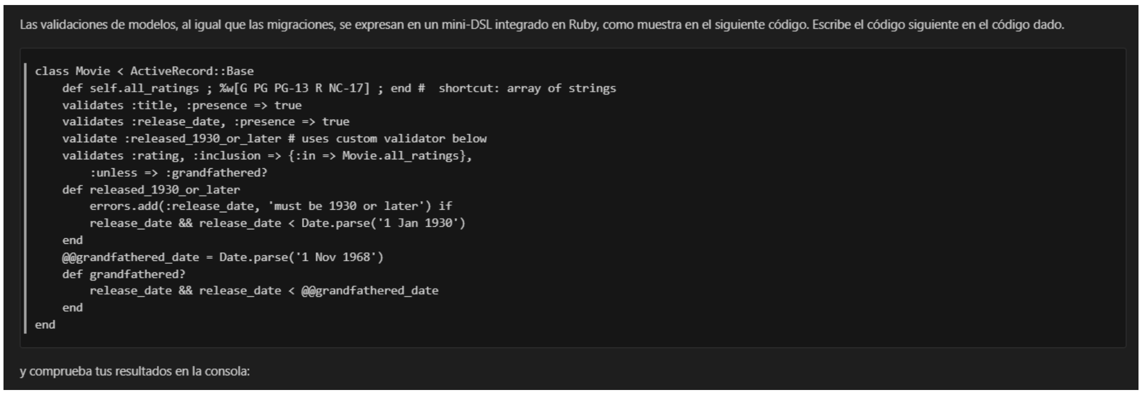
Tarea PC3

Veremos los enunciados de las preguntas:

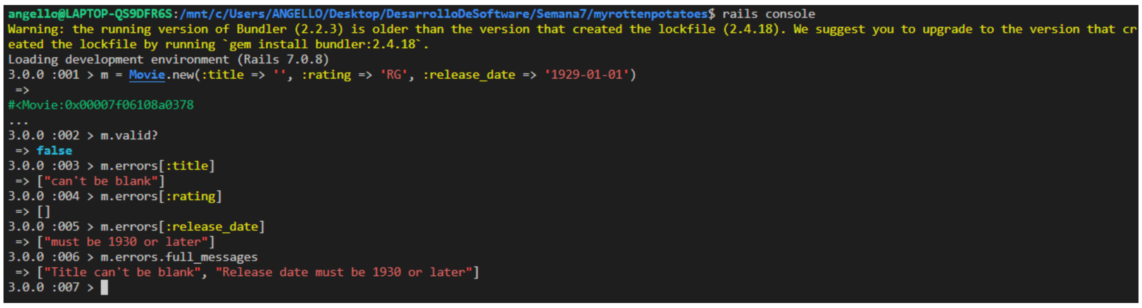


Las validaciones de modelos, al igual que las migraciones, se expresan en un mini-DSL integrado en Ruby, como muestra en el siguiente código. Escribe el código siguiente en el código dado.

```ruby
class Movie < ActiveRecord::Base
    def self.all_ratings ; %w[G PG PG-13 R NC-17] ; end #  shortcut: array of strings
    validates :title, :presence => true
    validates :release_date, :presence => true
    validate :released_1930_or_later # uses custom validator below
    validates :rating, :inclusion => {:in => Movie.all_ratings},
        :unless => :grandfathered?
    def released_1930_or_later
        errors.add(:release_date, 'must be 1930 or later') if
        release_date && release_date < Date.parse('1 Jan 1930')
    end
    @@grandfathered_date = Date.parse('1 Nov 1968')
    def grandfathered?
        release_date && release_date < @@grandfathered_date
    end
end
```

y comprueba tus resultados en la consola:

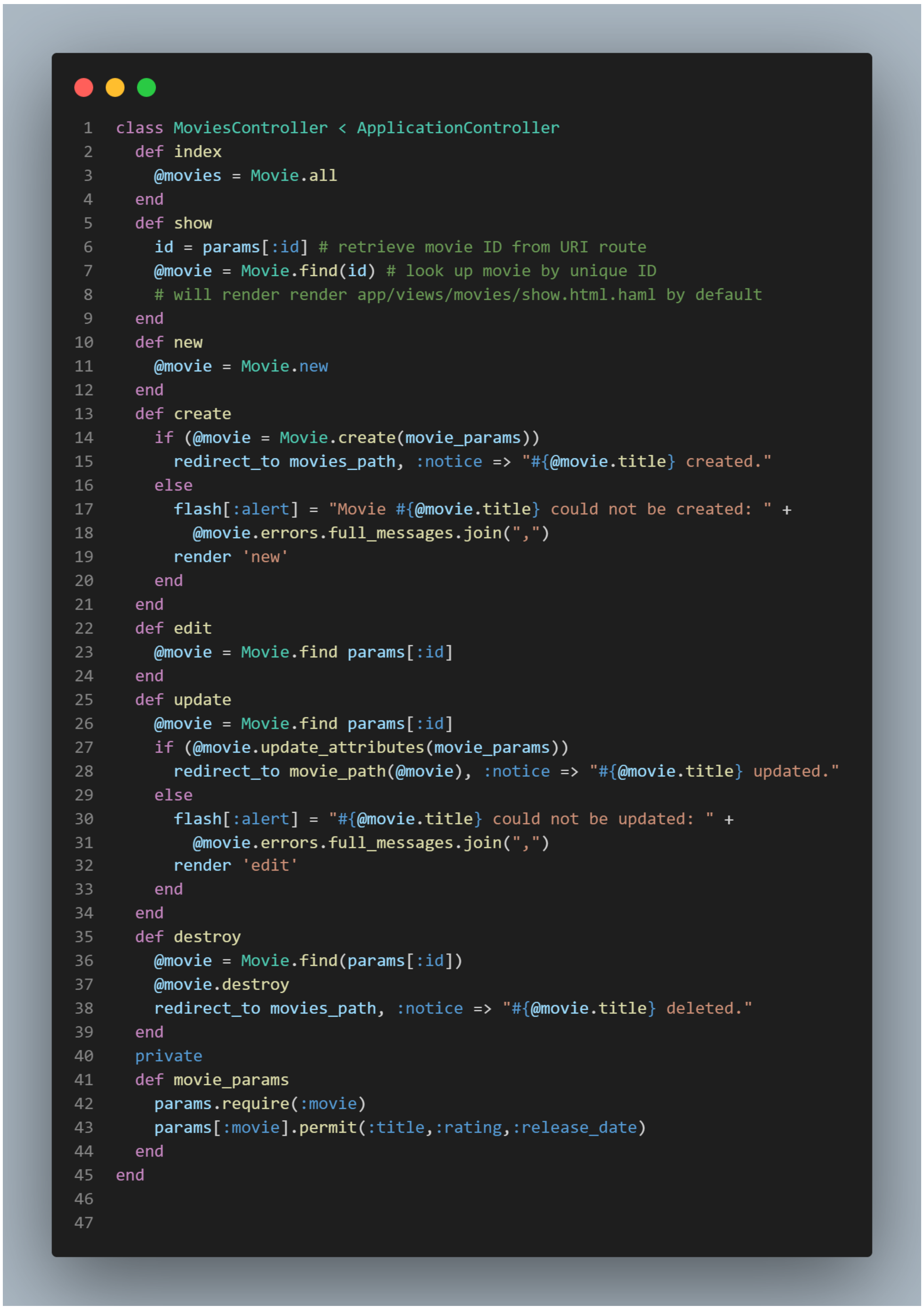
Comprobamos los resultados en consola:



```
angello@LAPTOP-QS9DFR6S:/mnt/c/Users/ANGELLO/Desktop/DesarrolloDeSoftware/Semana7/myrottenpotatoes$ rails console
Warning: the running version of Bundler (2.2.3) is older than the version that created the lockfile (2.4.18). We suggest you to upgrade to the version that cr
eated the lockfile by running `gem install bundler:2.4.18`.
Loading development environment (Rails 7.0.8)
3.0.0 :001 > m = Movie.new(:title => '', :rating => 'RG', :release_date => '1929-01-01')
 =>
#<Movie:0x00007f06108a0378
...
3.0.0 :002 > m.valid?
 => false
3.0.0 :003 > m.errors[:title]
 => ["can't be blank"]
3.0.0 :004 > m.errors[:rating]
 => []
3.0.0 :005 > m.errors[:release_date]
 => ["must be 1930 or later"]
3.0.0 :006 > m.errors.full_messages
 => ["Title can't be blank", "Release date must be 1930 or later"]
3.0.0 :007 > █
```

Comprobamos que el resultado es falso, es se debido a que la clase Movie valida algunos campos del objeto Movie, instanciamos pero la consola nos alerta que **m no es valido, porque no tiene titulo, y porque la ‘release_date’ debe ser en 1930 o después;**

Explica el siguiente código:



```ruby
class MoviesController < ApplicationController
  def index
    @movies = Movie.all
  end
  def show
    id = params[:id] # retrieve movie ID from URI route
    @movie = Movie.find(id) # look up movie by unique ID
    # will render render app/views/movies/show.html.haml by default
  end
  def new
    @movie = Movie.new
  end
  def create
    if (@movie = Movie.create(movie_params))
      redirect_to movies_path, :notice => "#{@movie.title} created."
    else
      flash[:alert] = "Movie #{@movie.title} could not be created: " +
        @movie.errors.full_messages.join(",")
      render 'new'
    end
  end
  def edit
    @movie = Movie.find params[:id]
  end
  def update
    @movie = Movie.find params[:id]
    if (@movie.update_attributes(movie_params))
      redirect_to movie_path(@movie), :notice => "#{@movie.title} updated."
    else
      flash[:alert] = "#{@movie.title} could not be updated: " +
        @movie.errors.full_messages.join(",")
      render 'edit'
    end
  end
  def destroy
    @movie = Movie.find(params[:id])
    @movie.destroy
    redirect_to movies_path, :notice => "#{@movie.title} deleted."
  end
  private
  def movie_params
    params.require(:movie)
    params[:movie].permit(:title,:rating,:release_date)
  end
end
```

index: Obtiene todos los registros de la tabla Movie y los asigna a la variable de instancia @movies. Esto se utiliza generalmente para mostrar una lista de películas en la interfaz de usuario.

show: Recupera el ID de la película desde la ruta URI, luego busca y asigna la película correspondiente a la variable de instancia @movie. La vista por defecto renderizada será app/views/movies/show.html.haml.

new: Inicializa una nueva instancia de Movie y la asigna a la variable de instancia @movie. Esto se usa para mostrar el formulario para crear una nueva película.

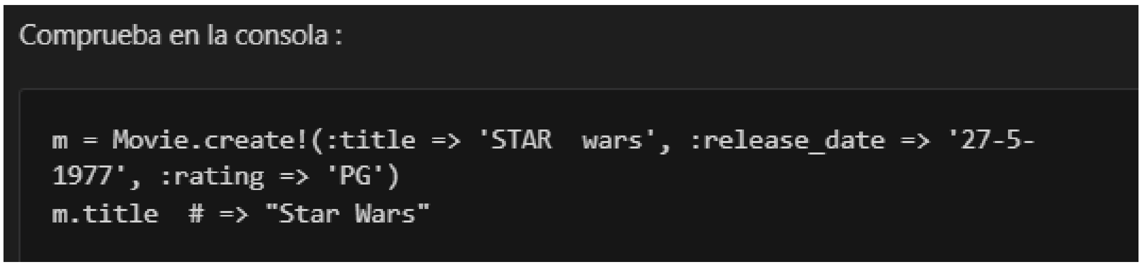
create: Intenta crear una nueva película con los parámetros proporcionados (movie_params). Si la creación es exitosa, redirige a la ruta de películas (movies_path) con un mensaje de éxito. Si hay errores, muestra un mensaje de error y vuelve a renderizar la vista '**new'.**

edit: Encuentra y asigna a la variable de instancia @**movie** la película que corresponde al ID proporcionado en los parámetros. Esto se usa para mostrar el formulario de edición de una película existente.

