varchar(255)</td>
<td>character varying</td>
<td>varchar(255)</td>
</tr>
<tr>
<td>:text</td>
<td>String</td>
<td>text</td>
<td>text</td>
<td>text</td>
</tr>
<tr>
<td>:time</td>
<td>Time</td>
<td>datetime</td>
<td>time</td>
<td>time</td>
</tr>
<tr>
<td>:timestamp</td>
<td>Time</td>
<td>datetime</td>
<td>timestamp</td>
<td>datetime</td>
</tr>
</tbody>
</table>
Schema Deltas In Migrations

- In addition to creating tables, the change method can also change existing tables
  - Modify columns of an existing table
    - `add_column`, `remove_column`, `rename_column`, `change_column`
  - Modify and delete tables
    - `change_table`, `drop_table`

- Example: `xxx_add_author_to_posts.rb`

```ruby
class AddAuthorToPosts < ActiveRecord::Migration
  def change
    add_column :posts, :author, :string
  end
end
```
Migrations as History

- Change defined by migration can be undone
  - Migrations give a *linear* history of deltas
  - Schema is the result of applying them (in order)
- Can move forward/backward in history
  - Create database only (no schema) defined in config/database.yml
    $ rails db:create
  - Update schema.rb (compare its version number to list of migrations) and apply to database
    $ rails db:migrate
  - Rollback schema.rb to earlier point in history
    $ rails db:rollback
  - Load schema defined in db/schema.rb
    $ rails db:schema:load
Schemas, Migrations, Models

- `schema.rb`
- `migrations`
- `models`
- `database.yml`
- `db:create`
- `db:migrate`
- `db:schema:load`
- `db:schema:dump`
Migrations vs Schema

- **Golden rule: Never edit schema.rb**
  - It is regenerated every time you do a migration
  - *Every* change in schema means writing a migration

- **Commit schema.rb to version control**
  - Deployment in fresh environment means loading schema, not reliving the full migration history

- **Commit migrations to version control**
  - Once a migration has been shared, to undo it you should create a *new* migration (preserve the linear history)
Summary

- **Databases: Tables, columns, rows**
  - Structure defined in a schema
  - Rails uses Ruby code to generate schema

- **Models**
  - Ruby classes that mirror database tables
  - Class names from table (singular vs plural)
  - Attributes from columns

- **Migrations**
  - Ruby code describing change to schema
  - Syntax look declarative
Rails: Associations and Validation
Schemas, Migrations, Models

- `schema.rb`
- `migrations`
- `models`
- `database.yml`

- `db:create`
- `db:schema:load`
- `db:migrate`
Recall: Migrations

class CreatePosts < ActiveRecord::Migration
  def change
    create_table :posts do |t|
      t.string :name
      t.string :title
      t.text :content
      t.timestamps
    end
  end
end
class Post < ApplicationRecord
  # attr_accessible :name, :title, :content
end
Generating Code: rails generate

- Notice: Two blobs of Ruby code need to be in sync
  - Migration (creates table and columns)
    - `db/migrate/xxx_create_students.rb`
  - Model (with matching name)
    - `app/models/student.rb`

- Easier: Generate *both* simultaneously
  - `$ rails generate model Student
    fname:string lname:string buckid:integer`
  - Use model name (singular) and attributes
  - Note: this does *not* generate the schema.rb (use rails)

- Migrations for table edits can also be generated
  - `$ rails generate migration AddNickNameToStudent
    nick:string`
  - Name is meaningful! (starts with add or remove)
  - Creates a migration that changes students table
Result of `generate model`

class CreateStudents < ActiveRecord::Migration
  def change
    create_table :students do |t|
      t.string :fname
      t.string :lname
      t.integer :buckid

      t.timestamps
    end
  end
end

class Student < ApplicationRecord
end
Demo with rails console

$ rails new demo # creates directory
  # no schema, migrations, or models
$ cd demo
$ rails generate model Student
  fname:string lname:string buckid:integer
  # see db/migrate, app/models
$ rails console
> Student.methods # lots available!
> Student.all    # error, no table
> s = Student.new # will this work?
Demo with rails console

$ rails new demo # creates directory
  # no schema, migrations, or models
$ cd demo
$ rails generate model Student
  fname:string lname:string buckid:integer
$ rails console
> Student.methods # lots available!
> Student.find :all # error, no table
> s = Student.new # error, no table
$ rails db:migrate # creates schema.rb
$ rails console
> Student.all #=> []
Working With Models

> s = Student.new

> s2 = Student.new fname: "Jo"

> s3 = Student.new fname: "Xi",
   buckid: 23

> Student.all #=> ?
Working With Models

> s = Student.new
> s2 = Student.new fname: "Jo"
> s3 = Student.new fname: "Xi", buckid: 23
> Student.all #=> [] still
> s.save
> Student.all #=> [<id: 1, ...>]
> s.fname = "Mary"
> s.save
Seeding the Database

- Quickly populate using `config/seeds.rb`
  - `$ rails db:seed`  # run `seeds.rb`
  - `$ rails db:reset`  # drop then reseed

- In `config/seeds.rb`:
  ```ruby
  30.times do
    Student.create!
      buckid: Faker::Number.unique
            .number(digits: 9),
      fname: Faker::Name.first_name,
      lname: Faker::Name.last_name
  end
  ```

- Useful gem: Faker
  - Add to Gemfile: `gem 'faker'`
  - `$ bundle install`
## Associations (1:N Relationship)

### students

<table>
<thead>
<tr>
<th>id (key)</th>
<th>buckid (integer)</th>
<th>team_id (foreign key)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>22352022</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>334432</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>34822039</td>
<td>6</td>
</tr>
</tbody>
</table>

### teams

<table>
<thead>
<tr>
<th>id (key)</th>
<th>name (string)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Wicked Wicky</td>
</tr>
<tr>
<td>2</td>
<td>The Happy Crew</td>
</tr>
<tr>
<td>6</td>
<td>No Names</td>
</tr>
</tbody>
</table>
Invariants

- A student belongs to exactly 1 team
  - Weaker: A student belongs to at most 1 team

- Same representation for either invariant
  - A column (of foreign keys) in students table

- Maintaining stronger invariant
  - Students can only be added with team_id set to something valid
  - Deleting a team deletes member students!

