



# Clase 09:

## Validadores y relaciones en Ruby on Rails



profesor: Patricio López Juri { [patricio@lopezjuri.com](mailto:patricio@lopezjuri.com) }  
créditos: 10  
horario: J:7-8  
sala: H3



**Supuesto que hago  
yo en este curso:**

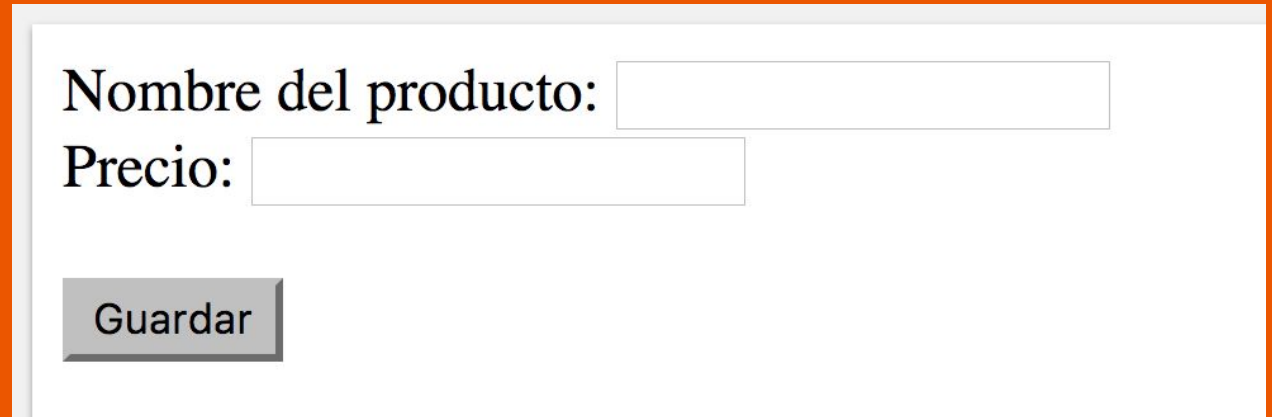
*Ustedes son personas inteligentes y pueden aprender y deducir reglas lógicas en base a ejemplos.*



# Repaso

- ¿Qué es y qué hace un *generador*?

# Veamos el siguiente formulario:



Nombre del producto:

Precio:

# ¿Qué es lo peor que puede pasar?

Nombre del producto:

Precio:

Guardar



## Un usuario podría ingresar:

- Datos vacíos, muy cortos o muy largos
- Datos incorrectos
- Datos inválidos según el contexto
  - Un correo sin @
  - Número negativos para un precio
  - Fechas pasadas o muy futuras
  - etc.




## **Los formularios suelen ser puntos delicados de una aplicación web.**

Es importante para evitar errores en la aplicación, consistencia de datos e incluso el fracaso de un proyecto puede deberse al mal contenido.

---

# VALIDADORES





## Existen al menos dos formas de validar el contenido en una app web.

1. Una opción es **validarlo a nivel de modelo**. Es decir, justo antes que se guarde en la **base de datos**.
  - a. Por ejemplo podemos impedir que la gente cree cuentas sin nombre, correos inválidos, etc.
  - b. Es decir, **validamos a nivel de datos**.
2. **Validar a nivel de controlador**. Esto es **validar las reglas de negocio**.
  - a. Ejemplo: que el usuario no pueda borrar un comentario que no es suyo.
  - b. Esto va más allá de los datos, sino que de las relaciones.

