Rails: Associations and Validation

Lecture 26
Schemas, Migrations, Models

migrations

models

database.yml

database

db:create

db:schema:load

db:migrate

schema.rb
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
Recall: Models

class Post < ActiveRecord::Base
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`

- Single point of control: Generate *both* automatically
  - $ 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 rake)

- Migrations for table edits can also be generated
  - $ rails g migration AddNickNameToStudent
    `nick:string`
  - Name is meaningful! (starts with add or remove)
  - Creates a migration that changes students table
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 < ActiveRecord::Base
end
$ rails new demo #creates directory
$ cd demo
$ rails generate model Student
name:string buckid:integer
$ 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
$ cd demo
$ rails generate model Student
name:string buckid:integer
$ rails console
> Student.methods #lots available!
> Student.find :all #error, no table
> s = Student.new #error, no table
$ rake db:migrate #creates schema.rb
$ rails console
> Student.all #=> []
Working With Models

```ruby
> s = Student.new
> s2 = Student.new name: "Jo"
> s3 = Student.new name: "Xi",
    buckid: 23
> Student.all #=> [] still
> s.save
> Student.all #=>[<id: 1, ...>]
> s.name = "Mary"
> s.save
```
**Associations (1:N Relationship)**

### teams

<table>
<thead>
<tr>
<th>id</th>
<th>name</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>

### 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>
Invariants

- A student belongs to exactly 1 team
  - Weaker: A student belongs to at most 1 team
- Driven by representation: Either invariant is supported by one column in students
- 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_column :students, :team_id, :integer
    add_index :students, :team_id  #for quick load
  end
end

class Student < ActiveRecord::Base
  belongs_to :team  #note singular form
               #creates :team method
end

class Team < ActiveRecord::Base
  has_many :students  #note plural form
end
Association Method

- Belongs_to creates method for accessing owner
  ```ruby
  @student = Student.find 1 #=> 22352022
  @student.team #=> 'Wicked Wicky'
  @student.team.name = 'Evil Wicky'
  ```

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

- Add a student to a team's association: student automatically saved (assuming team is stored in database)
  
  ```ruby
  t = Team.find 1
  t.students #=> []
  t.students << Student.new #gets an id
  t.students #=> [#<Student id: 1, ...>]
  ```

- Set a student's association to a team: student is not automatically saved
  
  ```ruby
  s = Student.find 1
  s.team = my_team
  s.reload #=> s's team is unchanged
  ```
Modifiers for belongs_to

class Student < ActiveRecord::Base
  belongs_to :team,
    :touch => :membership_updated
  belongs_to :project_group,
    :class_name => 'Team'
  belongs_to :major,
    :foreign_key => 'OSU_maj_code'
end
Modifiers for has_many

class Team < ActiveRecord::Base
  has_many :students,
    :limit => 5,
    :dependent => :destroy
end
More Relationships

- **1:1 (one-to-one)**
  - Use `belongs_to` with `has_one`
    - `has_one` is just `has_many` with limit 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

- Invariants on the data in a table
  - Every student has a (non-null) buckid
  - Buckids are unique
  - Team names are less than 30 characters
  - Usernames match a 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
```
Rails Implementation

- Model objects have an errors attribute
  - Errors is a hash
- 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's an overall key, `:base`
  - `s.errors[:buckid] = "is not a number"`
- The `valid?` method does the following:
  - Empties errors hash
  - Runs validations
  - Returns `errors.empty?`
Example

class Post < ActiveRecord::Base

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

end
Validates Method in Model

- **Uniqueness**
  - `:uniqueness => true`
  - `:uniqueness => {:message => 'Username taken'}`

- **Non-nullness (not the same as being true!)**
  - `:presence => {:message => 'Title needed'}`

- **Truth of a boolean field**
  - `:acceptance => {:message => 'Accept the terms'}`

- **Matching a regular expression**
  - `:format => {:with => /.*/, :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 => /.*/
validates_numericality_of :buckid,
  :only_integer => true
```
Summary

- **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