

Contenido

[*Nuevo Proyecto - Aplicaciones* 2](#_Toc52357512)

[*Gemas*: 2](#_Toc52357513)

[*Bundle Install:* 3](#_Toc52357514)

[*Comandos:* 4](#_Toc52357515)

[*Migraciones:* 4](#_Toc52357516)

[*Controladores:* 7](#_Toc52357517)

[Modelos: 11](#_Toc52357518)

[*Vistas* 13](#_Toc52357519)

[FORMS 13](#_Toc52357520)

[Mensajes de error: 14](#_Toc52357521)

[Mensajes de éxito: 14](#_Toc52357522)

[*CRUD mediante consola Rails* 14](#_Toc52357523)

[*Validaciones* 16](#_Toc52357524)

[Requisitos 16](#_Toc52357525)

[*Hacks de visual* 16](#_Toc52357526)

[EQUIVALE A 16](#_Toc52357527)

[*Rutas* 17](#_Toc52357528)

[Vínculos (Link To..) 17](#_Toc52357529)

[*Relaciones* 18](#_Toc52357530)

[Rails interacción con BDD o Estadísticas 22](#_Toc52357531)

[Find 23](#_Toc52357532)

[Each Do 24](#_Toc52357533)

[Operador “<<” 24](#_Toc52357534)

[Bycrypt: 25](#_Toc52357535)

[Map: 27](#_Toc52357536)

[Uniq 27](#_Toc52357537)

[Sum 27](#_Toc52357538)

[Include & Joins 28](#_Toc52357539)

[Joins + where 28](#_Toc52357540)

[Limit/First/Last 28](#_Toc52357541)

Supongamos que no sabes por dónde empezar, que eres nuevo que es la primera vez que te topas con rails, ¿Qué hago, ¿cómo, ¿dónde cuándo?, Para ti que eres iniciante y que te sientes perdido, te dejo este manual que te ayudará a desenvolverte en TFC, y que no cometas los errores que yo cometí

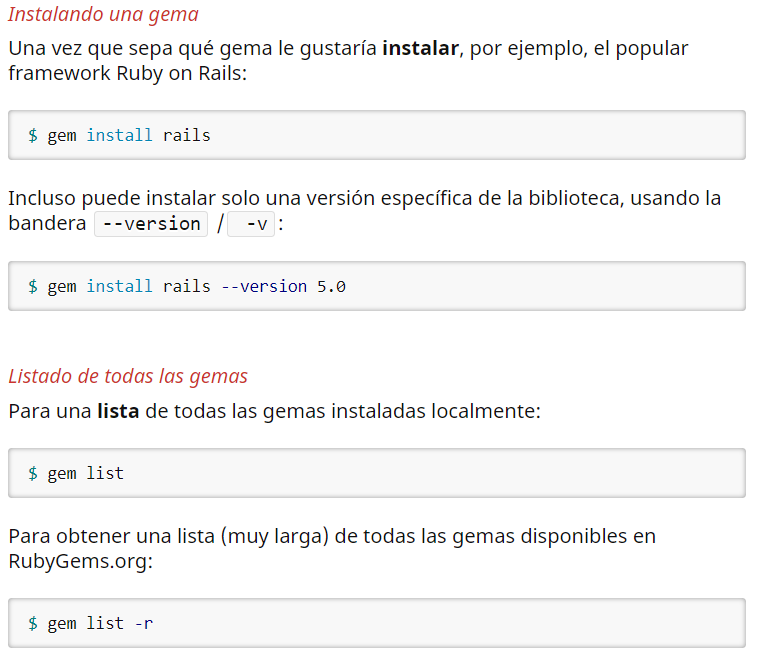
# Nuevo Proyecto - Aplicaciones

*rails new (aplicación)*

luego debes entrar en tu proyecto

# Gemas:

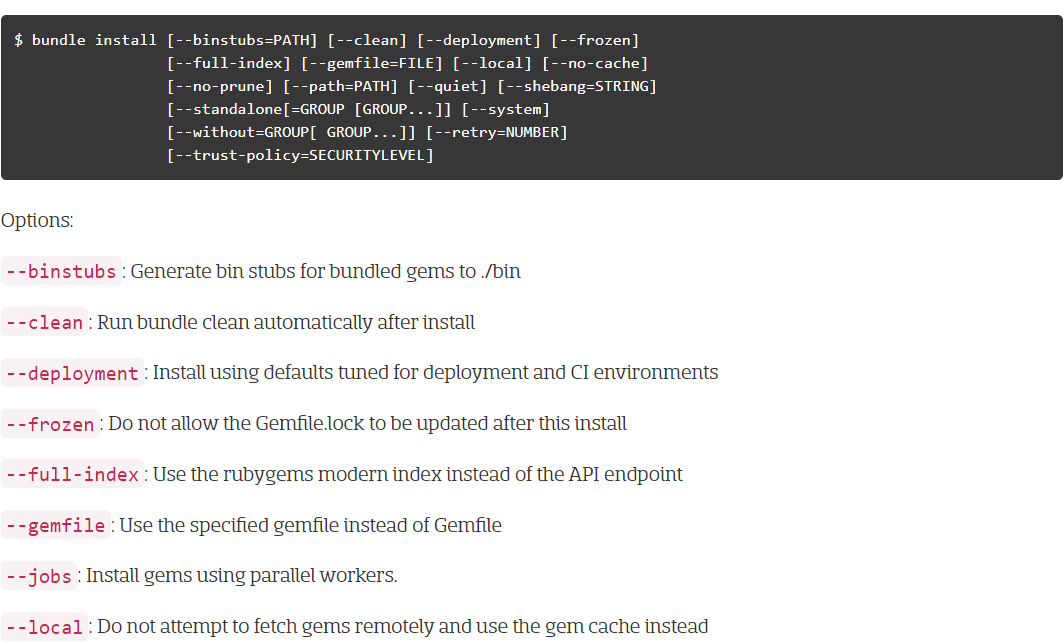
Gemas no deben ir de manera global. Deben ser de instaladas. Siempre es bueno sacar las gemas que no necesito del Gemfile.lock



Nota: debes estar en tu proyecto, no dentro de una de las carpetas del proyecto

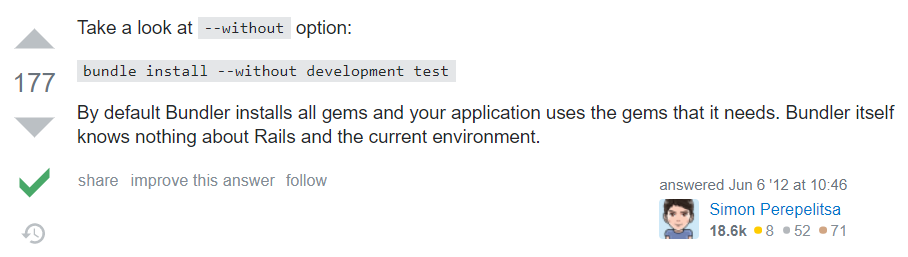
## Bundle Install:

*bundle install (Instala gemas que faltan)*



-- without: excluye las gemas que son parte de un grupo especificado.

