Guía de ejercicios - Relaciones N a N en los modelos

 ¡Hola! Te damos la bienvenida a esta nueva guía de estudio.

¿En qué consiste esta guía?

La siguiente guía de estudio tiene como objetivo practicar y ejercitar los contenidos que hemos visto en clase.

¡Vamos con todo!

Tabla de contenidos

¡Comencemos!

Actividad guiada: TravelGuide

A continuación, empezaremos a trabajar el proyecto de TravelGuide. Pedro siempre ha tenido interés por viajar a lugares donde parece que nunca se pasa mal. Sin embargo, el presupuesto de Pedro es limitado, por lo que quiere elegir muy bien su destino antes de realizar un gasto tan grande. Por esto te ha pedido desarrollar una plataforma donde los usuarios puedan subir sus historias de viaje, que otros usuarios puedan comentarlas y reaccionar a su historia para obtener reseñas de lugares de primera mano. Su hermano es desarrollador front-end y se encargará de los estilos de la página, por lo que solo debemos preocuparnos por las funcionalidades. Para nuestra primera entrega nos solicita los siguientes requisitos:

- Un usuario puede autenticarse en la aplicación.
- Un usuario puede comentar en las publicaciones de viajes.
- Los usuarios deben tener foto de perfil.
- Los usuarios pueden reaccionar tanto a los comentarios como a las publicaciones.
- Toda persona puede ver el índice de publicaciones.
- Las publicaciones de viajes pertenecen a un país en específico.

Desarrollo:

- **Paso 1:** Creamos la aplicación llamada `TravelGuide` con base de datos PostgreSql

```
rails new TravelGuide -d postgresql
```

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

```
rails db:create
```

- **Paso 3:** Identificamos los modelos a integrar

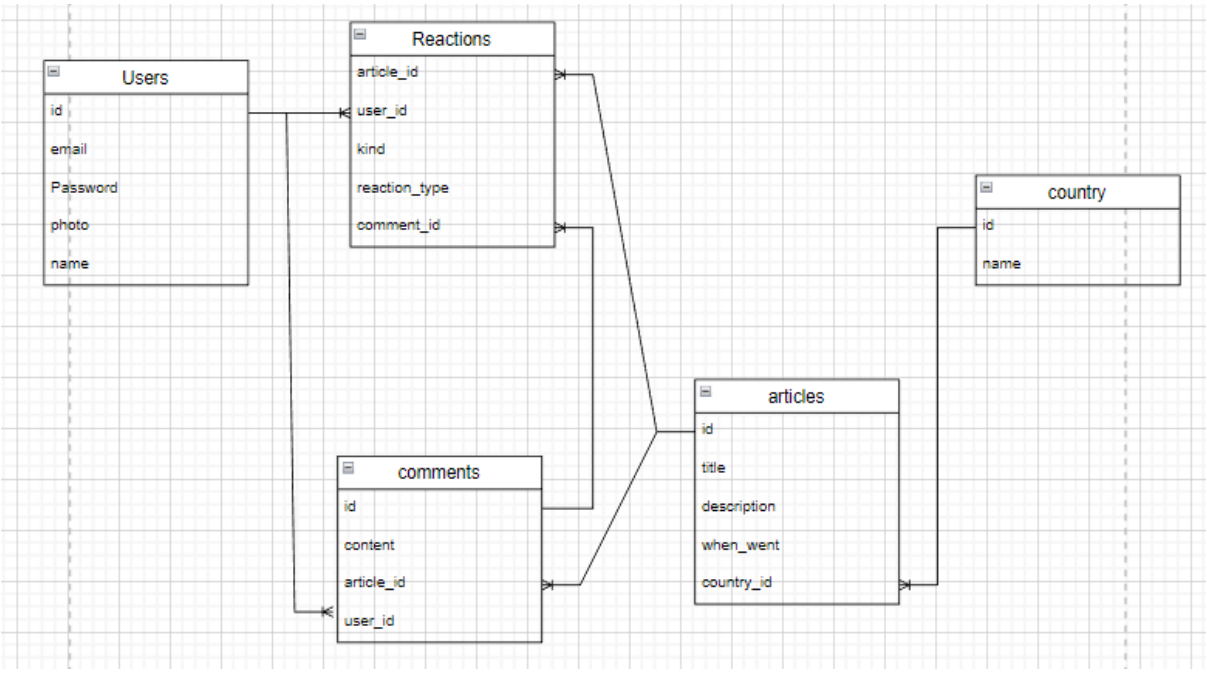

Imagen 1. DIagrama entidad relación
Fuente: Desafío Latam.

