



Universidad Luterana Salvadoreña

Facultad de Ciencias del Hombre y la Naturaleza

Licenciatura en Ciencias de la Computación.

Programación III

Sábado de 9:40 a.m. a 12:10 p.m.

Ciclo 02-2021

Catedrático:

Lic. Jorge Alberto Coto Zelaya

Nombre del Proyecto:

Investigación Laravel

Integrantes:

Aguilar Sánchez, Henry David

AS01134801

San Salvador, Domingo 9 de Septiembre de 2021.

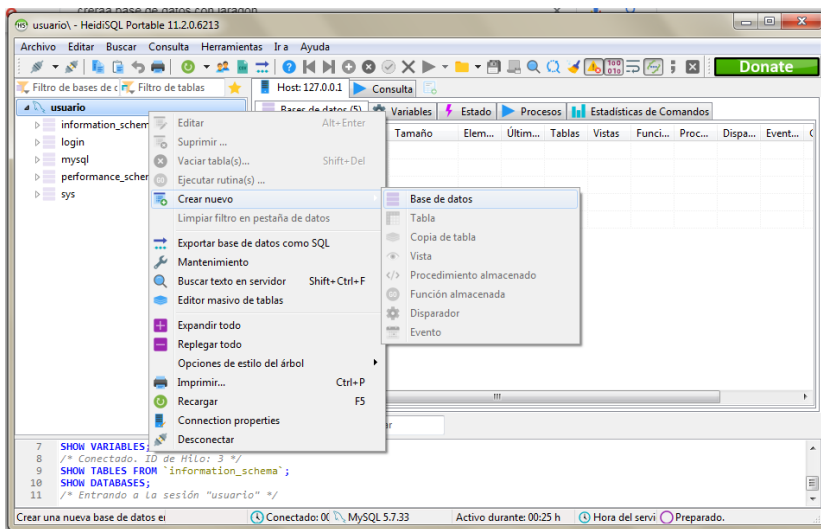
INDICE

¿Como crear una base de datos el laravel?	3
¿Que son las migraciones y como crear tablas utilizando migraciones?	5
¿Que son los Seeder en Laravel y como se genera de igual manera como implementarlos? (Crear y generar 10 seeder en el proyecto APP_usuarios, tabla users y profesión)	8
Investigar sobre el constructor de consultas SQL	11
Investigar sobre Eloquet ORM	14
Referencias bibliográficas	17

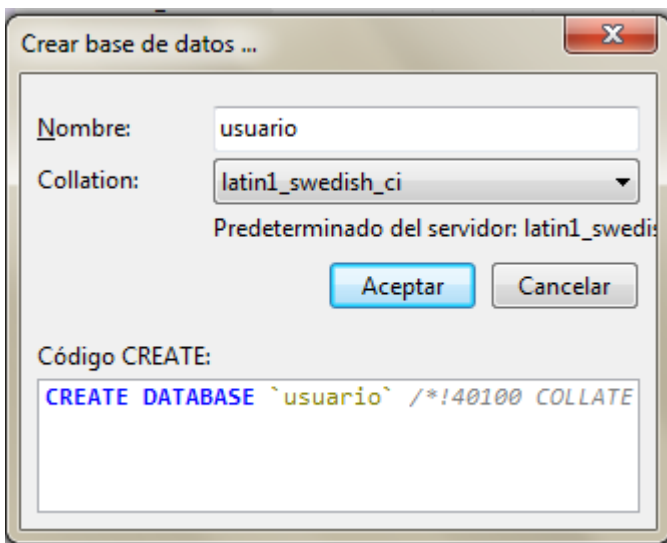
¿Como crear una base de datos el laravel?

Para trabajar con base de datos en Laravel existe múltiples opciones, este caso trabajaremos con la herramienta Laragon y su gestor de BD para crear la BD que posteriormente conectaremos con nuestro proyecto de Laravel.

