

Módulo 9.

Desarrollo de aplicaciones con ruby on rails (5 unidades)

Temario

Unidad 1

a) Relaciones 1 a N

Temas: Creación de modelos con relaciones 1 a N. Probar las relaciones 1 a N en rails console. Creación de valores asociados con build. Borrar elementos en cascada.

b) Diagrama de relaciones

Temas: Asociando elementos con: Select. Radio. Rutas anidadas. Formularios anidados.

Unidad 1

Relaciones 1 a N

Relaciones 1 a N, o 1 a Muchos

Las asociaciones se utilizan para definir relaciones entre tablas de una base de datos, para ello existen 2 tipos. Pero en este caso veremos solo 1 a N, o 1 a Muchos el cual se puede modelar en una base de datos relacional. ¿Y por qué necesitamos asociaciones entre modelos?, por que nos ayuda a facilitar operaciones comunes en nuestro código, ya que cada registro de una tabla está relacionado a varios registros de otra tabla.

Por ejemplo: Tenemos 2 modelos, un modelo llamado árbol y otro llamado naranja, en este caso cada árbol puede tener muchas naranjas.

Diagrama relacional Rails

En una base de datos relacional las tablas se relacionan a través de llaves primarias y llaves foráneas. Las columnas se componen de: Un nombre, Un tipo de dato (string, integer, text, boolean, date, etc), Tamaños, valores de ausencia (null), etc.

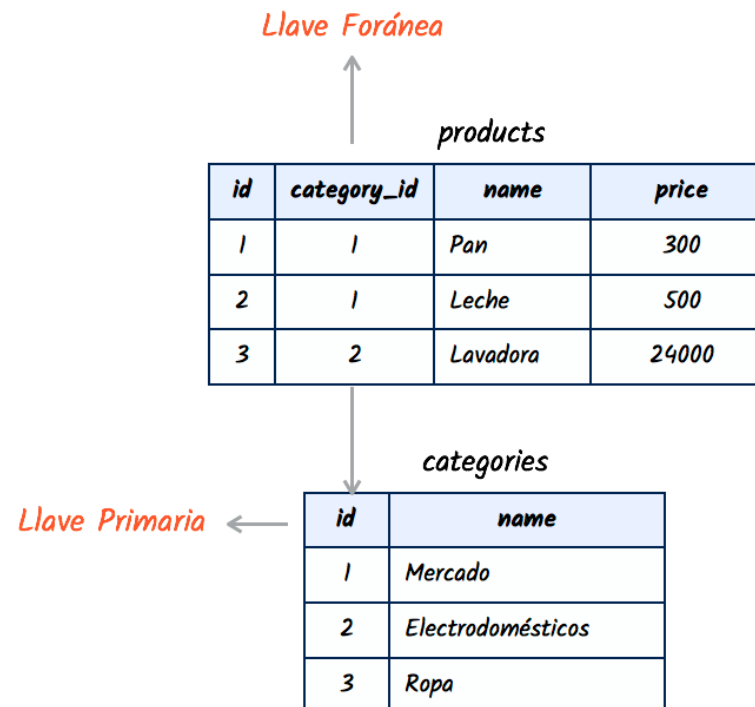
Cada tabla tiene una llave primaria, que tienen una o mas columnas. La columna que se denomina id es la llave primaria donde Una tabla sólo puede tener una única llave primaria. Además de la llave primaria existen llaves foráneas que son columnas que se referencian a las llaves primarias.

Diagram illustrating a table structure with columns and records.

| name | email | genre | birthday |
|-------|-----------------|-------|------------|
| Juan | juan@gmail.com | M | 1982-08-24 |
| Pedro | pedro@test.com | M | 1994-01-23 |
| Diana | diana@gmail.com | F | 1983-08-11 |

