Escuela de Informática y Sistemas

Jornadas de Administración e Informática

Año 2009

Conferencia: Ruby on Rails

Sergio A. Alonso

Analista en Sistemas

sergio@eim.esc.edu.ar

http://obelix.bunker.org.ar

Tercer día: jueves 18 de octubre

Objetivos

Realizar, paso a paso, en los Windows de los alumnos, la aplicación (demostrada) segundo día con Linux.

Primeras acciones con InstantRails

- Crear la aplicación
- Crear un “Hola Mundo”
- Comentarios respecto de conveniencia de trabajar en inglés, y el modo ***Convenir antes que Configurar***
- Configurar una base pequeña con sqlite o mysql, con sus managers, o directamente en la consola o MSDOS.
- Jugar con la base de datos utilizando la consola Rails.

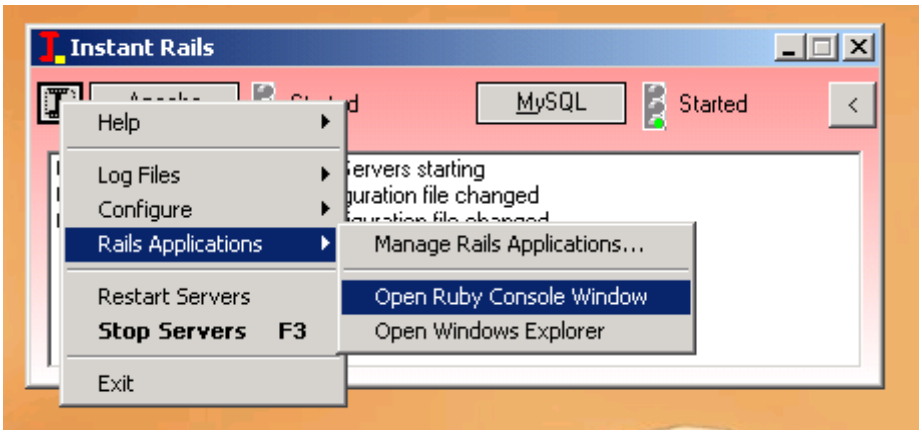
  Técnicamente: abstraer la sintaxis SQL mediante el ORM provisto por ActiveRecord.

Indice

Índice de contenido

Actividad:

- Abrir una consola (o MSDOS) vía InstantRails → Rails Applications → Open Ruby Console
- Ejecutar el siguiente comando:



 **`rails demo`**

Explicación

Este comando fabrica el esqueleto de la aplicación Rails. Ademas, preconfigura el entorno para trabajar con una base de datos llamada SQLite.

Si deseamos en cambio trabajar con otra base de datos, tal como mysql, oracle, postgresql, ibm_db, etc, podemos preconfigurar algunos archivos escribiendo, por ejemplo

 **`rails demo --database=mysql`**

Actividades

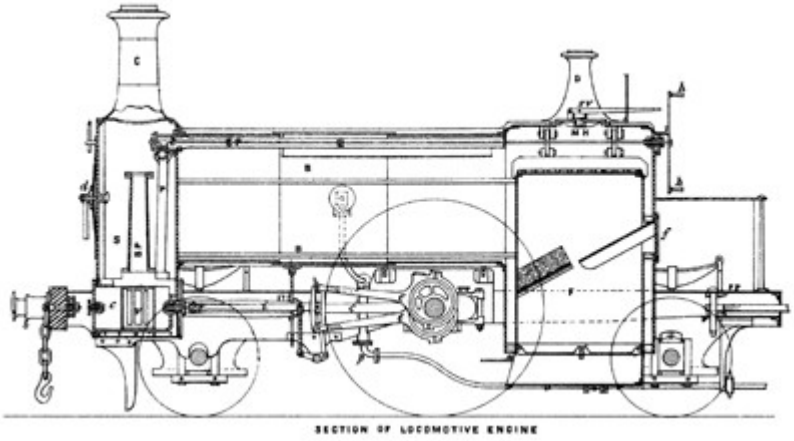
- Opcional bajo Windows: Mediante el editor **RoRed**, abrir la carpeta

  `C:\InstantRails\rails_apps\demo`

- Observar las carpetas generadas.
- Observar las bases definidas en **`config/databases.yml`**

Temas

- SQLite3
- YML
- Carpetas que se utilizan para el Modelo Vista Controlador




```
script/server
```

```
ruby script\server
```

Explicación

Corre un server web (escrito en Ruby) para pruebas.

Actividades:

- Cargar http://localhost:3000 en el navegador
- Observar el servidor Web logueando las consultas y errores

```
rm public/index.html
```

```
del public\index.html
```

Explicación

Borra el archivo de bienvenida

Actividades:

- Abrir una segunda consola (o MSDOS) vía InstantRails → Rails Applications → Open Ruby Console
- Recargar la pagina http://localhost:3000
- Observar el error, tanto en el navegador como en la salida del server Web

Temas:

Rails analiza la URL y busca una acción asociada. Las acciones están definidas dentro de los controladores, quienes deciden que se hace con la URL cargada en el navegador.

 **script/generate controller hola**

 **ruby script\generate controller hola**

Explicación:

Fabrica un controlador

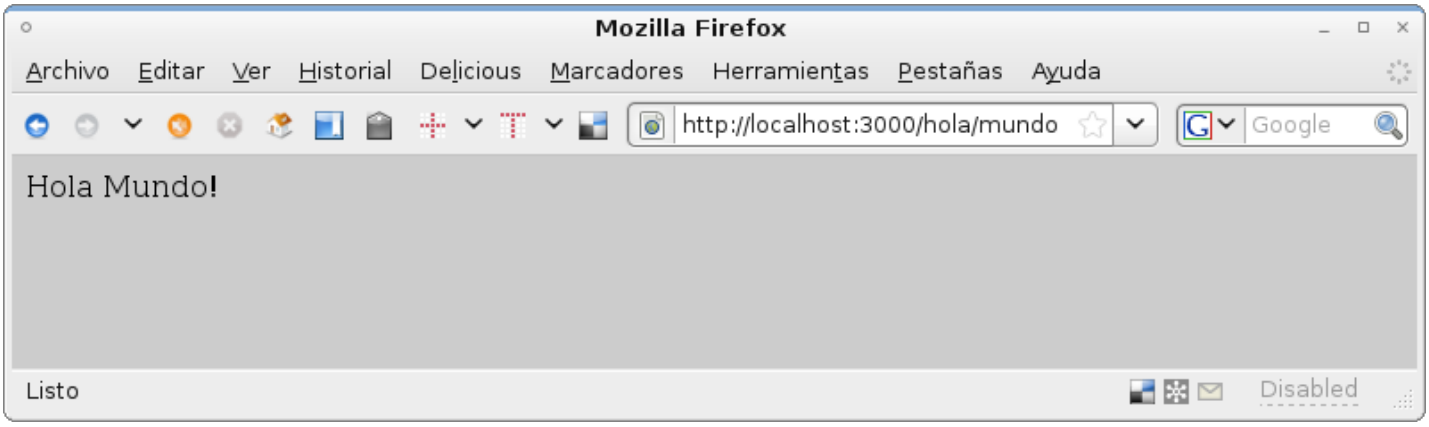
Actividades

1- Abrir **app/controllers/hola_controller.rb** y agregarle dos acciones:

```
class HelloController < ApplicationController
  def index
  end
  def mundo
    render :text => "Hola Mundo!"
  end
end
```

