Gastón Ramos - ramos.gaston@gmail.com

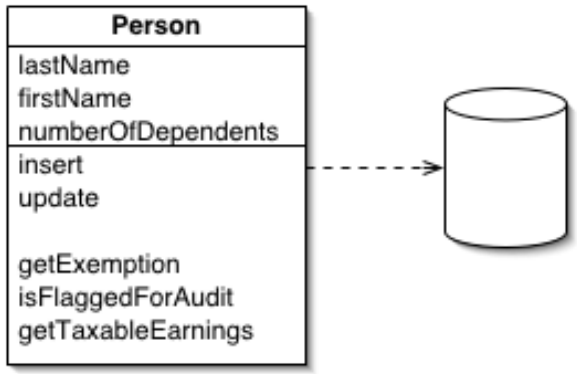
- http://gastonramos.wordpress.com/
- http://rubyargentina.soveran.com/
- http://www.rubylit.com.ar/

Agenda.

Active Record es un Patrón de Diseño.

Basado en el patrón ActiveRecord de Martin Fowler ("Patterns of Enterprise Architecture")

"Un objeto que engloba una fila de una tabla o vista de la base de datos, encapsula el acceso a la base de datos, y agrega lógica del dominio del problema sobre estos datos."



La biblioteca de Ruby Active Record .

"Nunca he visto una implementación de Active Record tan completa y tan útil como la de rails."

Martin Fowler

Active Record sigue el standard de ORM.

Active record sigue el standard de ORM y se diferencia de los demás por que minimiza la cantidad de configuración mediante el uso de un conjunto de convenciones.

Active Record sigue el standard de ORM.

- Una clase por tabla.
- Un objeto por registro.
- Las columnas como atributos de estos objetos.

Convención sobre configuración

Una clase por Tabla.

Código SQL para crear la tabla users:

```
CREATE TABLE `users` (
  `id` int(11) NOT NULL auto_increment,
  `login` varchar(255) default NULL,
  `crypted_password` varchar(255) default NULL,
  `email` varchar(25,5) default NULL,
   PRIMARY KEY  (`id`) ) ENGINE=InnoDB
```

Una clase por Tabla.

Código del modelo en Active Record (Ruby):

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

Uso de convenciones:

- Código

- Errores

+ Productividad

Convención sobre configuración.

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

- Sín Archivos XML.
- Reflexión y extensiones en runtime.
- Parece magia.

Columnas y atributos.

- Los objetos de Active Record se corresponden con las filas o registros de una tabla de la base de datos.
- Sin embargo hemos visto que no hay atributos en nuestas definiciones de clases.
- Esto por que Active Record los determina **dinámicamente** en runtime.
- Active Record "mira" el esquema dentro de la base de datos y configura las clases que mapean las tablas.

Expresividad en el código

Mucha Información en pocas líneas de código

```
class User < ActiveRecord::Base
  has_many                    :posts
  belongs_to                  :group

  validates_presence_of       :login, password
  validates_uniqueness_of     :login
  validates_confirmation_of   :password
end
```

Expresividad: Código más bello

+ Motivación

- Stress

+ Ganas

+ Productividad

La biblioteca de Ruby Active Record .

Active Record fue construida para **Ruby on Rails**, y hace que sea fácil el CRUD.

Ya va por la versión 2.1.0 y viene con la versión 2.1 de RoR.

- Create.
- Read.
- Update.
- Delete.

Tablas y clases.

Por defecto **Active Record asume que el nombre de la tabla es la forma plural de nombre de la clase**.
Si el nombre de la clase contiene múltiples palabras capitalizadas, el nombre de la tabla lleva guión bajo entre estas palabras.

Class Name	Table Name	Class Name	Table Name
Order	orders	LineItem	line_items
TaxAgency	tax_agencies	Person	people
Batch	batches	Datum	data
Diagnosis	diagnoses	Quantity	quantities

Pero sin no te gusta, lo podés cambiar!.

```
class Persona < ActiveRecord::Base
  set_table_name "persona"
end
```

Primary Key.

- Active Record asume que cada tabla ha de tener una clave primaria (normalmente llamada id).
- Se asegura que este campo id sea único para cada registro agregado en la tabla.
- Esta es una convención que puede no gustarle a algunos puristas.

¿Por que debemos usar una clave primaria artificial como id ?

Primary Key.

La razón es puramente práctica!

El formato de los datos externos puede cambiar con el paso del tiempo.