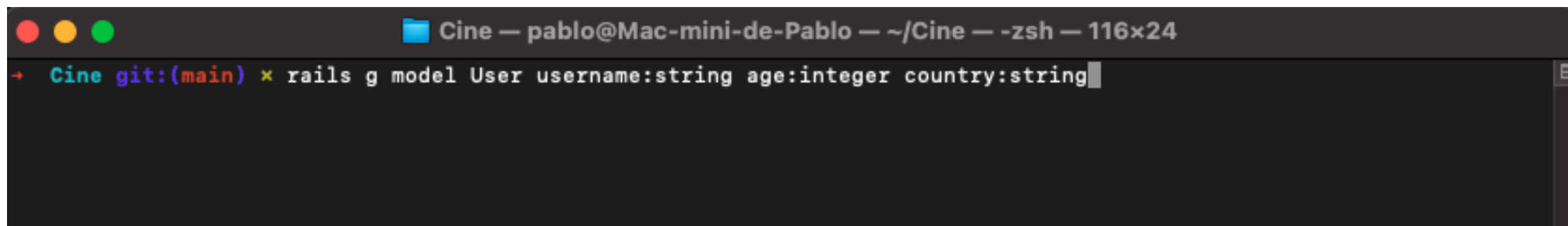
Columns: name, email, genre, birthday

Records: Juan, Pedro, Diana



Creando modelo 1 a N, o 1 a Muchos

Creamos un modelo llamado **User** con los siguientes parametros:

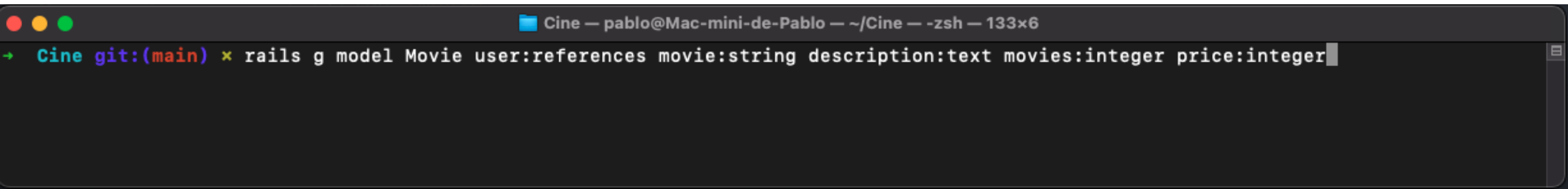


```
Cine — pablo@Mac-mini-de-Pablo — ~/Cine — zsh — 116x24  
→ Cine git:(main) ✖ rails g model User username:string age:integer country:string
```

Nota: Recordar sobre el diagrama relacional Rails

Creando modelo 1 a N, o 1 a Muchos

Creamos el modelo que va a estar relacionado al anterior utilizando **references** para crear la referencia.




```
Cine — pablo@Mac-mini-de-Pablo — ~/Cine — -zsh — 133x6  
→ Cine git:(main) ✕ rails g model Movie user:references movie:string description:text movies:integer price:integer
```

Cada campo debe llevar un tipo de dato ya sea **string, integer, text**, etc

Modelo creado Movie, 1 a N, o 1 a Muchos

Como resultado se agrega automáticamente **belongs_to :user** a **Movie**


```
app > models >  movie.rb  
1  class Movie < ApplicationRecord  
2    | belongs_to :user  
3  end
```

Otra forma de hacer esto es generar los 2 modelos **User** y **Movie** manualmente y agregar en **movie.rb** **belongs_to :user**

Recordar que el modelo **movie.rb** se encuentra en **app/models/movie.rb**

Modelo creado User, 1 a N, o 1 a Muchos

Se le agrega manualmente **has_many :movies**

```
app > models >  user.rb
1  class User < ApplicationRecord
2    has_many :movies
3  end
```

Otra forma de hacer esto es generar los 2 modelos **User** y **Movie**, y manualmente agregar en **user.rb** **has_many :movies**

Recordar que el modelo **user.rb** se encuentra en **app/models/user.rb**

Migrando, 1 a N, o 1 a Muchos

Al generar los 2 modelos y creando la referencia con la llave foránea, migramos.

```
X00 mingw32 X00 mingw32 X04 mingw32 java 1
== 20210413050803 CreateUsers: migrating =====
-- create_table(:users)
--> 0.0033s
== 20210413050803 CreateUsers: migrated (0.0034s) =====

→ Cine git:(main) ✕ rails s
```

Recordar que las migraciones se encuentra en **db/migrate** , Rails es tan inteligente que estas también pueden ser modificables en cualquier momento.

