



Clase 10:

Relaciones en Ruby on Rails



profesor: **Patricio López Juri** { 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

- ¿Por qué es importante validar en los modelos?
- Habían dos tipos de validaciones, ¿cuáles eran?

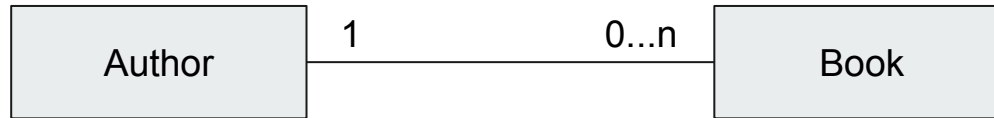
RELACIONES


Nos gustaría poder hacer relaciones entre modelos.

Por ejemplo: Que un *Autor* tenga *Libros*.




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 numérico 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

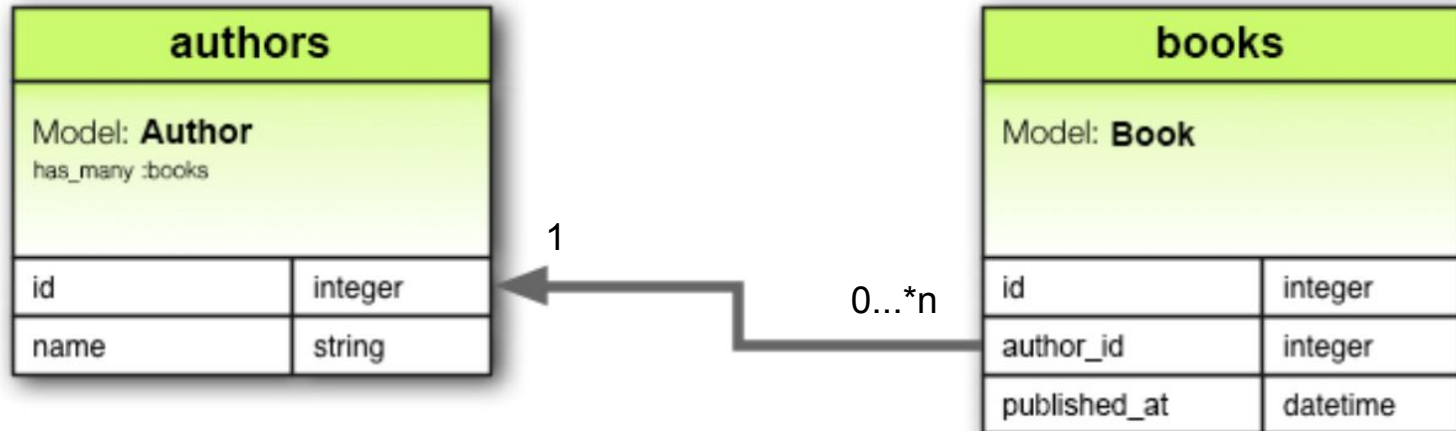


```
class Author < ApplicationRecord
  has_many :books
end
```

