Relaciones N a N en los modelos

Modelos con relaciones N a N (Parte I)

{desafío} latam_

Implementar relaciones N a N en los modelos utilizando Ruby on Rails para satisfacer un requerimiento.

- Unidad 1: Relaciones 1 a N.
- Unidad 2: Mecanismos de autenticación y control de accesos en una aplicación web.
- Unidad 3: Relaciones N a N en los modelos.
- Unidad 4: Utilizar Amazon S3 en proyecto en Rails.
- Unidad 5: Pruebas unitarias y funcionales.

Te encuentras aquí

{desafío} latam_

¿Qué aprenderás en esta sesión?

- *Implementación de relaciones N a N con dos modelos.*

¿Cuándo es necesario crear las vistas de devise?

{desafío} latam_

Modelos con relaciones N a N (Parte I)

Conceptos Importantes

- Modelos
- Relaciones N a N
- E-commerce
- Relaciones 1 a N
- Normalización

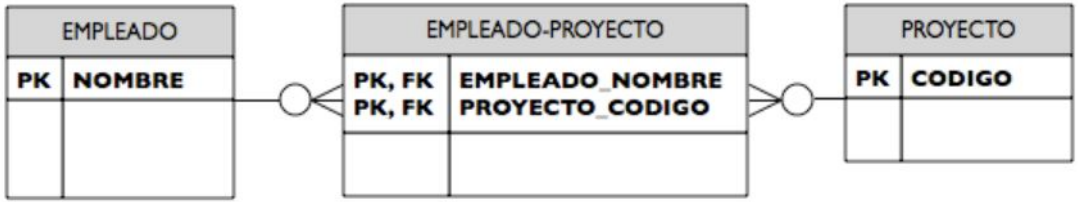
{desafío} latam_

/* Implementación de relaciones N a N con dos modelos. */

Implementación de relaciones N a N con dos modelos.

2 modelos 3 tablas

- Las relaciones N a N no existen en una base de datos normalizada.
- Toda relación N a N debe ser convertida a una serie de relaciones 1 a N con tabla intermedia.
- La tabla intermedia no siempre se ve reflejada en los modelos de rails.



Ejercicio guiado:

Catálogo de productos

{desafío}
latam_

Catálogo de productos

Contexto

Necesitamos desarrollar un catálogo de productos donde el usuario pueda revisar los productos que se encuentran en él y poder determinar entre muchas opciones cuál es su postura ante el producto. Además, los productos tendrán categorías asociadas para poder ordenarlos en un futuro.

A lo largo de las siguientes presentaciones iremos completándolo paso a paso para poder cumplir con estos requerimientos.

Catálogo de productos

- **Paso 1:** Creamos el proyecto, utilizando PostgreSql como base de datos.

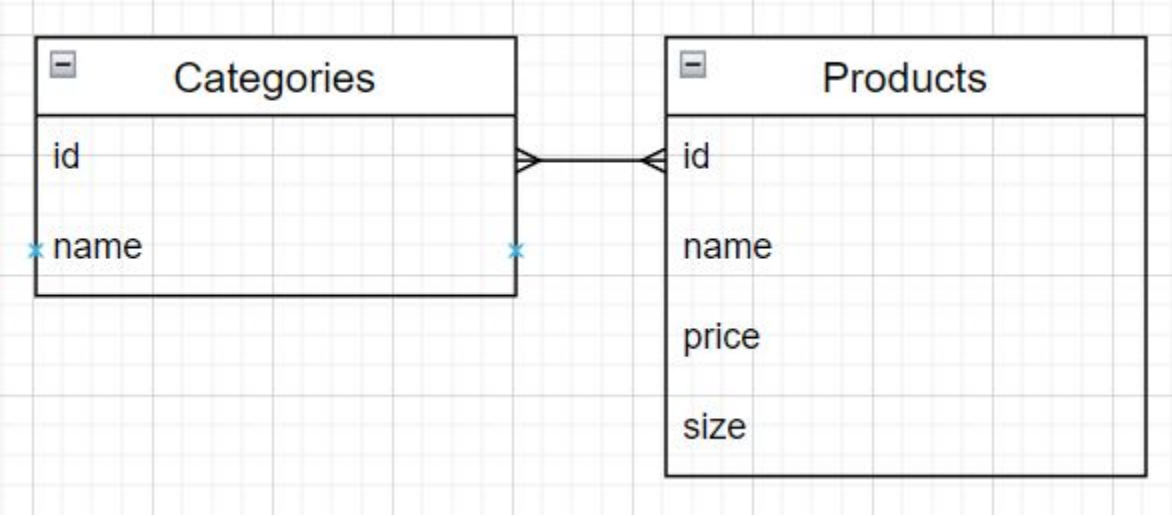
```
#bashrc
rails new Catalog -d postgresql
```