Probando resultado 1 a N, o N a Muchos en rails console

Para probar todo lo que acabamos de hacer anteriormente, escribimos **rails c** que es la consola de Rails este desplegará una consola igual a IRB en el cual escribiremos lo siguiente:

```
Cine — rails c — rails — ruby bin/rails c — 116x24
→ Cine git:(main) × rails c
The dependency tzinfo-data (>= 0) will be unused by any of the platforms Bundler is installing for. Bundler is installing for ruby but the dependency is only for x86-mingw32, x86-mswin32, x64-mingw32, java. To add those platforms to the bundle, run `bundle lock --add-platform x86-mingw32 x86-mswin32 x64-mingw32 java`.
Running via Spring preloader in process 10495
Loading development environment (Rails 6.1.3.1)
2.7.2 :001 > User.create(username: "Pabloski", age: 23, country: "Chile")
```

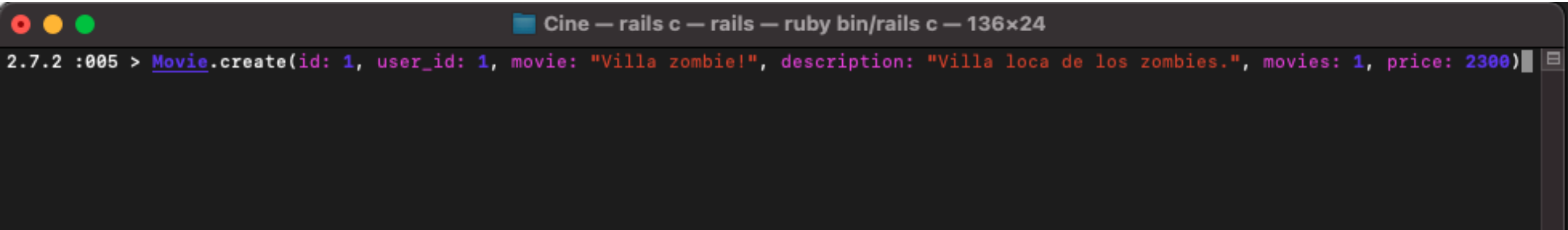
Colección User.

Tomamos todo lo del modelo **User** y usamos, **.create** con este crearemos lo siguiente, en este caso tenemos 3 parámetros.

Atributo **nombre**: con texto **"Pabloski"** así con **age**: y **country**:

Probando resultado 1 a N, o N a Muchos en rails console

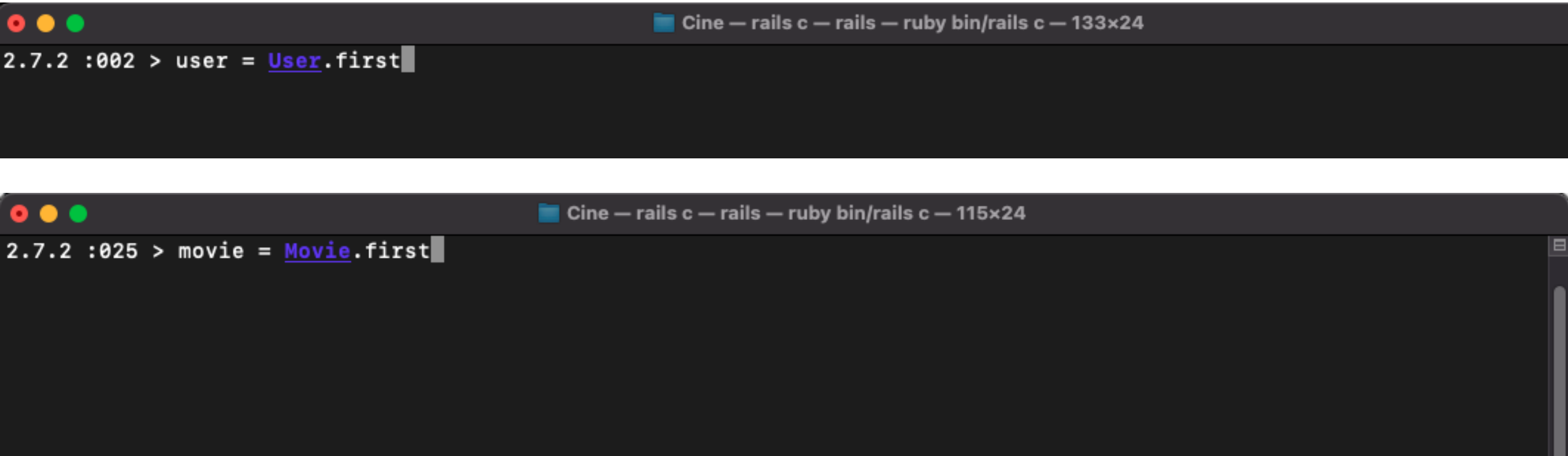
Como segunda prueba nuevamente en **rails console**, creamos ahora una con **Movie**,
Con los siguientes parámetros:



```
Cine — rails c — rails — ruby bin/rails c — 136x24
2.7.2 :005 > Movie.create(id: 1, user_id: 1, movie: "Villa zombie!", description: "Villa loca de los zombies.", movies: 1, price: 2300)
```