Ejemplo:



Para TFC: --no-ri --no-rdoc --noducument

Ver más en: <https://bundler.io/bundle_install.html>

# Comandos:

* Rails new myApp
* rails generate controller [nombre controlador, siempre Mayúscula y en plural]
* rails d controller (nombre controlador)
* rails generate model [nombre modelo, siempre minúscula y en singular]
* rails db:migrate
* rails db:rollback (regresa a la última migración)
* rails generate migration Add**Atributo**To**Modelo** atributo:tipo atributo:tipo
* rails console (Abre una cosola de rails)

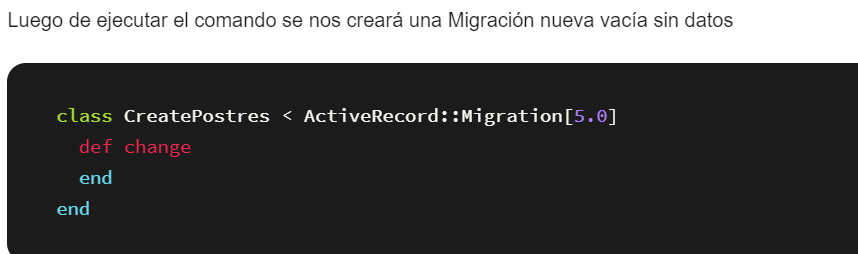
# Migraciones:

Una migración es un archivo que se crea dentro de la carpeta **db/migrate** y que contiene instrucciones para modificar el esquema de la base de datos (crear tablas, agregar columnas, eliminar columnas, eliminar tablas, etc.).

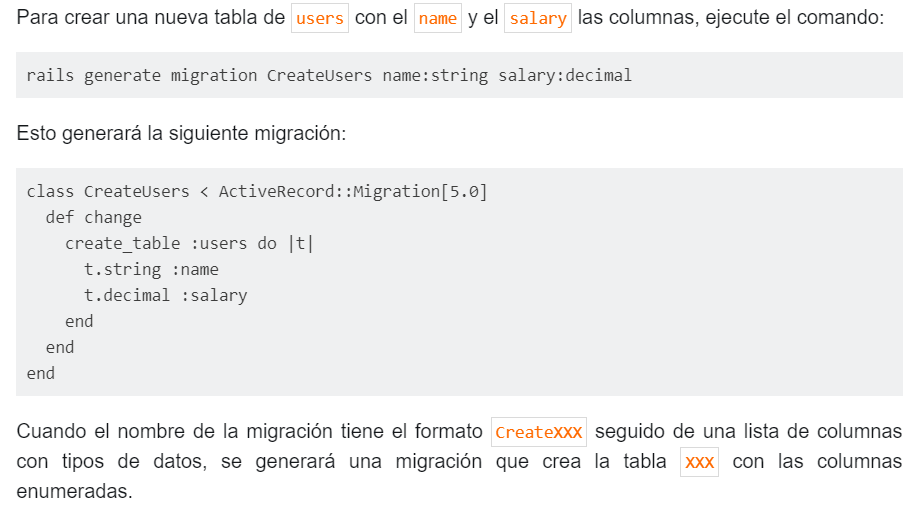
Cuando creas un modelo desde la línea de comandos con el generador de Rails, automáticamente se crea una migración con las instrucciones para crear la tabla.

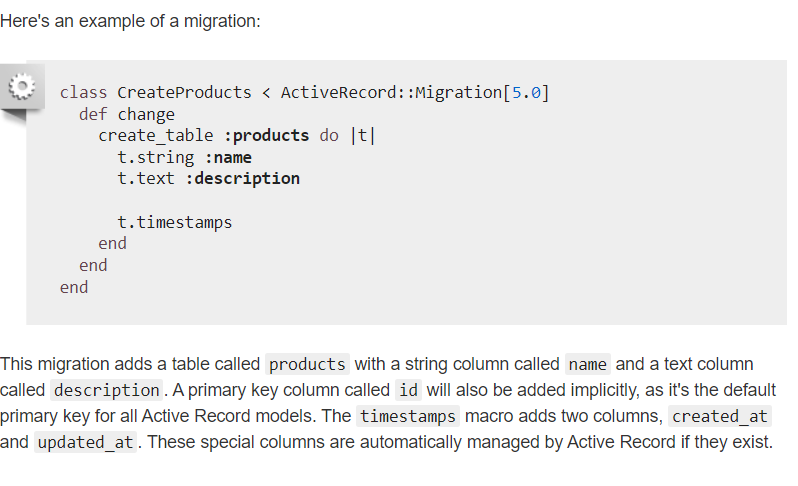
rails generate migration (create\_algo) este es el nombre de la migracion, piensa en una migración como en una versión de una tabla, que si esta no existe pues pones:

*rails generate migration create\_products*, el create es obligatorio, esto invoca al active record



También podemos crear una migracion con los campos de la tabla ya definidos:





Sin embargo, también puedes crear migraciones para agregar, remover o cambiar columnas de una tabla, entre otros.



dentro de mi base de datos creo la tabla “algo” y sus columnas

Nota: Si necesitas que algún campo tenga un valor por defecto: <https://stackoverflow.com/questions/7098602/add-a-default-value-to-a-column-through-a-migration>

Recuerda que si haces una migración e intentas hacer otra vez la migración:

*rails db:migrate*

no te genera uno nuevo. sino que se actualiza el esquema,

***Si quiero regresar a mi última migración:***

*rails db:rollback*

***Si quiero regresar a mi última migración****:*

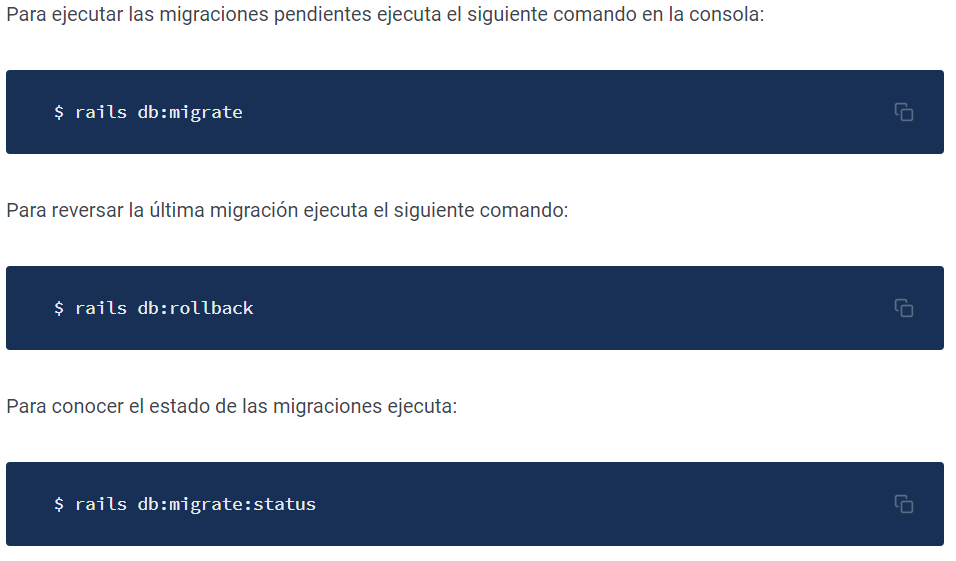
*rails db:rollback*

pero no es bueno si trabajas con Teams en proyectos, porque si compartes repositorios y un miembro descarga tu código y luego haces el rollback ese cambio no se reflejará en sus máquinas por tanto SIEMPRE debemos crear una nueva migración para que se reflejen los cambios directo en la base

Entonces entran en juego los famosísimos TimeStamps

Si quiero borrar una migración:

Esto sucede cuando en una tabla hay un atributo de más o el tipo de dato no es, o algo por el estilo, entonces la manera de borrar **es creando una nueva migración en la que se borre la tabla o bien, dar de baja el estatus**



si no quiero archivos innecesarios:

--no-frame

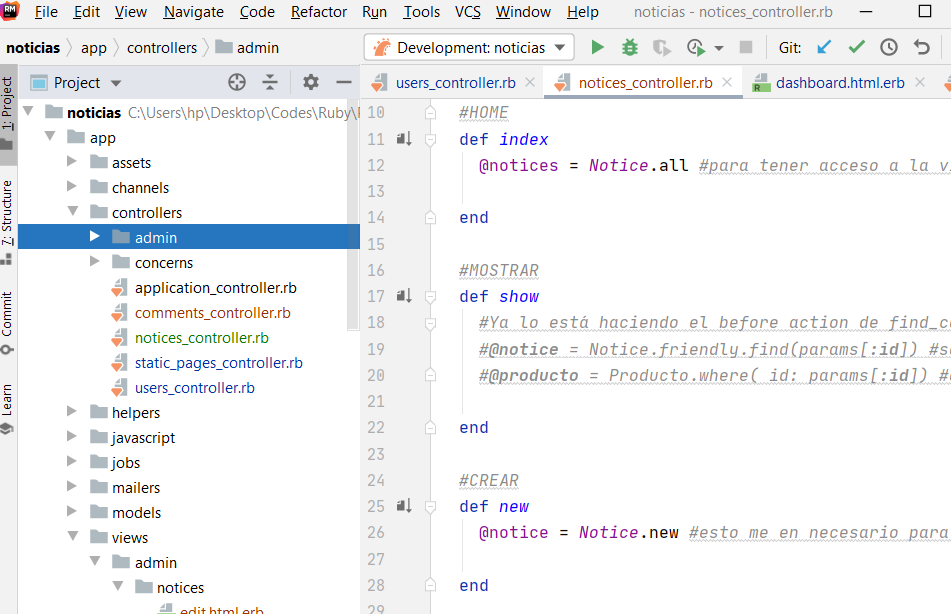
--no-test-unit

--no-assets

(Los ignora al rato de migrar)

# Controladores:

Tienen este aspecto:



***Crear un controlador:***

rails g controller (nombre controlador) o también:

rails g controlle (algo/controlador)

***Eliminarlo:***

rails d controller (nombre controlador)

Nota: sea eliminar o crear:

--no-test-unit --no-assets (Los ignora, sea al momento de elimnar o crearlos)

***La función home o index:***

*@algo = Algo.all #para tener acceso a la vista, @algo es como que global*

***La función mostrar:***

esta es igual a la de home, en el sentido de que me carga datos, pero la diferencia es que no me trae todos, solo de uno específico, por tanto:

*@algo = Algoo.find(params[:id]) #solo los detalles del producto (id)*

en este caso estoy buscando el "id" pero realmente podría ser cualquier otro atributo

***La función new:***

*@algo =algo.new*

esto me en necesario para instanciar un objeto al momento de que cargue el crear, si no tengo nada creado aun, no me puede dar error, esta función viene junto con otra, créate:

***La función create:***

*@algo = Algo.new(algo\_params)*  
if *@algo.save*  
 flash[:notice] = "Registro Exitoso!" #Notice debe ir siempre, es un mensaje  
 *redirect\_to* *(admin)\_algo\_path(@algo)* #Esto me redirecciona al new (

#(admin) si este método está dentro de un cotrolador para adminelse  
 *render* 'new'  
end

***La función edit:***

*@algo = Algoo.find(params[:id]) #solo los detalles del producto (id)*

Viene acompañada de update

***La función update:***

if *@algo.update(algo\_params)* flash[:notice] = "Modificado con exito!"  
 *redirect\_to (admin)\_algo\_path(@algo)*  
else  
 *render* 'edit'  
end

**La *función destroy:***

@patient = *Patient*.find(params[:id])*#solo detalles del producto (id)*@patient.destroy  
 *redirect\_to* room\_path

**Métodos privados:** solo acceden los métodos dentro de este contriller

*private*

*def producto\_params*

*params.require(:algo).permit(:atributo1, :atributo2…)*

*end*

Este método privado es el que le pasamos como parámetro y entonces se ejecuta todo el código de allí

“De los parámetros que voy a recibir, revisa que siempre venga un parámetro “algo(singular)”, luego de validar, permite que el usuario grabe el nombre y la cantidad (atributos como tal)

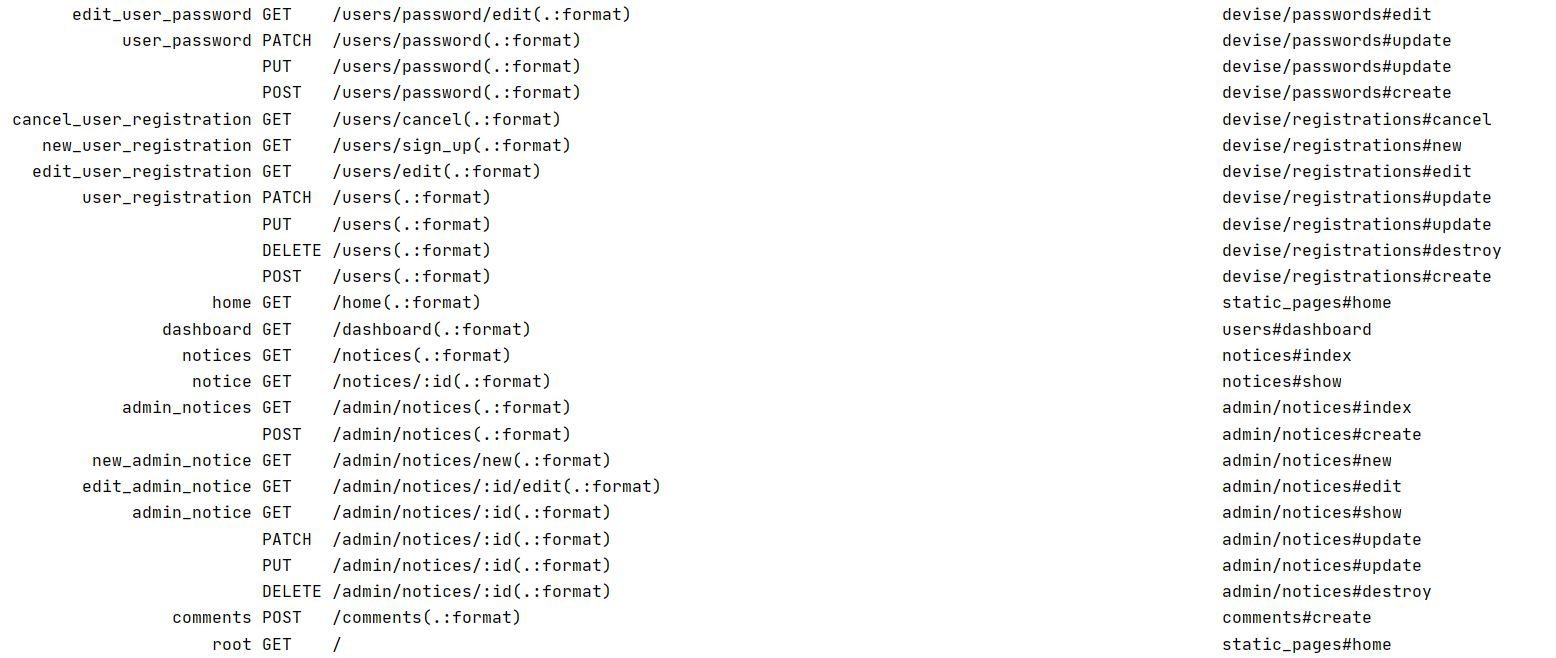
El :algo siempre es el singular porque referencia al modelo, no a la tabla

**Redireccionar**

¿Qué pasa cuándo completa una acción como crear o eliminar? simplemente se queda ahí, no sabe a dónde redireccionar, para redireccionar una vez que haya terminado una acción, Si usamos el comando*:*

*rails routes*

veremos algo como esto:



Prefix Verb IRL PATTERN

algo GET /algo(.:format) algo#index

algo GET /algo/:id(.:format) algo#show

root GET / algo#index

vemos que tenemos un prefijo "algo" (el /algo/) eso significa que si queremos usar ese prefijo

dentro de nuestra app, debemos agregarlo en la ruta, siguiendo el patrón que necesite, es decir para mostrar por id tenemos /algo/:id

Tonces nosotros a más de dar el prefijo debemos dar a dónde queremos que vaya

*redirect\_to (admin)\_algo\_path(@algo)*

Ejemplo

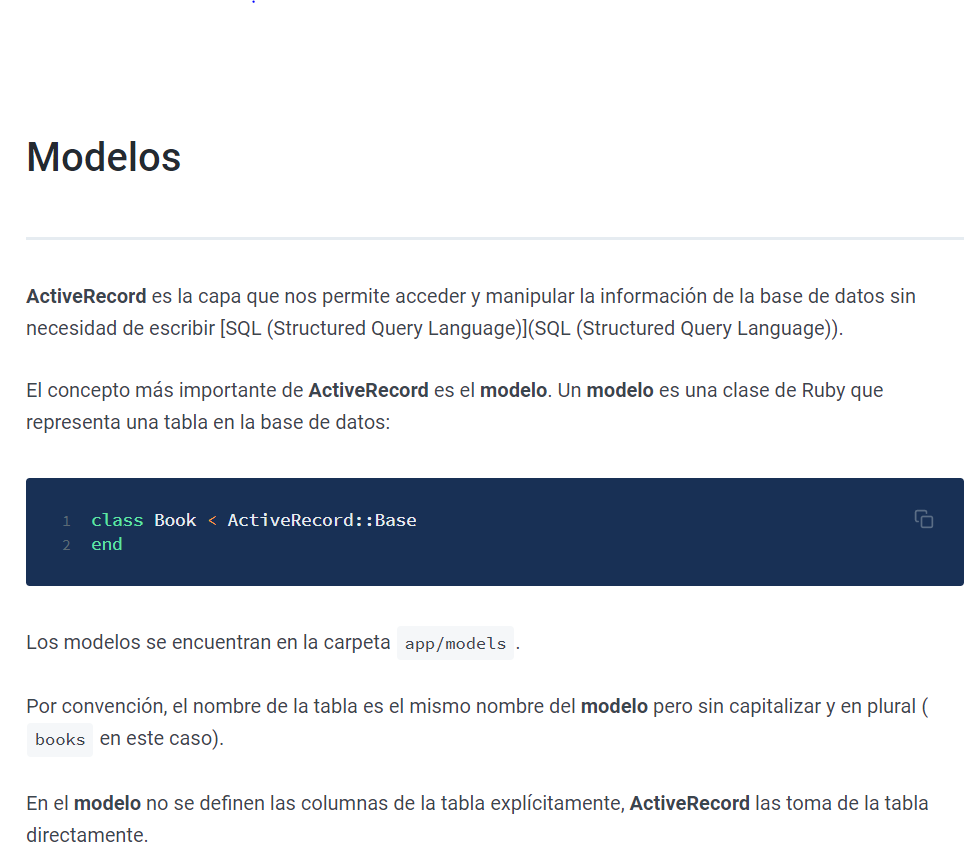
*redirect\_to admin\_notices\_path(@notice)* (Es el index de noticias de admin)

*redirect\_to notices\_path(@notice)* (Es el index de noticias)

**Mensajes de Confirmación**

Si se registra es necesario que se le advierta al usaurio porque si no se hace su registro, no hay nada que le diga y eso no está chévere, si regresas a ver en la acción “crear” vas a ver un notice, ese es el que nos notifica que la acción se ha llevado a cabo satisfactoriamente

# Modelos:



Para crear un modelo*: rails g model Modelo atributo:tipo atributo2:tipo atributo3:tipo ......*

creamos manualmente un algo.rb en la carpeta models

luego escribimos:

*class (Modelo)< ApplicationRecord*

*end*

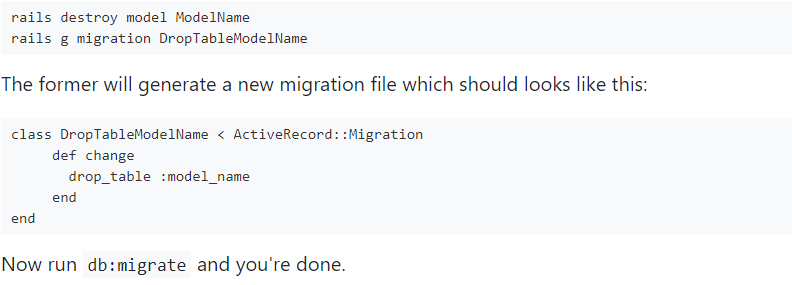
El modelo nos proporciona métodos getter y setter para comunicarnos con la base de datos

Recuerda siempre tipo Capital es el Modelo

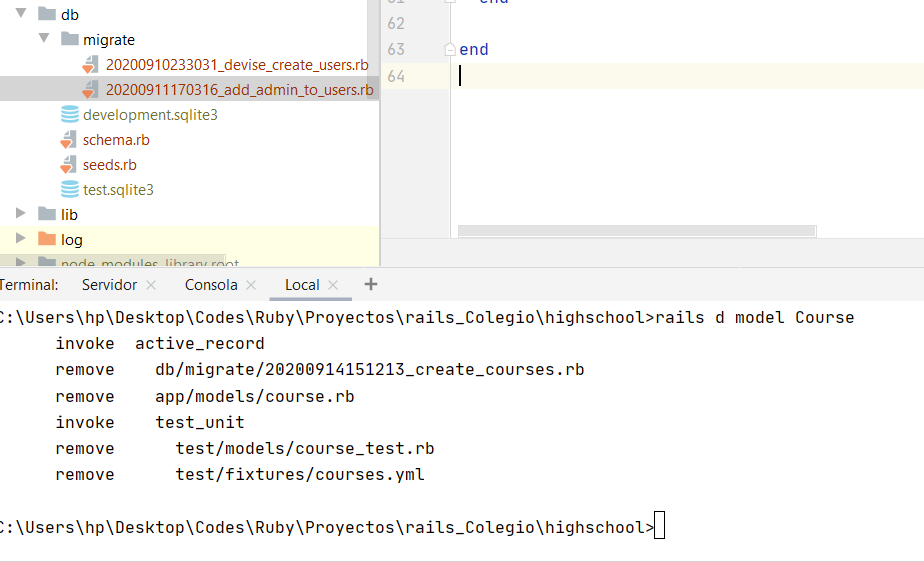


Para eliminar un modelo:

Verifica que el estado de la migración esté bajado



Automáticamente verás que la migración que estaba allí, ha desaparecido y solo quedarán las que tenías (Eso es gracias a que aún no habías hecho las migraciones):



Nota:

--no-test-unit --no-assets (Los ignora al rato de migrar)

# Vistas

Las vistas se crean teniendo ya el controlador y el modelo, estas se crean de manera manual, es decir sin comandos, simplemente nos desplazamos hasta **app/views** y crear una carpeta "algo" (por ejemplo, **si mi controlador es users\_controller**, esta **carpeta debería ser: users y dentro debería crear la vista de la acción correspondiente: create.html.erb, show.html.erb, etc.**

luego dentro de esa iría la plantilla de este algo! (Ten presente que el html.erb de new es el mismo para el edit)

## FORMS

Si en nuestro controlador, tenemos admin/controlle tenemos una función “create” necesitamos en la vista, proporcionar el modelo con el que tabajamos

*<%= form\_with model: [:admin, @algo], local: true do |f| %>*

*<%= end %>*

Si por otro lado tenemos un “algo\_controller” y tenemos un create en su él, necesitamos en la vista en su form:

*<%= form\_for(@algo, html: {class:"col-s12" }) do |elemento| %>*

*<%= end %>*

Una vez que termiamos de diseñar el form de nuevo registro al rato de dar click en "Guardar" debería invocarse el método "Crear"

Si en nuestro controlador, tenemos la función “delete”, necesitamos en la vista, proporcionar la solicitud:

*<%= link\_to "borrar", algoo\_path(algo), method: :delete, data: { confirm: "Estas seguro?"}*

*%>*

Ahora si ya le dije que le rootee a ese elemento y que me procese una solicitud de eliminación

Mensajes de error: Si queremos manejar también lo que son errores, que pueden presentarse a un usuario por algún motivo (contraseña incorrecta, etc), podemos al inicio del archivo **/new.html.erb**

*<% if @producto.errors.any? %>*

*<% @producto.errors.full\_messages.each do |errores| %>*

*<%= errores %>*

*<% end %>*

*<% end %>*

Mensajes de éxito: Si queremos manejar también lo que son mensajes de exito, que pueden presentarse a un usuario por algún motivo (logeo exitoso, etc), podemos (Siempre que hayamos establecido una confirmación en algún método en el controlador) al inicio del archivo **/application.html.erb en la sección del body:**

*<%= flash.each do |nombre, mensaje| %>*

*<%= mensaje %>*

*<% end %>*

# CRUD mediante consola Rails

Si entramos a: **rails console** , es ahí donde podemos hacer el código que posteriormente podría implementarse en la lógica del negocio, también podríamos hacer pruebas de conexión, etc.

***CREAR*** un nuevo “algo”:

En la consola de rails (rails c):

*Modelo.create(atributo: "nombre del atributo", atributo2: "nombre del atributo 2")*

mira que no le especifico el tipo, porque las columnas como que son por defecto String, pero lo ideal es especificarle el tipo de dato, así como tienes para el modelo “Book”

Luego también se crea un atributo id automáticamente

**OTRA** manera de crear es usando variables

*variable = (Un nuevo objeto de tipo Modelo)*

*variable = Algo.new*

Luego si queremos trabajar sobre los atributos de esa tabla (digamos las filas cabeceras)

*varaible.atributo = "algo chevere"*

*variable.nombre = “Zapatilla”*

Luego si queremos crear tanto el nuevo atributo más su valor:

*variable = Algo.new(atributo: "nombre del atributo" , atributo: "nombre del atributo" )*

¡Listo!

Luego que ya esté todo creado:

*variable.save*

***CONSULTAR*** un Producto

para uno especifico con un id especifico

*Algo.find(numero del id)*

¡Listo!

pero ¿Qué pasa si este artículo se elimina? el id desaparecería, porque el id es único entonces ¿Cómo saber cuál es el primero? (En caso de eliminar el id 1)

*Algo.first*

*Algo.last*

***CLASE VS VARIABLE***

La diferencia entre usar directamente la referencia a la clase y una variable es que con la referencia en la clase no puedo acceder al objeto del "Algo" que acabé de encontrar

¡Siempre es mejor crear una variable y asignarla a un objeto!

Lo que sería:

Algo.all == variable

Algo.find(id) == valriableSegmentada (enlista solo esa sección, esa fila de la tabla)

Algo.find(id).metodo no existe! pero,

variableSegmentada.metodo sí que existe!

Nota: variableSegmentada = variable.first, variable.where(id:3), variable.nombre, etc.

***EDITAR*** un algo

Getters

*varaibleSegmentada.atributo (me devuelve el valor de este atributo)*

con la referencia al modelo no hay como

Setters

*variavleSegmentada = "editacion de algo"*

ahora se ha editado, pero no a nivel de la base de datos,

¡CUIDADO!

¿Qué debemos hacer?

*variableSegmetnada.save*

***ELIMINAR*** un algo

*variableSegmentada.destroy*

**ERRORES** para ver los errores

variable.error.full\_messages

# Validaciones

Guardar “algo” con valores nulos no es viable para nuestro negocio.

Vamos al modelo, aplicamos el atributo “Precence”

class Producto < ApplicationRecord

valdiates: (atributo), presenece: true

end

De ese modo ya implementamos que si el atributo tiene valor nulo al momento de usar algo.save no se guarda de ese modo ya implementamos que si el atributo tiene valor nulo al momento de usar: algo.save no se guarda si el valor del atributo es nulo.

## Requisitos

luego de valdiates: (atributo), presenece: true , va el ,length: {}

validates :nombre, presence: true, length: { minimun: 6, maximum: 50 }

# Hacks de visual

CTRL + P (Abre el buscador de un archivo)

CTRL + F (Busca una palabra dentro de archivo actual)

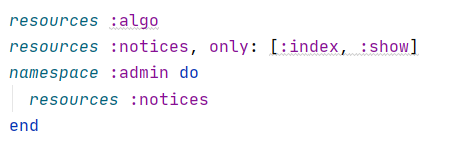
En la terminal, desplazarse hasta un proyecto y escribir: code . (Abirá visual studio en el archivo con todas las carpetas)

## EQUIVALE A

* ls:dir
* grep: findstr
* ls -a: dir /A

# Rutas

Hay un archivo dentro de config/routes.rb el cual contiene las rutas asociadas a cada método que existe en un controlador:



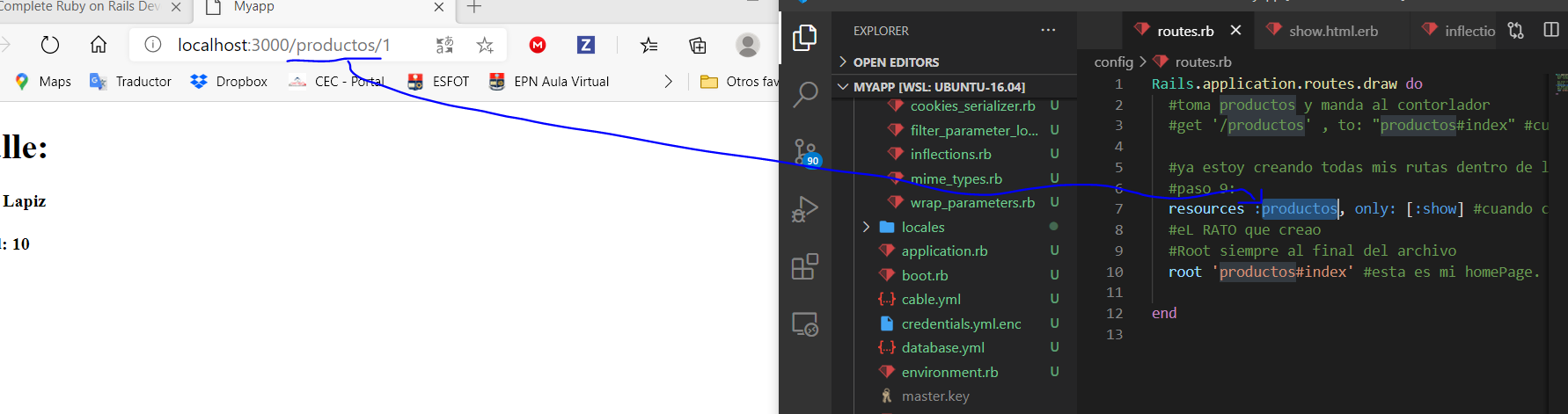
esto me va a dar todas las rutas que se necesiten para mi método de mi controlador, y sus respectivas acciones son (GET, POST, PUT…)

También podemos delimitar los verbos, para que el método solo tenga acciones determinadas, como ves en notices, por medio de “only”

Ejemplo:

resources :(algo), only: [:show] (:show es la única acción que estoy diciendo de la que quiero una ruta, pero puede ser, index, create, new, etc.)