- **Paso 4:** Creamos los modelos en base de datos mediante scaffold (opcional, puedes realizarlo a mano si lo deseas).
    a. User con Devise.

```
#Gemfile
gem 'devise'
```

```
#bashrc
bundle
```

```
#bashrc
rails generate devise:install
```

```
#bashrc
rails g devise:views
```

```
#bashrc
rails generate devise user
```

b. Agregamos campos a User.

```
#bashrc
rails g migration AddDetailsToUsers photo name
```

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

c. Agregamos strong params.

```
#app/controllers/application_controller.rb
class ApplicationController < ActionController::Base
  before_action :configure_permitted_parameters, if: :devise_controller?

  protected

  def configure_permitted_parameters
    devise_parameter_sanitizer.permit(:sign_up, keys: [:name, :photo])
    devise_parameter_sanitizer.permit(:account_update, keys: [:name,
:photo])
  end
end
```

d. Agregamos campos al formulario de registro.

```
#app/views/devise/registrations/new.html.erb
  <div class="field">
    <%= f.label :name %><br />
    <%= f.text_field :name, autocomplete: "name" %>
  </div>

  <div class="field">
    <%= f.label :photo %><br />
    <%= f.text_field :photo, autocomplete: "photo" %>
  </div>
```

e. Creamos modelo de Country.

```
#bashrc
rails g model Country name
```

f. Hacemos scaffold de Articles.

```
#bashrc
rails g scaffold articles title description when_went:datetime
country:references
```

g. Creamos modelos Comment.

```
#bashrc
rails g model comment content article:references user:references
```

h. Creamos modelo Reaction

```
#bashrc
rails g model reaction article:references user:references kind
reaction_type comment:references
```

¡No olvides revisar la migración, en este caso necesitas que `article_id` y `comment_id` no tengan "`null: false`"!

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

- **Paso 5:** Relacionar las tablas en los modelos.

```
#app/models/article.rb
class Article < ApplicationRecord
  belongs_to :country
  has_many :comments, dependent: :destroy
  has_many :reactions, dependent: :destroy
  has_many :users, through: :reactions #or comments doesn't matter
end
```

```
#app/models/comment.rb
class Comment < ApplicationRecord
  belongs_to :article
  belongs_to :user
  has_many :reactions, dependent: :destroy
end
```

```
#app/models/country.rb
class Country < ApplicationRecord
  has_many :articles, dependent: :destroy
end
```

```
#app/models/reaction.rb
class Reaction < ApplicationRecord
  belongs_to :article, optional: true
  belongs_to :user
  belongs_to :comment, optional: true
end
```

```
#app/models/user.rb
class User < ApplicationRecord
  # Include default devise modules. Others available are:
  # :confirmable, :lockable, :timeoutable, :trackable and :omniauthable
  devise :database_authenticatable, :registerable,
         :recoverable, :rememberable, :validatable
  has_many :reactions
  has_many :comments
  has_many :articles, through: :reactions
end
```

- **Paso 6:** Arreglamos el problema de inicio de sesión de Devise.

```
#config/initializers/devise
config.navigational_formats = ['*/*', :html, :turbo_stream]
```

- **Paso 7:** Integramos la gema Faker.

```
#Gemfile
gem 'faker'
```

```
#bashrc
bundle
```

- **Paso 8:** Agregamos datos de prueba para artículos.