Probando resultado 1 a N, o N a Muchos en rails console

Luego de crear los 2 objetos, los almacenamos en una variable



The image shows two screenshots of a Rails console window. The top screenshot shows the command `user = User.first` being entered. The bottom screenshot shows the command `movie = Movie.first` being entered. Both screenshots show the console window title as "Cine — rails c — rails — ruby bin/rails c — 133x24" and "Cine — rails c — rails — ruby bin/rails c — 115x24" respectively.

```
2.7.2 :002 > user = User.first
```

```
2.7.2 :025 > movie = Movie.first
```

Resultado asociaciones 1 a N, o 1 a Muchos en rails console

Escribimos los siguiente y como resultado anterior ambos **movie** y **user con otros ids** se generaron, y se cumple la asociación 1 a muchos. Pero esto no es todo, podemos requerir las películas del usuario, o cuanto cuesta las películas del usuario, etc.

```
Cine — rails c — rails — ruby bin/rails c — 133x24
2.7.2 :008 > user.movies
Movie Load (0.2ms)  SELECT "movies".* FROM "movies" WHERE "movies"."user_id" = ? /* loading for inspect */ LIMIT ?  [["user_id", 1], ["LIMIT", 11]]
=> #<ActiveRecord::Associations::CollectionProxy [#<Movie id: 1, user_id: 1, movie: "Villa zombie!", description: "Villa loca de los zombies.", movies: 1, price: 2300, created_at: "2021-04-13 06:35:48.092504000 +0000", updated_at: "2021-04-13 06:35:48.092504000 +0000">]>
2.7.2 :009 > 
```

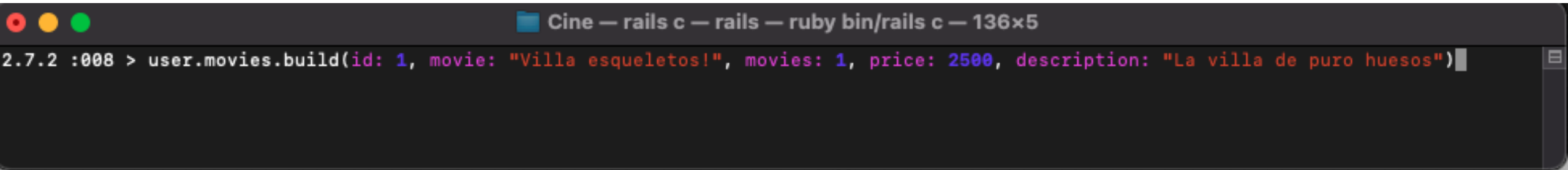
Creación de valores asociados con el método build

Para empezar con el **método build** debemos aprender de este.

El método **build** es un **alias de new**, este devuelve un nuevo objeto del tipo de colección que se ha instanciado con atributos y vinculado a este objeto, pero que aún no se ha guardado. Puede pasar un arreglo de atributos hash, esto devolverá una arreglo con los nuevos objetos.