2- Reiniciar (**`Ctrl + C`**) el servidor Web

3- Cargar en el navegador la dirección **`http://localhost:3000/hola/mundo`**



El archivo `config/routes.rb`

Busque las lineas

Install the default routes as the lowest priority.

```
  map.connect ':controller/:action/:id'
```

Explicación

Estas líneas indican ni mas ni menos la forma en que Rails **mapea** las direcciones, de tal forma que sean fáciles de recordar, tanto por *personas* como por *spiders* de buscadores.

Ejemplo: **http://www.mascotas.org.ar/adopciones/modificar/46**

… pero

```
render :text => "Hola Mundo!"
```

¿Qué es lo que hace? pues... fabrica una ***Vista*** por el momento. En realidad, deberíamos haber tenido un

archivo llamado **app/views/hola/mundo.erb**

Actividades

Probemos el otro *action*, (el *index*), escribiendo

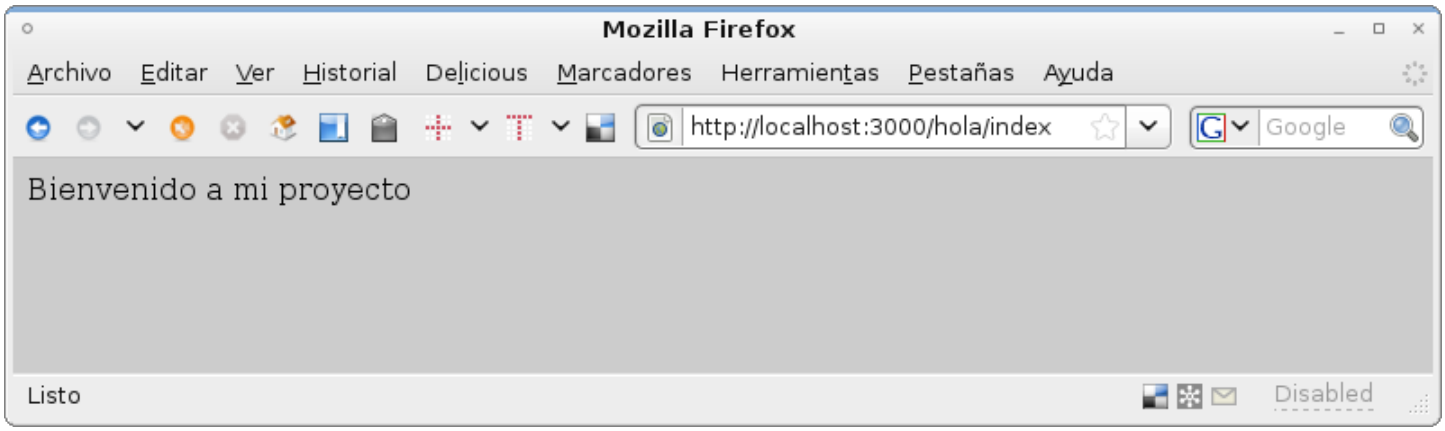
http://localhost:3000/hola

o también

http://localhost:3000/hola/index

Efectivamente, el **controlador** debería tener asociada una **vista**. Escribamos una vista muy simple, creando un archivo en **app/views/hola/index.erb**

```
<html>
   Bienvenido a mi proyecto
</html>
```


Mozilla Firefox
Archivo Editar Ver Historial Delicious Marcadores Herramientas Pestañas Ayuda
http://localhost:3000/hola/index
Google
Bienvenido a mi proyecto
Listo
Disabled

ActiveRecord

Se trata de una biblioteca capaz de realizar un Mapeo Relacional de Objetos (ORM), accediendo a los datos de los motores SQL sin preocuparse por la sintaxis de cada motor en particular.

Esta librería puede ahorrarnos montones de líneas de código. Sin embargo, para lograr su magia, realiza una especie de pacto con el programador, que conviene respetar.

Para empezar, Rails es un framework que utiliza **Convención sobre Configuración**. Una de sus convenciones es trabajar en ingles. Esto no sorprende ni molesta a ningún programador acostumbrado a trabajar en factorías de software. Incluso es muy útil si se desea abrir el código y acelerar el desarrollo del proyecto con programadores del ambiente de la comunidad de Software Libre.

Así, si vamos a tener una tabla “Compañías”, la trataremos en inglés, y cuidando mayúsculas y tiempos verbales. Ejemplo:

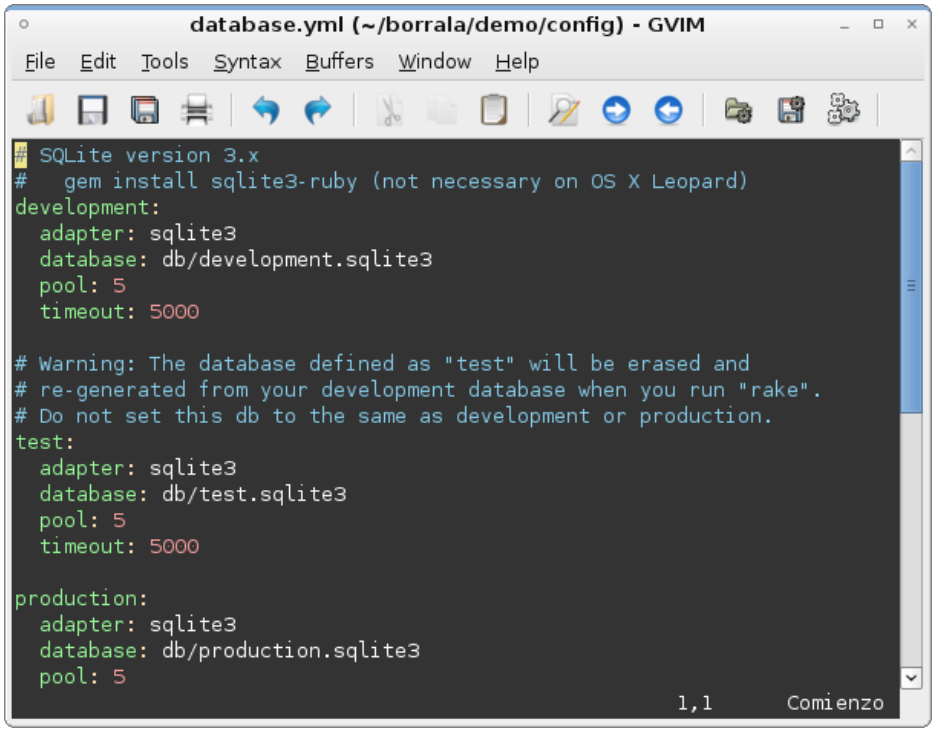
- Tabla: **`companies`**
- Modelo: **`Company`**
- Archivo que contendrá la definición: **`app/model/company.rb`**
- Las claves principales de las tablas siempre serán **`id`**, no “id_company” o “cod_company”
- Las claves externas siempre terminarán en **`_id`**, ejemplo: **`category_id`**

Actividades

1- Abra el archivo **`config/databases.yml`** :

Tip: si tiene vim instalado (y si en Windows lo instaló como Full), puede llamar al editor escribiendo

`gvim .`



Explicación:

Esto quiere decir, que las bases, si es que decidimos utilizar SQLite3, van a permanecer en la subcarpeta **`/db`**

Por que utilizar SQLite? En realidad puede utilizar toda clase de motores SQL. Pero este pequeño motor se recomienda mucho en la comunidad de software libre por las siguientes razones:

- Es muy rápido para consultas

- Es muy portable. De hecho, forma parte del motor Ajax en su pequeño pero poderoso plugin "Google Gears".

- Permite llevarse la base en cada migración, y reconstruirla en el destino, sin tener que molestarse en dumpear la base cada vez.

… y muuuchas otras razones.

El principio de trabajo de SQLite es muy simple. **Podríamos** crear la base realizando un simple

sqlite3 db\development.sqlite3

… pero esto implicaría que *queremos* aprender a hacer las cosas al estilo SQLite3. De esto es lo que precisamente quiere ORM que **no nos preocupemos.**

Scaffolds

Un poco de acción rápida:

```
ruby script/generate scaffold Company name:string address:string active:boolean
```

```
rake db:migrate
```

Abrir Firefox y escribir

http://localhost:3000/companies

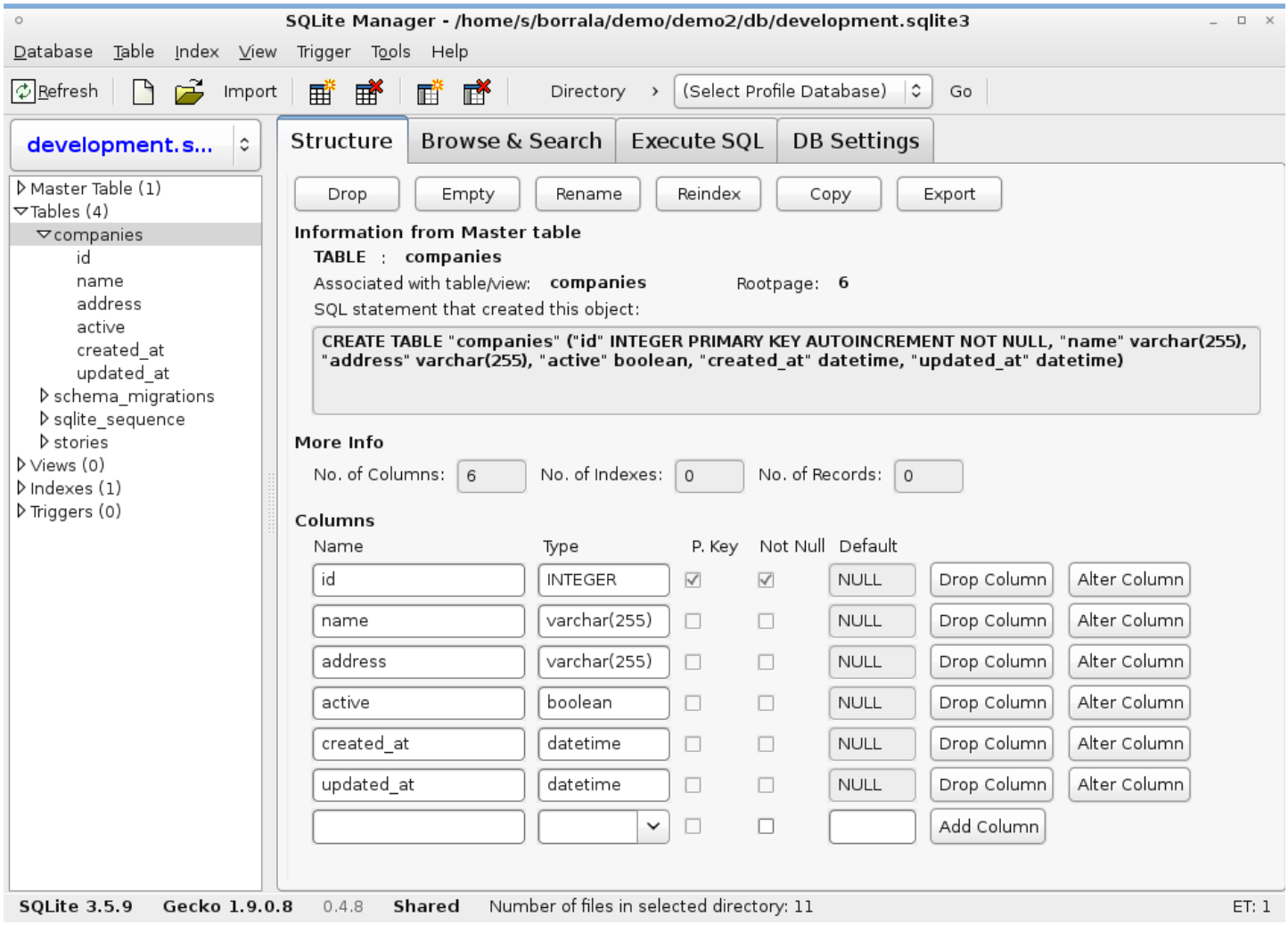
¡Nada mal para 3 (tres) líneas escritas en MSDOS!