update: Encuentra la película correspondiente al ID proporcionado y intenta actualizar sus atributos con los parámetros proporcionados (movie_params). Si la actualización es exitosa, redirige a la ruta de la película actualizada con un mensaje de éxito. Si hay errores, muestra un mensaje de error y vuelve a renderizar la vista **'edit'.**

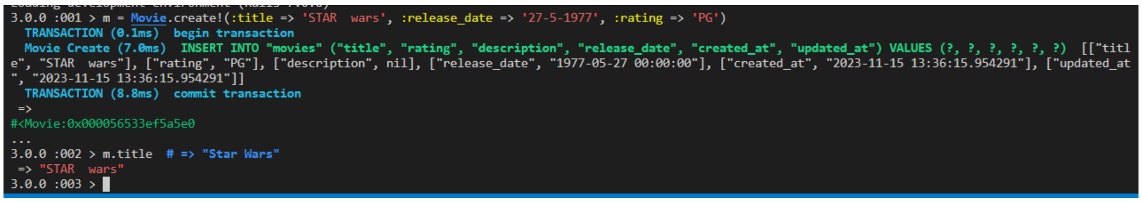
destroy: Encuentra la película correspondiente al ID proporcionado y la elimina de la base de datos. Luego, redirige a la ruta de películas con un mensaje de éxito.

movie_params: Método privado utilizado para filtrar y permitir solo los parámetros específicos necesarios para crear o actualizar una película. En este caso, permite solo los parámetros **title, rating y release_date** dentro de los parámetros **movie** proporcionados en la solicitud.


Comprueba en la consola :

```
m = Movie.create!(:title => 'STAR  wars', :release_date => '27-5-
1977', :rating => 'PG')
m.title  # => "Star Wars"
```

Comprobamos:

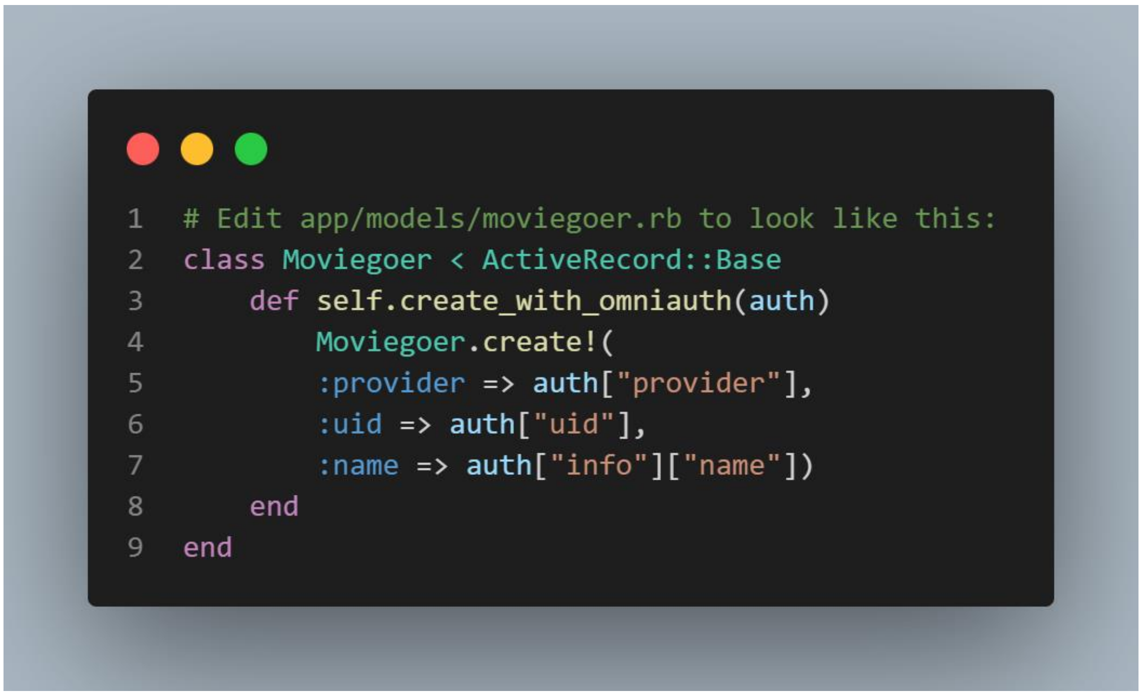


```
3.0.0 :001 > m = Movie.create!(:title => 'STAR  wars', :release_date => '27-5-1977', :rating => 'PG')
  TRANSACTION (0.1ms)  begin transaction
  Movie Create (7.0ms)  INSERT INTO "movies" ("title", "rating", "description", "release_date", "created_at", "updated_at") VALUES (?, ?, ?, ?, ?, ?)  [["titl
e", "STAR  wars"], ["rating", "PG"], ["description", nil], ["release_date", "1977-05-27 00:00:00"], ["created_at", "2023-11-15 13:36:15.954291"], ["updated_at
", "2023-11-15 13:36:15.954291"]]
  TRANSACTION (8.8ms)  commit transaction
 =>
#<Movie:0x000056533ef5a5e0
...
3.0.0 :002 > m.title  # => "Star Wars"
 => "STAR  wars"
3.0.0 :003 >
```

Usamos el método **create!** para validar los datos que consignamos antes de insertar la tupla en la base de datos, por ello usa **transaction**; si la validación falla, manda una excepción.