#db/seeds

```
until Country.count == 20 do
  Country.create(name: Faker::Address.country) if
!Country.pluck(:name).include?(Faker::Address.country)
end

countries = Country.all

until Article.count == 100 do
  Article.create(title: Faker::Book.title, description:
Faker::Lorem.paragraph_by_chars(number: 200, supplemental: false), when_went:
Faker::Date.between(from: 10.years.ago, to: Date.today), country_id:
countries.sample.id)
end
```

- **Paso 9:** Agregamos índice de artículos como nuestra ruta raíz

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

- **Paso 10:** Agregamos tipos de reacciones como constante.

```
#app/models/article.rb
Kinds = %w[like dislike not_interested neutral].freeze
KindsSpanish = {"like" => "Me gusta", "dislike" => "No me gusta",
"not_interested" => "No me interesa", "neutral" => "Neutral"}.freeze
```

- **Paso 11:** Agregamos controlador, método y ruta para crear reacciones.

```
#bashrc
rails g controller reactions
```

```
#app/controllers/reactions_controller.rb
class ReactionsController < ApplicationController

  def new_user_reaction
    @user = current_user
    @type = params[:reaction_type]
    @article = Article.find(params[:article_id]) if params[:article_id]
    @comment = Comment.find(params[:comment_id]) if params[:comment_id]
    @kind = params[:kind]
    respond_to do |format|
```

```
      (@type == "comment") ? reaction_comment = Reaction.find_by(user_id: @user,
comment_id: @comment.id) : reaction_article = Reaction.find_by(user_id: @user.id,
article_id: @article.id)

      if reaction_article || reaction_comment
        format.html { redirect_to article_path(@article), notice: 'You already reacted
to this article' }
      else
        (@type == "article") ? @reaction = Reaction.new(user_id: @user.id, article_id:
@article.id, reaction_type: @type, kind: @kind) : @reaction = Reaction.new(user_id:
@user.id, comment_id: @comment.id, reaction_type: @type,  kind: @kind)
        if @reaction.save!
          format.html { redirect_to article_path(@article), notice: 'Reaction was
successfully created.' }
        else
          format.html { redirect_to article_path(@article), notice: 'Something went
wrong' }
        end
      end

    end
  end

end
```

```
#config/routes.rb
post '/new_user_reaction', to: 'reactions#new_user_reaction', as:
'new_user_reaction'
```

- **Paso 12:** Agregar botón para crear reacción en vista show del artículo.

```
#app/views/articles/show.html.erb
<p style="color: green"><%= notice %></p>

<%= render @article %>

<div>
  <%= link_to "Edit this article", edit_article_path(@article) %> |
  <%= link_to "Back to articles", articles_path %>

  <%= button_to "Destroy this article", @article, method: :delete %>

  <% Article::Kinds.each do |kind| %>
    <%= button_to "#{Article::KindsSpanish[kind]}",
new_user_reaction_path(article_id: @article.id, reaction_type:
"article", kind: kind), method: :post %>
  <% end %>
```

```
</div>
```

- **Paso 13:** Agregar controlador de creación de comentarios con su ruta.

```
#bashrc
rails g controller comments
```

```
#app/controllers/comments_controller.rb
class CommentsController < ApplicationController

  def create
    @article = Article.find(params[:comment][:article_id])
    @comment = Comment.new(comment_params)
    @comment.user = current_user
    respond_to do |format|
      if @comment.save
        format.html { redirect_to article_path(@article.id), notice:
'Comment was successfully created.' }
      else
        format.html { redirect_to article_path(@article.id), notice:
'Comment was not created.' }
      end
    end
  end

  private

  def comment_params
    params.require(:comment).permit(:content, :article_id)
  end

end
```

```
#config/routes.rb
resources :comments, only: [:create]
```

- **Paso 14:** Agregar formulario de comentarios a vista show de artículos.

```
#app/views/comments/_form.html.erb
<%= form_with(model: @comment, local: true) do |f| %>
```

```
    <% if @comment.errors.any? %>
        <div id = 'error_explanation'>
            <h2>Este comentario no se pudo crear por las siguientes
razones</h2>
            <ul>
                <% @comment.errors.full_messages.each do |msg|%>
                    <li><%= msg %></li>
                <% end %>
            </ul>
        </div>
    <% end %>

    <%= f.hidden_field :article_id, value: @article.id %>

    <div class = 'field'>
        <%= f.label :content, as: "Contenido" %>
        <%= f.text_field :content %>

    <div class = 'actions'>
        <%= f.submit %>
    </div>
<% end %>
```

```
#app/views/articles/show.html.erb
<%= render 'comments/form' %>
```