En resumen: **build no guarda el objeto**, lo crea. **si el guardado falla**, devuelve falso y no se cumple, no se crea y tira un raise, create!

Creación con el método build en rails console



```
Cine — rails c — rails — ruby bin/rails c — 136x5
2.7.2 :008 > user.movies.build(id: 1, movie: "Villa esqueletos!", movies: 1, price: 2500, description: "La villa de puro huesos")
```

En este escenario estamos construyendo el objeto de los atributos de Movie para el User, relación 1 a Muchos usando, **user.movies.build** hay 2 formas para añadirse unos **Hash rocket** con su “valor”, o como parametro y su “valor”


En este caso le agregamos un nuevo **id, movie, movies, price, description**.

Eliminación de elementos en Cascada 1 a Muchos, o 1 a N

En Rails tenemos la opción de realizar una eliminación en cascada desde nuestro modelo. En este caso, al eliminar un usuario que también se eliminen sus notas. Para ello nos podremos apoyar de dos métodos. **destroy y delete_all.**

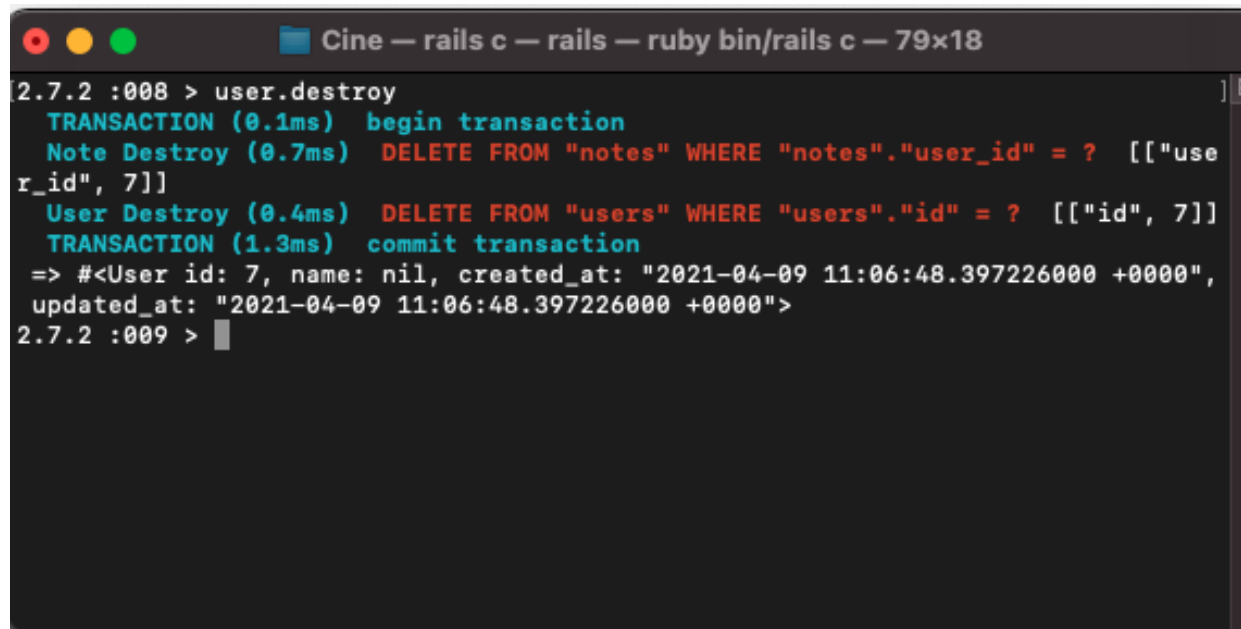
Apoyandonos en la relación 1 a Muchos, o 1 a N.

