

Optimización y Desnormalización del Modelo de Datos

Fernando Blat

fernando.blat@the-cocktail.com

<http://www.inwebwetrust.net>

Índice

- ¿Qué es ActiveRecord?
- Modelo de datos de ejemplo
- Normalización / Desnormalización del modelo
- Tags en La Coctelera
- Observers
- Conclusiones

Introducción a ActiveRecord

¿Qué es ActiveRecord?

- Librería de Rails encargada de mapear los objetos de negocio en la base de datos
- Crear un sistema de clases que podemos utilizar en nuestra aplicación
- Implementa el patrón Object Relational Mapping (ORM)
- Lo extiende proponiendo dos nuevos conceptos: relaciones y herencia

Filosofía

- Olvida la base de datos
- Toda la magia de Ruby para trabajar con el modelo de datos
- Minimizar el número de líneas y maximizar la productividad
- Independiente de Rails

¿Qué se puede hacer?

```
Post.find(:all, :conditions => "status='publish'")  
# SELECT * from posts WHERE  
  status='publish'
```

```
Post.find_by_nicetitle('como-optimizar-active-record')  
# SELECT * from posts WHERE  
  nicetitle='como-optimizar-active-record' LIMIT 1
```

```
post.categories  
# SELECT * from categories WHERE post_id='3'
```

```
post.author  
# SELECT * from people WHERE id='10'
```

Características

- Mapeo automático entre clases y tablas: tabla posts corresponde a la clase Post
- Asociaciones entre clases
- Validación de atributos
- `acts_as_(list|tree)`
- *Callbacks*
- Observers
- Herencia
- Transacciones
- Manipulación directa de objetos
- Abstracción de la base datos
- Soporte para *logging*

Usando AR fuera de Rails

```
require 'rubygems'
requiregem 'activerecord'

ActiveRecord::Base.establish_connection({
  :adapter => "sqlite3",
  :dbfile  => "db/db.posts.sqlite3"
})

class Post < ActiveRecord::Base
  set_table_name 'posts'
end

posts = Post.find_all_by_date('2006-11-25')
```

Nuestro pequeño modelo de datos

Nuestro pequeño modelo de datos

Categoría

```
class Category < ActiveRecord::Base
  has_many :posts
end
```

Persona

```
class Person < ActiveRecord::Base
  has_many :posts
end
```

Nuestro pequeño modelo de datos

Post

```
class Post < ActiveRecord::Base
  has_many :categories
  belongs_to :author, :class_name => 'Person'

  validates_presence_of :title
  validates_presence_of :body
end
```

Nuestro pequeño modelo de datos

Y en la base de datos...

```
create_table "posts" do |t|
  ...
  t.column "user_id", :integer
  ...
end

create_table "posts_categories", :id => false do |t|
  t.column "post_id", :integer
  t.column "category_id", :integer
end

create_table "categories" do |t|
  ...
end
```

(des) normalización

La Normalización...

- no es un capricho
- impide la duplicación de los datos
- evita la inconsistencia
- organiza las tablas y los datos de forma más natural
- produce esquemas más fáciles de entender

Así que si no normalizas...

- ten cuidado con la inconsistencia
- modificar un mismo valor implica actualizar varios campos
- aumentan las posibilidades de cometer errores

category.posts

```
SELECT * FROM posts
  INNER JOIN
    posts_categories
  ON
    posts.id = posts_categories.post_id
  WHERE
    (posts_categories.category_id = 41609)
  ORDER BY date DESC;

# 0.135517 segs.
```

No es para tanto

- en este caso, y posiblemente muchos otros, no es para tanto tener que trabajar con un modelo normalizado
- hay pocos datos: número pequeño de categorías, y de posts en dichas categorías
- siempre se trabaja en el dominio del usuario

Mamá, quiero tags

acts_as_taggable

- plugin desarrollado por DHH
- polimórfico
- introduce dos modelos: `Tag` y `Tagging`
- `Tag` contiene el identificador y el nombre
- `Tagging` contiene la relación con otros modelos

Tag

```
class Tag < ActiveRecord::Base

  has_many :taggings
  validates_uniqueness_of :name

end
```

Tagging

```
class Tagging < ActiveRecord::Base

  belongs_to :tag
  belongs_to :taggable, :polymorphic => true

  belongs_to :user

end
```

El plugin permite...

```
user.tags
```

```
post.tags
```

```
post.tag_with("css stylshet css4 w3c")
```

```
Post.find_tagged_with("rails")
```

¿Qué hacer con los tags?