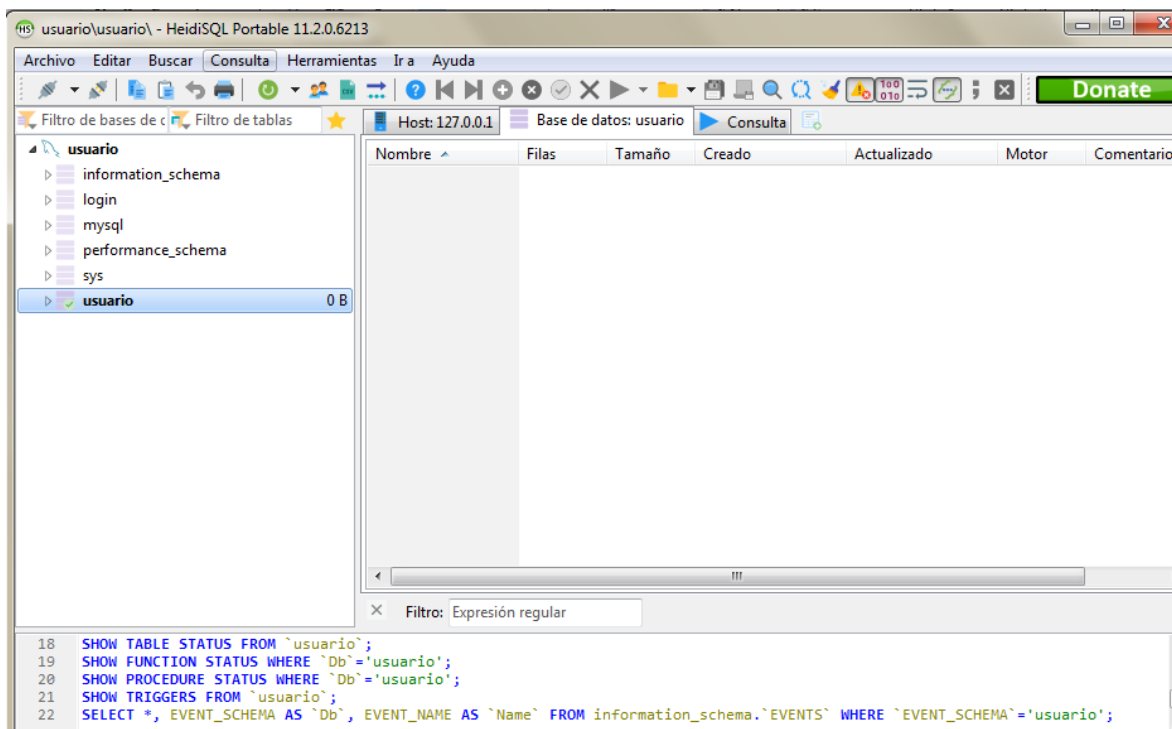
Paso1: Abrimos la herramienta y procedemos a dar click derecho sobre la sesión, nos dirigimos al apartado de “Crear nuevo -> Base de datos”



Paso 2: Colocamos el nombre de la base de datos y la Collation en el cuadro que se desplegara.



Posteriormente se creara nuestra Base de datos.



¿Que son las migraciones y como crear tablas utilizando migraciones?

El tema de migraciones en Laravel tiene que ver con el diseño de tablas de nuestra base de datos, y aunque podemos directamente crear nuestra tabla en MySQL crear el controlador y el modelo, lo vamos hacer usando migraciones, cual es la diferencia? Pues la diferencia es que usando migraciones diseñas las tablas como si de modelos se tratarán y luego las puedes generar a MySQL con un sólo comando, además puedes llevar la cronología de la creación de tus tablas y si algo salió mal, por ejemplo te olvidaste de crear un campo, fácilmente haces un rollback y deshaces los cambios.

La tabla que vamos a crear para el ejemplo se va llamar usuarios y contendrá los campos: nombre, email, profesión, password.

```
php artisan make:migration create_usuarios_table
```

se creó un nuevo archivo dentro de la carpeta database->migrations

```
C:\laragon\www\usuarios (main)
λ php artisan make:migration create_usuarios_table
Created Migration: 2021_09_04_180956_create_usuarios_table
```

Es una clase que hereda de la clase Migration, en esta clase definimos los campos que va contener la tabla Libros, en esta clase hay dos métodos down que se llama cuando ejecutamos un rollback y

```
1  <?php
2
3  use Illuminate\Support\Facades\Schema;
4  use Illuminate\Database\Schema\Blueprint;
5  use Illuminate\Database\Migrations\Migration;
6
7  class CreateUsuariosTable extends Migration
8  {
9      /**
10       * Run the migrations.
11       *
12       * @return void
13       */
14     public function up()
15     {
16         Schema::create('usuarios', function (Blueprint $table) {
17             $table->increments('id');
18             $table->timestamps();
19         });
20     }
21
22     /**
23      * Reverse the migrations.
24      *
25      * @return void
26      */
27     public function down()
28     {
29         Schema::dropIfExists('usuarios');
30     }
31 }
```

up que es donde crearemos los campos para nuestra tabla, el archivo debe quedar como se muestra a continuación:

Procedemos a ingresar los campos que contendrá nuestra tabla

```
class CreateUsersTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('users', function (Blueprint $table) {
            $table->increments('id');
            $table->string('name');
            $table->string('email')->unique();

            $table->unsignedInteger('profession_id')->nullable();
            $table->foreign('profession_id')->references('id')->on('profession');

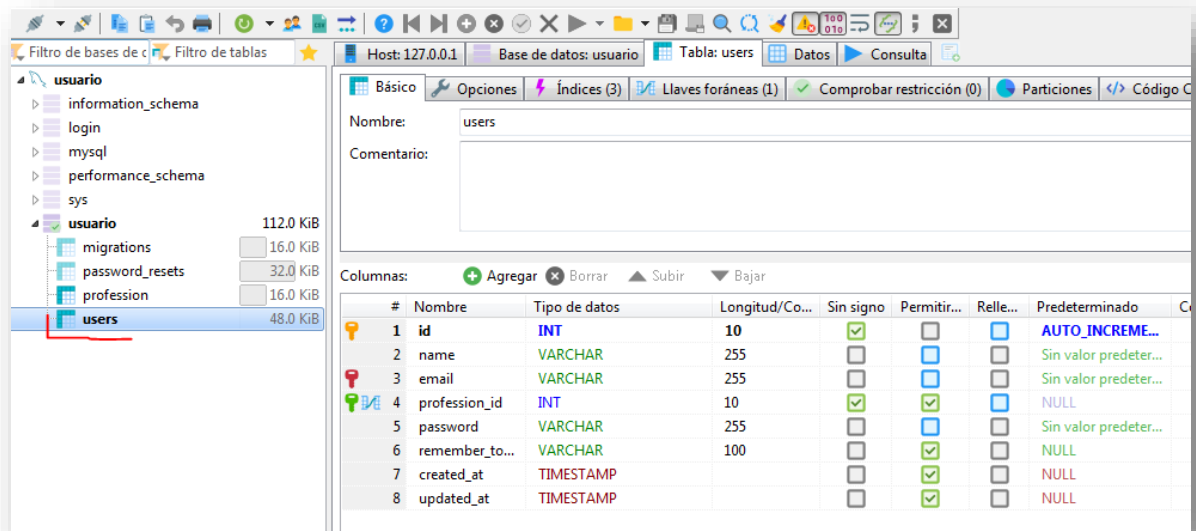
            $table->string('password');
            $table->rememberToken();
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::dropIfExists('users');
    }
}
```

Ejecutamos el siguiente comando para hacer la migración:

```
C:\laragon\www\usuarios (main) X Filtro: Expresión regul
λ php artisan migrate
Migrating: 2021_09_05_233203_create_users_table
Migrated: 2021_09_05_233203_create_users_table
```

Como se puede observar se ha creado nuestra tabla través de migraciones con laravel



The screenshot shows a database management interface. On the left, a tree view displays the database structure, with the 'usuario' database selected and the 'users' table highlighted. The main panel shows the 'users' table structure with the following columns:

#	Nombre	Tipo de datos	Longitud/Co...	Sin signo	Permitir...	Relle...	Predeterminado
1	id	INT	10	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	AUTO_INCREME...
2	name	VARCHAR	255	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Sin valor predeter...
3	email	VARCHAR	255	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Sin valor predeter...
4	profession_id	INT	10	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	NULL
5	password	VARCHAR	255	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Sin valor predeter...
6	remember_to...	VARCHAR	100	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	NULL
7	created_at	TIMESTAMP		<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	NULL
8	updated_at	TIMESTAMP		<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	NULL

¿Que son los Seeder en Laravel y como se genera de igual manera como implementarlos? (Crear y generar 10 seeder en el proyecto APP_usuarios, tabla users y profesión)