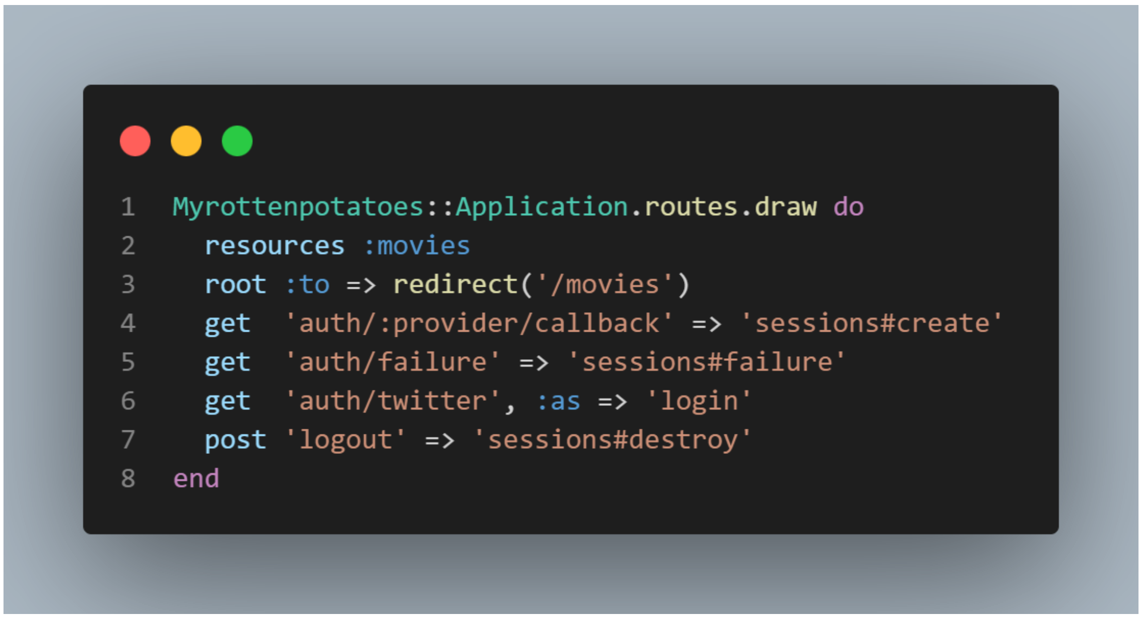
SSO y autenticación de terceros

Generamos el modelo moviegoers y una migración



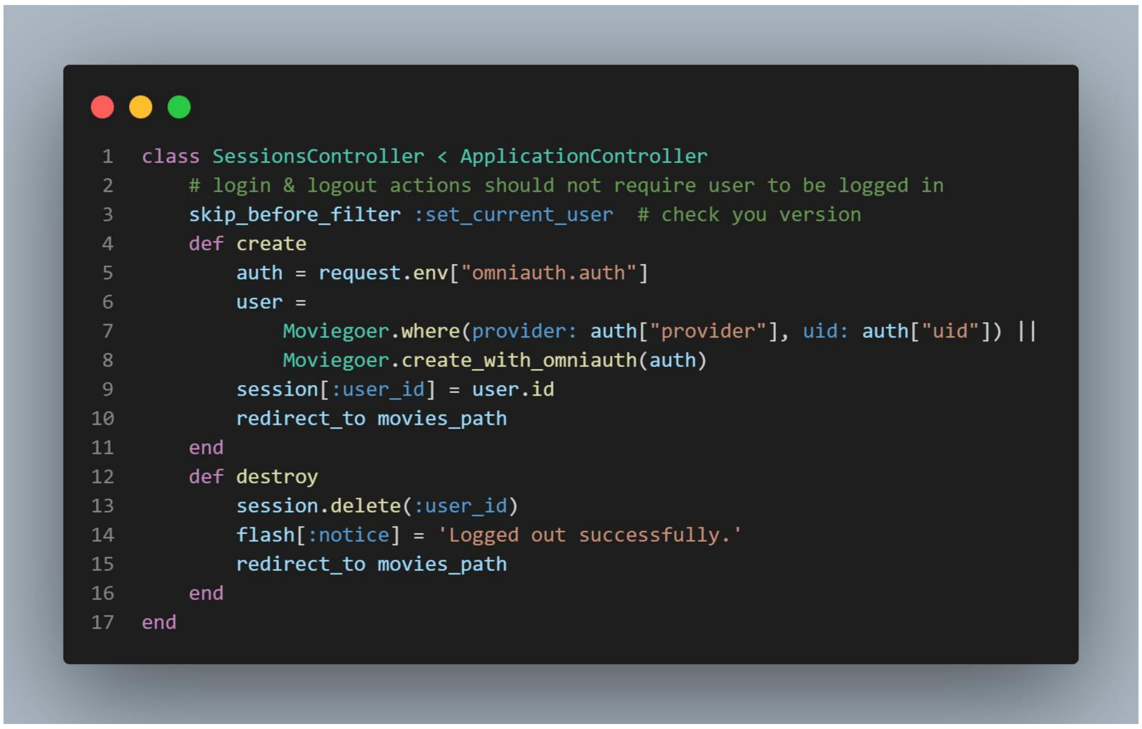
```ruby
# Edit app/models/moviegoer.rb to look like this:
class Moviegoer < ActiveRecord::Base
    def self.create_with_omniauth(auth)
        Moviegoer.create!(
        :provider => auth["provider"],
        :uid => auth["uid"],
        :name => auth["info"]["name"])
    end
end
```