- La primera línea construye los modelos, es decir, las clases que mapean a un modelo relacional.
- La segunda linea crea la base de datos, y una tabla **companies** (notar el plural).

Explicación: ¿Y la base donde está?

Podemos constatar el trabajo realizado mediante algún administrador para SQLite, tal como **SQLite Manager** (Plugin de Firefox). Efectivamente, **rake** nos ha construido todos los campos, mas el **id**, y dos campos muy útiles: **created_at** y **updated_at**

El piso de datos: SQLite



¿Cómo sabía **rake** que cosas crear? Pues la linea anterior (*generate scaffold*) nos construyó los *andamios* necesarios:

- Revisar algunas vigas: el archivo **`db/migrate/20090506030728_create_companies.rb`**

¿Como lo hizo **rake**? Es un buen momento para revisar el archivo de bitácora

 **`more log/development`**

 **`more log\development.log`**

Allí nos encontraremos con un montón de instrucciones SQL que hecho ActiveRecord por nosotros:

instrucciones CREATE TABLE, CREATE DATABASE, etc.

Nota 1: En **Windows,** debido a que este archivo está escrito en utf8, y MSDOS trabaja en ascii, la salida en pantalla se verá plagada de caracteres extraños.

Nota 2: En **Linux** se acostumbra a usar **tail -f** en lugar del anticuado **more**

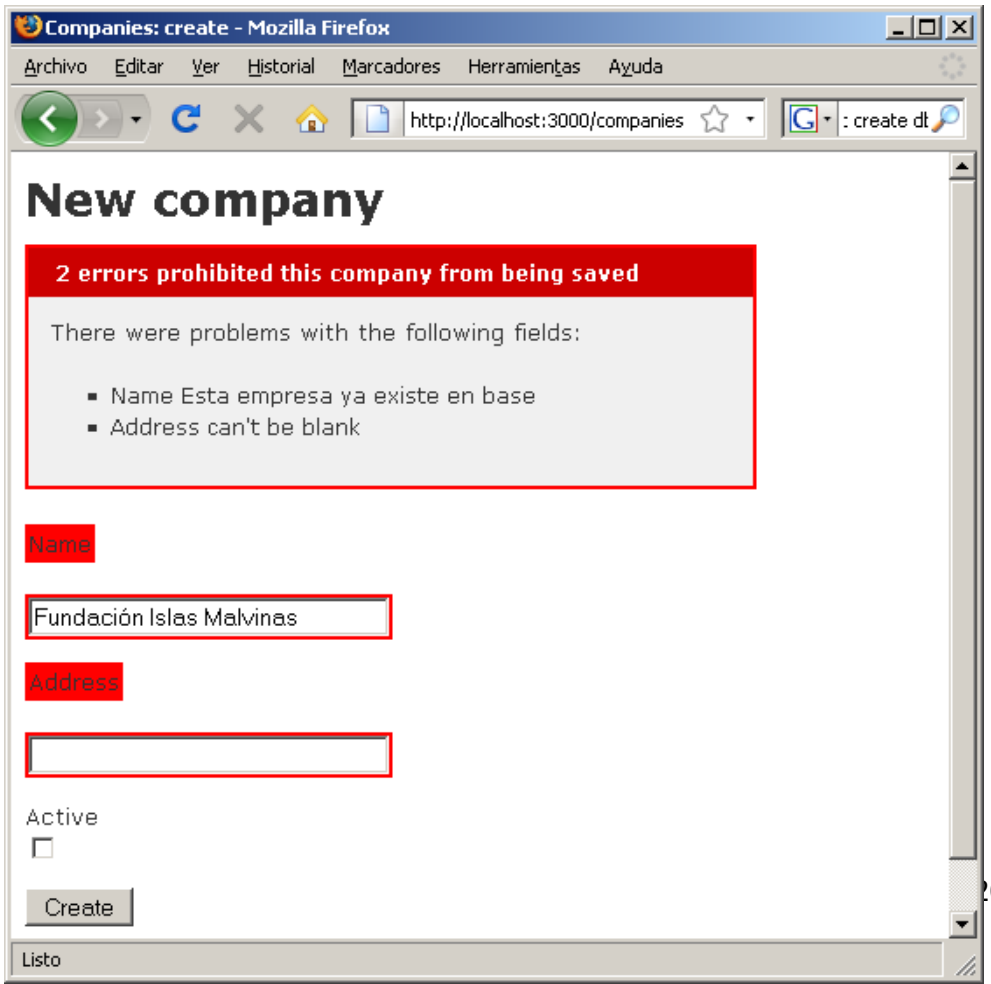
Validaciones

Validar no puede ser mas sencillo en Rails. Debemos aclararlo en el modelo.

Actividad: abrir **app\models\company.rb**, y dejarlo de la siguiente forma:

```
class Company < ActiveRecord::Base
     validates_uniqueness_of :name, :message => "Esta empresa ya existe en base"
     validates_presence_of :name, :message => "necesito este valor "
validates_presence_of
:address
end
```

Si todo anda bien...



La imagen no es nada: Layouts y CSS

Vamos a mejorar un poco la presentación

1) Entre a la plataforma http://www.campus.uner.edu.ar
2) Entre a Grado → Facultad de Ciencias de la Administración → Ruby on Rails
3) Encuentre Apuntes y Actividades Tercer Día
4) Descargue layouts_css_imagenes_extras.rar
5) Descomprima el contenido **sobre** la raíz de su proyecto:

C:\InstantRails-2.0-win\rails_apps\demo

Concretamente, el archivo comprimido contiene:

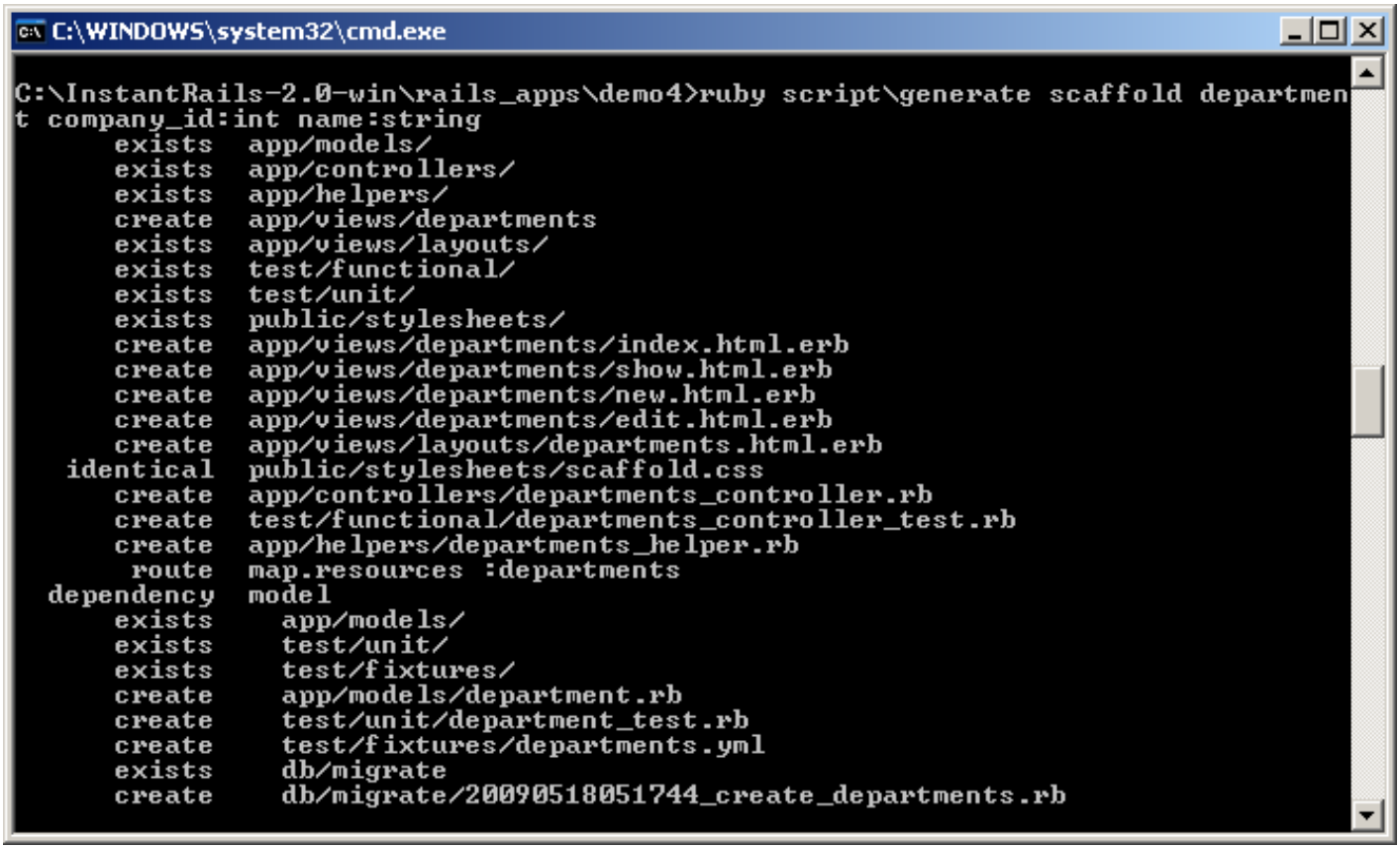
Archivo ejemplo	**Explicación**
`app\models\company.rb`	Las validaciones que ya vimos.
`app\views\`**`layouts`**`\companies.html.erb`	Ciertas zonas ( ) que Rails “renderea” **antes** que las Vistas (edit.html.erb, index.html.erb, new.html.erb, etc.)
`public\images\avatar_sergio.png`	La imagen de arriba a la izquierda.
`public\stylesheets\sergio.css`	Desarrollo de las zonas ( ) solicitadas por el layout. Esto es ubicación, margenes, colores, etc

Tablas Combinadas

Las compañias poseen mas de un departamento. Generaremos modelo y controlador de departamentos en un solo paso:

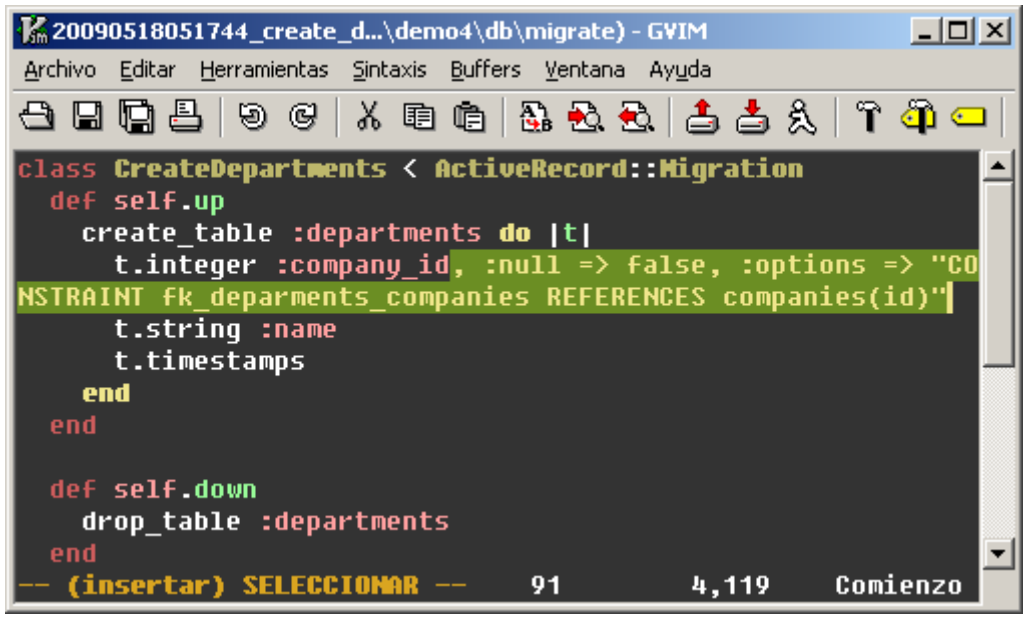
`C:\InstantRails-2.0-win\rails_apps\demo4>`**`ruby script\generate scaffold department company_id:int name:string`**

Si observamos la salida generada, veremos que Rails ha generado archivos para cuando estemos listos para “migrar”, esto es, crear en el motor SQL un reflejo funcional de nuestro modelo.