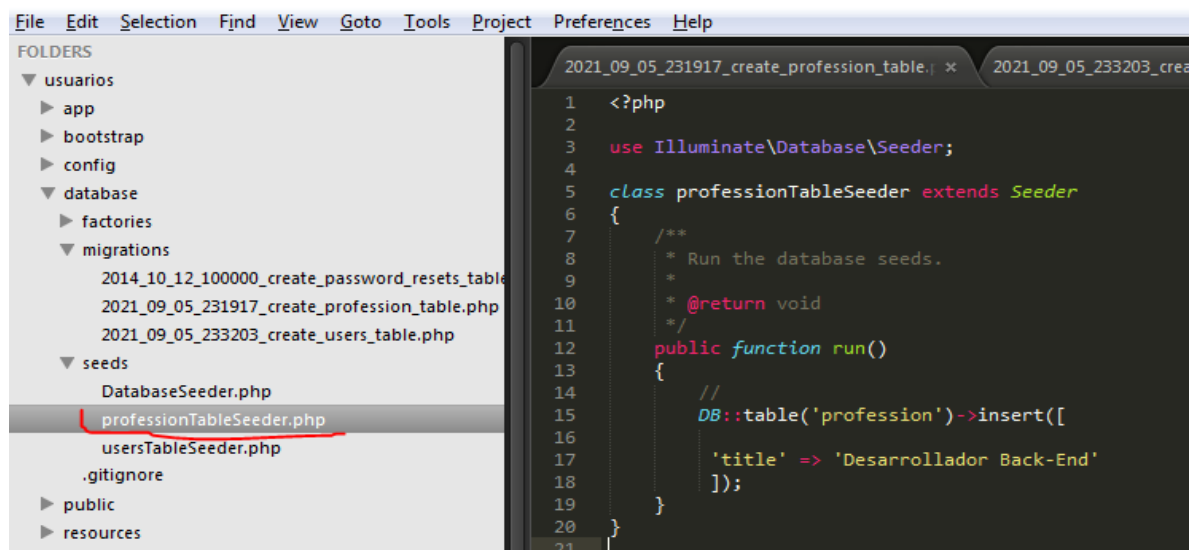
Laravel incluye un método simple para sembrar su base de datos con datos de prueba utilizando clases de semillas. Todas las clases de semillas se almacenan en el `database/seeds` directorio. Las clases de semillas pueden tener el nombre que desee, pero probablemente deberían seguir alguna convención sensata, como `UsersTableSeeder`, etc. De forma predeterminada, `DatabaseSeeder` define una clase para usted. Desde esta clase, puede utilizar el `call` método para ejecutar otras clases de inicialización, lo que le permite controlar el orden de inicialización.

Para generar una sembradora, ejecute el `make:seeder` comando `Artisan`. Todas las sembradoras generadas por el marco se colocarán en el `database/seeds` directorio:

```
php artisan make:seeder UsersTableSeeder
```

Una clase sembradora sólo contiene un método por defecto: `run`. Este método se llama cuando se ejecuta el `db:seed` comando `Artisan`. Dentro del método `run`, puede insertar datos en su base de datos como desee. Puede usar el generador de consultas para insertar datos manualmente o puede usar las fábricas de modelos Eloquent.

Al ejecutar el comando se nos creara un nuevo archivo en donde colocaremos los valores a subir con el seeders

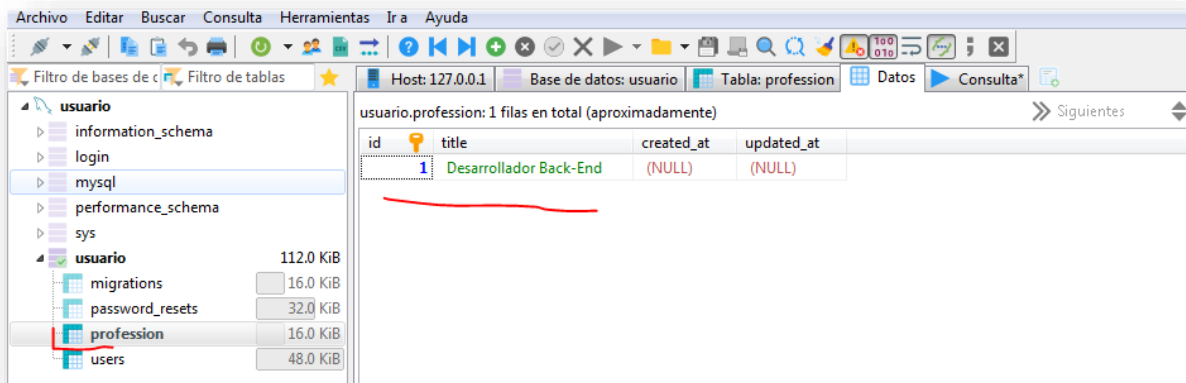


```
1 <?php
2
3 use Illuminate\Database\Seeder;
4
5 class professionTableSeeder extends Seeder
6 {
7     /**
8      * Run the database seeds.
9      *
10     * @return void
11     */
12     public function run()
13     {
14         //
15         DB::table('profession')->insert([
16             'title' => 'Desarrollador Back-End'
17         ]);
18     }
19 }
20
21
```