El reosurces es lo que se ve en el URL:

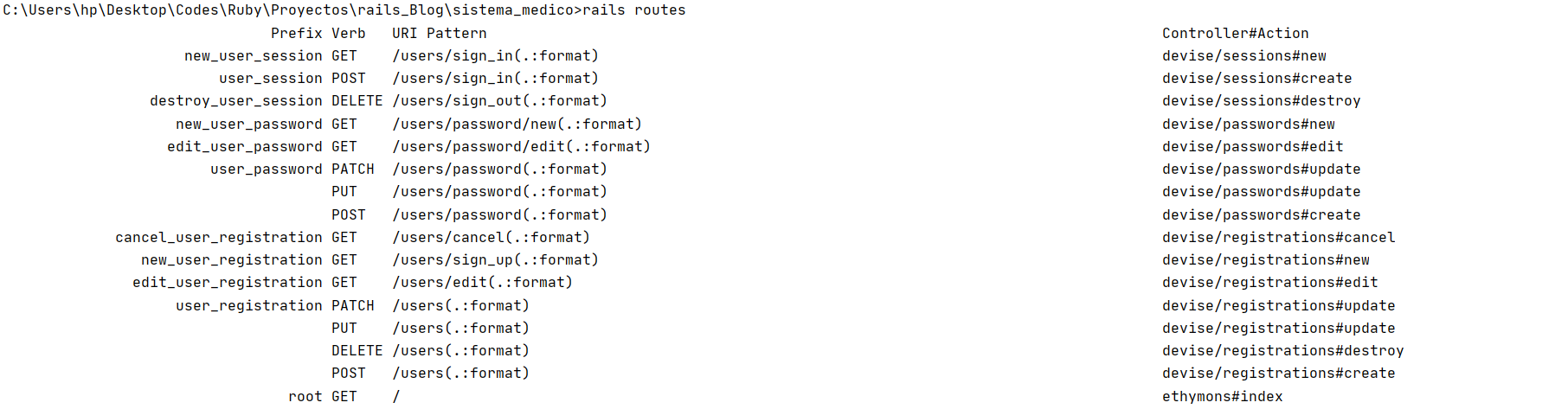


## Vínculos (Link To..)

Seguramente te has topado con estos y no sabes de qué va



Si usamos el comando rails routes veremos



La estructura que tenemos es Prefijo, verbo, URI, Controlador y Acción; es importante que sepas que el controlador es el que está a la izquierda y el método de ese controlador a la derecha: controladro#método

Cada método de ese controlador tiene su URL propia, ya que el verbo POST, GET, UPDATE así lo requiere para hacer.

Por útlimo, tenemos el prefijo, estos son la clave; si te fijas por ejemplo, el controlador ethymons tiene la acción index, y su prefijo es “root”

Si queremos vincular como tenemos en la imagen, simplemente debemos añadir “\_path” al prefijo y de ese modo ya estaríamos dirigiéndonos a esa acción

Para más facilidad puedes usar el comando rails routes -c “controlador” para ver las rutas de ése controlador en particular

# Relaciones

Existen relaciones de tipo 1 -m, m-m

Ejemplo:

Básicamente vamos a relacionar dos tablas

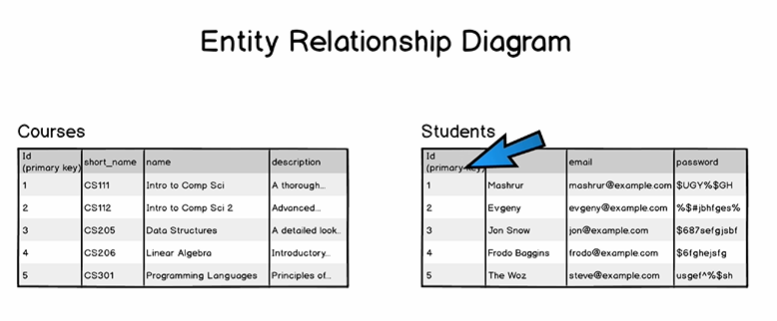
usuarios: username, email, password

productos: nombre, cantidad, estado

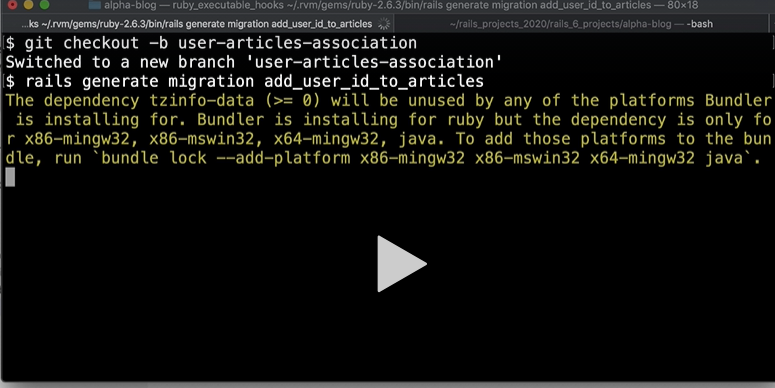
Vamos a relaciona un usuario a un producto ¿Cómo?

introduciendo un "user\_id" en la tabla de productos siendo este una "FK" que hará referencia la "PK" de usuarios que es el id. Por tanto, esta relación es 1 a m.

ERD: entity relation diagram



Lo que hacmos es hacer una migración para que la columna alumns tenga el id del curso al que está perteneciente

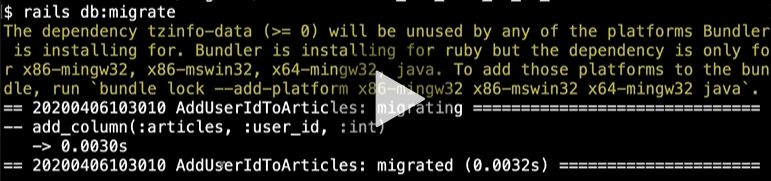


Si vamos a las migraciones veremos:

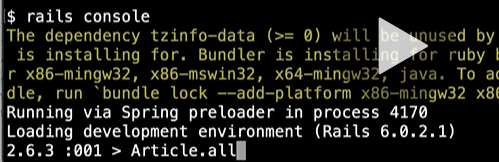


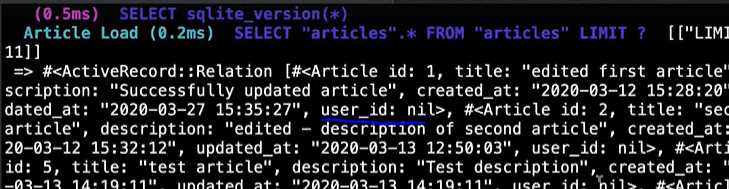
En este caso lo que se hizo es añadir la comuna a la tabla “ariculos”, qué columna? La del user\_id que es de tipo “int”

Ejecutamos la migración:



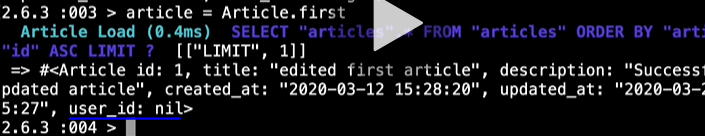
Comprobamos





Efectivamente se ha añadido el nuevo campo a nuestra tabla, pero por ahora tiene el valor nulo.

Observa que, en este caso hay un solo creador de un arituclo, con la asociación que hemos hecho lo que hacemos es encontrar al creador de un articulo haciendo referencia al usuario del articulo, es decir



Digamos que estas haciendo de un curso y un alumno, debería ser:

Course.first tener el id de un alumno?

O debería ser Alumn.firts tener el id de un curso?

Exacto, debería ser Alumn.first y a el estar asociado el id del curso, entonces, es a la tabla alumno a la que se le debe añadir el id del curso!