Concretamente, y entre muchas otras cosas, ha generado un archivo

`db\migrate\20090518051744_create_departments.rb`



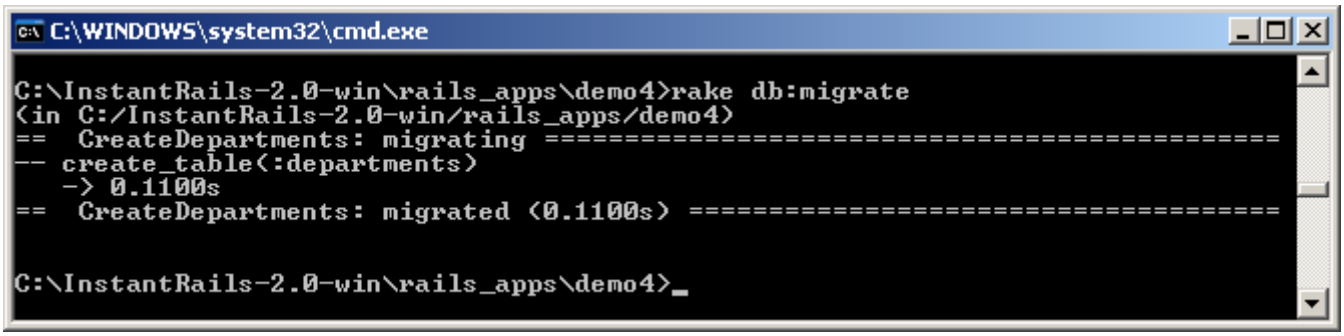
En este archivo aparece un detalle: existe un campo llamado a propósito company_id: esto significa dos cosas:

- Debe existir una clase (modelo) Company por algún lado.
- Según la convención de Rails, cuando aparece un campo terminado en _id, es por que lo estamos declarando como clave externa de otra tabla.
- En la pagina 143 del libro Agile Web Development with Rails (Tercera Edición), se recomienda modificar el archivo de migraciones, y en la línea de creación de estos campos clave, agregar en lenguaje DDL los datos necesarios para crear el índice. Lo llamaremos **fk_deparments_companies**

  Por esta razón la línea t.integer … quedaría de esta manera:

  ```
  t.integer :company_id, :null => false, :options => "CONSTRAINT
  fk_deparments_companies REFERENCES companies(id)"
  ```

Corremos la migración



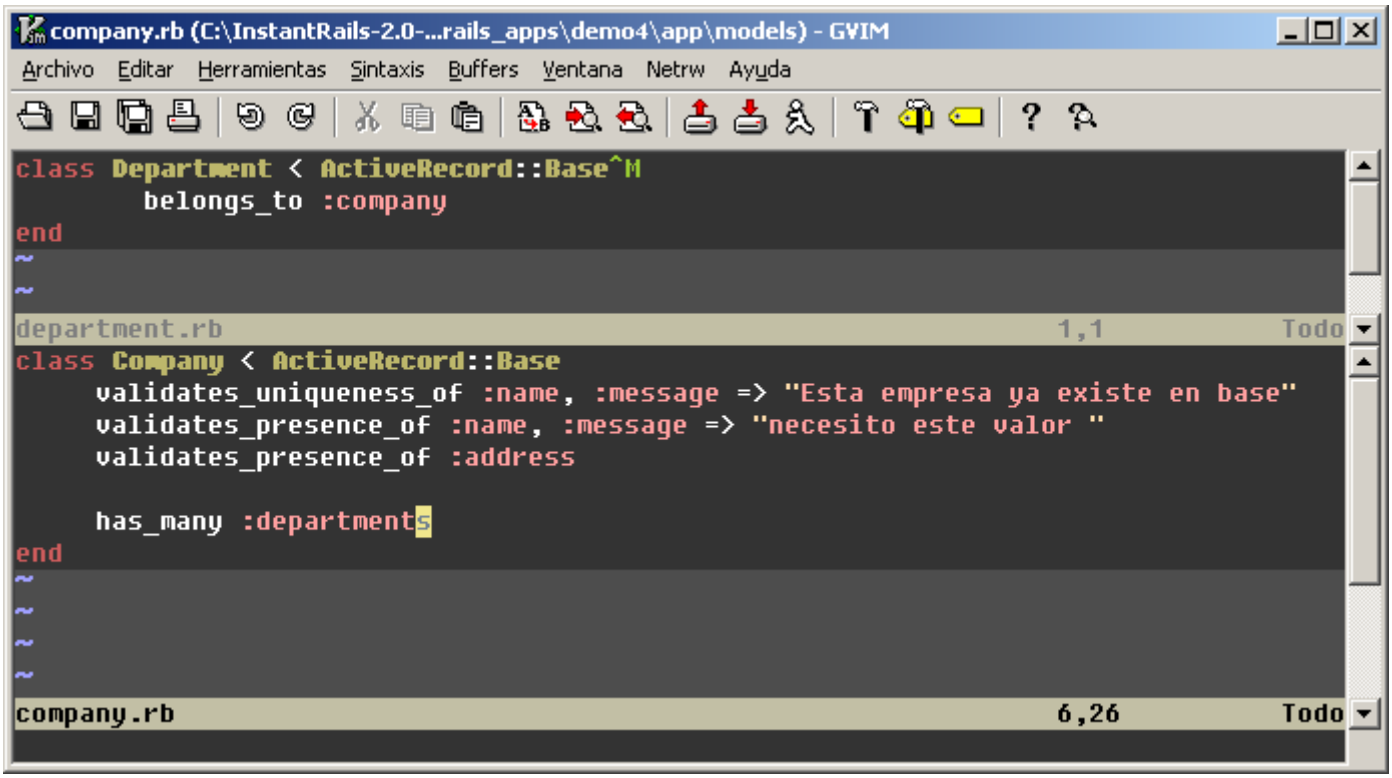
Es decir que por abajo, si miramos el archivo **log\development.log**, nos encontraríamos una charla entre ActiveRecord y el motor SQLite, de tipo CREATE TABLE, etc.

Formalizar la relación

Si bien hemos establecido la relación entre ambas tablas, no hemos aclarado si es una relacion de Uno a Uno, de Uno a Muchos, o de Muchos a Muchos.

En terminos generales, abrimos ambos modelos, y les explicamos a Rails que una compañía puede tener varios departamentos. En tanto que un departamento pertenecerá a una sola compañía.