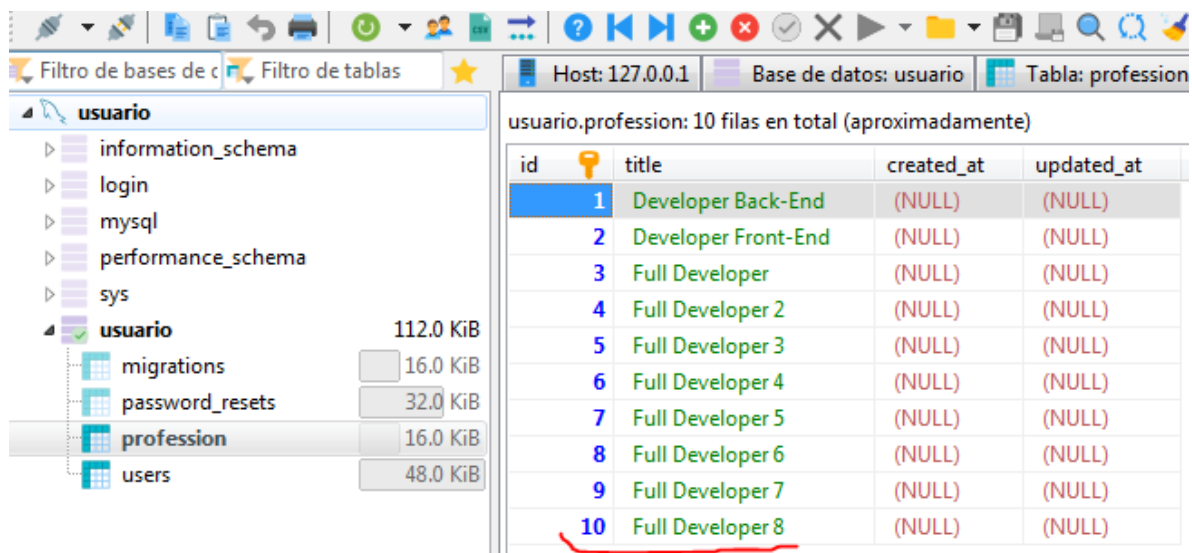

Ahora ejecutamos el siguiente código para sembrar nuestros datos en la base de datos

```
C:\laragon\www\usuarios (main)
λ php artisan db:seed
Seeding: professionTableSeeder
```

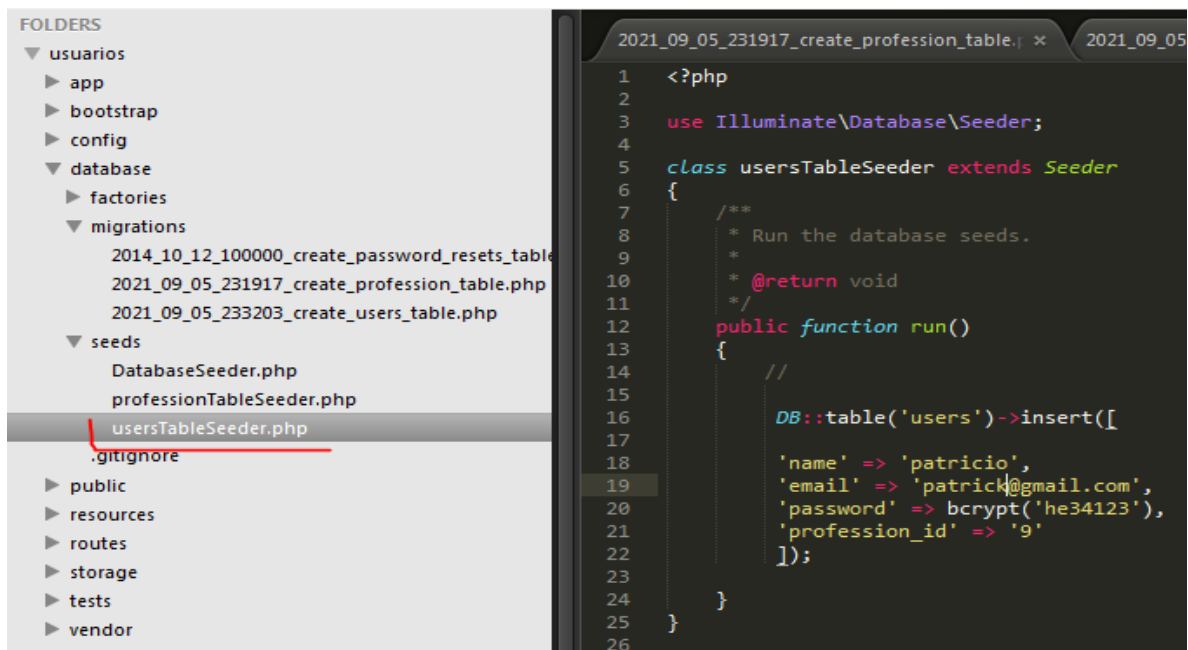
Posteriormente consultamos nuestra base de datos para corroborar que si se subió la información:



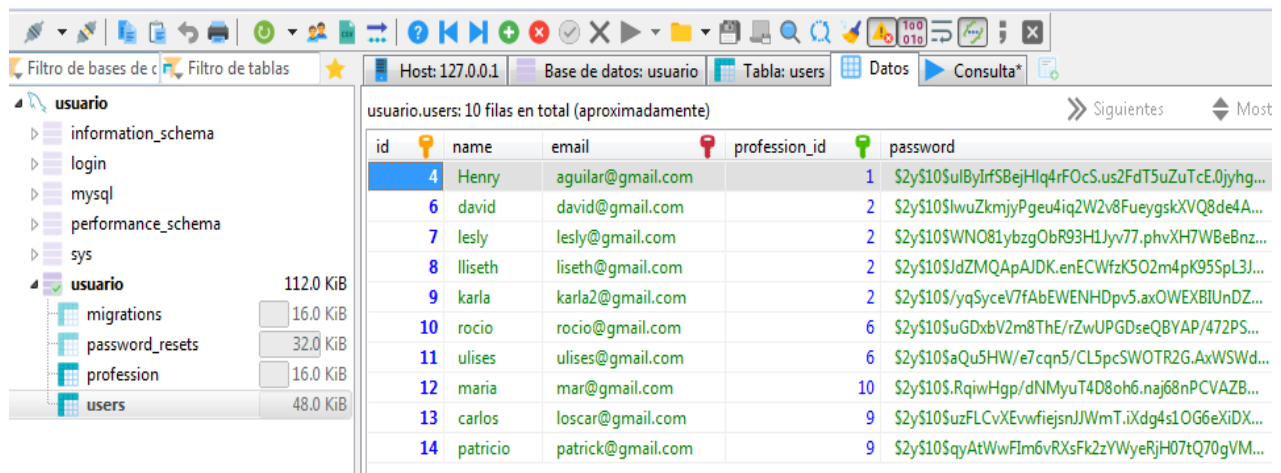
Posteriormente aremos 9 seeders más para completar 10 registros



También crearemos 10 registros para la tabla users



```
1 <?php
2
3 use Illuminate\Database\Seeder;
4
5 class usersTableSeeder extends Seeder
6 {
7     /**
8      * Run the database seeds.
9      *
10     * @return void
11     */
12     public function run()
13     {
14         //
15
16         DB::table('users')->insert([
17
18             'name' => 'patricio',
19             'email' => 'patrick@gmail.com',
20             'password' => bcrypt('he34123'),
21             'profession_id' => '9'
22         ]);
23     }
24 }
25
26
```



usuario.users: 10 filas en total (aproximadamente)

id	name	email	profession_id	password
4	Henry	aguiar@gmail.com	1	\$2y\$10\$uLbYlrfSBejHlq4rFOcS.us2FdT5uZuTcE.0jyhg...
6	david	david@gmail.com	2	\$2y\$10\$slwuZkmjyPgeu4iq2W2v8FueygskXVQ8de4A...
7	lesly	lesly@gmail.com	2	\$2y\$10\$WNO81ybzgObR93H1Jyv77.phvXH7WBeBnz...
8	liseth	liseth@gmail.com	2	\$2y\$10\$jdZMQApAJDK.enECWfzK5O2m4pK95SpL3J...
9	karla	karla2@gmail.com	2	\$2y\$10\$/yqSyceV7fAbEWENHDpv5.axOWEXBIUnDZ...
10	rocio	rocio@gmail.com	6	\$2y\$10\$uGDxbV2m8ThE/rZwUPGDseQBYAP/472PS...
11	ulises	ulises@gmail.com	6	\$2y\$10\$aQu5HW/e7cqns/CL5pcSWOTR2G.AxWSWd...
12	maria	mar@gmail.com	10	\$2y\$10\$.RqiWgpg/dNMMyuT4D8oh6.naj68nPCVAZB...
13	carlos	loscar@gmail.com	9	\$2y\$10\$SuzFLCvXEvvfiejsnJJWmT.iXdg4s1OG6eXiDX...
14	patricio	patrick@gmail.com	9	\$2y\$10\$yqAtWwFlm6vRXsFk2zYWyeRjH07tQ70gVM...

