ERB FILE--

<% %>

Executes the ruby code within the brackets.

<%= %>

Prints something into erb file.

<% -%>

Avoids line break after expression.

<%# %>

Comments out code within brackets; not sent to client

So, these are the steps we need to follow to create models:

>rails generate model Person first\_name:string last\_name:string --force

This creates a migration file which creates tables and fields in the database.

[location: db/migrate] The ruby code in this file is converted into database directives.

\*Migrate can be applied to the database by

>rake db:migrate

\*Special console to see database tables is

>rails dbconsole

(It is by default a sqlite console for ruby on rails)

\*Now we have special feature which is fully connected to the live database using

>rails console

Where we can run ruby commands to create database tales, columns, delete or update or save them

\*Use scope, default\_scope and custom class methods to build queries that can be called in single line from any controller that needs them

\*Queries are lazy. When you issue a query, it creates a query object but doesn’t make a change in the database yet.

\*In order to create and update or work upon the database in db console without errors, we need to update the constraints in the models/person.rb

\*reload! command will successfully update all the code from application so we can run it in the console

\*ActiveRecords have another feature of callbacks that are similar to before and after filters in controllers but these can be run before or after any event in the series of events that occur in the lifecycle of model.

\*Write query code in the models and use named\_scopes

Use the word scope and choose a name for your new query

Scope :all\_lazy

Lambda can take a block of code as an argument

\*First step for running any rails 3 application-

>gem install bundler

>bundle

>rake db:create db:migrate #to create and migrate database

Rails will update schema.rb file after this.

CREATE PROJECT SCAFFOLD

>rails generate scaffold Project title:string

>rake db:migrate

>rails server

To add reference data which is same in development and production, in db/seeds.rb

>rake db:seed

(File is plural(db/seeds) since it is talking about data but the task is singular since it is talking about verb)

Use puts(‘message’) for output. Rails.logger.info won’t work.

>rake –T db shows the possible commands on db

Class Level: respond\_to

Here you specify the formats that the Responder should handle.

# Responds to html and json on all actions

respond\_to :html, :json

# Responds to html and json on index and show actions only.

respond\_to :html, :json, :only => [:index,:show]

# Responds to html for everything except show, and json only for index, create and update

respond\_to :html, :except => [:show]

respond\_to :json, :only => [:index, :create, :update]

Instance Level: respond\_with

The options here are those that would be passed to render or redirect\_to in your controller, but they are only included for success scenarios. For GET actions these would be the render calls, for other actions this would be the options for redirect. Probably the most useful of these is the :locationoption, which can be used to override that redirect path in case the arguments for respond\_with are not sufficient to build the right URL.

# These two are essentially equal

respond\_with(:admin, @user, @post)

respond\_with(@post, :location => admin\_user\_post(@user, @post)

# Respond with a 201 instead of a 200 HTTP status code, and also

# redirect to the collection path instead of the resource path

respond\_with(@post, :status => :created, :location => posts\_path)

# Note that if you want to pass a URL with a query string

# then the location option would be needed.

# /users?scope=active

respond\_with(@user, :location => users\_path(:scope => 'active'))

Performance Optimization is Memoization.

-Only use memorization with expensive operations that aren’t already cached by activeRecord.

Formtastic by justinfrench