

TALLER CRUD Avanzada

Tatiana Lizbeth Cabrera Vargas

PHP Avanzado

Valery Juliana Morantes

Ficha 2184573

Bogotá

2022

ADSI

INTRODUCCION

El presente trabajo lo llevamos trabajando y siguiendo la metodología enseñada por la instructora Tatiana cabrera, aprendimos muchos términos, y procedimientos con unas buenas practicas implementando las tablas relacionadas, los required que son tan fundamentales en la programación posterior a eso mostrando el paso a paso para llevar a cabo con el trabajo final que es esta crud avanzada

TALLER CRUD Avanzada

Iniciamos implementado un método que nos permite eliminar las fotos desde la carpeta storage en dado caso que nuestra tabla tenga una foto para agregar

```
*/
public function destroy($id)
{
    // DB::table('productos')-> where('id','=', $id)->delete();
    $producto = producto::findOrFail($id);
    if(Storage::delete('public/'.$producto->Foto)){
        producto::destroy($id);
    }





    return redirect('producto')->with('msn','producto eliminada exitosamente');
```

localhost / 127.0.0.1 / crud | php | x Gestion de Productos x +

localhost/crud/public/producto

Administración de Mercancia Productos Marcas Usuarios Iniciar Sesión Registrar

Bienvenido

ID	Nombre del producto	costo	precio	Cantidad	Marca	Foto
6	gferytyoooooo	12	12	1	airpods	
7	PRUEBA45	12	12	1	airpods	
9	ghyh	12	1	1	tablets	
11	gferger	12	12	1	airpods	

Ahora nos dirigimos en el controlador y realizamos al final de cada return

```
public function store(Request $request)
{
    $campos=[
        'nombreProducto'=>'required|string|max:50|',
        'costo'=>'required|numeric|max:150|',
        'precio'=>'required|numeric|max:150|',
        'cantidad'=>'required|numeric|max:150|',
        'marca'=>'required|string|max:150|',
        'foto'=>'required|string|mimes:jpg,jpeg,bmp,png|max:500'
    ];
    $this->validate($request,$campos);
    $datoP=request()->except('_token','Enviar');
    if($request->hasFile('foto')){
        $datoP ['foto'] =$request->file('foto')->store('uploads','public');
    }
    Producto::insert($datoP);
    return redirect('producto/');
}
```

Realizamos los required en el controlador de producto en las funciones store, update y destroy

Posteriormente realizamos las validaciones, nos dirigimos al controlador en este caso ProductoController nos dirigimos a los métodos mencionados. creamos una variable campos abrimos llaves y llamamos los campos de nuestras tablas de la siguiente manera

```
public function update(Request $request, $id)
{
    $campos=[
        'nombreProducto'=>'required|string|max:50|',
        'costo'=>'required|numeric|max:150|',
        'precio'=>'required|numeric|max:150|',
        'cantidad'=>'required|numeric|max:150|',
        //'marca'=>'required|string|max:150|',
        'foto'=>'required|string|mimes:jpg,jpeg,bmp,png|max:500'
    ];
    $this->validate($request,$campos);
    $datosProducto=request()->except('_token','_method');
    if($request->hasFile('Foto')){
        $datosProducto['Foto']=$request->file('Foto')->Store('uploads','public');
    }
    producto::where('id','=',$id)->update($datosProducto);

    //return view('Marca.edit',compact('marca'));

    return redirect('producto/')->with('msn','producto actualizado correctamente');
}
```

```

public function destroy($id)
{
    // DB::table('productos')-> where('id','=', $id)->delete();
    $producto = producto::findOrFail($id);
    if(Storage::delete('public/'.$producto->Foto)){
        producto::destroy($id);
    }

    return redirect('producto')->with('msn','producto eliminada exitosamente');
}
}

```

Cerramos con esta variable para traer la información en este caso las validaciones de CAMPOS