Investigar sobre el constructor de consultas SQL

Método insert

Con el método `DB::insert` podemos escribir consultas SQL de forma manual para insertar contenido dentro de nuestra base de datos. Por ejemplo, el código del seeder `ProfessionSeeder.php` que utiliza el método `DB::table`:

```
DB::table('professions')->insert([  
    'title' => 'Desarrollador back-end',  
]);
```

Puede ser re-escrito utilizando el método `DB::insert`, directamente con código SQL:

```
DB::insert('INSERT INTO professions (title) VALUES ("Desarrollador back-end")');
```

Aunque `DB::insert` nos da el mismo resultado que `DB::table`, cuando realizamos consultas de este tipo y recibimos datos ingresados por un usuario debemos tener mucho cuidado, ya que nos exponemos a ataques de inyección de SQL.

Inyección de SQL

Nuestro código es vulnerable a inyecciones de SQL cuando insertamos variables dentro de una solicitud de SQL. Por ejemplo, si tuvieras una consulta donde seleccionas una serie de artículos dependiendo de un autor:

```
$sql = "SELECT * FROM articles WHERE id_author = $id";
```

Esta consulta trae todos los artículos escritos por un determinado autor. Sin embargo, dentro de esta consulta estamos insertando contenido de forma directa al colocar la variable `$id`.

Supón que ingresamos a los artículos del autor desde una URL, pasando como argumento el id del autor (`?id=1` o `articulos/{id}`), en este caso retornaríamos todos los artículos escritos por el autor cuyo id sea igual a 1. Sin embargo, como nuestro código es vulnerable, un usuario malintencionado podría cambiar la URL por `?id=1 UNION SELECT password FROM users`. La consulta realmente se estaría realizando de esta forma:

```
SELECT * FROM articles WHERE id_author = 1 UNION SELECT password FROM users;
```

Esta consulta selecciona todos los artículos y luego selecciona todas las contraseñas almacenadas en la tabla `users`.

Al almacenar contraseñas en una base de datos asegúrate de siempre encriptarlas.

Parametros dinámicos

Para evitar ataques de inyección de SQL podemos utilizar parámetros dinámicos. Laravel utiliza internamente el componente PDO de PHP y debido a esto podemos colocar marcadores en nuestra consulta. Laravel nos permite pasar sus valores en un array como segundo argumento del método:

```
DB::insert('INSERT INTO professions (title) VALUES (?)', ['Desarrollador back-end']);
```

Otra forma de pasar los parametros es usando como marcador un parametro de sustitución con nombre y como segundo argumento pasamos un array asociativo de los respectivos parámetros con sus valores:

```
DB::insert('INSERT INTO professions (title) VALUES (:title)', ['title' => 'Desarrollador back-end']);
```

Al hacer esto estaremos protegidos de ataques de inyección de SQL puesto que los parámetros dinámicos serán escapados de forma automática y segura.

Método select

Utilizando el método DB::select podemos construir una consulta SELECT de SQL de forma manual:

```
DB::select('SELECT id FROM professions WHERE title = ?', ['Desarrollador back-end']);
```

Por otro lado, utilizando el constructor de consultas podemos realizar una consulta SQL de tipo SELECT, de la siguiente forma:

```
$professions = DB::table('professions')->select('id')->take(1)->get();
```

El resultado de esta consulta es un objeto de la clase Illuminate\Support\Collection que encapsula el array de datos y esto nos trae algunas ventajas extras: una interfaz orientada a objetos con la cual trabajar y muchas funciones extras. Por ejemplo, podemos utilizar el método first para obtener el primer resultado de la consulta (en el caso de este ejemplo, la primera profesión):

```
$professions->first(); // en vez de $professions[0]
```

Consultas con condicionales (WHERE)

El constructor de consultas también nos permite realizar consultas condicionales utilizando where:

```
$profession = DB::table('professions')->select('id')->where('title', '=', 'Desarrollador back-end')->first();
```

El operador = dentro del método where es opcional. Pasando el nombre de la columna junto con el valor, Laravel asumirá que quieres usar el operador de comparación de igualdad (=):

```
where('title', 'Desarrollador back-end')
```

El método where también acepta un array asociativo, donde indicamos el nombre de la columna y el valor que esperamos encontrar:

```
where(['title' => 'Desarrollador back-end'])
```

Métodos dinámicos

También podemos utilizar métodos dinámicos:

```
$profession = DB::table('professions')->whereTitle('Desarrollador back-end')->first();
```

En este caso whereTitle es lo equivalente a escribir where('title', '=', 'Desarrollador back-end').