Lo hacemos en la carpeta app\models



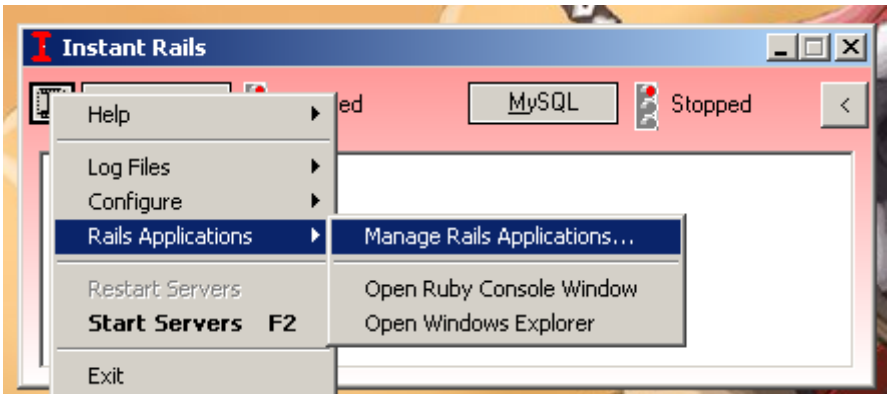
Jugar con Tablas Combinadas en Rails Console

Estando ambas tablas creadas, es una buena ocasión para ver si el modelo responde a nuestras expectativas.

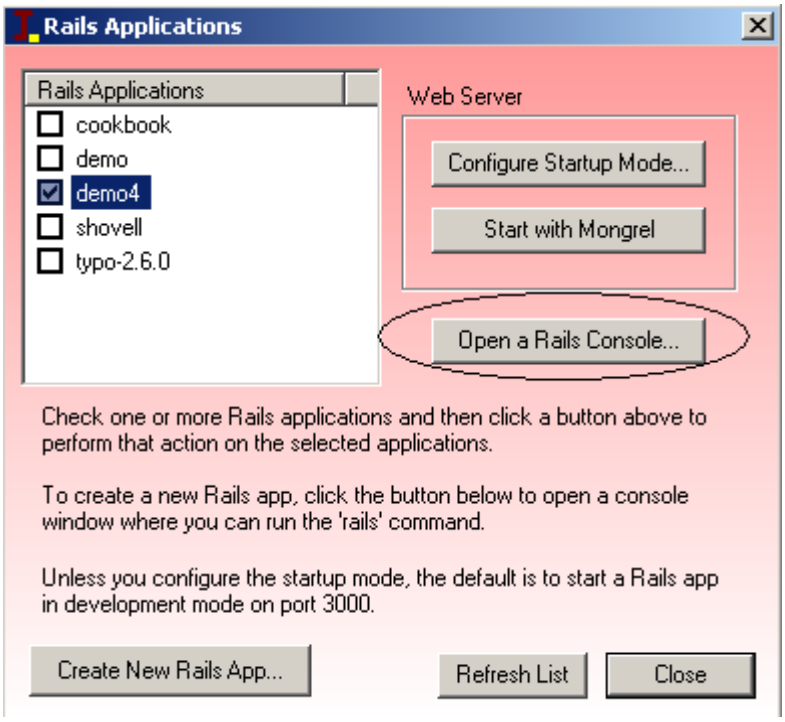
1. Creamos una Compañía y tomamos nota de su índice, creado automáticamente. Por ejemplo: 3
2. Creamos un Departamento, y en el campo company_id, fijamos el valor 3
3. Cuando queremos saber a que compañía pertenece un Departamento, en lugar de realizar un

   ```
   SELECT companies.name

   FROM companies, departments

   WHERE companies.id = departments.companies_id
   ```

   … simplemente instanciamos un objeto de tipo Departament. Una de sus propiedades es la Company a la que pertenece. A esto se le llama expresarlo en forma de ORM.

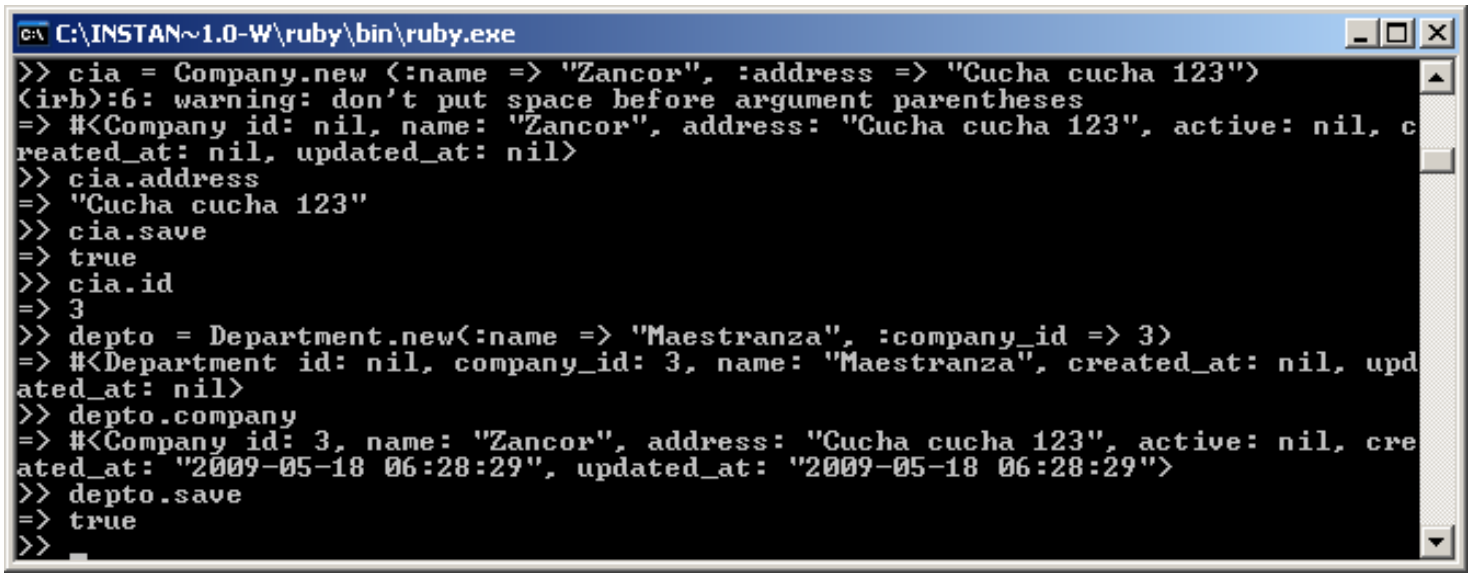


Si bien estos pasos los podemos hacer creando cada registro en el **SQLite Manager,** conviene aprender a escribirlo en formato ORM, ya que así lo expresaremos cuando programemos **controladores**. Para ello llamamos a una versión del IRB (el interactive Ruby) ya precargado con la clase ActiveRecord y sus herencias... nuestros modelos.