**Migraciones**

rails generate migration create\_users

voy a mi carpeta **db/migrate**  y allí voy a poder ver mis migrciones que acabe de crear

class CreateUsers < ActiveRecord::Migration[5.2]

def change

create\_table :users do |t|

end

end

end

Procedemos a llenar la tabla con los aributos y tipos de datos:

* t.string :username
* t.string :email
* t.timestamps
* etc

Por último, generamos la migración

rails db:migrate

Luego nos aseguramos de tener en el modelo, **app/models/** un archivo del mismo nombre del “**algo**” pero en **singular**. En este caso sería:

class User < ApplicationRecord

end

Lo último que nos queda por ver sería la parte de estadísticas.

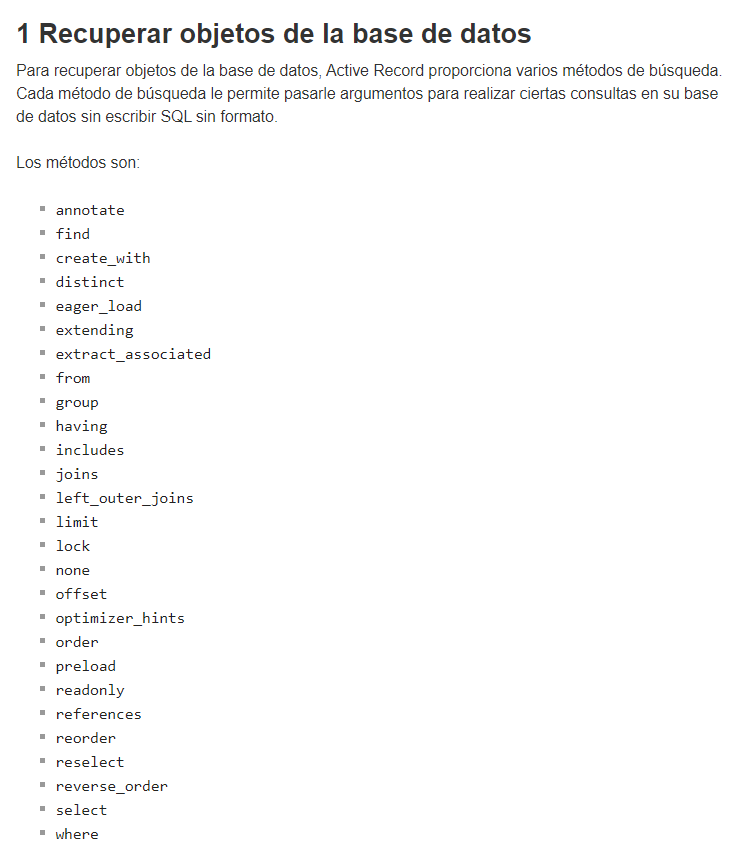
Las migraciones que haga no deben ser borradas manualmente, den ser borradas así:

1. Crear una migracion que borre la tabla
2. Dar de baja el status

Nota:

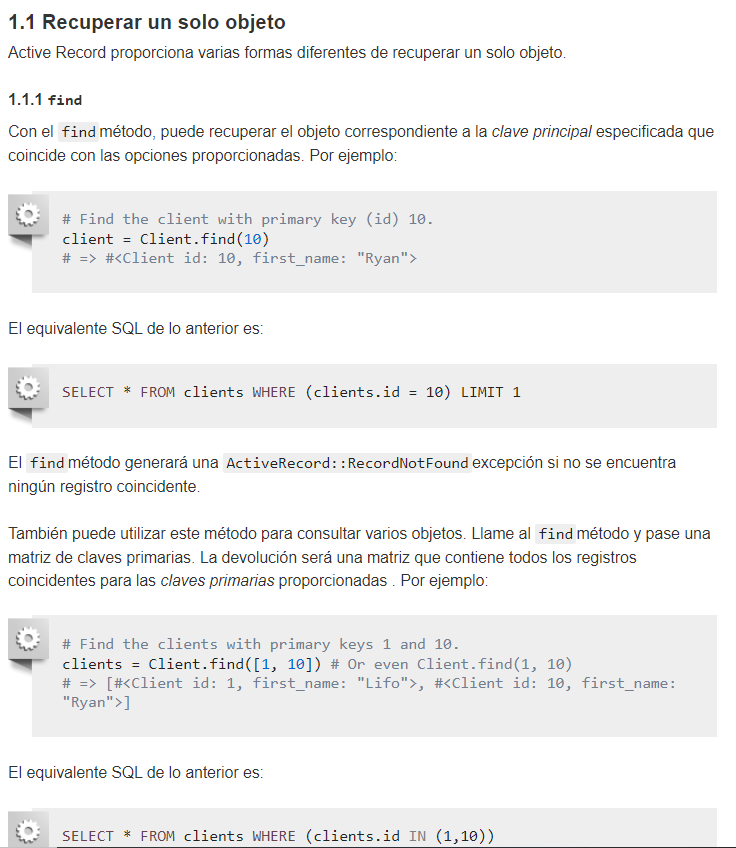
--no-test-unit --no-assets (Los ignora al rato de migrar)

# Rails interacción con BDD o Estadísticas

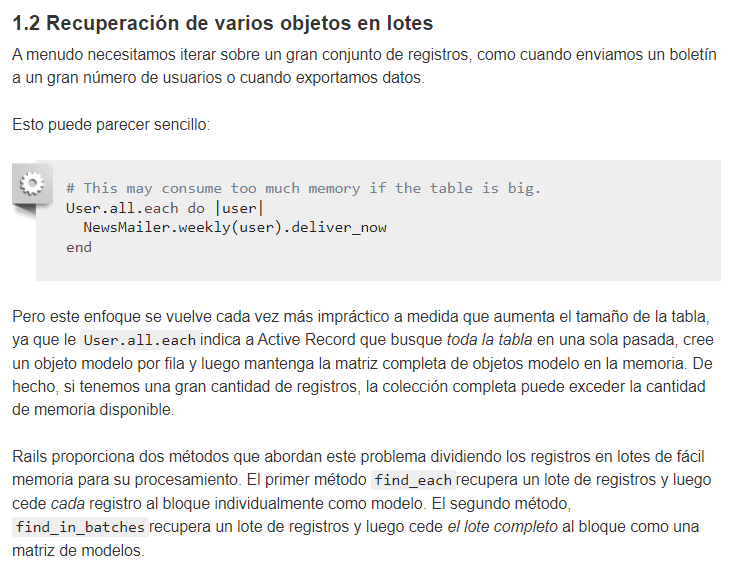
  
Cuando tengo un conjunto de datos, uso **do** para iterar sobre el array de objetos