Omitir el método select de DB::table

Omitiendo el método select al utilizar DB::table podemos retornar todas las columnas:

```
$profession = DB::table('professions')->where('title', '=', 'Desarrollador back-end')->first();
```

Investigar sobre Eloquent ORM

Para comenzar, cree un modelo Eloquent. Los modelos suelen vivir en el appdirectorio, pero puede colocarlos en cualquier lugar que pueda cargarse automáticamente de acuerdo con su composer.jsonarchivo. Se amplían todos los modelos Eloquent Illuminate\Database\Eloquent\Model.

Definiendo un modelo elocuente

```
class User extends Model {}
```

También puede generar modelos Eloquent usando el make:modelcomando:

```
php artisan make:model User
```

Tenga en cuenta que no le dijimos a Eloquent qué tabla usar para nuestro Usermodelo. El "caso de serpiente", el nombre en plural de la clase se utilizará como nombre de la tabla a menos que se especifique explícitamente otro nombre. Entonces, en este caso, Eloquent asumirá que el Usermodelo almacena registros en la userstabla. Puede especificar una tabla personalizada definiendo una tablepropiedad en su modelo:

```
class User extends Model {  
  
    protected $table = 'my_users';  
  
}
```

Consultas básicas

Devuelve una colección de objetos de todos los usuarios.

```
// retorna object(Illuminate\Database\Eloquent\Collection)
$users = User::all();
$users = User::get();
```

Si buscamos un solo usuario, nos devuelve el modelo User.

```
// retorna object(App\User)
$user = User::find(10);
$user = User::all()->first();
```

Para crear un registro.

```
$user = new User;
$user->username = "user";
$user->save();
// save retorna un boolean, podrían usarlo así:
if( $user->save() ){
    var_dump($user->id);
}
```

Para actualizar un registro.

```
$user = User::find(10);
$user->username = "new user";
$user->save();
```

Para eliminar un registro.

```
$user = User::find(10);
$user->delete();
```

Usando where.

```
$user = User::where("estado", "=", 1)->find(10);
```

Para que nos devuelva una instancia de `Illuminate\Pagination\LengthAwarePaginator` y pague cada 10 registros.

```
$users = User::where("estado", "=", 1)->paginate(10);
// En la vista
foreach ($users as $key => $user) {
    // $user es una Instancia de la clase User
}
```

Seleccionamos solo algunas columnas.

```
$users = User::where("estado","=",1)->select("id","username")->paginate(10);
```

Si queremos usar alguna función de mysql, podemos usar `DB::raw()`.

```
$users = User::where("estado","=",1)
->select(DB::raw("id,username, DATE_FORMAT(created_at,'%d/%m/%Y %h:%i %p') AS
fecha"))
->paginate(10);
```


Referencias bibliográficas

Septiembre 2021, ECODEUP-PROGRAMACION WEB FULL STACK,
<https://www.ecodeup.com/como-instalar-y-configurar-laravel-5-5/>

Septiembre 2021, ECODEUP-PROGRAMACION WEB FULL STACK,
<https://www.ecodeup.com/como-crear-un-crud-en-laravel-5-5-desde-cero/>

Septiembre 2021, Laravel 5.5: 'Method foreing does not exist',
<https://es.stackoverflow.com/questions/126781/laravel-5-5-method-foreing-does-not-exist-error-con-llave-for%C3%A1nea>

Septiembre 2021, Database: Seeding,
<https://laravel.com/docs/5.5/seeding#introduction>

Septiembre 2021, Laravel Eloquent ORM y Query Builder,
https://desarrollowebtutorial.com/laravel-eloquent-orm-query-builder-consultas-sql/#Laravel_Utilizando_Eloquent_ORM

Septiembre 2021, Eloquent ORM,
<https://laravel.com/docs/5.0/eloquent>

Septiembre 2021, Constructor de consultas SQL de Laravel,
<https://styde.net/constructor-de-consultas-sql-de-laravel/>