# 

TEMAS:

* Conexión DB
* Comando Rake
* Configurar conexión
* Modelos
* Migraciones

OBJETIVO:

* Conectar Rails con postgres
* Interactuar y manipular modelos (Tablas SQL)

MARCO TEÓRICO

Introducción

Ruby on Rails usa Sqlite3 como gestor de datos por default. Sqlite3 a diferencia de los sistemas de gestión de base de datos cliente-servidor no es un proceso independiente con el que el programa principal se comunica, En lugar de esto se enlaza con el programa pasando a ser parte integral del mismo. El programa utiliza las funciones la funcionalidad de SQLite a través de llamadas simples a subrutinas y funciones. El conjunto de la base de datos (definiciones, tablas,índices y los propios datos), son guardados como un sólo fichero estándar en la máquina donde ejecutamos el programa. Este gestor puede no ser la suficientemente escalable para una aplicación, o puede carecer ciertos beneficios que nos aportan otros gestores de más robustos como MySql o PostgreSql.

Instalar / Agregar gema PG

En este caso las instrucciones pueden variar según sea su caso. un caso puede ser que ya tengas el proyecto creado o el otro es que inicias la aplicación desde cero.

En el caso de que no tengas el proyecto creado, instala la gema desde consola:

gem install pg

Si ya tenía el proyecto de rails creado, tendrá que agregar la gema a tu gemfile:

gem ‘pg’

Y después instalarla:

bundle update

Configuración de conexión a base de datos

Cuando se crea una aplicación Ruby on Rails sin especificar el gestor de base de datos, por defecto la configuración se encontrara definida para SQLite.

Para configurar la base de datos con la que se comunicara Rails usaremos un archivo de configuración llamada database.yml, ubicado en:

RAILS\_ROOT/config/database.yml

**Note:** RAILS\_ROOT es la carpeta raíz de nuestro proyecto.

El archivo database.yml es usado por Rails para conectarse a la base de datos apropiada dependiendo el actual entorno de desarrollo. esto usa YAML, un estándar para la serialización de datos. Aquí se enlistan las base de datos para diferentes entornos de desarrollo; development, test, and production configurados previamente por el generador de Rails. Por defecto Rails espera que uses diferentes bases de datos para cada uno de los entornos de la aplicación, esto tiene mucha lógica ya que, por ejemplo. la base de datos para pruebas estará basi y se podrá reconstruir cada vez que se ejecuten las pruebas de la aplicación. Para cada base de datos, se debe asegurar que coincidan con el usuario y contraseña que se definieron en Postgres.

Una vez configurado, su database.yml debe contener algo como esto:

default: &default

adapter: postgresql  
 encoding: unicode  
   
 pool: 5  
 username: myapp  
 password: password1

development:

<<: \*default  
 database:

test:

<<: \*default  
 database: myapp\_test

production:

<<: \*default  
 database: myapp\_production

Luego puede ejecutar:

rake db: setup

Esto creará bases de datos de desarrollo, prueba y produccion, establecerá sus propietarios para el usuario especificado y creará tablas "schema\_migrations" en cada uno. Esta tabla se usa para registrar sus migraciones a esquemas y datos.

**Comando Rake**

En la tradición de Unix, la utilidad Make ha jugado un papel importante en la creación de programas ejecutables a partir del código fuente. Rake es Ruby make, un lenguaje de Make-like escrito en Ruby.

Antes de Rails 5, Ruby on Rails usaba Rake extensivamente, por lo que para aplicaciones Rails heredadas es importante saber cómo usarlo. Probablemente los dos comandos de Rake más comunes en un contexto de Rails son rake db:migrate (para actualizar la base de datos con un modelo de datos) y rake test (para ejecutar el conjunto de pruebas automatizado). En estos y otros usos de rake, es importante asegurarse de que el comando utilice la versión de Rake correspondiente al Gemfile de la aplicación Rails, que se lleva a cabo utilizando el comando bundle exec de Bundler. Por lo tanto, el comando de migración

rake db:migrate

puede ser escrito como:

bundle exec rake db:migrate

Comandos Rake más comunes para bases de datos.

rake db:create # Crea la Base de datos

rake db:drop # Elimina la base de datos

rake db:migrate # Ejecuta las migraciones pendientes

rake db:migrate:status # Muestra el estatus de las migraciones

rake db:rollback # Retrocede la última migración

rake db:schema:load # Load a schema.rb file into the database

rake db:seed # Carga datos semilla de db/seeds.rb

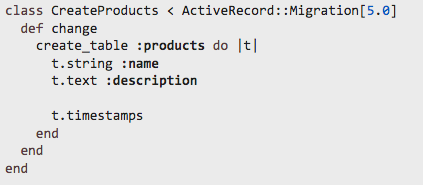
rake db:setup # Crea base de datos, carga el esquema y corre los datos semilla

Migraciones

Rails proporciona un lenguaje específico de dominio para administrar un esquema de base de datos llamado migraciones. Las migraciones se almacenan en archivos que se ejecutan en cualquier base de datos que Active Record admite utilizando rake.