- **Paso 2:** Creamos la base de datos.

```
#bashrc
rails db:create
```

Catálogo de productos

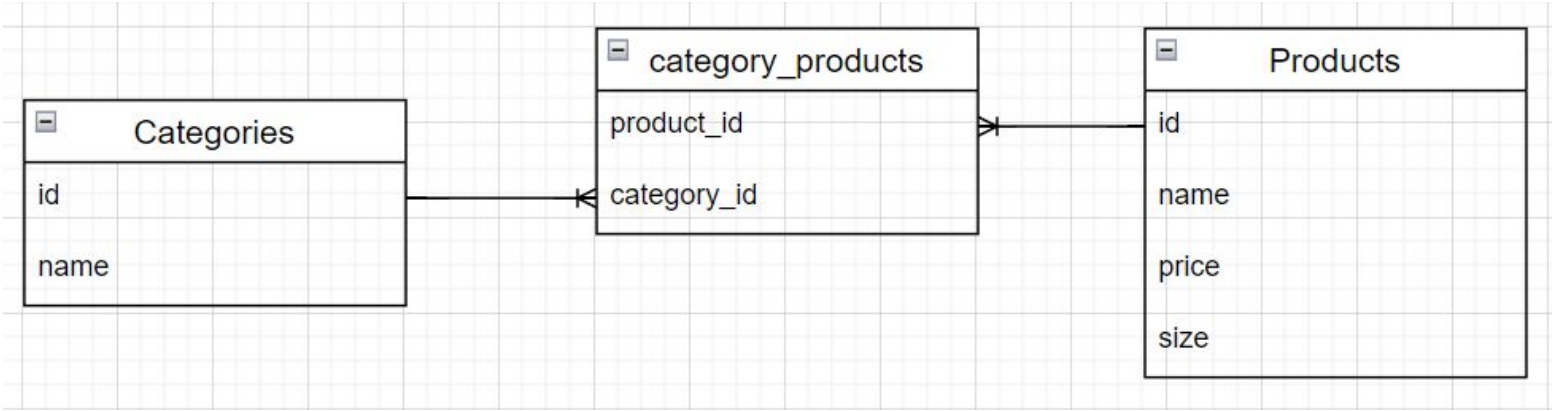
- **Paso 3:** Identificar entidades



{desafío} latam_

Catálogo de productos

- **Paso 4:** Normalizar tablas.



Catálogo de productos

Solución

- **Paso 5:** Creamos el modelo category.

```
#bashrc
rails g scaffold category name
```

- **Paso 6:** Creamos el modelo productos.

```
#bashrc
rails g scaffold product name price:integer size:integer
```

Catálogo de productos
Solución

- **Paso 7:** Creamos tabla intermedia entre categorías y productos
  7.1: Creamos migración

```
#bashrc
rails g migration CreateJoinTableCategoriesProducts category product
```

7.2: Revisamos migración

```ruby
class CreateJoinTableCategoriesProducts < ActiveRecord::Migration[7.0]
  def change
    create_join_table :categories, :products do |t|
      # t.index [:category_id, :product_id]
      # t.index [:product_id, :category_id]
    end
  end
end
```

Catálogo de productos

Solución

- **Paso 8:** Corremos la migración en la base de datos.

```
#bashrc
rails db:migrate
```

- **Paso 9:** Agregamos relaciones a los modelos.