- Maintaining weaker invariant
  - Students can be added with null team_id
  - Deleting a team null-ifies members' team_id
Rails Migration and Models

class AddTeamForeignKeys < ActiveRecord::Migration
  def change
    add_reference :students, :team,
                  index: true  # for quick load
  end
end

class Student < ApplicationRecord
  belongs_to :team  # note singular form
                  # have Student#team method
end

class Team < ApplicationRecord
  has_many :students  # note plural form
                    # have Team#students method
end
Association Methods

- **Belongs_to** creates method for accessing owner
  
s = Student.find 1 #=> 22352022
  s.team #=> 'The Happy Crew'
  s.team.name = 'The(tm) Happy Crew'

- **Has_many** creates method for accessing members
  
t = Team.find 1
  t.students #=> array of students
  t.students.first
  t.students.size
  t.students.destroy_all
  t.students.any? { |s| ... }
Asymmetry in Writes to Assoc.

- Add a student to a team’s association: Student is automatically saved (assuming team is stored in database)
  \[
  t = \text{Team}.\text{find 1}
  \]
  \[
  t.\text{students} \Rightarrow []
  \]
  \[
  t.\text{students} \ll \text{Student}.\text{new} \# \text{ gets an id}
  \]
  \[
  t.\text{students} \Rightarrow [\#<\text{Student id: 1, ...}>]
  \]

- Assign a team student’s association: Student is *not* automatically saved
  \[
  s = \text{Student}.\text{find 1}
  \]
  \[
  s.\text{team} = \text{my_team}
  \]
  \[
  s.\text{reload} \Rightarrow s's\text{ team is unchanged!}
  \]
Modifiers for `belongs_to`

class Student < ApplicationRecord

  belongs_to :greek_house,
  optional: true
  # allows foreign key to be null

  belongs_to :project_group,
  class_name: 'Team'
  # default is Project_Group

  belongs_to :major,
  foreign_key: 'OSU_code'
  # default is major_id

  belongs_to :team,
  touch: :membership_updated

end
Modifiers for has_many

class Team < ApplicationRecord
  has_many :students,
    limit: 5,
    # max number of members
  dependent: :destroy,
    # what happens to dependents
  # when parent is destroyed?
  class_name: 'OSUStudent'
    # default is Student
end
More Relationships

- **1:1 (one-to-one)**
  - Use `belongs_to` with `has_one`
    - `has_one` is just `has_many` with limit of 1
  - Same asymmetry in writing exists

- **N:M (many-to-many)**
  - A third, intermediary table is used with 2 columns (for foreign keys from two tables)
  - In rails, use `has_many :through` association
Validations

- An invariant on data in a single table
  - Every student has a (non-null) buckid
  - Buckids are unique
  - Team names are less than 30 characters
  - Usernames match a given regular expression

- To maintain invariant:
  - Must be true initially
  - Must be satisfied by each insertion

- These validations are in the model
  - A model instance can be checked
  - Invalid objects can not be saved

```ruby
student = Student.new lname: 'Vee'
student.valid? #=> false (no buckid)
student.save #=> false
```
Example

class Post < ApplicationRecord

  validates :name, presence: true
  validates :title, presence: true, length: { minimum: 5, maximum: 50 }

end
Rails Implementation

- Model object has an `errors` attribute
  - This attribute is a hash (of problems)
- Failing a validity check adds an item to the `errors` hash
  - Empty hash corresponds to valid object
  - Each attribute is a key in the `errors` hash (plus there is a general key, `:base`)
    ```ruby
    s.errors[:buckid] = "is not a number"
    ```
- The `valid?` method does the following:
  - Empties `errors` hash
  - Runs validations
  - Returns `errors.empty?`
Validates Method in Model

validates :column, condition

- Uniqueness
  uniqueness: true
  uniqueness: {message: 'Username already taken'}

- Non-nullness (not the same as truth, see next)
  presence: {message: 'Title needed'}

- Truth of a boolean field
  acceptance: {message: 'Accept the terms'}

- Matching a regular expression
  format: {with: /[A-Z].*/, message: ...}
  format: /[A-Za-z0-9]+/

- Being a number
  numericality: {only_integer: true}

- Having a length
  length: {minimum: 5}
Alternative: Declarative Style

- Special methods for each flavor of validation

```ruby
validates_uniqueness_of :username
validates_presence_of :password
validates_acceptance_of :terms
validates_format_of :name, with: /[A-Z].*/
validates_numericality_of :buckid, only_integer: true
```
Summary

- **Code generation**
  - Database schema generated by schema.rb
  - Schema.rb generated by rails on migrations
  - Migrations and models can be generated by rails

- **Associations**
  - 1:N (or 1:1) relationships via foreign keys
  - Rails methods `belongs_to`, `has_many`
  - Create association attributes, which can be read and written
  - Asymmetry in writing owner vs member

- **Validations**
  - Invariants checked before saving
  - Errors hash contains list of problems
  - Declarative style for common case checks
  - Custom validity checkers possible too