Temario

Unidad 3

a) Modelos con relaciones N a N

Temas: has_many :through vs has_and_belongs_to_many. Implementación de relaciones N a N con dos modelos. Implementación de relaciones N a N con tres modelos. Borrado en cascada en relaciones N a N. Asignar recursos vía checkbox, siguiendo convenciones REST e ingresarlas como recurso anidado.

Relaciones N a N, o Muchos a Muchos

Un **User** puede tener muchos **Movies** y un **User** puede tener muchos **Employees**. Es una relación muchos a muchos. En una asociación muchos a muchos es un tipo de asociación en donde **un registro de una tabla puede estar relacionada a muchos registros de otra otra tabla**. Y para definir una relación muchos a muchos se debe **crear una tabla intermedia** que relacione las dos tablas. a **continuación** sabremos el significado de **has_many :through y has_and_belongs_to_many**, el cual también tiene su diferencia.

has_many :through y has_and_belong_to_many

Por ejemplo,el modelo **User,** el modelo **Movie** puede tener muchos del modelo **Movie**, y ese modelo **Movie** no tienen por qué pertenecer exclusivamente al **User;** podrían tener otros **Users**. Ahí es cuando entra en juego la asociación de muchos a muchos. un objeto puede tener muchos objetos que le pertenecen, pero no exclusivamente.

El otro ejemplo es has_manu :through su función es insertar una tabla nueva a los modelos, este no tiene relación y se puede usar los atributos de esa tabla.

has_many :through y has_and_belong_to_many

En este caso usaremos el segundo, insertándole una tabla que entra a hacer el juego de Mucho a Muchos.

```ruby
class User < ApplicationRecord
  has_many :assistants, dependent: :destroy # Acá iniciamos el mucho a muchos con through
  accepts_nested_attributes_for :assistants
  has_many :movies, through: :assistants

  validates :username, presence: true
  validates :age, presence: true, length: { maximum: 2}
  validates :country, presence: true
  # Include default devise modules. Others available are:
  # :confirmable, :lockable, :timeoutable, :trackable and :omniauthable
  devise :database_authenticatable, :registerable,
         :recoverable, :rememberable, :validatable
end
```

```ruby
class Movie < ApplicationRecord
  has_many :assistants, dependent: :destroy # Acá iniciamos el mucho a muchos con through
  has_many :user, through: :assistants
end
```

Creamos un controlador llamado Assistant y generamos otro modelo en relación Muchos a Muchos:

En este caso usaremos el tercero, Creamos un controlador llamado Assistant y generamos otro modelo en relación Muchos a Muchos:

```
81  class Assistant < ApplicationRecord
82    belongs_to :user
83    belongs_to :movie
84
85    accepts_nested_attributes_for :user
86  end
```

Borrado en cascada en relaciones N a N:

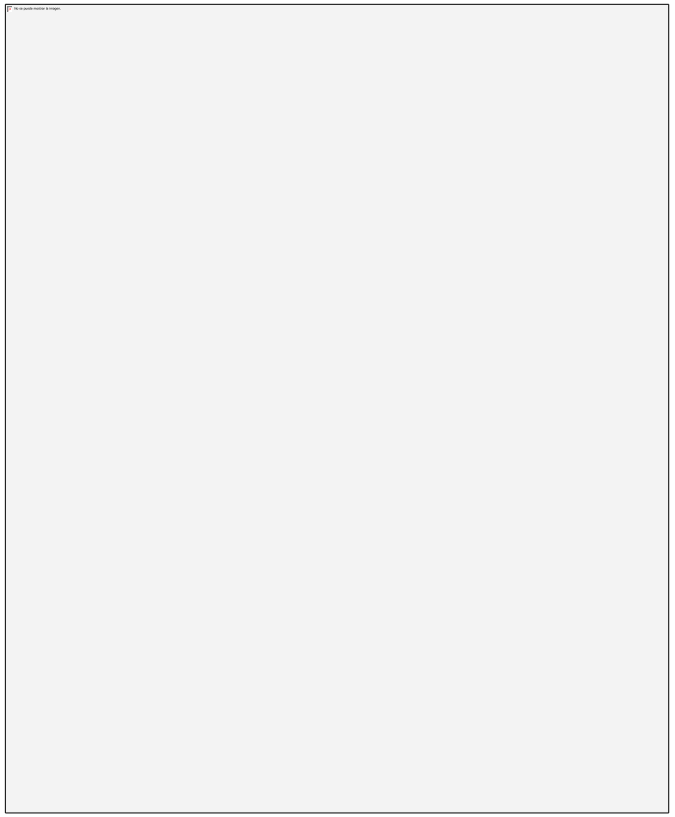
En este caso usaremos el tercero, Creamos un controlador llamado Assistant y generamos otro modelo en relación Muchos a Muchos:

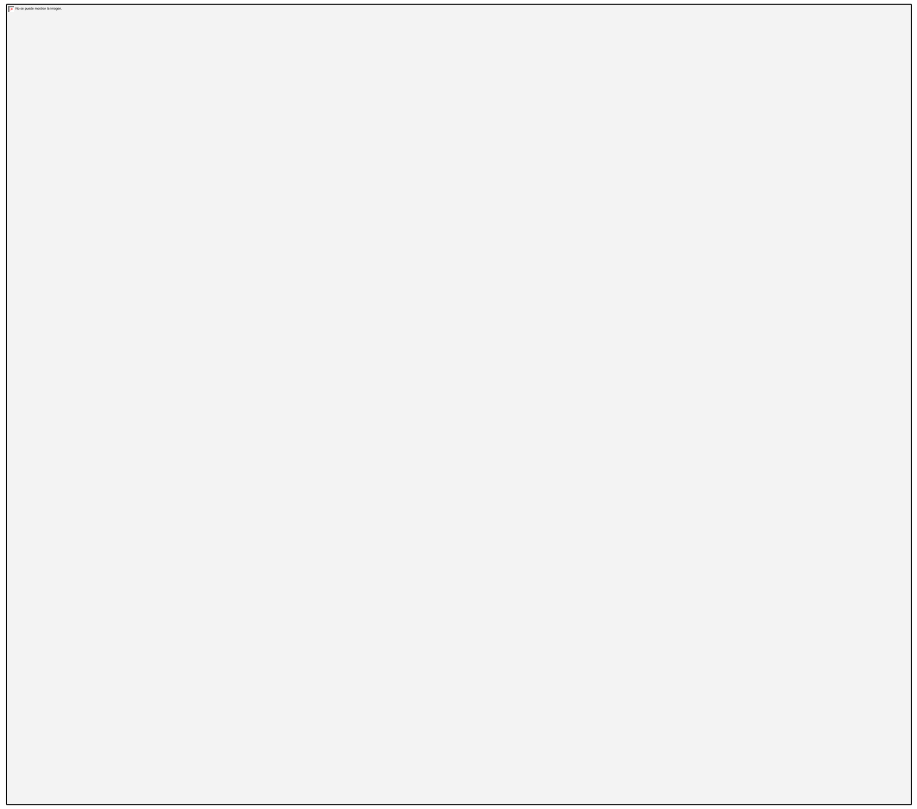
Esto usando: **collection.delete(object, …)**
Esto usando: **collection.clear**
y a sí con el tercer modelo de puente assistant.

```
user = User.find(1)
movie = user.movies.find(1)

user.movies.delete(movie)


user = User.find(2)

user.movies.clear
```

Relaciones polimorficas

Relaciones polimórficas:
En las relaciones del tipo has_many <=> belongs_to nos permite asociar por ejemplo **1** a Muchos, siendo categoría y artículo el mismo tipo de objetos.





Formularios multipart

Una tarea común es cargar algún tipo de archivo, ya sea una imagen de una persona o un archivo CSV que contenga datos para procesar. Lo más importante a recordar con las subidas de archivos es que la codificación de la forma renderizada **DEBE** establecerse en "multipart/form-data". Si utiliza form_for, esto se hace automáticamente. Si utiliza form_tag, debe configurarlo usted mismo, como en el ejemplo siguiente
Rails proporciona el par habitual de helpers: el barebones file_field_tag y el orientado a modelos file_field. La única diferencia con otros helpers es que no se puede establecer un valor predeterminado para los inputs de archivos ya que esto no tendría ningún significado. Como es de esperar en el primer caso el archivo subido está en params[:picture]y en el segundo caso en params[:person][:picture].

Unidad 4

Subida de archivos con actionstorage y AmazonS3

Active Storage que es y como se implementa

ActiveStorage es una herramienta que facilita la subida de archivos a almacenamientos en la nube como AWS, Google Cloud Storage o Microsoft Azure Storage y adjunta esos archivos convirtiéndolos en Active Record objects. Viene con un servicio local basado en disco para desarrollo y pruebas, y admite la duplicación de archivos a otros servicios para copias de seguridad y migraciones.

Con ActiveStorage, una aplicación puede transformar las cargas de imágenes, generar representaciones de imágenes de cargas sin imágenes como archivos PDF y videos, y extraer metadatos de archivos arbitrarios.

ActiveStorage en Rails

Active Storage es una gema incluida por defecto en tu aplicación (o API) de Rails. Sirve tanto para cargar imágenes a nivel local, como para sincronizarlas o respaldarlas en servicios de la nube como Amazon S3, Google Cloud Storage o Microsoft Azure Storage.

En caso de querer usar otro servicio solo se debe sustituir :local por el nombre del servicio a emplear.

Adjuntar archivos a un registro

Material complementario de la unidad

Link a video relacionado

1. https://www.youtube.com/watch?v=GsfRL8OVZ7k

Link a lectura complementaria

1. https://medium.com/@jesusapd4/agregar-im%C3%A1genes-a-tu-proyecto-con-active-storage-8ec62c8eef3f
2. https://codigofacilito.com/articulos/active-storage

Link a investigación relacionada

1. https://sa-east-1.signin.aws.amazon.com/
2. https://medium.com/@jesusapd4/agregar-im%C3%A1genes-a-tu-proyecto-con-active-storage-8ec62c8eef3f