- Por ejemplo, tenemos una base de datos con expedientes, el número del expediente bien podría ser nuestra clave primaria.
- ¿Que pasa si después de un año se decide anteponerle una letra X a todos los números de expedientes?
- Tendríamos que cambiar todos los id en nuestro esquema, y actualizar todas las relaciones a la tabla expediente.
- Todas estas cosas requieren trabajo.

Primary Key.

- Todos estos problemas desaparecen si usamos nuestro propio valor interno como primary key.
- Si vamos a comenzar un proyecto nuevo trataremos de seguir las convenciones, para tener que trabajar menos.
- Y si tenemos que trabajar con una base de datos existente podemos cambiar esta convención.

```
class Expediente < ActiveRecord::Base
  self.primary_key = "nro_expediente"
end
```

Primary Key, Composite Key.

- Normalmente Active Record se toma el cuidado de crear un nuevo valor de primary key cuando agregamos un nuevo registro a un tabla.
- Si nosotros no seguimos la convención y utilizamos nuestro propio campo de id, debemos encargarnos de poner un valor de id único antes de guardar un nuevo registro.
- ¿Qué pasa con las claves primarias compuestas?
- En principio AR no las soporta, pero podemos utilizar algún plugin, **http://compositekeys.rubyforge.org/**

Conexión con la Base de Datos

- Active Record viene con soporte para DB2, Firebird, Frontbase, MySQL, Openbase, Oracle, Postgres, SQLite, SQL Server, and Sybase databases.
- Una de las formas de conectarnos a la Base de Datos es mediante el uso del método de clase **establish_connection**
- Cada conector tiene un pequeño conjunto de parámetros conexion diferente.

Conexión con la Base de Datos

```
ActiveRecord::Base.establish_connection(
  :adapter  => "mysql",
  :host     => "rubylit.com.ar",
  :database => "wiki",
  :username => "railsuser",
  :password => "securepw"
)
```

Asociaciones

```
belongs_to

has_one

has_many

has_and_belongs_to_many
```

Asociaciones

```
class Project < ActiveRecord::Base

  belongs_to                  :portfolio
  has_one                     :project_manager
  has_many                    :milestones
  has_and_belongs_to_many     :categories

end
```

Asociaciones

```
Project#portfolio

Project#project_manager

Project#milestones

Project#categories
```

Asociaciones

```
Project#portfolio = portfolio

Project#portfolio.nil?

Project#project_manager = project_manager,

Project#project_manager.nil?

Project#milestones.empty?
```

Asociaciones

```
Project#milestones.size

Project#milestones << milestone

Project#milestones.delete(milestone)

Project#milestones.find(milestone_id)

Project#milestones.find(:all, options)
```

Asociaciones

```
Project#milestones.create

Project#categories.empty?

Project#categories.size

Project#categories << category1

Project#categories.delete(category1)
```

Uno a Uno

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

class Office < ActiveRecord::Base
  belongs_to :employee
end
```

One to Many

```
class Manager < ActiveRecord::Base
  has_many :employees
end

class Employee < ActiveRecord::Base
  belongs_to :manager
end
```

Muchos a muchos

Hay dos formas de construir una relación de muchos a muchos, la primer forma usa **has_many** con la opción **:through** y un modelo de unión.

Muchos a muchos

```
class Assignment < ActiveRecord::Base
  belongs_to :programmer  # foreign key - programmer_id
  belongs_to :project     # foreign key - project_id
end

class Programmer < ActiveRecord::Base
  has_many :assignments
  has_many :projects, :through => :assignments
end

class Project < ActiveRecord::Base
  has_many :assignments
  has_many :programmers, :through => :assignments
end
```

Muchos a muchos

La segunda forma usa **has_and_belongs_to_many** en ambos modelos.

```
# foreign keys en la tabla de unión

class Programmer < ActiveRecord::Base
  has_and_belongs_to_many :projects
end

class Project < ActiveRecord::Base
  has_and_belongs_to_many :programmers
end
```

Find es el método principal en AR.

```
User.find(1)
```

```
#<User id: 1, name: "Pablo", login: "gaston", password: "pepe">

# SQL ejecutado:

SELECT * FROM `users` WHERE (`users`.`id` = 1)

# ---------------------------------------------------
```

```
User.first
```

```
#<User id: 1, name: "Pablo", login: "gaston", password: "pepe">

# SQL ejecutado:

SELECT * FROM `users` LIMIT 1
```

Find es el método principal en AR.