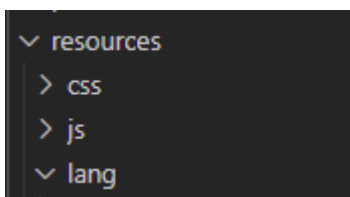
`$this->validate($request,capos$)`

```

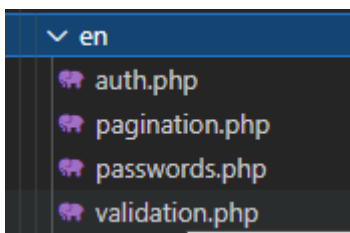
];
$this->validate($request,$campos);

```

Vamos a la carpeta en recursos -> Lang para cambiar el idioma



Validation



Traducimos

```
return [
    'accepted'          => ':attribute debe ser aceptado.',
    'accepted_if'       => ':attribute debe ser aceptado cuando :other sea :value.',
    'active_url'        => ':attribute no es una URL válida.',
    'after'             => ':attribute debe ser una fecha posterior a :date.',
    'after_or_equal'    => ':attribute debe ser una fecha posterior o igual a :date.',
    'alpha'            => ':attribute sólo debe contener letras.',
    'alpha_dash'       => ':attribute sólo debe contener letras, números, guiones y guiones bajos.',
    'alpha_num'        => ':attribute sólo debe contener letras y números.',
    'array'            => ':attribute debe ser un conjunto.',
    'before'           => ':attribute debe ser una fecha anterior a :date.',
    'before_or_equal'   => ':attribute debe ser una fecha anterior o igual a :date.',
    'between'          => [
        'array'    => ':attribute tiene que tener entre :min - :max elementos.',
        'file'     => ':attribute debe pesar entre :min - :max kilobytes.',
        'numeric' => ':attribute tiene que estar entre :min - :max.',
        'string'  => ':attribute tiene que tener entre :min - :max caracteres.',
    ],
    'boolean'          => 'El campo :attribute debe tener un valor verdadero o falso.',
    'confirmed'        => 'La confirmación de :attribute no coincide.',
    'current_password' => 'La contraseña es incorrecta.',
    'date'             => ':attribute no es una fecha válida.',
    'date_equals'      => ':attribute debe ser una fecha igual a :date.',
    'date_format'      => ':attribute no corresponde al formato :format.',
    'declined'         => ':attribute debe ser rechazado.',
    'declined_if'      => ':attribute debe ser rechazado cuando :other sea :value.',
    'different'        => ':attribute y :other deben ser diferentes.',
    'digits'           => ':attribute debe tener :digits dígitos.',
    'digits_between'   => ':attribute debe tener entre :min y :max dígitos.',
    'dimensions'       => 'Las dimensiones de la imagen :attribute no son válidas.',
```

Y así se vería al momento de realizar alguna acción así en cada una de las tablas

AGREGAR:

public/producto/create

A

Registro Producto

!

Corrige los siguientes errores:

- El campo nombre producto es obligatorio.
- El campo costo es obligatorio.
- El campo precio es obligatorio.
- El campo cantidad es obligatorio.
- El campo marca es obligatorio.
- El campo foto es obligatorio.

X

Nombre del producto:

Costo del producto: \$

Precio del producto: \$

Cantidad:

Marca:

airpods

Foto:

Elegir archivo

No se ha selecci...do ningún archivo

Esta es la foto actual:

Editar Marca

Nombre Marca:

airpods

Descripcion:


electrodomesticos

Foto:

Elegir archivo


No se ha seleccionado ningún archivo

Esta es la foto actual:



Enviar

Volver



Ahora nos dirigimos al form.blade de la tabla padre que en este caso es marca y hacemos un if para que nos muestre los errores con la función errors esto hará que nos muestre las validaciones hechas en el controlador en el caso que los datos no sean los correctos o el campo quede vacío quedaría de la siguiente manera.