---

# Validando a nivel de modelo

**Documentación oficial:**

[http://guides.rubyonrails.org/active\\_record\\_validations.html](http://guides.rubyonrails.org/active_record_validations.html)



Si se nos olvida los campos de un modelo los podemos ver en **db/schema.rb**

```
ActiveRecord::Schema.define(version: 20171012224913) do
  create_table "products", force: :cascade do |t|
    t.string "name"
    t.text "description"
    t.integer "cost"
    t.boolean "available"
    t.datetime "created_at", null: false
    t.datetime "updated_at", null: false
  end
end
```

Definición de *Product*



## El modelo *Product*, pero sin validaciones

```
class Product < ActiveRecord::Base  
end
```

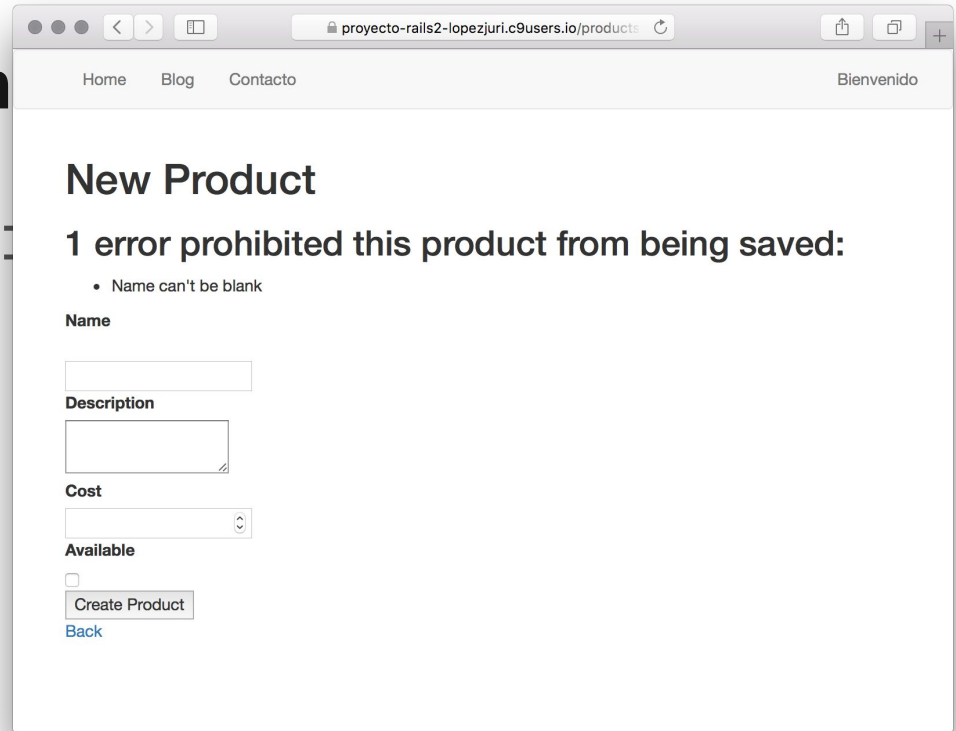


**Validemos que el nombre (*name*) no esté vacío**

```
class Product < ActiveRecord::Base
  validates :name, presence: true
end
```

Validemos que el nom

```
class Product < ActiveRecord::Base
  validates :name,
end
```



The screenshot shows a web browser window with the address bar displaying 'proyecto-rails2-lopezjuri.c9users.io/products'. The page has a navigation bar with links for 'Home', 'Blog', and 'Contacto', and a 'Bienvenido' message on the right. The main content area is titled 'New Product' and displays a validation error: '1 error prohibited this product from being saved:'. Below this, a bulleted list indicates the error: '• Name can't be blank'. The form fields are labeled 'Name', 'Description', 'Cost', and 'Available'. The 'Name' field is empty, while the others contain placeholder text. At the bottom of the form, there is a 'Create Product' button and a 'Back' link.

Home Blog Contacto Bienvenido

## New Product

1 error prohibited this product from being saved:

- Name can't be blank

Name

Description

Cost

Available

Create Product

[Back](#)

