acts_as_catalog , real_fk y acts_as_magic_model — Tres plugins simples para toda ocasión

Gunnar Wolf — gwolf@gwolf.org
http://www.gwolf.org/soft/rails

Instituto de Investigaciones Económicas, UNAM
Desarrollador del proyecto Debian

1er RubyCamp UNAM
5 de junio de 2009

Esto puede ser raro

Varios puntos de esta presentación pueden parecerte raros.

- Si bien soy desarrollador de Rails –y lo disfruto constantemente– no comparto una cantidad importante de puntos de la *cultura Rails*
- Algunas de las cosas que mencionaré existen ya de otro modo. Otras van directamente en contra de algunas prácticas bien establecidas.
- ...Pero a mí me funcionan, y espero que a ustedes también.

A continuación, tres plugins de Rails, desde triviales hasta sencillos, cortos, y probablemente útiles para aprender

Temas

El tipo más comun de tabla del mundo...

Constantemente en nuestros sistemas nos enfrentamos con *catálogos* — Tablas que tienen únicamente ID y nombre:

Definición de tabla

```
create_table :things do |t|
  t.string :name, :null => false
end
```

Modelo de tabla

```
class Thing < ActiveRecord::Base
  validates_presence_of :name
  validates_uniqueness_of :name
end
```

Recuerda: ¡No te repitas! (DRY)

Uno de los principios de diseño de Rails es darnos lo necesario para no repetir código redundante — En vez de repetir algo así de simple en cada uno de nuestros catálogos, hagámoslo más sencillo:

Definición de tabla

```
create_catalogs :things, :stuff, (...)
```

Claro, acompañado de su contraparte

```
drop_catalogs :things, :stuff, (...)
```

Modelo de tabla

```
class Thing < ActiveRecord::Base
  acts_as_catalog
end
```

Y ya que andamos entrados en gastos... ¡Colecciones!

¿Por qué no agregar otros métodos comunmente asociados a los catálogos?
Los catálogos muy comunmente los utilizamos como colecciones para llenar campos `select` o similares... Agreguemos pues:

Trabajando con colecciones completas

```
>> Thing.collection_by_id
=> [[1, "Una cosa"], [2, "Otra cosa"], [3, "Algo más"]]
>> Thing.collection_by_name
=> [[3, "Algo más"], [2, "Otra cosa"], [1, "Una cosa"]]
```

Infraestructura para I18N

La convención en varios marcos de internacionalización (I18N) es que, para traducir sin ambigüedad elementos dinámicos de una tabla, prefijemos a las cadenas con el nombre de la tabla y un separador.
Llamemos a esto *nombre cualificado*. Entonces:

Nombres cualificados

```
>> Thing.find(1).qualified_name
=> "Thing|Una cosa"
>> Thing.qualified_collection_by_id
=> [[1, "Thing|Una cosa"], [2, "Thing|Otra cosa"], [3, "Thing|Algo más"]]
>> Thing.qualified_collection_by_name
=> [[3, "Thing|Algo más"], [2, "Thing|Otra cosa"], [1, "Thing|Una cosa"]]
```

¿Y dónde está acts_as_catalog ?

Página	`http://actsascatalog.rubyforge.org/`
Rubyforge	`http://rubyforge.org/projects/actsascatalog`
Depósito Git	`git clone git://rubyforge.org/actsascatalog.git`
Ver el código	`http://actsascatalog.rubyforge.org/git?p=actsascatalog.git;a=tree`

Temas

real_fk — Llaves foráneas verdaderas

- El modelo de base de datos que ha impulsado Rails es el de una base de datos *tonta*
- Las recomendaciones generales para el desarrollo con Rails parten de la premisa de que *toda la lógica del sistema estará implementada en un sólo lenguaje: Ruby*
- Por mucho tiempo, la base de datos por omisión al iniciar un proyecto Rails era MySQL, y mucha gente la usaba también en producción — Ahora que ha cambiado a SQLite, hay una mayor conciencia.