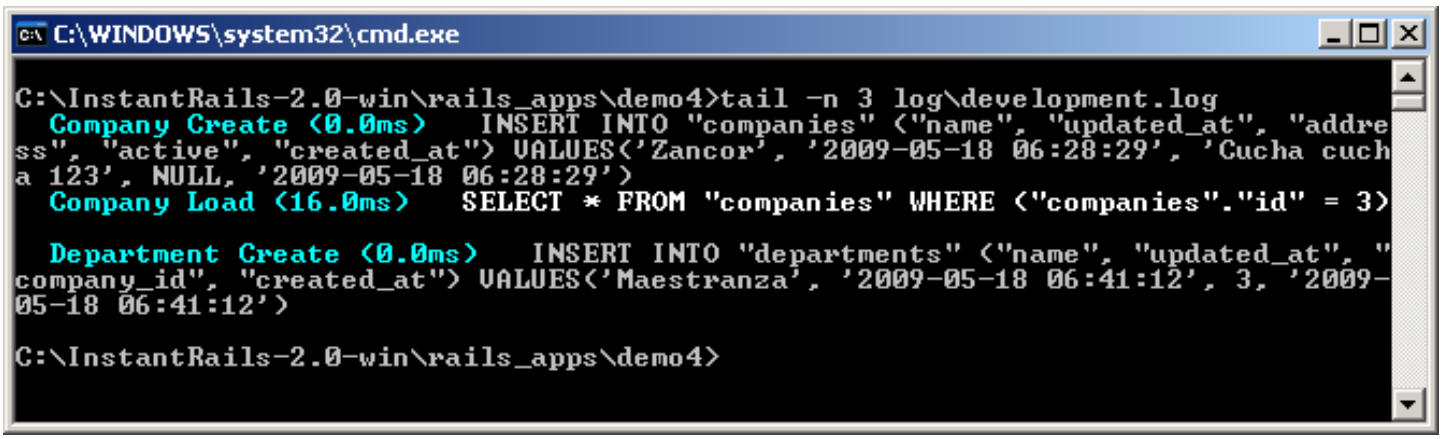


Esto se le conoce como la **Rails Console**.

Observar atentamente el “chateo” con el modelo. Podemos jugar a ser **controladores** antes de sentarnos a programar.



Aquí podemos ver los pasos realizados en un Rails Console. Por abajo están ocurriendo las verdaderas transacciones SQL.



SELECT / Combo / Caja desplegable

Crear en el controlador `[app/controllers/departments_controller.rb]`, accion `edit`, un objeto, llamado @`companias`, compuesto por TODAS las compañias que le proporcione ActiveRecord.

Se lo crea con @ adelante para que tengan acceso a él desde las vistas.

```
  @companias = Company.find(:all)
```

Y en la vista, `app/views/departments/edit.html.erb`, cambiamos la modesta caja que nos permitía escribir el id de la compañía a la cual pertenecía el departamento:

```
  <p>
    <%= f.label :company %><br />
    <%= f.text_field :company_id %>
  </p>
```

Y lo reemplazamos por una caja desplegable, mucho mas agradable:

```
<select name="department[company_id]">
   <% @companias.each do |cia| %>
       <option value="<%= cia.id %>"
         <%= 'selected' if cia.id == @department.company_id %> >
         <%= cia.name %>
       </option>
   <% end %>
```

ORM... de nuevo

Recordemos: en todo ORM (Object Relational Map), las clases mapean tablas, los atributos a los campos, los métodos a las acciones, y un objeto puede ser un registro o una colección de ellos.

Abrimos una consola irb mediante el comando

```
script/console
```

y probamos

1. Crear un objeto **deptos** que represente una colección de todos los registros disponibles
2. Lo iteramos. Es decir, lo recorremos, mostrando con el comando p (puts para holgazanes).

```
deptos=Department.find(:all)
    ==> [#<Department id: 1, company_id: 1, name: "Secretaria Academica",
created_at: nil, updated_at: nil>, #<Department id: 2, company_id: 3, name:
"Maestranza", created_at: "2009-05-18 06:41:12", updated_at: "2009-05-18
06:41:12">]

irb --> deptos.each do |i|
    \-+ p i.name
>> end
"Secretaria Academica"
"Maestranza"
deptos[1].name
    ==> "Maestranza"
```

También lo podríamos haber recorrido con un for, un do loop, etc. Pero horrorizaría a cualquier programador de Ruby.

Partials

Basado en Pag 166 de Foundation Rails

Se utiliza para formularios muy grandes, en donde es muy laborioso mantener actualizados los mismos campos (nombre, apellido, dirección, dni, etc) en **cada** formulario New, y en cada Update.

Simplemente creamos un archivo “parcial”, común a ambos archivos, que es llamado y renderizado cada vez que hace falta. Los partial llevan un guión bajo “_” para ser identificados.

1) Abrimos app/views/departments/edit.html.erb, y **cortamos** la sección:

```
<% form_for(@department) do |f| %>
  <%= f.error_messages %>
  <p>
    <%= f.label :name %><br />
    <%= f.text_field :name %>
  </p>
<p>
    <select name="department[company_id]">
        <% @companias.each do |cia| %>
            <option value="<%= cia.id %>"
                <%= 'selected' if cia.id == @department.company_id %> >
                <%= cia.name %>
            </option>
        <% end %>
     </select>
</p>
<p><%= f.submit "Update" %></p>
<% end %>
```

La pegamos en un archivo nuevo, ubicado en **app/views/departments/** que llamaremos **_form.html.erb**

El ultimo párrafo que dice “**Update**” lo cambiamos por algo mas genérico: “**submit**”.

Y en **app/views/departments/edit.html.erb**, lo dejamos de la siguiente manera:

```
<h1>Editing department</h1>

<%= render :partial => 'form' %>

<%= link_to 'Show', @department %> |
<%= link_to 'Back', departments_path %>
```

Probamos si anda todo bien. Ahora , para que el formulario “New” se beneficie tambien de cualquier cambio en el partial, actualizamos tanto la accion new, como la vista app/views/departments/new.html.erb

Crear automáticamente el diagrama

Fixme: probar en Windows si hace falta graphviz

```
sudo gem install railroad
railroad -M | dot -Tsvg > modelos.svg
railroad -C | dot -Tsvg > controladores.svg
```

 `nautilus .`

 `start .`

En el administrador de archivos, debería aparecer un diagrama de los modelos y otro de los controladores. El formato es .svg, es decir vectorial xml, que se lo puede formatear con cualquier herramienta de dibujo como Inkscape, Corel Draw, Ilustrator, etc.

Si Windows no trae ningún software, cambiar svg (vectorial, modificable) por .png (bitmap, deficil de modificar)

Mas info: http://railroad.rubyforge.org