Actualizamos routes.db



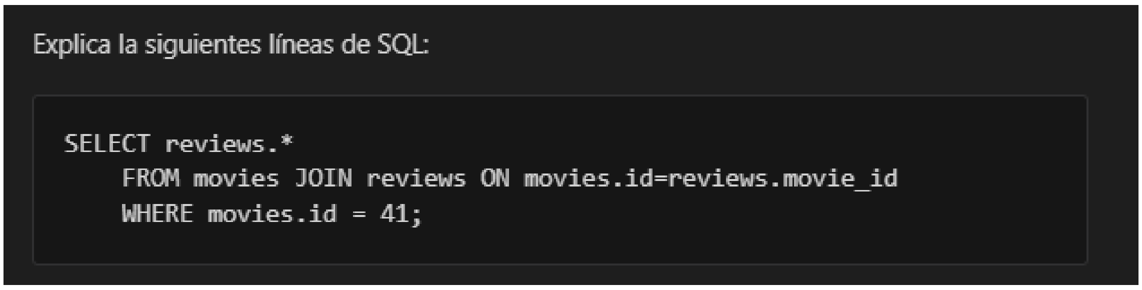
```ruby
Myrottenpotatoes::Application.routes.draw do
  resources :movies
  root :to => redirect('/movies')
  get  'auth/:provider/callback' => 'sessions#create'
  get  'auth/failure'  => 'sessions#failure'
  get  'auth/twitter', :as => 'login'
  post 'logout' => 'sessions#destroy'
end
```

Y creamos su controlador sessions_controller:



```ruby
class SessionsController < ApplicationController
    # login & logout actions should not require user to be logged in
    skip_before_filter :set_current_user  # check you version
    def create
        auth = request.env["omniauth.auth"]
        user =
            Moviegoer.where(provider: auth["provider"], uid: auth["uid"]) ||
            Moviegoer.create_with_omniauth(auth)
        session[:user_id] = user.id
        redirect_to movies_path
    end
    def destroy
        session.delete(:user_id)
        flash[:notice] = 'Logged out successfully.'
        redirect_to movies_path
    end
end
```

Claves Foráneas:



Explica la siguientes líneas de SQL:

```sql
SELECT reviews.*
    FROM movies JOIN reviews ON movies.id=reviews.movie_id
    WHERE movies.id = 41;
```

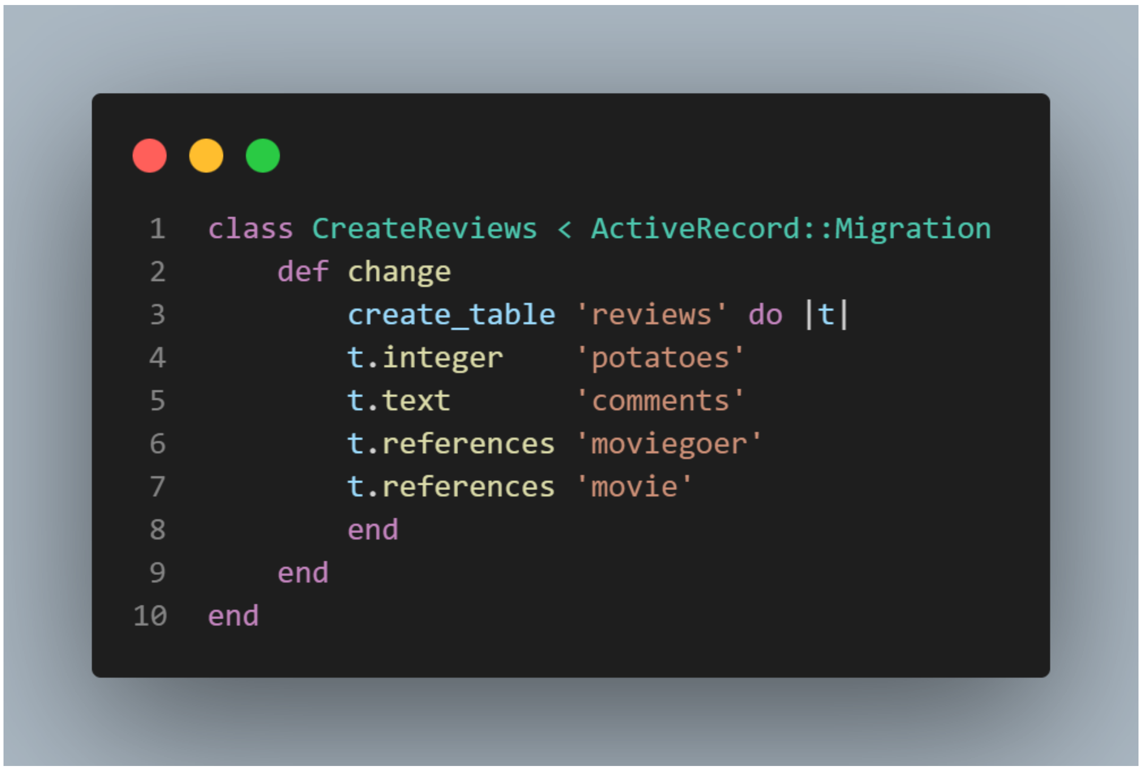
SELECT reviews.*: Selecciona todas las columnas de la tabla reviews.

FROM movies JOIN reviews ON movies.id=reviews.movie_id: Realiza una operación de JOIN entre las tablas movies y reviews utilizando la condición de igualdad **movies.id = reviews.movie_id**.

WHERE movies.id = 41: Filtra las filas para seleccionar solo aquellas donde el id de la tabla movies es igual a 41.