Puede pensar en cada migración como una nueva 'versión' de la base de datos. Un esquema comienza con nada en él, y cada migración lo modifica para agregar o eliminar tablas, columnas o entradas. Active Record sabe cómo actualizar su esquema a lo largo de esta línea de tiempo, llevándolo de cualquier punto del historial a la última versión. Active Record también actualizará su archivo db / schema.rb para que coincida con la estructura actualizada de su base de datos.

Aquí hay un ejemplo de una migración:



Esta migración agrega una tabla llamada products con una columna de string llamada name y una columna de texto llamada description. También se agregará implícitamente una columna de clave principal llamada id, ya que es la clave primaria predeterminada para todos los modelos de Active Record. La macro timestamps agrega dos columnas, created\_at y updated\_at. Estas columnas especiales son administradas automáticamente por Active Record si existen.

Tenga en cuenta que definimos el cambio que queremos que suceda en el tiempo. Antes de ejecutar esta migración, no habrá una tabla. Después, la tabla existirá. Active Record también sabe cómo revertir esta migración: si retrotraemos esta migración, se eliminará la tabla.

En las bases de datos que admiten transacciones con declaraciones que cambian el esquema, las migraciones se envuelven en una transacción. Si la base de datos no es compatible con esto, cuando falla una migración, las partes que tuvieron éxito no se revertirán. Tendrá que deshacer los cambios que se hicieron a mano.

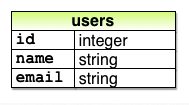
Tenga en cuenta que el código anterior es independiente de la base de datos: se ejecutará en MySQL, PostgreSQL, Oracle y otros.

Modelos

El modelo está formado por una clase heredada de ApplicationRecord, cada clase es un modelo, y cada modelo representa una tabla en la base de datos. por lo tanto el Modelo es el encargado de trabajar la lógica de la base de datos. Podríamos tener un modelo User y una tabla users, desde la clase User trabajaremos todo lo relacionado a la tabla users.

**Crear Modelo User**

Hay tantas opciones para un modelo de datos de usuario como diferentes formas de registro en la web; por simplicidad, vamos a ir con un enfoque claramente minimalista. Los usuarios de nuestra aplicación tendrán un identificador único llamado id (de tipo entero), un nombre visible públicamente (de tipo string) y una dirección de correo electrónico (también de tipo string) que se duplicará como un nombre de usuario único.



Implementaremos el modelo de datos de los usuarios, junto con una interfaz web para ese modelo. La combinación constituirá un recurso de los usuarios, que nos permitirá pensar en los usuarios como objetos que se pueden crear, leer, actualizar y eliminar a través de la web a través del protocolo HTTP. Nuestro recurso de Usuarios será creado por un programa de generador scaffold , que viene con cada proyecto de Rails.

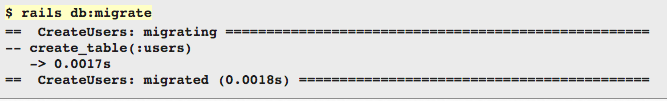
El argumento del comando scaffold es la versión singular del nombre del recurso (en este caso, User), junto con los parámetros opcionales para los atributos del modelo de datos:

**$ rails generate scaffold User name:string email:string  
 invoke active\_record  
 create db/migrate/20160515001017\_create\_users.rb  
 create app/models/user.rb  
 invoke test\_unit  
 create test/models/user\_test.rb  
 create test/fixtures/users.yml  
 invoke resource\_route  
 route resources :users  
 invoke scaffold\_controller  
 create app/controllers/users\_controller.rb  
 invoke erb  
 create app/views/users  
 create app/views/users/index.html.erb  
 create app/views/users/edit.html.erb  
 create app/views/users/show.html.erb  
 create app/views/users/new.html.erb  
 create app/views/users/\_form.html.erb  
 invoke test\_unit  
 create test/controllers/users\_controller\_test.rb  
 invoke helper  
 create app/helpers/users\_helper.rb  
 invoke test\_unit  
 invoke jbuilder  
 create app/views/users/index.json.jbuilder  
 create app/views/users/show.json.jbuilder  
 invoke assets  
 invoke coffee  
 create app/assets/javascripts/users.coffee  
 invoke scss  
 create app/assets/stylesheets/users.scss  
 invoke scss  
 create app/assets/stylesheets/scaffolds.scss**

Al incluir name:string y email:string, hemos dispuesto que el modelo de User tenga el formulario que contiene dos campos (Tenga en cuenta que no es necesario incluir un parámetro para el ID, Rails lo crea automáticamente para utilizarlo como clave principal en la base de datos).



Para continuar con la aplicación de , primero tenemos que migrar la base de datos usando rails db:migrate



Herramientas

PostgreSQL, Ruby, Rails

Ejercicios

Configurar su proyecto para trabajar con Postgres, realizar el primer modelo y migración de la base de datos.