- **Paso 15:** Agregar variable @comment a métodos necesarios en controlador articles

```
#app/controllers/artciles_controller.rb
  def show
    @comment = Comment.new
  end
```

¡Manos a la obra! - Quiero ver los comentarios

Pedro nos ha pedido que los comentarios se carguen dentro de la vista del artículo, además quiere muchos comentarios dentro de la página para poder aplicarle estilo después. Los comentarios deben verse con la foto de perfil del usuario junto al contenido del mismo y no le gusta el campo `when_went`, quiere que aparezca como "Cuando fui".

- Mostrar comentarios en vista show del artículo.
- Rellenar el seed con más datos ficticios.
- Cambiar la forma en que se muestra `when_went` en el parcial de artículos.

¡Manos a la obra! - ¡Quiero verlo!

Pedro nos ha dicho que le gustó mucho nuestra implementación, pero que necesita tener la aplicación en su computador para hacer pruebas.

- Realizar deploy a heroku.

En la plataforma tendrás acceso al código finalizado de esta aplicación con el nombre **Material de apoyo - Guía de ejercicios - Relaciones N a N en los modelos.**

Solución Manos a la Obra 1

- **Paso 1:** Agregamos comentarios al controlador show de artículos.

```
#app/controllers/articles_controller.rb
def show
  @comment = Comment.new
  @comments = @article.comments
end
```

- **Paso 2:** Agregamos los comentarios a la vista show del artículo.

```
#app/views/comments/_article_comments.html.erb
<div>
    <% @comments.each do |comment| %>
        <p>
            <%= image_tag(comment.user.photo) %>
            <%= comment.content %>
        </p>
    <% end %>
</div>
```

```
#app/views/articles/show.html.erb
<%= render 'comments/article_comments' %>
```

- **Paso 3:** Agregamos comentarios y reacciones al seed.

```
#db/seeds.rb
until Country.count == 20 do
  Country.create(name: Faker::Address.country) if
!Country.pluck(:name).include?(Faker::Address.country)
end

countries = Country.all

until Article.count == 100 do
  Article.create(title: Faker::Book.title, description:
Faker::Lorem.paragraph_by_chars(number: 200, supplemental: false),
when_went: Faker::Date.between(from: 10.years.ago, to: Date.today),
country_id: countries.sample.id)
end
```

```
i = 0
until User.count == 20 do
  User.create(email: "test#{i}@gmail", password: "asdasdasd",
"password_confirmation": "asdasdasd", photo: Faker::Avatar.image, name:
Faker::Name.name)
  i += 1
end

articles = Article.all
users = User.all

until Comment.count == 1000 do
  Comment.create(content: Faker::Lorem.paragraph_by_chars(number: 200,
supplemental: false), article_id: articles.sample.id, user_id:
users.sample.id)
end

r_type = %w[article comment]
comments = Comment.all
kinds = Article::Kinds

until Reaction.count == 1000 do
  rel_type = r_type.sample
  if rel_type == "article"
    Reaction.create(article_id: articles.sample.id, user_id:
users.sample.id, kind: kinds.sample, reaction_type: rel_type)
  else
    Reaction.create(comment_id: comments.sample.id, user_id:
users.sample.id, kind: kinds.sample, reaction_type: rel_type)
  end

end
```

- **Paso 4:** Cambiamos la forma en que se muestra el parametro when_went

```
#app/views/articles/_article.html.erb
 <p>
   <strong>Cuando fui:</strong>
   <%= article.when_went %>
 </p>
```

Solución Manos a la Obra 2

- Realizar deploy a heroku.
    a. Tenemos que logearnos en heroku

```
heroku login
```

    b. Creamos una aplicación

```
heroku create
```

    c. Subimos los cambios a heroku

```
git push heroku main
```

    d. Corremos las migraciones para el correcto funcionamiento

```
heroku run rake db:migrate
```

    e. Corremos el seed para tener datos

```
heroku run rake db:seed
```

En la plataforma te compartimos el código de este ejercicio con el nombre **"Material de apoyo - Guía de ejercicios - Relaciones N a N en los modelos"**