Añadiendo del dependiente delete_all

```
app > models >  movie.rb  
1   class Movie < ApplicationRecord  
2     belongs_to :user, dependent: :delete_all  
3   end
```

Hay que saber la diferencia entre **destroy** y **delete_all**

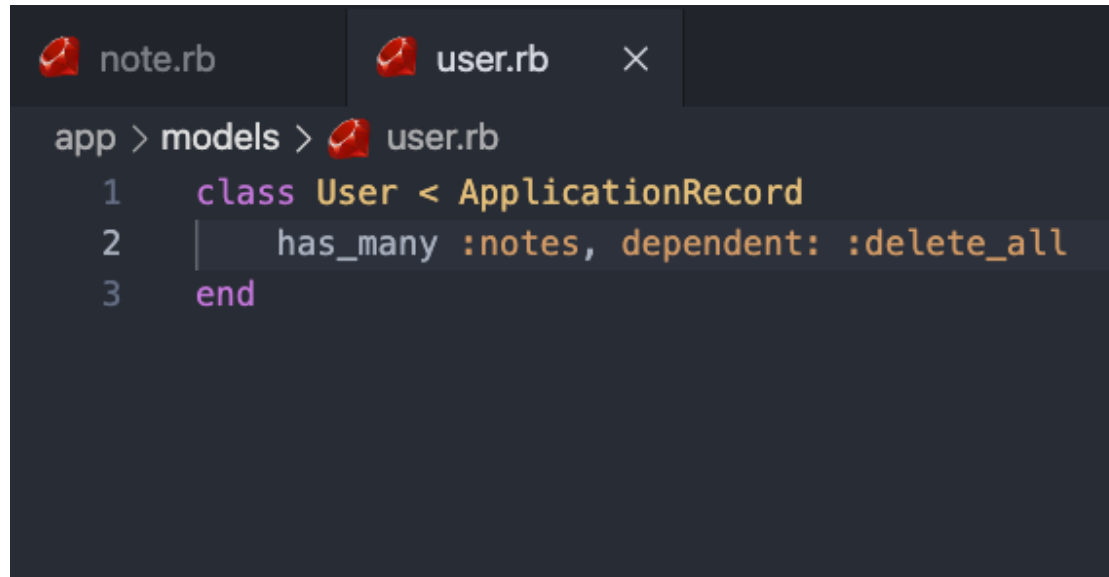
Ejemplo usando destroy en user, con rails console



```
2.7.2 :008 > user.destroy
TRANSACTION (0.1ms) begin transaction
Note Destroy (0.7ms) DELETE FROM "notes" WHERE "notes"."user_id" = ? [{"user_id", 7}]
User Destroy (0.4ms) DELETE FROM "users" WHERE "users"."id" = ? [{"id", 7}]
TRANSACTION (1.3ms) commit transaction
=> #<User id: 7, name: nil, created_at: "2021-04-09 11:06:48.397226000 +0000",
updated_at: "2021-04-09 11:06:48.397226000 +0000">
2.7.2 :009 >
```

Como podemos observar al ejecutar el comando **user.destroy** elimina las **movies del usuario**. Las notas se instancian y **eliminan de una en una**.

Añadiendo el dependiente delete_all



```
note.rb  user.rb  ×  
app > models > user.rb  
1  class User < ApplicationRecord  
2    has_many :notes, dependent: :delete_all  
3  end
```

Una vez que hayamos cambiado el **dependiente** a **:delete_all** no es necesario renombrar los **callbacks**

Al hacer uso de **dependent: :delete_all** en nuestro modelo, los registros que tengan relación con el objeto a eliminar serán eliminados mediante la sentencia **SQL, DELETE**, es decir, los objetos relacionados **no** serán instanciados. Eliminar los registros directamente de la base de datos hará que los **callbacks** en nuestros modelos no sean ejecutados.

Material complementario de la unidad

Link a video relacionado

1. https://www.youtube.com/watch?v=TNF_ptAX2-M

Link a lectura complementaria

1. https://guides.rubyonrails.org/association_basics.html
2. <https://guias.makeitreal.camp/bases-de-datos/bases-de-datos-relacionales>

Link a investigación relacionada

1. https://guides.rubyonrails.org/association_basics.html
2. <https://guias.makeitreal.camp/ruby-on-rails-i/activerecord-asociaciones>
3. <https://stackoverflow.com/questions/783584/ruby-on-rails-how-do-i-use-the-active-record-build-method-in-a-belongs-to-rel>

Unidad 1

Diagrama de relaciones

Selectores, radios, rutas anidadas y formularios anidados

En este temario veremos selectores, radios, rutas anidadas y formularios anidados.