  9.1: Agregamos relación a category

```
#app/models/category.rb
class Category < ApplicationRecord
  has_and_belongs_to_many :products
end
```

Catálogo de productos
Solución

- **Paso 9:** Agregamos relaciones a los modelos.
  9.2: Agregamos relación a product

```
#app/models/product.rb
class Product < ApplicationRecord
  has_and_belongs_to_many :categories
end
```

- **Paso 10:** Agregamos la gema Faker.

```
#bashrc
bundle add faker
```

Catálogo de productos
Solución

- **Paso 11:** Validamos que la categoría tenga nombre único.

```
#app/models/category.rb
validates :name, uniqueness: true
```

Catálogo de productos

Solución

- **Paso 12:** Rellenamos las categorías con datos de prueba, para ello utilizaremos el archivo **seeds.rb**.

```
#db/seeds.rb
while Category.count < 10
  if !Category.pluck(:name).include?(Faker::Game.genre)
    unique_name = Faker::Game.genre
    Category.create(name: unique_name)
  end
end
```

Catálogo de productos

Solución

- **Paso 13:** Definimos **/products** como la ruta raíz de la aplicación.

```
#config/routes.rb
root "products#index"
```

Catálogo de productos
Solución

- **Paso 14:** Al crear un producto le asociamos una categoría.
  14.1: Agregar campo al controlador

```
#app/controllers/product_controller.rb
def product_params
 params.require(:product).permit(:name, :price, :size, :category_ids => [])
end
```

Catálogo de productos
Solución

- **Paso 14:** Al crear un producto le asociamos una categoría.
  14.2: Agregar campos en formulario

```
#app/views/products/_form.html.erb
<div>
  <%= form.label :category_ids, style: "display: block" %>
  <%= form.select :category_ids, options_from_collection_for_select(@categories,
:id, :name) %>
</div>
```

Catálogo de productos
Solución

- **Paso 14:** Al crear un producto le asociamos una categoría.
  14.3: Modificamos métodos del controlador

```
#app/controllers/product_controller.rb
before_action :set_categories, only: %i[ new edit create update]

def set_categories
  @categories = Category.all
end
```

Catálogo de productos

Solución

- **Paso 14:** Al crear un producto le asociamos una categoría.
  14.4: Modificamos el método create

```
#app/controllers/product_controller.rb
def create
  @product = Product.new(product_params)
  respond_to do |format|
    if @product.save
      format.html { redirect_to product_url(@product), notice: "Product was successfully created." }
      format.json { render :show, status: :created, location: @product }
    else
      format.html { render :new, status: :unprocessable_entity }
      format.json { render json: @product.errors, status: :unprocessable_entity }
    end
  end
end
```

Catálogo de productos

Solución

- **Paso 14:** Al crear un producto le asociamos una categoría.
  14.5: Creamos nuestro primer producto

Catálogo de productos

Solución

- **Paso 14:** Al crear un producto le asociamos una categoría.
  14.6: Revisamos resultado en terminal

{desafío} latam_

Catálogo de productos

Solución

- **Paso 15:** Revisamos creación del objeto por consola

```
3.1.1 :001 > ap Product.last
  Product Load (0.2ms)  SELECT "products".* FROM "products" ORDER BY
#<Product:0x00007efcafd0f2c0> {
            :id => 5,
          :name => "Metaverse",
         :price => 2000,
          :size => 45,
    :created_at => Thu, 08 Dec 2022 22:50:52.434203000 UTC +00:00,
    :updated_at => Thu, 08 Dec 2022 22:54:16.208803000 UTC +00:00
```

Catálogo de productos

Solución

- **Paso 16:** Revisamos la relación entre productos y categorías por consola.

```
3.1.1 :002 > ap Product.last.categories
  Product Load (0.4ms)  SELECT "products".* FROM "products" ORDER BY "p
  Category Load (0.4ms)  SELECT "categories".* FROM "categories" INNER
[
    [0] #<Category:0x00007efcafa8ea78> {
                  :id => 1,
                :name => "Simulation",
          :created_at => Thu, 08 Dec 2022 21:37:29.948726000 UTC +00:00,
          :updated_at => Thu, 08 Dec 2022 21:37:29.948726000 UTC +00:00
    }
]
```

Ejercicio propuesto

“Agregando Usuarios”

{desafío} latam_

Agregando Usuarios

Debemos agregar usuarios a nuestro catálogo de productos, como primeros requisitos se pide lo siguiente:

- Realizar autenticación básica con Devise.
- Agregar Bootstrap al proyecto (método libre).
- Un usuario puede ir a cerrar sesión, iniciar sesión y/o registrarse, desde un navbar de navegación.

Si utilizamos 2 modelos, 3 tablas ¿Podemos acceder a la información de la tabla intermedia?

Próxima sesión...

- *Implementación de relaciones N a N con tres modelos.*
- *Has_many :through vs Has_and_belongs_to_many*
- *Borrado en cascada en relaciones N a N.*

{desafío}
latam_

Academia de
talentos digitales