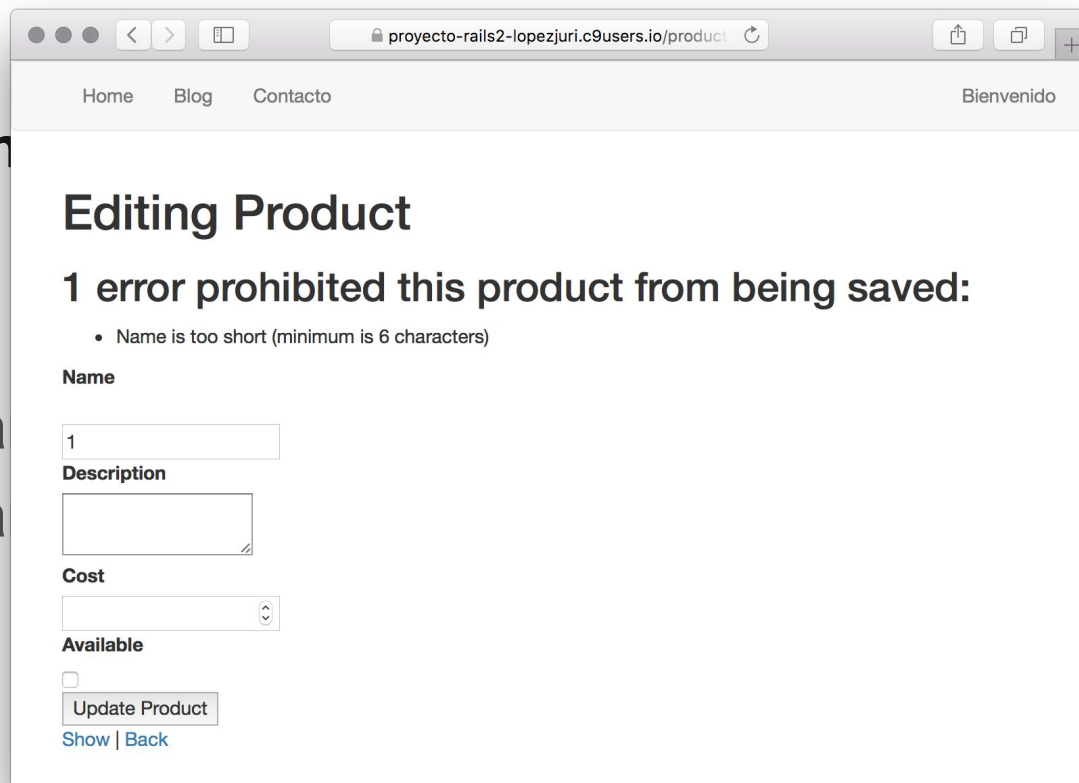


**Pero ahora si pone una sola letra? Además validemos el largo.**

```
class Product < ActiveRecord::Base
  validates :name, presence: true
  validates :name, length: { minimum: 6 }
end
```

Pero ahora si pone un  
largo.

```
class Product <
  validates :name, length: { minimum: 6 }
end
```



The screenshot shows a web browser window with the address bar displaying "proyecto-rails2-lopezjuri.c9users.io/product". The page has a navigation bar with links for "Home", "Blog", and "Contacto", and a "Bienvenido" message on the right. The main content area is titled "Editing Product" and displays a validation error: "1 error prohibited this product from being saved:". Below this, a bulleted list states "Name is too short (minimum is 6 characters)". The form includes fields for "Name" (containing "1"), "Description" (empty), "Cost" (a numeric input field), and "Available" (a checkbox). At the bottom of the form are an "Update Product" button and two links: "Show" and "Back".



# Actividad en clase

---



## A lo que ya llevamos vamos a agregar validaciones.

- Validar que el nombre del producto sea más largo que 6
- Validar que el largo la biografía sea mayor que 16 y menor o igual que 300.
- Validar que el costo del producto sea positivo y sin decimales.

### Avanzado:

- Validar además que la biografía pueda ir vacía. Es decir, puede estar vacía o con largo entre 16 y 300. Debe funcionar con ambos casos.
- Al momento de ingresar un nombre vacío, debe mostrar un mensaje de error personalizado.



---

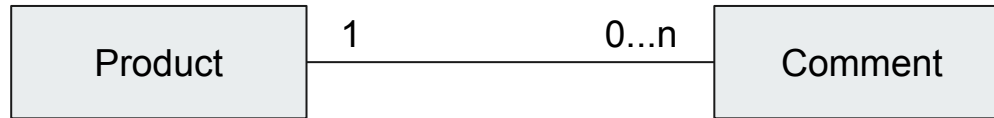
# RELACIONES


**Nos gustaría poder hacer relaciones entre modelos.**

Por ejemplo: Que un Producto tenga Comentarios.




