Rails Magic

Database Config

database.yml

```
development:
  adapter: postgresql
  database: hello_class
  username: hello_class_user
  password: secret123
```

Database Associations

Conventions

- model_id for key columns
- foreign keys are unnecessary
    - but advisable

How?

- Class Macros
  - has_one
  - belongs_to
  - has_many
  - has_many :through

has_one

```
class User < ActiveRecord::Base
  has_one :office
end
```

<diagram showing has_one database relationship>

belongs_to

```
class User < ActiveRecord::Base
  belongs_to :organization
end
```

<diagram showing belongs_to relationship>

has_many

```
class Organization < ActiveRecord::Base
  has_many :users
end
```

<diagram showing has_many relationship>

has_many :through

```
class Organization < ActiveRecord::Base
  has_many :users
  has_many :offices, :through => :users
end
```

<diagram showing has_many :through relationship>

Using Associations

Just like any other attribute!

```
Organization.first.offices

User.first.office

Organization.first.users

User.first.organization
```

ARel

ActiveRecord Query DSL

- where
- select
- order
- group
- limit
- offset
- joins

where

```
# Builds a where clause

User.where("age > ?", 21)
# select * from users where age > 21

User.where("first_name = ?", "Bob")
# or
User.where(:first_name => "Bob")
# select * from users where first_name = "Bob"
```

where continued

```
# Be careful of SQL injection!
# String interpolation is not sanitized!

User.where("first_name =
          '#{params[:first_name]}'")
# select * from users
# where first_name = 'Bobby'; drop tables;'

# Do this instead
User.where("first_name = :first_name",
         params)
```

select

```
# You can specify fields to select

User.
  where(:first_name => "Bob").
  select(:last_name, :age)
# select first_name, age from users
# where first_name = 'Bob'
```

order, limit and offset

```
User.order("age ASC")
# select * from users order by age ASC

User.order("age ASC").limit(5).offset(10)
# select * from users limit 5 offset 10
```

group

```
age_distribution =
  User.
    select("COUNT(*) as user_count, age").
    group("age")

# select COUNT(*) as user_count, age
# from users
# group by age

age_distribution.each do |count|
  puts "Age: #{count.age} Users: #{count.user_count}"
end
```

joins

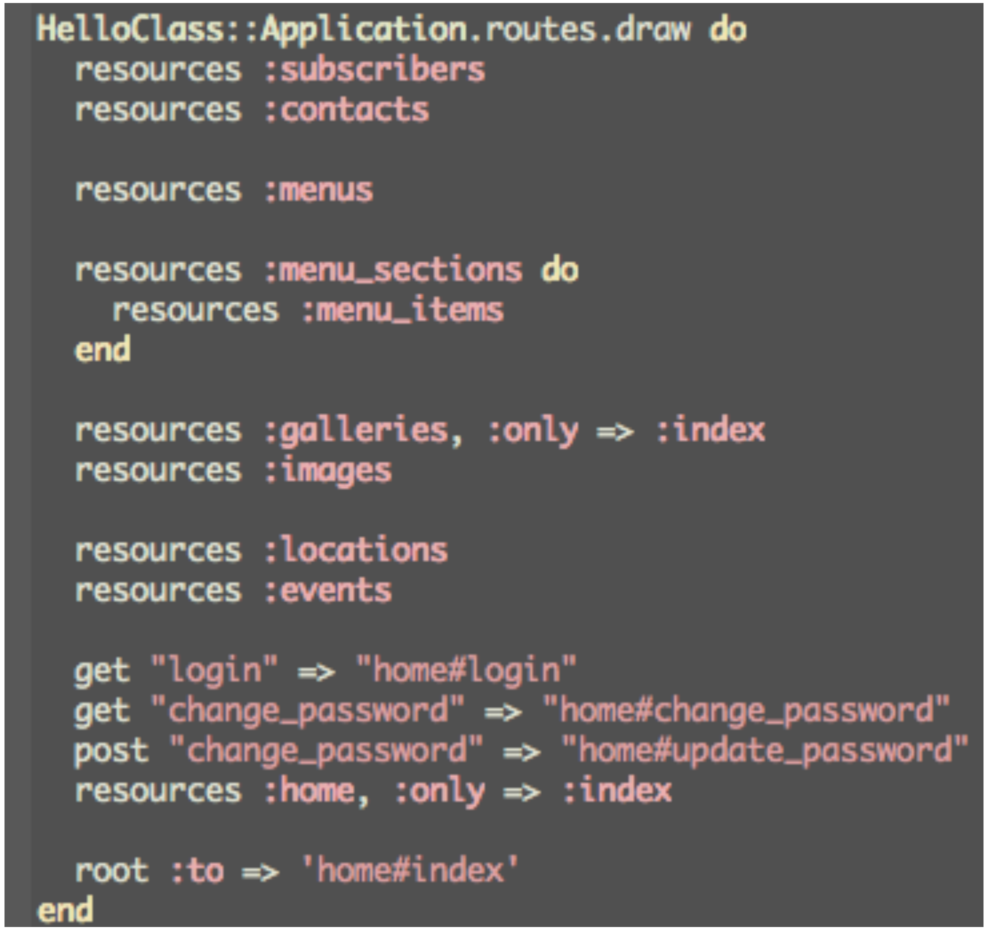
```
User.
  joins(:organization).
  where("organizations.name = ?", "Foo")
# select users.* from users join organizations
# on users.organization_id = organizations.id
# where organizations.name = 'Foo'
```

Chaining

- DSL methods can be chained together
- ActiveRecord is lazy by default
    - You can return a query object from a method
    - Queries will only be executed if needed

Routing

routes.rb



```ruby
HelloClass::Application.routes.draw do
  resources :subscribers
  resources :contacts

  resources :menus

  resources :menu_sections do
    resources :menu_items
  end

  resources :galleries, :only => :index
  resources :images

  resources :locations
  resources :events

  get "login" => "home#login"
  get "change_password" => "home#change_password"
  post "change_password" => "home#update_password"
  resources :home, :only => :index

  root :to => 'home#index'
end
```

Routing DSL

```
# All requests for /hello_world are forwarded
# to the HelloWorldController's index action
match "hello_world" => "hello_world#index"

get "users" => "users#index"

post "users" => "users#create"

put "users/:id" => "users#update"

delete "users/:id" => "users#delete"
```

Routing DSL continued

```
# Defines all of the routes needed for a
# REST resource
resources :users

# Requests to /admins are forwarded to the
# index action with params[:admin] set to true
get "admins" => "users#index", :admin => true

# Requests are forwarded to the by_name action
# with params[:first_name] and
# params[:last_name] populated
get "users/:first_name/:last_name" => "users#by_name"
```