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



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



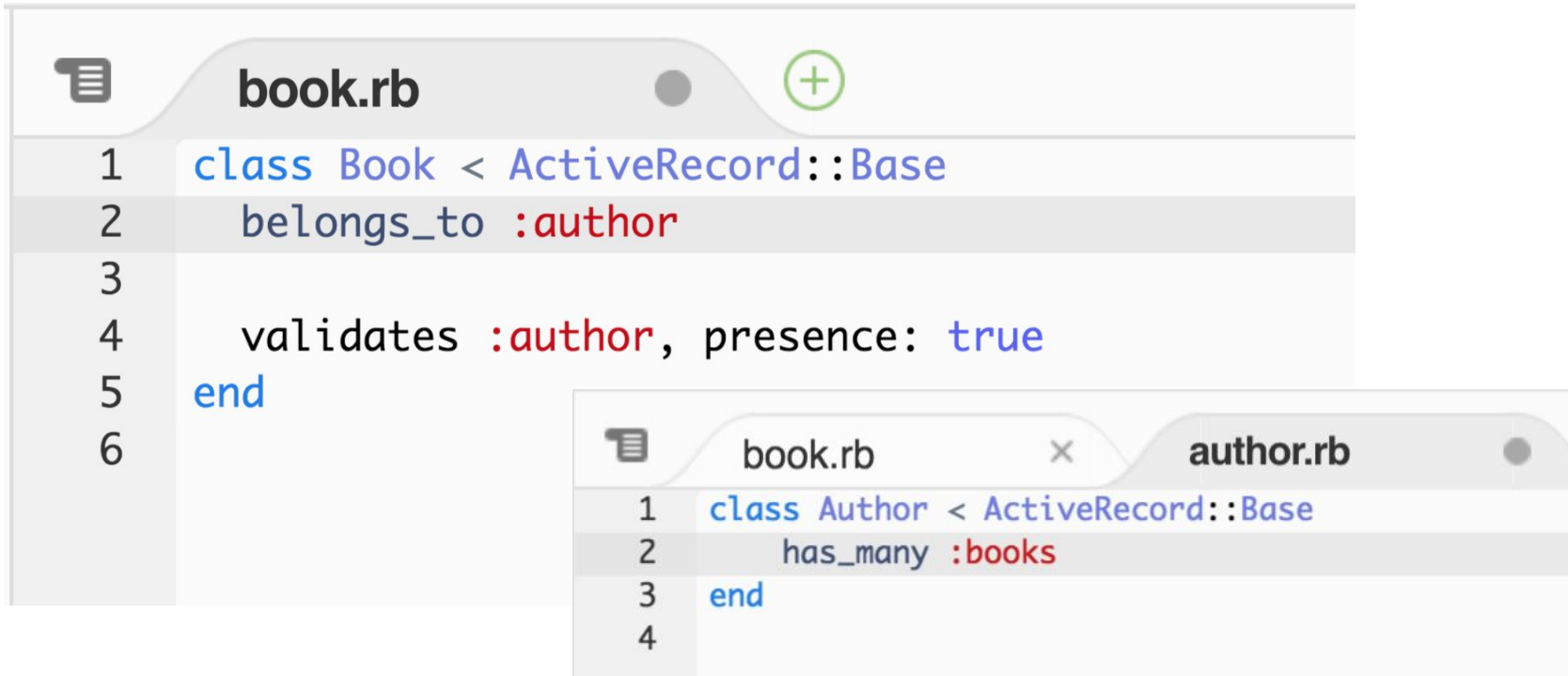


La manera más sencilla de hacer asociaciones es al momento de generar con scaffold un modelo nuevo.

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

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

Hay que modificar `app/models/author.rb` en este caso.



```
1 class Book < ActiveRecord::Base
2   belongs_to :author
3
4   validates :author, presence: true
5 end
6
```

```
1 class Author < ActiveRecord::Base
2   has_many :books
3 end
4
```

Luego ejecutar las migraciones con:
`rake db:migrate`

...y si ya creamos el modelo
¿Cómo lo modificamos?

agregar campos, crear relaciones, etc.