Todos estas creaciones serán utilizados mediante los **controladores**.

Selectores, radios, rutas anidadas y formularios anidados serán creados con la relación 1 a Muchos, o 1 a N.

Creando el controlador Users


Ahora que ya tenemos nuestros modelos con sus nuevas migraciones para poder usar las asociaciones y poder usarlo con **select y radio**, generaremos un **controlador** llamado **Users** para luego asociarlo y llamar el modelo desde el controlador, usamos:

rails g controller Users index

Esto nos creará el **controlador con las rutas, vistas, test, helper y assets**.

Creando variable de instancia @user

Una vez que hayamos creado el controlador **Users** en el vamos a escribir lo siguiente:

```
app > controllers >  users_controller.rb  
1   class UsersController < ApplicationController  
2     def index  
3       @user = User.all  
4     end  
5
```

Esto se refiere a que estamos llamando todo lo del modelo **User** con **all** y encapsularlo todo en esta **variable de instancia** llamado **@user**.

Implementando collection_select

En este escenario estamos creando un formulario con el **modelo User**, el cual tomamos del controlador **users_controller** de la variable de instancia **@user**. El cual funciona de la siguiente manera, escribimos:

```
1 <h1>Formulario y collection_select, radio</h1>
2
3 <%= form_with model: @user, local: true do |f| %>
4   <%= f.collection_select :user_id, User.all, :id, :age %>
5 <% end %>
6
```

Implementando radio_button

Una vez que hayamos implementado **collection_select**, ahora vamos a implementar **radio_button** y escribimos lo siguiente:

```
1  <h1>Formulario y collection_select, radio</h1>
2
3  <%= form_with model: @user, local: true do |f| %>
4    <%= f.label :email %>
5    <%= f.radio_button :contact, "email", checked: false %>
6  <% end %>
```

En el formulario iteramos y dentro ponemos un **label** con propiedad **:email** este puede ser editado y estilizado
Luego pondremos un **radio_button** con el nombre **:contact**, valor **"email"** y si este está **comprobado en falso o verdadero**

Definición rutas anidadas

Para definir los recursos de las rutas usaremos **resource**, **resource** es un helper de enrutamiento de recursos, que le permite declarar rápidamente todas las rutas comunes de un controlador. Una sola llamada a los recursos puede declarar todas las rutas necesarias para sus acciones.

Para aplicar las rutas anidadas también necesitaremos **realizar cambios en el enrutamiento predeterminado** que nos proporcionó Rails para que los recursos de **Movie** sean hijos de los recursos de **User**. para aplicar el helper **resource** debemos ir a **routes.rb** y especificar que vamos a usar el controlador **:users** y **:movies** este usará todas las rutas en **verbo HTTP**.

Para verificar las rutas y rutas anidadas con resource, escribimos en la consola **rails routes**

Nota: Recordar el cuadro de los Verbos HTTP y las Acciones, con ese cuadro sabremos como usar bien las rutas, resource y resources anidados.