```
User.all
```

```
[#<User id: 1, name: "Pablo", login: "gaston", password: "pepe">,
#<User id: 2, name: "David Bner", login: "david", password: "ppp">,
#<User id: 3, name: "Pepito", login: "pepe", password: "1234">]

# SQL ejecutado:

SELECT * FROM `users`
```

Find con condiciones.

```
User.find(:all,
    :conditions => {:name => "david"}
)

# SQL ejecutado:

SELECT * FROM `users` WHERE (`users`.`name` = 'david')

# ----------------------------------------------------

User.find(:all,
   :conditions =>{ :first_name => "Bruce",
                   :last_name => "Lee" } )

# SQL ejecutado:

SELECT * FROM users WHERE
                (first_name ='Bruce' and last_name = 'Lee');
```

Order by.

```
User.find(:all, :order => "name desc")
```

```
[#<User id: 3, name: "Pepito", login: "pepe", password: "1234">,
#<User id: 1, name: "Pablo", login: "gaston", password: "pepe">,
#<User id: 2, name: "David Bner", login: "david", password: "ppp">]


#  SQL ejecutado ------------------------------

 Select * from Users ORDER BY name desc;
```

Group by.

```
User.find(:all, :group => "language")
```

```
# ----------------------------------------------
 SQL ejecutado:
 Select * from Users GROUP BY language;
```

Joins.

```
User.find(:all,
:join => "LEFT JOIN comments ON
                          comments.post_id = id")

# ---------------------------------------------

 SQL ejecutado:

 SELETC * FROM Users LEFT JOIN comments
 ON comments.post_id = id;
```

Creación.

```
user = User.new(:name => "David",
                :occupation => "Code Artist")

user.save

user.name # => "David"

# ---------------------------------------

User.create(:name => "Pepito", :login => "pepe",
            :password => "1234")
```

Otros Finds

Finds alternativos:

```
User.find_by_sql("SELECT * from users")

User.find_by_name_and_login("David", "dhh")

User.find_or_create_by_name('Bob', :age => 40)
```

Y más... Finds

Finds, like, select

```
User.find( :all,
 :conditions => ["name like ?" , "#{name}%" ])
```

```
Talks.find( :all,
          :select => "title, speaker, recorded_on" )
```

Validaciones

- Active Record puede validar el contenido de objeto del modelo.
- Estas validaciones se realizan automáticamente cuando el objeto se graba en la BD.
- Si las validaciones fallan el objeto no se guarda y queda en memoria con un estado inválido.
- Active Record puede distinguir entre objetos que corresponden a registros en la BD y los que no.
- `User.save User.new_record?`
- Update o insert según cada caso.

Validaciones

- `validate`

  (en cada operación de grabado)
- `validate_on_create`
- `validate_on_update`
- `User.valid?`

  (lo podés consultar en cualquier momento)

Validation Helpers

- Active Record tiene un conjunto de métodos "helpers" que agregan validaciones a nuestros modelos.
- El nombre no puede estar vacío.
- La edad debe ser entre 18 y 90 años, etc.
- Estas validaciones "comunes" las hacen los helpers.

Validation Helpers

```
validates_format_of

validates_uniqueness_of

validates_acceptance_of

validates_associated

validates_confirmation_of

validates_exclusion_of

validates_inclusion_of

validates_length_of

validates_numericality_of
```

Ejemplos de Validation Helpers

```
class User < ActiveRecord::Base

  validates_confirmation_of :password

end
```

```
class User < ActiveRecord::Base

  validates_length_of :password, :in => 6..20

  validates_length_of :address, :minimum => 10,
                      :message => "seems too short"

end
```

Callbacks

- Usando callbacks, Active Record te permite "pariticipar" en este proceso de monitoreo.
- Con los callbacks podemos escribir código se invocará en cada evento significante del objeto.
- Active Records define 20 callbacks.
- Por ejemplo **before_destroy** que se ejecuta antes de que el método **destroy** se llame.

model.save()

Nuevo registro

- before_validation
- before_validation_on_create
- after_validation
- before_save
- before_create

Insert

- after_create
- after_save

model.save()

Registro existente

- before_validation
- before_validation_on_update
- after_validation
- after_validation_on_update
- before_save
- before_update

Update

- after_update
- after_save

model.destroy()

- before_destroy

Delete

- after_destroy

Final

Si tenemos tiempo vemos un poco de práctica.

Final, Referencias

Fin, Gracias por escuchar

Referencias:

http://ar.rubyonrails.com/

Agile Web Development with Rails - Second Edition

ISBN: 0-9776166-3-0