Estas son las *migraciones*

En la terminal cuando usamos

```
rake db:migrate
```

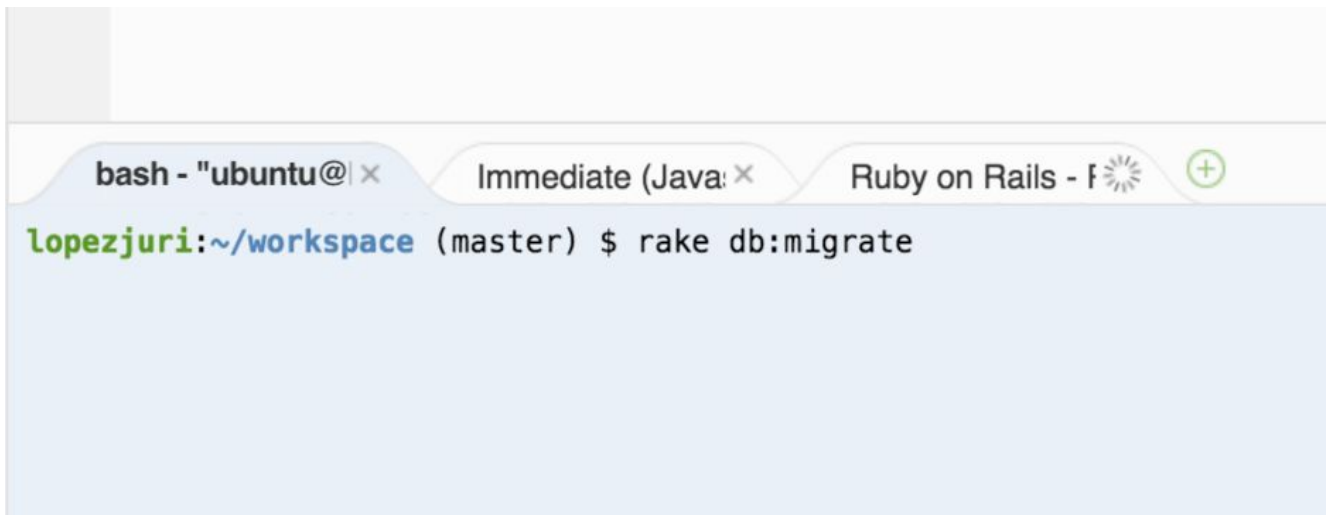


Migraciones son para ejecutar cambios en la base de datos

- Las bases de datos SQL tienen esquemas estrictos.
- Modificar algo en los modelos en Ruby no modificará la base de datos.
- Para hacer los cambios se hacen migraciones.
- Estas se pueden crear manualmente o las puede generar Rails.

En la terminal, las ejecutamos con:

rake db:migrate

A screenshot of a terminal window with three tabs: 'bash - "ubuntu@| x', 'Immediate (Java: x', and 'Ruby on Rails - f' with a loading icon and a green plus icon. The active tab is 'bash - "ubuntu@| x'. The terminal text shows the prompt 'lopezjuri:~/workspace (master) \$' followed by the command 'rake db:migrate'.

```
lopezjuri:~/workspace (master) $ rake db:migrate
```

Un ejemplo de una migración que agrega un campo a un modelo: agregar pages a Book

```
rails generate migration AddPagesToBooks pages:integer
```

ruby - "ubuntu@l x

Immediate (Java: x

Ruby on Rails - f



```
lopezjuri:~/workspace (master) $ rails generate migration AddPagesToBooks pages:integer
```

```
Running via Spring preloader in process 4250
```

```
  invoke  active_record
```

```
  create  db/migrate/20171026163830_add_pages_to_books.rb
```

```
lopezjuri:~/workspace (master) $ █
```



Veamos si entendieron: escriba las migraciones necesarias para:

- Agregar un **apodo** a los **usuarios**
- Agregar **color**, **tamaño** y **marca** a los **productos**.
- Quitar el color a los productos.
- Agregar **comentarios** a los **productos**.



Veamos si entendieron: escriba las migraciones necesarias para:

- `rails g migration AddNicknameToUsers nickname:string`
- `rails g migration AddColorToProducts color:string size:integer brand:string`
- `rails g migration RemoveColorFromProducts color:string`
- `rails g migration AddProductToComments product:references`



Actividad en nuestro proyecto

Vamos a usar git para respaldar los cambios. En la terminal (*bash*) ejecutar:

- `git init`
- `git status` # deberían aparecer muchos archivos en rojo
- `git add --all`
- `git status` # deberían aparecer muchos archivos en verde
- `git commit -m "Inicio"`
- `git status` # no deberían aparecer archivos



Actividad en nuestro proyecto

- Crear un modelo con scaffold llamado BlogPost, este debe tener los campos que ustedes crean necesarios.
- Crear el modelo BlogComment con los campos que crean necesarios.

Debe existir una asociación entre ellos. Un BlogPost puede tener muchos comentarios. Un BlogComment debe tener siempre un BlogPost.

Si algo sale mal:

- `git clean -fd && git checkout -- . # borra los cambios nuevos`
- `rake db:reset # resetea la base de datos`

Ver video en YouTube:

Relaciones en Ruby on Rails

https://youtu.be/iz6L_9xD7Ck