**Las relaciones nos permiten asociar instancias de modelos,** esto es lo mismo que se ve en el ramo de Base de Datos






**Consideración:** en base de datos solemos asignar un `primary_key` a un atributo que creemos único como el RUT o Email.



**Consideración:** en base de datos solemos asignar un `primary_key` a un atributo que creemos único como el RUT o Email.

En Ruby on Rails, por defecto, **se usa un ID autogenerado número y que incrementa automáticamente.**



**Consideración:** en base de datos solemos asignar un `primary_key` a un atributo que creemos único como el RUT o Email.

En Ruby on Rails, por defecto, **se usa un ID autogenerado número y que incrementa automáticamente.**

## ¿Por qué?





# Tipos de relaciones

Como es de esperarse:

- 1 a 1
- 1 a  $n$
- $n$  a  $n$



# Tipos de relaciones en Rails

En el caso de Ruby on Rails se usa la siguiente declaración en los modelos:

- `belongs_to`
- `has_many`
- `has_many :through`
- `has_one`
- `has_one :through`
- `has_and_belongs_to_many`



Usaremos el ejemplo oficial de Rails:

[http://guides.rubyonrails.org/association\\_basics.html](http://guides.rubyonrails.org/association_basics.html)

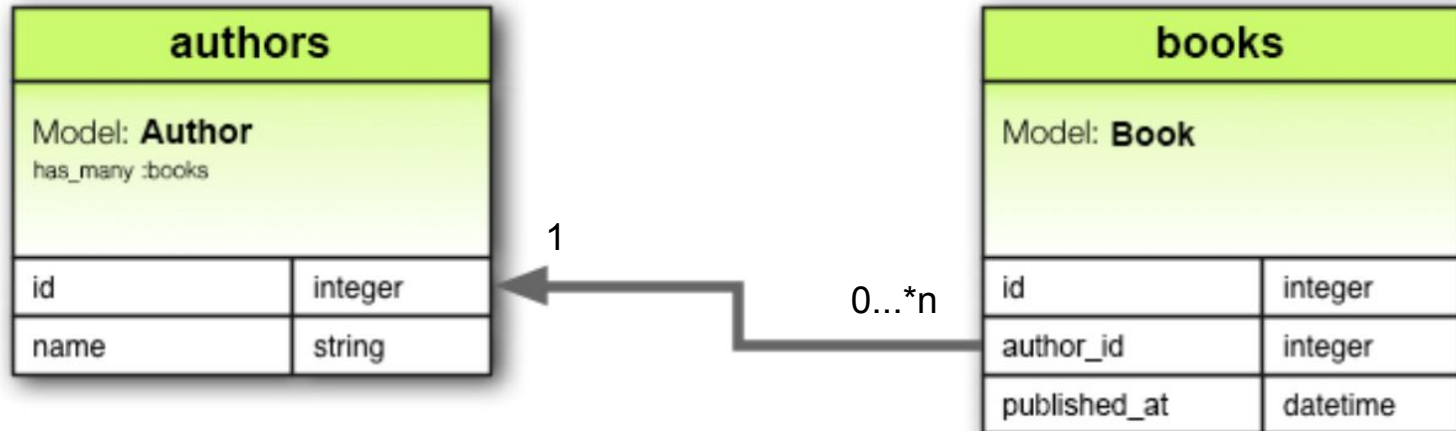


```
class Author < ApplicationRecord
  has_many :books
end
```

Un autor tiene muchos libros.  
Un libro tiene un autor.



```
class Book < ApplicationRecord
  belongs_to :author
end
```





La manera más sencilla de hacer asociaciones es al momento de generar con scaffold un modelo.

```
rails generate scaffold Author name:string
```

```
rails generate scaffold Book title:string author:references
```