Cuando tengo un **find** (un especifico) accedo a los atributos

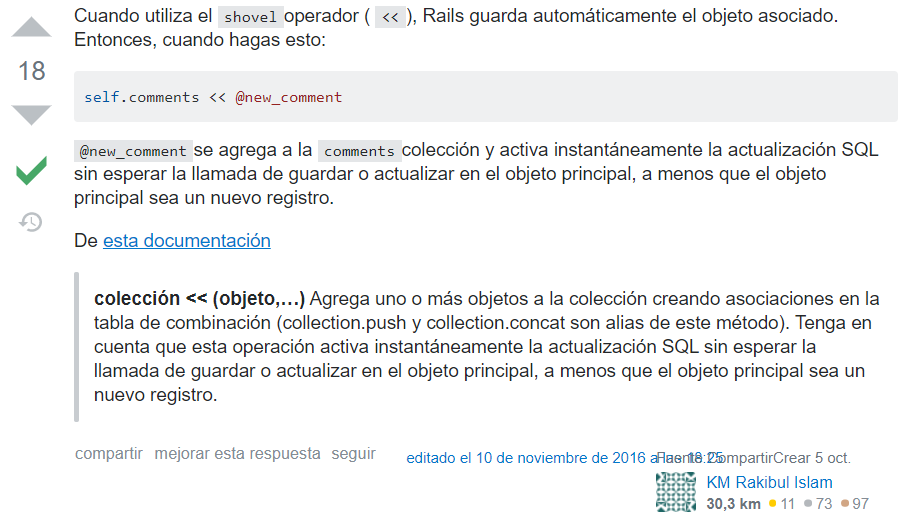
## Find



## Each Do



## Operador “<<”



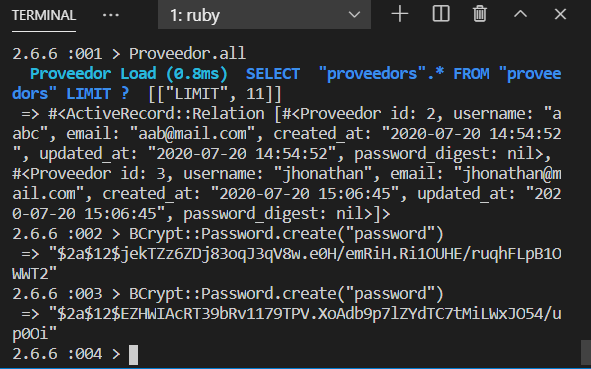
Ejemplo

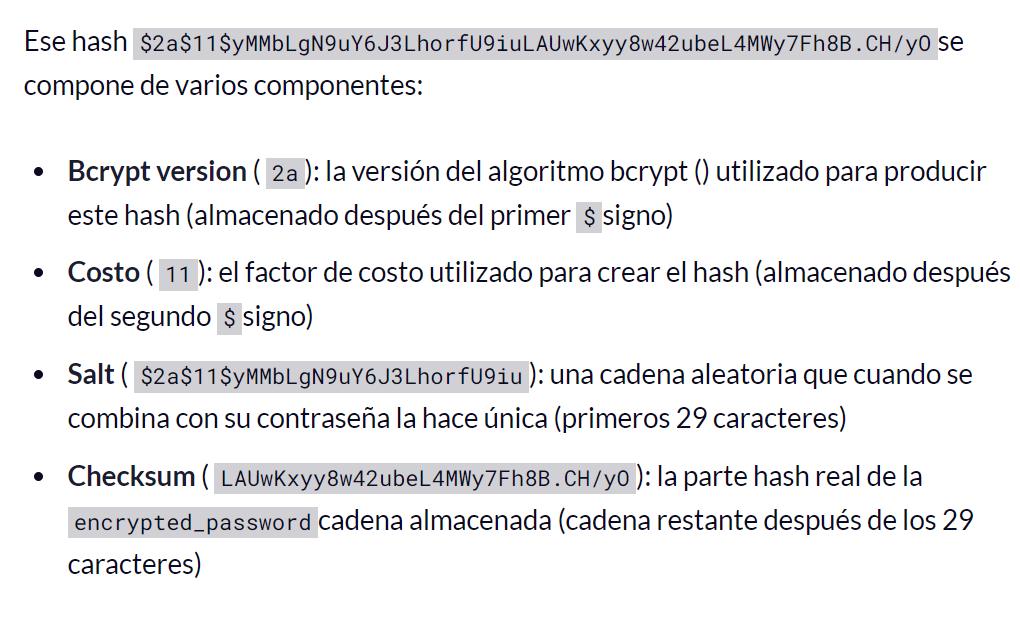


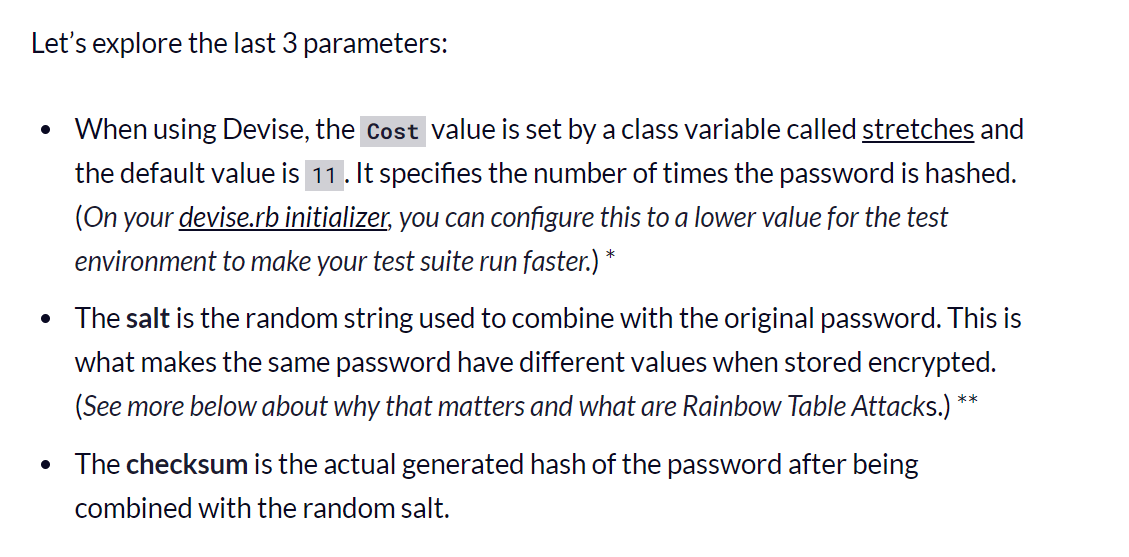
## Bycrypt:

bcrypt () es un algoritmo hash sofisticado y seguro diseñado por el proyecto OpenBSD para contraseñas hash. La gema bcrypt Ruby proporciona un contenedor simple para manejar contraseñas de forma segura.

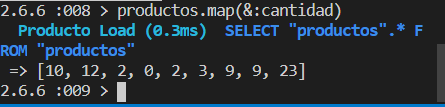
Ejemplo:





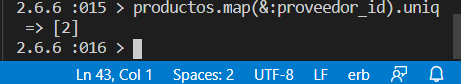


Map: 

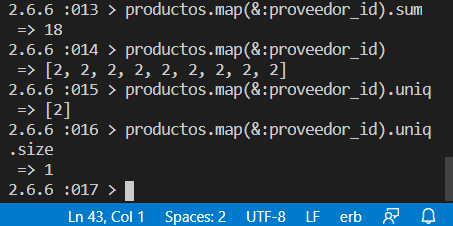


## Uniq

Devuelve sólo valores únicos, no repetidos de un array, rango o diccionario



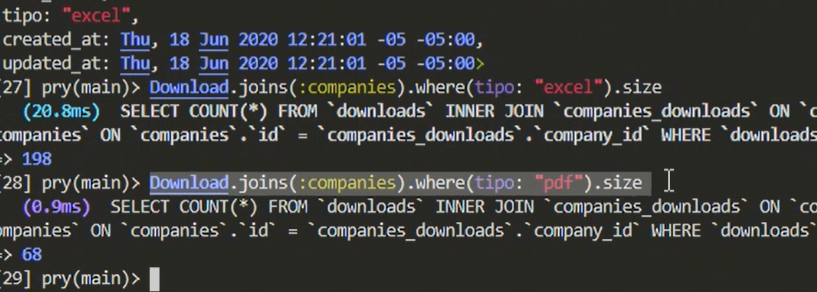
## Sum



## Include & Joins



## Joins + where



## Limit/First/Last

cuando pongo first, o last se trasnformna en array

limit mantiene como relación y puedo acceder a unos objetos



Bibliografías:

https://www.tutorialspoint.com/ruby/ruby\_if\_else.htm