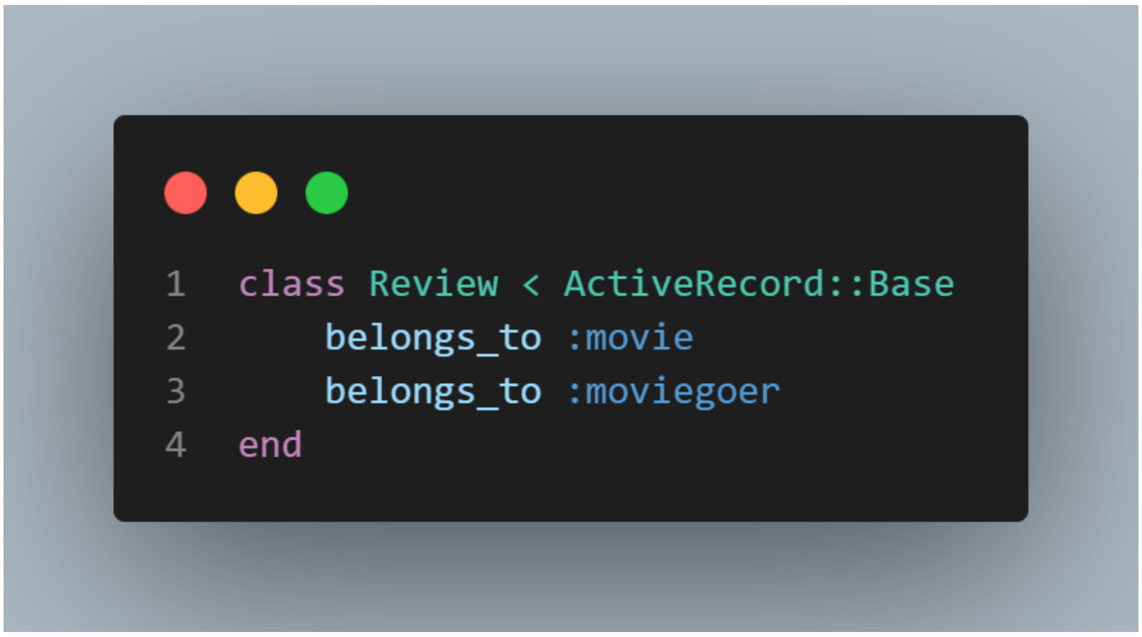
La consulta busca y selecciona todas las revisiones asociadas a una película específica identificada por el id 41 en la tabla movies.

Ahora creamos la migración y la tabla reviews:



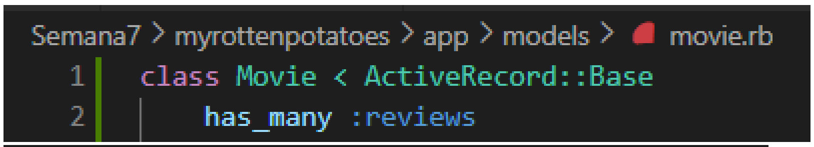
```ruby
class CreateReviews < ActiveRecord::Migration
    def change
        create_table 'reviews' do |t|
        t.integer    'potatoes'
        t.text       'comments'
        t.references 'moviegoer'
        t.references 'movie'
        end
    end
end
```

También creamos un nuevo modelo review.rb

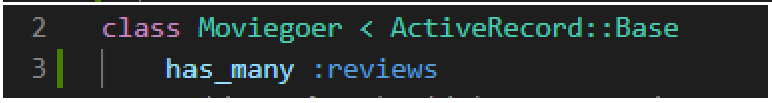


```ruby
class Review < ActiveRecord::Base
    belongs_to :movie
    belongs_to :moviegoer
end
```

Inseramos la línea de codigo:

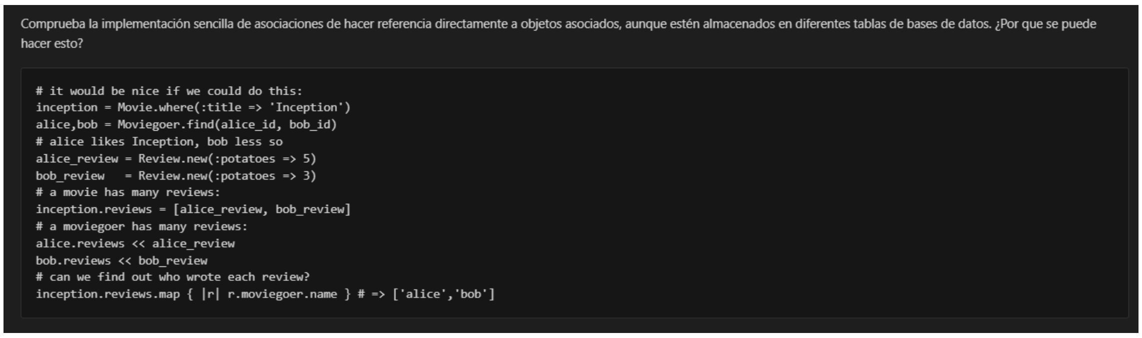


```
Semana7 > myrottenpotatoes > app > models > movie.rb
class Movie < ActiveRecord::Base
    has_many :reviews
```



```ruby
class Moviegoer < ActiveRecord::Base
    has_many :reviews
```

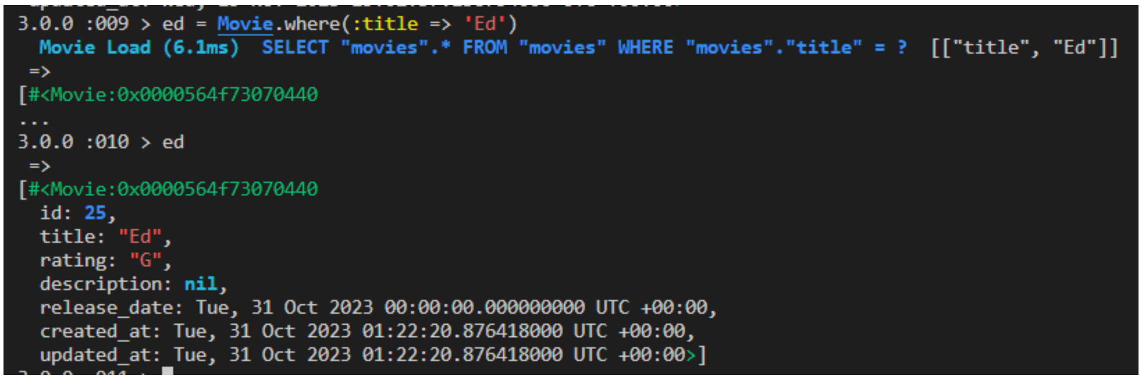
Pregunta:



Comprueba la implementación sencilla de asociaciones de hacer referencia directamente a objetos asociados, aunque estén almacenados en diferentes tablas de bases de datos. ¿Por que se puede hacer esto?

