Rails:
Models

Lecture 28
Rails Overview

Ruby on Rails
Web Applications

Browser or client

Requests → Web Server

HTTP, RSS, ATOM or SOAP

Web Server

Forwards → Dispatcher

Apache, WeBrick or Lighttpd

Invokes FastCGI, mod_ruby or CGI processor

Dispatcher

Loads → Controller

Handles some validations

Controller

CRUDs → Active Record

Responds

Active Record

Queries → Database

Data or Errors

Database

MySQL, PostgreSQL or Oracle

Action Mailer

Delegates

Delivers

Action WebServices

Renders

Action View

Responds

Displays

XML response

XHTML, CSS, JS & images, XML

XML response
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

Controller

"change data"

"new state"

"change display"

Model

View

"user action"

Output

"change data"
Basic Web App Skeleton: 3-Tier

User Interface

http

HTML, CSS, Javascript

Application

http

php

Apache Tomcat

SQL

Data

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
- Web Server
- Dispatcher
- Controller
- Active Record
- Database

Requests → Forwards → Loads → Redirects

- HTTP, RSS, ATOM or SOAP
- HTML, CSS, JS & images, XML
- Apache, WeBrick or Lighttpd
- Invokes FastCGI, mod_ruby or CGI processor
- Handles some validations

- XML response
- Action View
- Action WebServices
- Action Mailer

CRUDs

Responds

Queries

Data or Errors

MySQL, PostgreSQL or Oracle
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>name</th>
<th>Iname</th>
<th>buckid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Marco</td>
<td>Pantani</td>
<td>22352022</td>
</tr>
<tr>
<td>Primo</td>
<td>Carnera</td>
<td>334432</td>
</tr>
<tr>
<td></td>
<td>Cher</td>
<td>34822039</td>
</tr>
</tbody>
</table>
Models

- Programmatic way for application to interact with database
  - Model = a Ruby class
  - Extends ApplicationRecord
  - Found in app/models

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

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
  
  ```ruby
  p = Post.new author: 'Xi'
  p.save # commits change to database
  ```

- Read/write attributes like an ordinary Ruby class
  
  ```ruby
  p = Post.find_by author: 'Xi'
  p.title #=> nil
  p.title = 'A Successful Project'
  p.save # don't forget to save!
  ```

- To delete a row from the table
  
  ```ruby
  p.destroy # 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>Column Type</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>boolean</td>
<td>tinyint(1)</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>decimal</td>
</tr>
<tr>
<td>float</td>
<td>float</td>
<td>float</td>
<td>float</td>
</tr>
<tr>
<td>integer</td>
<td>integer</td>
<td>int(11)</td>
<td></td>
</tr>
<tr>
<td>varchar(255)</td>
<td>character varying</td>
<td>varchar(255)</td>
<td></td>
</tr>
<tr>
<td>text</td>
<td>text</td>
<td>text</td>
<td>text</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>id (key)</td>
<td>id (key)</td>
<td>id (key)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

**Key**
- **id**: Unique identifier for each record.
- **owner**: Key for vehicles.
- **major**: Key for students.
- **program**: Key for programs.
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: 2022_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
Migrations

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 `20220319145307_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`
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>:id (key)</th>
<th>:name (string)</th>
<th>:title (string)</th>
<th>:content (text)</th>
<th>:created_at (datetime)</th>
<th>:updated_at (datetime)</th>
</tr>
</thead>
</table>

:posts
# Column Type Mappings

<table>
<thead>
<tr>
<th>Migration</th>
<th>Ruby</th>
<th>SQLite</th>
<th>Postgresql</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</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: schema: load
- db: schema: dump
- db: migrate
- database
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