Rails Models + Migrations + Validations

Objectives

By the end of today you should be able to...

- Generate models & migrations in Rails
- Use the rails console
- Add validations to our models

Definitions

ORM

ORM stands for object relational mapping. ORM describes a software system that maps SQL queries and results into objects. **Active Record** is the ORM system that is used in Rails.

Models

Models are Ruby classes. They talk to the database, store and validate data, perform the business logic and otherwise do the heavy lifting.

Migrations

A **migration** is a set of database instructions. Those instructions are Ruby code, which migrates our database from one state to another. Essentially they describe database changes.

Migrations are a way for us to manage the creation and alteration of our database tables in a structured and organized manner.

Each migration is a seperate file, which Rails runs for us when we instruct it. Rails keeps track of what's been run, so changes don't get attempted more than once.

We describe the DB changes using Ruby, and it doesn't matter which DB engine we use - Rails has connectors for each different DB engine we might use, which translates the ruby structure into the appropriate DB commands.

Validations

Validations are used to ensure that only valid data is saved into your database. For example, it may be important to your application to ensure that every user provides a valid email address and mailing address.

Let's get started!

Please type all this by hand so you're not blindly copying & pasting and you remember it better.

- Generate a new Rails project:

```
rails new models_example --database=postgreql --skip-test-unit
```

- This command tells Rails to use `postgresql` and adds the `pg` gem to our `Gemfile`. Otherwise, Rails uses `SQLite3` out of the box.
- Enter your app's directory.

`cd models_example/`

- Create our database.

`rake db:create`

Generate a Model & a Migration

WE DO:

- Create a new model called `User` with `first_name` and `last_name` properties that are strings. Look at the output and try to decifer what files it just made for you.

`rails generate model User first_name:string last_name:string`

- NOTE: if we don't specifiy a data type Rails will assume String by default
- Migrate our database to create the **users** table.

`rake db:migrate`

Familiarize yourself with the Rails Console

WE DO:

To enter, go to terminal and in the root of your rails app type

`rails console` or `rails c`

(This is IRB with your rails app loaded in.)

Inside of your Rails console, create a new User object.

`irb(main):001:0> albert = User.new`

Set the name of the user.

```
irb(main):002:0> albert.first_name = "Albert"
irb(main):002:0> albert.last_name = "Einstein"
```

Save your user to the database.

`irb(main):003:0> albert.save`

Retrieve all of the users in the database and store then im a users variable.

`irb(main):004:0> users = User.all`

Exit the console.

`exit`

Modify the existing DB with another Migration

- Rails gives us some help to generate migration files - we can list the fields and their types in the generate command, and if we name the migration appropriately, Rails even guesses the name of the table.
- by putting Add....To.... Rails knows we are adding these columns to which table, and the migration can be written automatically
- Let's also store a user's age along with their names. Generate a new migration file named `AddAgeToUsers`:

```
rails generate migration AddAgeToUsers age:integer
```

- This will have created a migration file in our `RAILS_ROOT/db/migrate` folder. The purpose of this file is to describe what actions we want to take to move our DB schema from its current state to the new state, and also, what would need to happen to move the migration back to the old state again (should we need to).
- available column types:
  - :binary
  - :boolean
  - :date
  - :datetime
  - :decimal
  - :float
  - :integer
  - :primary_key
  - :string
  - :text
  - :time
  - :timestamp

Run the migration so that the column is added to the table.

```
rake db:migrate
```

We can check that the migration ran successfully.

```
rake db:migrate:status
```

CRUD the users in the Console

Create

- `user = User.create(first_name: "Abraham", last_name: "Lincoln")`
- `user = User.create(first_name: "Abraham", last_name: "Maslow")`

NB: See all your users with `User.all`

- NOTE: `create` combines the `new` and `save` actions.

Update

- Find - `user = User.find(1) #the number '1' passed into the find method corresponds to the id of the user it will find`
- Set - `user.first_name = "Taco"`
- Save - `user.save`

or

- Find — `user = User.find(1)`
- Update — `user.update_attributes(first_name: "Taco")`

Delete

- Find - `user = User.find(1)`
- Destroy - `user.destroy`
- Count - `User.all.count`

More Finding

- `User.all` -> returns an array of allusers
- `User.find_by_last_name('Lincoln')` -> returns the first user that meets the criteria
- `User.where(first_name: 'Abraham')` -> returns an array of users that meet the criteria
- `User.first` -> finds first user
- `User.last` -> finds last user

YOU DO:

- Add 2 new users to your database via new/save and 2 via create.

Let's add another column to our Users table via Migration

So far, we dropped and recreated our tables when we wanted to add columns to them. But this is not a practical, real-world solution. So we use migrations to do this in Rails.

Rails gives us some help to generate migration files - we can list the fields and their types in the generate command, and if we name the migration appropriately, Rails even guesses the name of the table:

WE DO:

- Let's add a `hometown` field to our `User` table:

```
rails g migration AddHometownToUsers hometown
```

- by putting Add....To.... Rails knows we are adding these columns to which table, and the migration can be written automatically

YOU DO:

- Add a new column to your `Users` table using a migration (`fav_food` , `nickname` , `pet`). Add the new attribute to each of your Users.

Schema.rb

When migrations run, Rails also updates the schema.rb file - it contains the migration commands all combined into each table.

The schema is the snapshot of your current database tables and fields.

Rollbacks

We can undo running migrations with:

`rake db:rollback`

- Beware:
  - don't rollback migrations which have been run on other machines (essentially, if they're in source code control)
  - instead, write a new migration to undo the changes

We can rollback to a specific migration like so:

`bin/rake db:migrate VERSION=20150213144026`

Run `rake db:migrate` to get back to the current version of your schema.

Removing columns from tables

We can use the same naming convention as create to automatically generate the migration file:

```
rails g migration RemoveNameFromCustomers name:string
```

Validations

We set our validations in our app/models/user.rb model.

Let's make sure each user definitely has a first name and a last name before they're saved.

```
class User < ActiveRecord::Base
  validates :first_name, presence: true
  validates :last_name, presence: true
end
```

- Type `reload!` into the console to update your model validations.
- Try saving a user with no first or last name and see what error is thrown.
- Try calling `valid?` on a new user.
- Walk through the Rails Docs and show that you can add error messages to your validations (e.g.- `too_short: "must have at least %{count} words"`).

WE DO:

- Add the following validation to `user.rb`:

`validates :age, length: { is: 2 }`

or add `length` to the name fields:

```
validates :last_name, presence: true, length: { minimum: 4 }
```

- Create a user and make it fail. Then make it pass.

YOU DO:

- Add a validation to the new table column you created above.

Further Reading

- Active Record Overview
- Migrations
- Validations