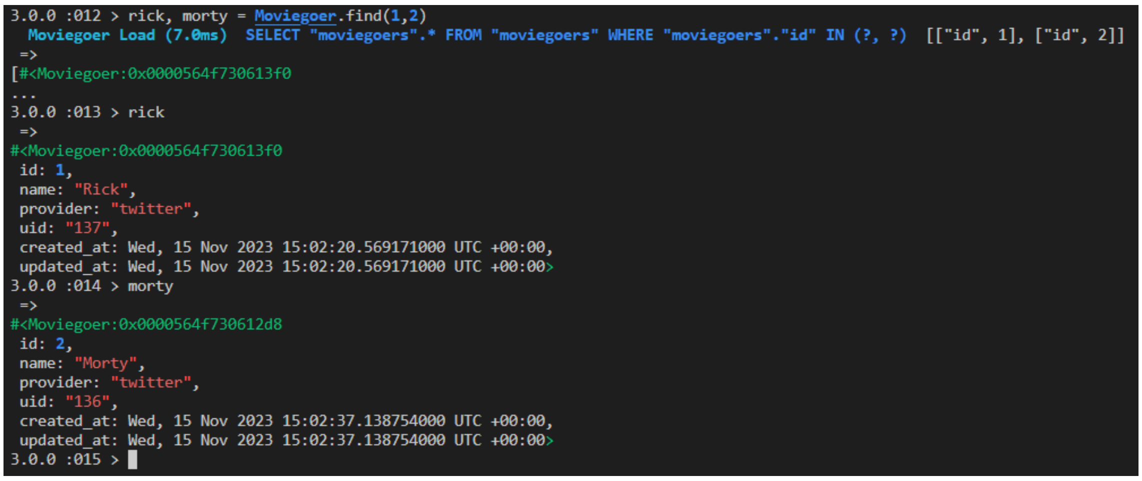
```ruby
# it would be nice if we could do this:
inception = Movie.where(:title => 'Inception')
alice,bob = Moviegoer.find(alice_id, bob_id)
# alice likes Inception, bob less so
alice_review = Review.new(:potatoes => 5)
bob_review   = Review.new(:potatoes => 3)
# a movie has many reviews:
inception.reviews = [alice_review, bob_review]
# a moviegoer has many reviews:
alice.reviews << alice_review
bob.reviews << bob_review
# can we find out who wrote each review?
inception.reviews.map { |r| r.moviegoer.name } # => ['alice','bob']
```

Guardamos la película Ed en la variable ed:



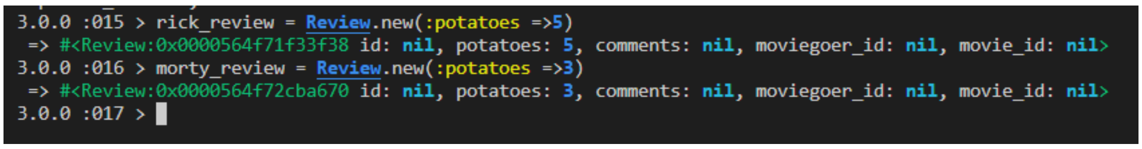
```
3.0.0 :009 > ed = Movie.where(:title => 'Ed')
  Movie Load (6.1ms)  SELECT "movies".* FROM "movies" WHERE "movies"."title" = ?  [["title", "Ed"]]
 =>
[#<Movie:0x0000564f73070440
...
3.0.0 :010 > ed
 =>
[#<Movie:0x0000564f73070440
  id: 25,
  title: "Ed",
  rating: "G",
  description: nil,
  release_date: Tue, 31 Oct 2023 00:00:00.000000000 UTC +00:00,
  created_at: Tue, 31 Oct 2023 01:22:20.876418000 UTC +00:00,
  updated_at: Tue, 31 Oct 2023 01:22:20.876418000 UTC +00:00>]
```

Creamos las variables Rick y morty que van a guardar los registros con id 1 y 2 respectivamente



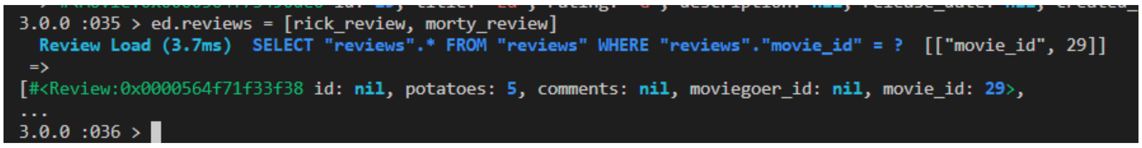
```
3.0.0 :012 > rick, morty = Moviegoer.find(1,2)
  Moviegoer Load (7.0ms)  SELECT "moviegoers".* FROM "moviegoers" WHERE "moviegoers"."id" IN (?, ?)  [["id", 1], ["id", 2]]
 =>
[#<Moviegoer:0x0000564f730613f0
...
3.0.0 :013 > rick
 =>
#<Moviegoer:0x0000564f730613f0
 id: 1,
 name: "Rick",
 provider: "twitter",
 uid: "137",
 created_at: Wed, 15 Nov 2023 15:02:20.569171000 UTC +00:00,
 updated_at: Wed, 15 Nov 2023 15:02:20.569171000 UTC +00:00>
3.0.0 :014 > morty
 =>
#<Moviegoer:0x0000564f730612d8
 id: 2,
 name: "Morty",
 provider: "twitter",
 uid: "136",
 created_at: Wed, 15 Nov 2023 15:02:37.138754000 UTC +00:00,
 updated_at: Wed, 15 Nov 2023 15:02:37.138754000 UTC +00:00>
3.0.0 :015 >
```