Un plugin orientado a las migraciones

- No hay como un verdadero motor de gestión de bases de datos — Y la integridad referencial es el punto de inicio.
- El punto donde se lleva a cabo casi toda la acción es en las migraciones
- A futuro, vale la pena extender este plugin para incorporarlo a `ActiveRecord::Base` — Especialmente para evitar excepciones a través de validaciones (explicaré más adelante).

Referencias sencillas — has_many, belongs_to

Las referencias más sencillas –y más básicas– son las simples — has_many y belongs_to

Definiendo referencias simples

```
create_table :students {|t| (...) }
create_table :groups {|t| (...) }
add_reference :students, :groups, :null => false
```

- Crea una columna group_id en la tabla students
- Acepta los mismos argumentos que al crear una columna en la tabla (en este caso, :null => false)
- En PostgreSQL, agrega un CONSTRAINT de llave foránea
  - ON DELETE RESTRICT. A futuro, pienso agregar la opción de especificar ON DELETE CASCADE.
- Claro está, hay un remove_reference correspondiente.

Referencias compuestas - has_and_belongs_to_many

Las referencias muchos-a-muchos son implementadas creando una tabla intermedia de relación.

Definiendo referencias con tablas intermedias

```
create_table :teachers {|t| (...) }
create_table :groups {|t| (...) }
create_habtm :teachers, :groups
```

- Crea una tabla intermedia de relación siguiendo la convención de Rails — En este caso, `groups_students`
- Es una tabla específicamente de relación — No tiene columna ID; sus dos únicas columnas son `group_id` y `teacher_id`
- Claro está, ambas columnas son creadas con `CONSTRAINT` de llave foránea

...A futuro

- Este plugin nos ahorra trabajo al preparar las migraciones, pero le falta aún integrarse a la lógica de la aplicación
- Si no incluimos explícitamente un `:validates_associated`, nos enfrentaremos a excepciones cuando la base de datos rechace al registro
- Este puede ser un buen caso para modificar el comportamiento base de `ActiveRecord::Base`, sin que requiera mención específica, con sólo tener el plugin disponible.

¿Y dónde está real_fk ?

Página	`http://realfk.rubyforge.org/`
Rubyforge	`http://rubyforge.org/projects/realfk`
Depósito Git	`git clone git://rubyforge.org/realfk.git`
Ver el código	`http://realfk.rubyforge.org/git?p=realfk.git;a=tree`

Temas

Declarar relaciones explícitamente... ¡buf! (1)

- Rails nos ha malacostumbrado a que las cosas son fáciles. Y automáticas.
- Todos los principios básicos –DRY, un mismo lenguaje para todo, convención sobre configuración, etc.- nos hacen especialmente sensibles a las repeticiones, al tiempo de programador optimizable
- Ya declaré las relaciones en mis migraciones (o de algún modo, manipulando la base de datos). **¿Por qué tenemos que repetirlo explícitamente?**

Declarar relaciones explícitamente... ¡buf! (2)

- Algunos sistemas requieren ser altamente flexibles — Permitir modificar la estructura de la BD en tiempo de ejecución
- Incluso si valoramos el que nuestro sistema en producción tenga todas las relaciones explicitadas, esto nos puee ser muy cómodo para prototipear

- Este plugin fue desarrollado de manera independiente y concurrente a uno muy similar hasta en el nombre y el URL — Dr. Nic's Magic Models, `http://magicmodels.rubyforge.org/`

Uso explícito (1)

