# Ruby on Rails

Scopes/Validations

# Overview

- default_scope
- Named scopes
- Validaciones de ActiveRecord
- Validaciones personalizadas

# Default Scope

**`default_scope`**

Es un método de clase para especificar cómo se van a obtener los registros por defecto de la base de datos, en lugar de descansar en el ordenamiento/presentación por defecto de la base de datos.

```
irb(main):001:0> Hobby.pluck :name
   (0.1ms)  SELECT "hobbies"."name" FROM "hobbies"
=> ["Programming", "Music"]
```

# Como evitar el default_scope

Se utiliza unscoped para evitar traer los registros como definimos en el default_scope.

```ruby
class Hobby < ActiveRecord::Base
  has_and_belongs_to_many :people

  default_scope { order :name }
end
```

```
irb(main):001:0> Hobby.pluck :name
   (0.1ms)  SELECT "hobbies"."name" FROM "hobbies"  ORDER BY "hobbies"."name" ASC
=> ["Music", "Programming"]
irb(main):002:0> Hobby.unscoped.pluck :name
   (0.3ms)  SELECT "hobbies"."name" FROM "hobbies"
=> ["Programming", "Music"]
```

# Named Scopes

- scope :name, lambda

```
class Person < ActiveRecord::Base
    has_one :personal_info, dependent: :destroy
    has_many :my_jobs, class_name: "Job"
    has_and_belongs_to_many :hobbies
    has_many :jobs
    has_many :approx_salaries, through: :jobs, source: :salary_range

    def max_salary
      approx_salaries.maximum(:max_salary)
    end

    scope :ordered_by_age, -> { order age: :desc}
    scope :starts_with, -> (starting_string){ where("first_name LIKE ?", "#{starting_string}%")}
end
```

```
irb(main):001:0> Person.ordered_by_age.pluck :age
   (0.1ms)  SELECT "people"."age" FROM "people"  ORDER BY "people"."age" DESC
=> [75, 57, 33, 30, 27, 27]
irb(main):002:0> Person.ordered_by_age.starts_with("Jo").pluck :age, :first_name
   (0.3ms)  SELECT "people"."age", "people"."first_name" FROM "people" WHERE (first_name LIKE 'Jo%')  ORDER BY "peopl
e"."age" DESC
=> [[57, "Josh"], [27, "John"], [27, "John"]]
irb(main):003:0> Person.ordered_by_age.limit(2).starts_with("Jo").pluck :age, :first_name
   (0.2ms)  SELECT  "people"."age", "people"."first_name" FROM "people" WHERE (first_name LIKE 'Jo%')  ORDER BY "peop
le"."age" DESC LIMIT 2
=> [[57, "Josh"], [27, "John"]]
```

Un scope siempre retorna un ActiveRecord::Relation

Puede hacer nuestras consultas ActiveRecord mas simples.

# Validaciones

Normalmente queremos tener algún control sobre qué va a almacenarse en la base de datos.

No cualquier entrada es apropiada para ingresar a la base de datos.

Si existiesen validaciones fallidas, la información no debe ser guardada en la base de datos.

ActiveRecord provee muchos validadores pre-construidos.

# :presence and :uniqueness

- `presence: true`
  - Se asegura de que el dato no sea nulo
- `uniqueness: true`
  - Realiza un chequeo para verificar de que no existan registros en la base de datos qeu ya tengan el valor del atributo especificado

job.rb

```ruby
class Job < ActiveRecord::Base
  belongs_to :person
  has_one :salary_range

  validates :title, :company, presence: true
```

```
~/advanced_ar$ rails c
Loading development environment (Rails 4.2.3)
irb(main):001:0> job = Job.new
=> #<Job id: nil, title: nil, company: nil, position_id: nil, person_id: nil, created_at: nil, updated_at: nil>
irb(main):002:0> job.errors
=> #<ActiveModel::Errors:0x007fe0b0b03590 @base=#<Job id: nil, title: nil, company: nil, position_id: nil, person_id:
 nil, created_at: nil, updated_at: nil>, @messages={}>
irb(main):003:0> job.save
   (0.2ms)  begin transaction
   (0.1ms)  rollback transaction
=> false
irb(main):004:0> job.errors
=> #<ActiveModel::Errors:0x007fe0b0b03590 @base=#<Job id: nil, title: nil, company: nil, position_id: nil, person_id:
 nil, created_at: nil, updated_at: nil>, @messages={:title=>["can't be blank"], :company=>["can't be blank"]}>
irb(main):005:0> job.errors.full_messages
=> ["Title can't be blank", "Company can't be blank"]
```

# Otros validadores

- `:numericality` – valida que la entrada sea numérica
- `:length` – valida que la entrada tenga cierta longitud
- `:format` – realiza una validación contra una expresión regular
- `:inclusion` – valida que el valor está dentro de un rango específico.
- `:exclusion` – valida que el valor está fuera de un rango específico

# Validadores personalizados

- Se puede escribir un método y llamar a **`errors.add(columnname, error)`** cuando se encuentra una condición de error.
- Se especifica el nombre del método como símbolo en la sentencia **`validate`**.

```
class SalaryRange < ActiveRecord::Base
  belongs_to :job

  validate :min_is_less_than_max

  def min_is_less_than_max
    if min_salary > max_salary
      errors.add(:min_salary, "cannot be greater than maximum salary!")
    end
  end
end
```

```
irb(main):001:0> sr = SalaryRange.create min_salary: 30000.00, max_salary: 10000.00
   (0.1ms)  begin transaction
   (0.0ms)  rollback transaction
=> #<SalaryRange id: nil, min_salary: 30000.0, max_salary: 10000.0, job_id: nil, created_at: nil, updated_at: nil>
irb(main):002:0> sr.errors
=> #<ActiveModel::Errors:0x007fe0acea87a8 @base=#<SalaryRange id: nil, min_salary: 30000.0, max_salary: 10000.0, job_
id: nil, created_at: nil, updated_at: nil>, @messages={:min_salary=>["cannot be greater than maximum salary!"]}>
irb(main):003:0> sr.errors.full_messages
=> ["Min salary cannot be greater than maximum salary!"]
irb(main):004:0> sr.save!
   (0.2ms)  begin transaction
   (0.0ms)  rollback transaction
ActiveRecord::RecordInvalid: Validation failed: Min salary cannot be greater than maximum salary!
```

# Entonces...

Las validaciones nos permiten tener control sobre lo que ingresa a la base de datos.

Para más información sobre ActiveRecord:

- http://guides.rubyonrails.org/active_record_basics.html
- http://guides.rubyonrails.orgactive_record_querying.html
- http://guides.rubyonrails.org/association_basics.html
- http://guides.rubyonrails.org/active_record_callbacks.html