Creamos las variables rick_review y morty_review con la calificación de cada uno



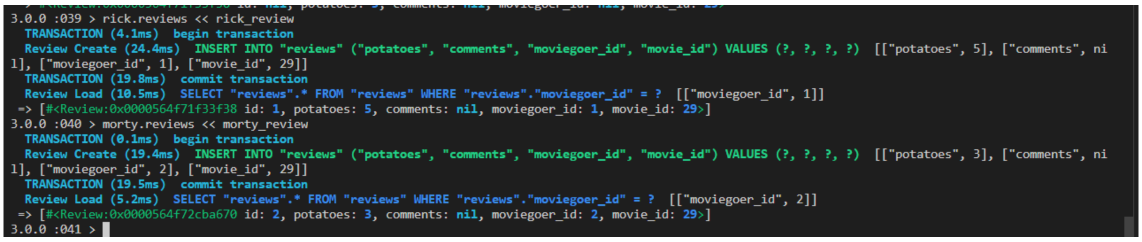
```
3.0.0 :015 > rick_review = Review.new(:potatoes =>5)
 => #<Review:0x0000564f71f33f38 id: nil, potatoes: 5, comments: nil, moviegoer_id: nil, movie_id: nil>
3.0.0 :016 > morty_review = Review.new(:potatoes =>3)
 => #<Review:0x0000564f72cba670 id: nil, potatoes: 3, comments: nil, moviegoer_id: nil, movie_id: nil>
3.0.0 :017 >
```

Ahora agregamos estas reviews recién creadas a la pelicula Ed cuya referencia esta guardada en ed



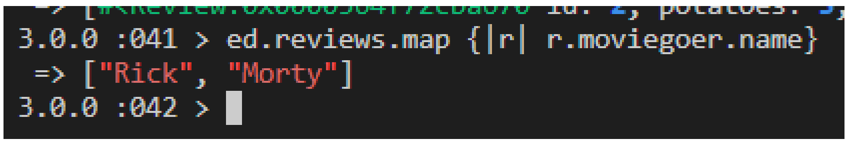
```
3.0.0 :035 > ed.reviews = [rick_review, morty_review]
  Review Load (3.7ms)  SELECT "reviews".* FROM "reviews" WHERE "reviews"."movie_id" = ?  [["movie_id", 29]]
 =>
[#<Review:0x0000564f71f33f38 id: nil, potatoes: 5, comments: nil, moviegoer_id: nil, movie_id: 29>,
...
3.0.0 :036 >
```

Por ultimo, guardamos el Rick_review dentro de las reviews de Rick, similarmente con Morty

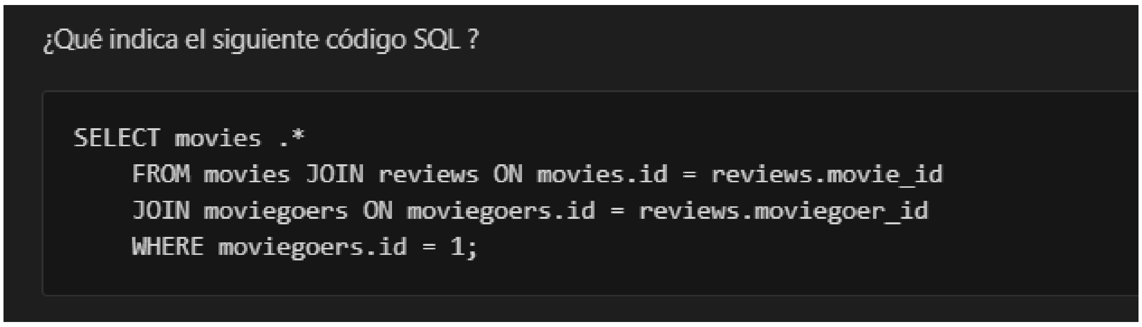


```
3.0.0 :039 > rick.reviews << rick_review
  TRANSACTION (4.1ms)  begin transaction
  Review Create (24.4ms)  INSERT INTO "reviews" ("potatoes", "comments", "moviegoer_id", "movie_id") VALUES (?, ?, ?, ?)  [["potatoes", 5], ["comments", ni
1], ["moviegoer_id", 1], ["movie_id", 29]]
  TRANSACTION (19.8ms)  commit transaction
  Review Load (10.5ms)  SELECT "reviews".* FROM "reviews" WHERE "reviews"."moviegoer_id" = ?  [["moviegoer_id", 1]]
 => [#<Review:0x0000564f71f33f38 id: 1, potatoes: 5, comments: nil, moviegoer_id: 1, movie_id: 29>]
3.0.0 :040 > morty.reviews << morty_review
  TRANSACTION (0.1ms)  begin transaction
  Review Create (19.4ms)  INSERT INTO "reviews" ("potatoes", "comments", "moviegoer_id", "movie_id") VALUES (?, ?, ?, ?)  [["potatoes", 3], ["comments", ni
1], ["moviegoer_id", 2], ["movie_id", 29]]
  TRANSACTION (19.5ms)  commit transaction
  Review Load (5.2ms)  SELECT "reviews".* FROM "reviews" WHERE "reviews"."moviegoer_id" = ?  [["moviegoer_id", 2]]
 => [#<Review:0x0000564f72cba670 id: 2, potatoes: 3, comments: nil, moviegoer_id: 2, movie_id: 29>]
3.0.0 :041 >
```

Y ahora podemos ver quien hizo las reviews de la película Ed:



```
3.0.0 :041 > ed.reviews.map {|r| r.moviegoer.name}
 => ["Rick", "Morty"]
3.0.0 :042 >
```


¿Qué indica el siguiente código SQL ?

```
SELECT movies .*
    FROM movies JOIN reviews ON movies.id = reviews.movie_id
    JOIN moviegoers ON moviegoers.id = reviews.moviegoer_id
    WHERE moviegoers.id = 1;
```

Respuesta:

La consulta está buscando todas las películas que han sido revisadas por el moviegoer con el id 1, y selecciona todas las columnas de esas películas.