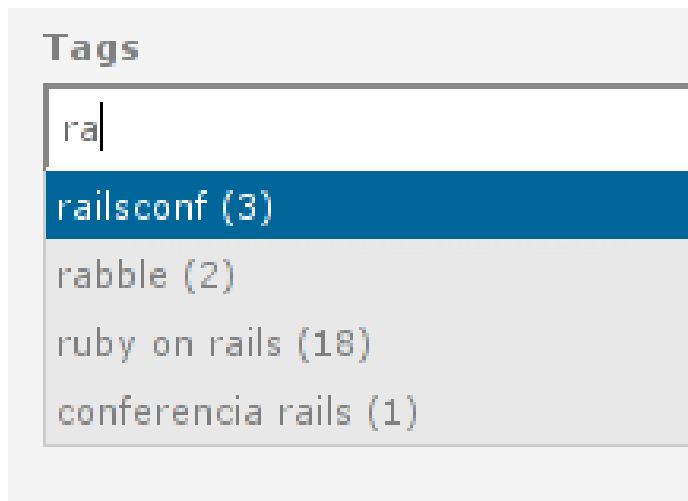
Nubes de tags

2006 actualidad agosto amigos amistad amor
aniversario antena3 arquitectura arte autor barcelona
blog blogs canarias canciones china ciencia cine citas
colombia comunicacion con cuatro cuentos cultura
curiosidades dbsk deporte deportes diario dios educacion
elecciones entrevista espana eta familia felicidad foto
fotografia **fotos** frases futbol gobierno google

- el primer día +4.000 tags diferentes
- ahora +40.000 tags diferentes
- +100.000 posts taggeados

¿Qué hacer con los tags?

Sugerir tags



- hay usuarios con +1.000 tags

¿Qué hacer con los tags?

Implementando una nube de tags:

ActiveRecord way:

```
tags = Tag.find(:all).sort do |a,b|  
  a.taggings.size <=> b.taggings.size  
end  
  
tags = tags[0...150]
```

- Tantos SELECT como tags tengamos

¿Qué hacer con los tags?

Implementando una nube de tags:

Optimizando con SQL:

```
select *,count(tag_id) as c from taggings group  
by tag_id order by c desc limit 150;
```

- 150 SELECT + 1

¿Qué hacer?

¡Cachear!

- No, si se está regenerando continuamente
- No, si, por muy poco que regeneremos la caché se genera una consulta de cientos de Mb que satura la máquina
- No es una solución a largo plazo

¿Entonces?

Desnormalizamos el modelo

- Combinamos en una tabla, atributos de varias
- Introducimos los campos calculados
- ...

Objetivo

- Reducir al mínimo el número de `SELECT`
- Poder utilizar el `LIMIT`
- Si has de ordenar, que sea a través de la BBDD
- Reducir al mínimo el número de atributos replicados
- Mantener la consistencia a base de reglas y *callbacks*
- Tratar cada situación de forma especial

temporal_tags

```
create_table "temporal_tags" do |t|
  t.column "name", :string
  t.column "count", :integer
end
```

temporal_user_tags

```
create_table "temporal_user_tags" do |t|
  t.column "tag_id", :integer
  t.column "user_id", :integer
  t.column "username", :string
  t.column "name", :string
  t.column "count", :integer
end
```

Mantenemos los modelos existentes y añadimos tablas, que no modelos, con los datos desnormalizados

¿Cómo mantener todo este caos
a raya?