Declarando modelos explícitamente como mágicos

```
# En la migración
create_catalogs :salary_levels
create_table :students { |t| (...) }
create_table :teachers { |t| (...) }
create_table :groups { |t| (...) }
add_reference :students, :groups
add_reference :teachers, :salary_levels
create_habtm :teachers, :groups

# Y los modelos respectivos
class Student < ActiveRecord::Base; belongs_to :group; end
class Group < ActiveRecord::Base; has_many :students; end
class Teacher < ActiveRecord::Base; acts_as_magic_model; end

# Desde la consola
>> Teacher.find(:first).groups[0].students[0].name
=> "Fulano de Tal"
>> Teacher.find(:first).salary_level.name
=> "Titular"
```

Uso explícito (2)

Del ejemplo anterior:

- Sólo declaré como `acts_as_magic_model` a una de las tablas; podría haberlas declarado a todas
- `acts_as_magic_model` modifica tanto al modelo en el cual es invocado como a los modelos en que éste se refleja — En este caso, agregó un `has_and_belongs_to_many` tanto en `Group` como en `Teacher`
- Si hay modelos referidos desde el modelo mágico que no existan, éstos son declarados (`SalaryLevel`)
- Si un modelo ya existe, la relación sólo es agregada a la definición existente

Uso mágico global

Podemos también ahorrarnos la declaración de todos los modelos que no requieran una implementación explícita

Declaración mágica (misma BD)

```
# En config/environments.rb
ActiveRecord::Base.auto_magic_models
```

- Todos los modelos son generados
- Naturalmente, los modelos que fueron declarados ya explícitamente siguen siendo válidos — Sólo se les agregan las relaciones

Otras características generales

- Reconoce y maneja correctamente las tablas de relación (`HABTM`)
- **No implementa (¿aún?) validaciones**
- Si el plugin `acts_as_catalog` está instalado y encuentra catálogos, agrega también métodos *`catálogo`*`_name` — En el ejemplo anterior, `Teacher#salary_level_name`.
- Su futuro... Es incierto. El plugin me gusta, funciona confiablemente y me representó una *gran* manera de aprender. Sin embargo, duplica la funcionalidad de un plugin bastante mejor conocido, probado y difundido.
- Bueno, pero no es un hecho que deje de desarrollarlo ;-)
  Sí funciona, y funciona muy bien

Otras características generales

- Reconoce y maneja correctamente las tablas de relación (`HABTM`)
- **No implementa (¿aún?) validaciones**
- Si el plugin `acts_as_catalog` está instalado y encuentra catálogos, agrega también métodos *`catálogo`*`_name` — En el ejemplo anterior, `Teacher#salary_level_name`.

- Su futuro... Es incierto. El plugin me gusta, funciona confiablemente y me representó una *gran* manera de aprender. Sin embargo, duplica la funcionalidad de un plugin bastante mejor conocido, probado y difundido.
- Bueno, pero no es un hecho que deje de desarrollarlo ;-)
  Sí funciona, y funciona muy bien

Otras características generales

- Reconoce y maneja correctamente las tablas de relación (`HABTM`)
- **No implementa (¿aún?) validaciones**
- Si el plugin `acts_as_catalog` está instalado y encuentra catálogos, agrega también métodos *`catálogo`*`_name` — En el ejemplo anterior, `Teacher#salary_level_name`.

- Su futuro... Es incierto. El plugin me gusta, funciona confiablemente y me representó una *gran* manera de aprender. Sin embargo, duplica la funcionalidad de un plugin bastante mejor conocido, probado y difundido.
- Bueno, pero no es un hecho que deje de desarrollarlo ;-)
  Sí funciona, y funciona muy bien

¿Y dónde está acts_as_magic_model ?

Página	`http://magicmodel.rubyforge.org/`
Rubyforge	`http://rubyforge.org/projects/magicmodel`
Depósito Git	`git clone git://rubyforge.org/magicmodel.git`
Ver el código	`http://magicmodel.rubyforge.org/git?p=magicmodel.git;a=tree`

Temas

¿Dudas? ¿Comentarios? ¿Cebollazos?

¡Gracias!

Gunnar Wolf — gwolf@gwolf.org
http://www.gwolf.org/soft/plugins_rails