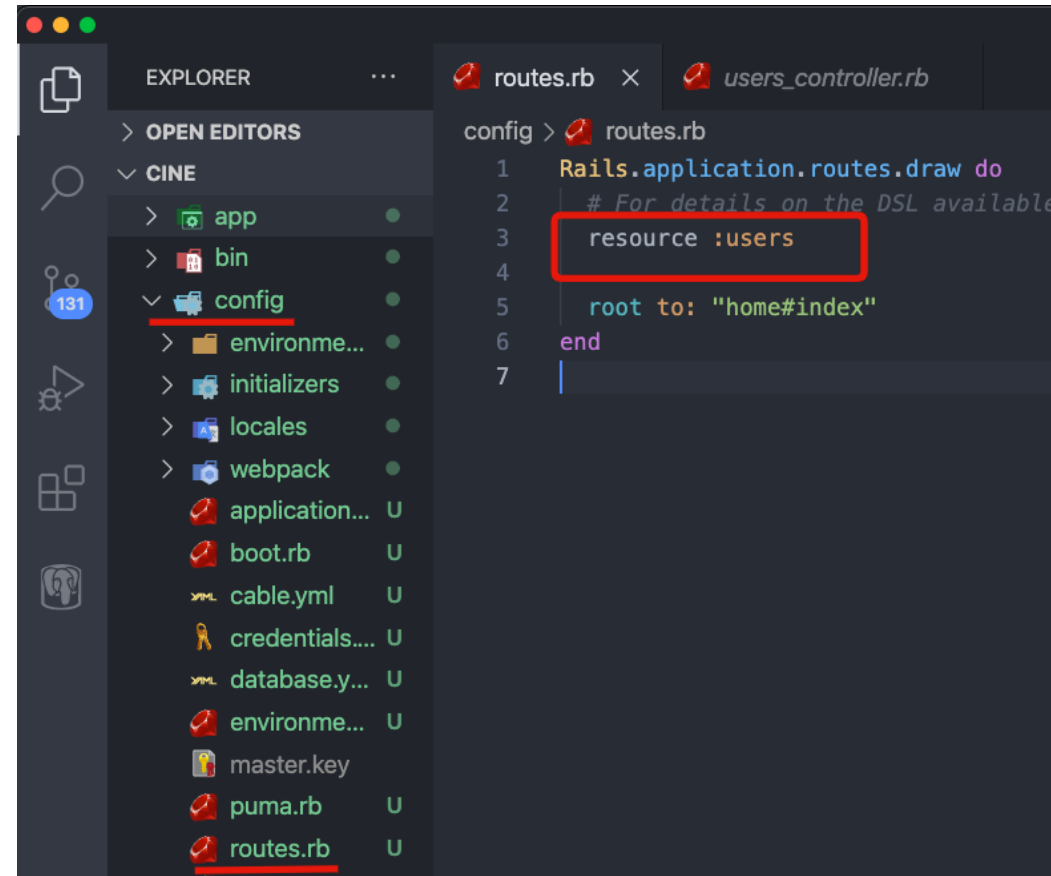
Cuadro con los verbos HTTP y las Acciones

En este cuadro podemos observar los Verbos HTTP y las Acciones

Los verbos mas comunes son GET, POST

| HTTP Verb | Path | Controller#Action | Used for |
|-----------|------------------|-------------------|--|
| GET | /photos | photos#index | display a list of all photos |
| GET | /photos/new | photos#new | return an HTML form for creating a new photo |
| POST | /photos | photos#create | create a new photo |
| GET | /photos/:id | photos#show | display a specific photo |
| GET | /photos/:id/edit | photos#edit | return an HTML form for editing a photo |
| PATCH/PUT | /photos/:id | photos#update | update a specific photo |
| DELETE | /photos/:id | photos#destroy | delete a specific photo |

Ejemplo aplicando resource ruta no anidada



The screenshot shows the Visual Studio Code interface. On the left, the Explorer sidebar displays the project structure under the 'CINE' folder. The 'config' folder is expanded, showing files like 'environment.rb', 'initializers', 'locales', 'webpack', 'application.rb', 'boot.rb', 'cable.yml', 'credentials.yml', 'database.yml', 'environment.rb', 'master.key', 'puma.rb', and 'routes.rb'. The 'routes.rb' file is selected and highlighted in red. On the right, the Editor pane shows the content of 'routes.rb' within the 'config' directory. The code is as follows:

```
config > routes.rb
1  Rails.application.routes.draw do
2    # For details on the DSL available
3    resource :users
4
5    root to: "home#index"
6  end
7
```

The line `resource :users` is highlighted with a red rectangle.

Nota: routes.rb siempre estarán en config, las rutas, resources, etc.

Ejemplo aplicando resource ruta anidada

```
config > routes.rb
1  Rails.application.routes.draw do
2    resources :users do
3      resources :movies
4    end
5  end
6
```

Agregamos **root to: home#index** ya que se usará a futuro. debajo del resource anidado.