Observers

- Un **observer** responde eventos sobre un objeto
- Se integra perfectamente con el modelo

```
config/environment.rb:
```

```
config.active_record.observers = :comment_observer
```

```
app/controllers/application.rb:
```

```
observer :comment_observer
```

```
class CommentObserver < ActiveRecord::Observer
  def after_save(comment)
    Notifier.deliver_comment("admin@lacoctelera.com",
      "Hay un nuevo comentario", comment)
  end
end
```

Callbacks

- `before_validation`
- `before_validation_on_create`
- `after_validation`
- `after_validation_on_create`
- `before_save`
- `before_create`
- `after_create`
- `after_save`

Funcionamiento

- Creamos un `tagging_observer`
- Definimos una acción para el *callback* `after_save`
- Se actualizan las tablas desnormalizadas globales, diarias y de usuario

```
class TTag < ActiveRecord::Base
  set_table_name "temporal_tags"
end

class TDDTag < ActiveRecord::Base
  set_table_name 'temporal_daily_tags'
end
```

tagging_observer.rb

```
class TaggingObserver < ActiveRecord::Observer
  def after_save(tag)
    begin #actualización
      t = TTag.find(tag.tag_id)
      t.update_attribute(:count, t.count+1)
    rescue # creación
      t = TTag.new
      t.id = tag.tag_id
      t.name = tag.name
      thetag = Tag.find_by_sql("select tags.id as id,
        tags.name as name, count(tag_id) as count
        from taggings, tags
        where taggings.taggable_type = '#{tag.taggable_type}'
        and taggings.tag_id = tags.id and tags.id = '#{tag.tag_id}'
        group by tag_id limit 1").first
      t.count = (thetag) ? thetag.count : 1
      t.save
    end
  end
end
```

tagging_observer.rb

```
class TaggingObserver < ActiveRecord::Observer
  def after_save(tag)
    begin #actualización
      t = TTag.find(tag.tag_id)
      t.update_attribute(:count, t.count+1)
    rescue # creación
      ...
    end
  end
end
end
```

tagging_observer.rb

```
...
rescue # creación
  t = TTag.new
  t.id = tag.tag_id
  t.name = tag.name
  thetag = Tag.find_by_sql("select tags.id as id,
    tags.name as name, count(tag_id) as count
    from taggings, tags
    where
      taggings.taggable_type='#{tag.taggable_type}'
    and
      taggings.tag_id = tags.id and
      tags.id = '#{tag.tag_id}'
    group by tag_id limit 1").first
  t.count = (thetag) ? thetag.count : 1
  t.save
end
end
end
```

Funcionamiento

- Repetiríamos el mismo método para el resto de tablas de tags
- *script* de carga “inicial” de los datos
- MySQL: `Engine=Memory`
- Tests

Tests

Tests: antes

```
def test_post_create_with_tags

  cine_before = TTag.find_by_name('citas')
  assert cine_before.count, 2

  terror_before = TTag.find_by_name('frases')
  assert_nil terror_before
```

Tests: durante

```
get :new, {}, {'user' => User.find(1000) }
assert_response :success

post :create,
  {"publish"=>"Publicado",
  "post"=>{"body"=>"Cuerpo sano en mente sana",
  "title"=>"Toma yogur y haz sudokus", "id"=>""},
  "user_id"=>"1000"},
  {"tag" => { "name" => "citas frases" } },
  {'user' => User.find(1000) }

assert_response :redirect
```

Tests: después

```
cine_after = Ttag.find_by_name('citas')
assert cine_after.count, 3

terror_after = Ttag.find_by_name('frases')
assert_not_nil terror_after
assert terror_after.count, 1
```

Tests

- Testear todos los modelos desnormalizados
- Testear todos los *callbacks* que utilicemos
- Testear cada posibilidad dentro de cada *callback*

No tiene porqué haber inconsistencia en los datos

Conclusiones

Conclusiones

- Ten cuidado con ActiveRecord y conoce a fondo en qué consultas se traduce
- Analiza logs y estudia comportamientos anormales: dispatch de cientos de megas, demasiadas consultas por petición, etcétera
- ¿Vale la pena empezar directamente con un modelo desnormalizado?
- Si tienes cuidado no tiene porqué llegar a darse una situación de inconsistencia
- Testea lo que haces



¿Preguntas?



Salamanca 17

Madrid – 28020
España

tel. +34 91 567 0605

www.the-cocktail.com