Antes de este paso cabe recalcar que es necesario ir a validaciones, y traducir lo que viene por defecto para una mejor comprensión

```
@if (count($errors)>0)
<div class="container alert alert-danger d-flex align-items-relative alert-dismissible fade show" role="alert">
  <svg xmlns="http://www.w3.org/2000/svg" width="32" height="32" fill="currentColor" class="bi bi-exclamation-triangle" viewBox="0 0 32 32">
    <path d="M16 8A8 8 0 1 1 0 8a8 8 0 0 1 16 0zM8 4a.995.995 0 0 0-.995 1.35 3.507a.552.552 0 0 0 1.1 1.1 1.1 1.1 0 0 0 1.1 1.1 3.507.552 0 0 0 1.1 1.1.995.995 0 0 0 1.35-.995Z">
    </path>
  </svg>
  <ul>
    <strong>Corrige los siguientes errores:</strong>
    @foreach ($errors->all() as $message)
      <li>{{ $message }}</li>
    @endforeach
  </ul>
  <button type="button" class="btn-close" data-bs-dismiss="alert" aria-label="Close"></button>
</div>
@endif

<div class="row mb-3">
  <label class="col-md-4 col-form-label text-md-end" for="nombreMarca">Nombre Marca: </label>
  <div class="col-md-6">
    <input class="form-control" type="text" name="nombreMarca" value="{{isset($marca->nombreMarca)?$marca->nombreMarca:''}}">
  </div>

  <label class="col-md-4 col-form-label text-md-end" for="descripcion">Descripcion: </label>
  <div class="col-md-6">
```


Y así quedaría en la vista:

- El campo foto es obligatorio.

Nombre del producto:

Costo del producto: \$

Precio del producto: \$

Cantidad:


Marca:

Foto:

Elegir archivo

No se ha selecci...do ningún archivo

Esta es la foto actual:

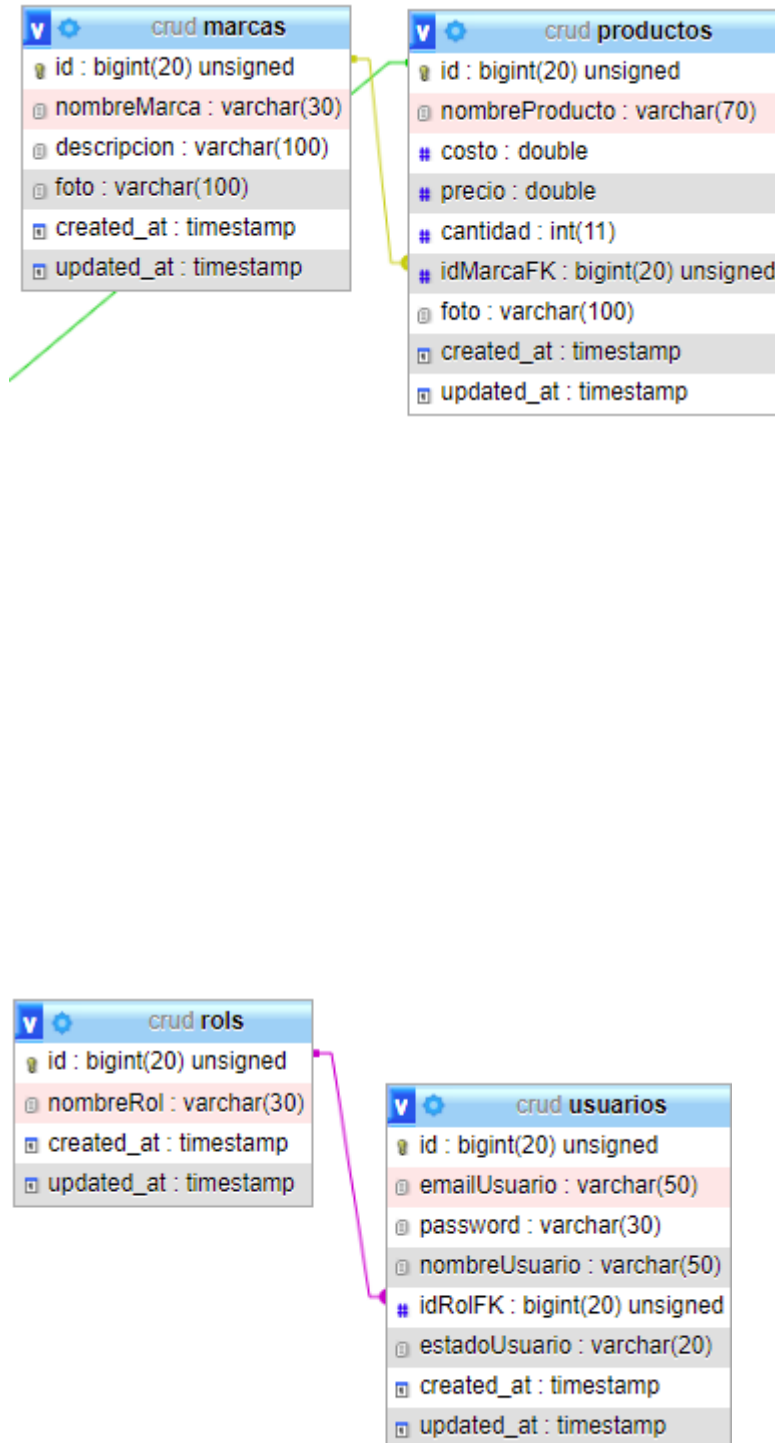
 Agregue una foto por favor

Enviar

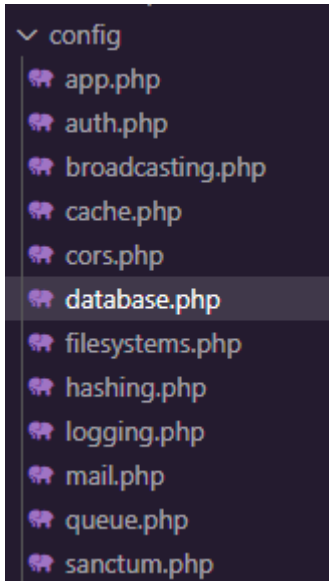
Volver

Y por último trabajamos las relaciones entre tablas

La tabla padre en este caso es marcas ya que tiene una llave primaria y la tabla hija seria productos dado con el mismo contexto la tabla padre seria rol ya que un rol lo tiene un usuario, la tabla hija es usuarios



Para incluir la integridad referencial en la base de datos mysql es necesario habilitar bases de datos tipo INNOBD yo lo realice de la siguiente manera me dirijo a la carpeta config y abro el que dice database.php



Luego buscamos mysql

```
'mysql' => [
    'driver' => 'mysql',
    'url' => env('DATABASE_URL'),
    'host' => env('DB_HOST', '127.0.0.1'),
    'port' => env('DB_PORT', '3306'),
    'database' => env('DB_DATABASE', 'forge'),
    'username' => env('DB_USERNAME', 'forge'),
    'password' => env('DB_PASSWORD', ''),
    'unix_socket' => env('DB_SOCKET', ''),
    'charset' => 'utf8mb4',
    'collation' => 'utf8mb4_unicode_ci',
    'prefix' => '',
    'prefix_indexes' => true,
    'strict' => true,
    'engine' => null,
    'options' => extension_loaded('pdo_mysql') ? array_filter([
        PDO::MYSQL_ATTR_SSL_CA => env('MYSQL_ATTR_SSL_CA'),
    ]) : [],
],
```

Buscamos el que dice engine y colocamos InnoDB

```
'mysql' => [
    'driver' => 'mysql',
    'url' => env('DATABASE_URL'),
    'host' => env('DB_HOST', '127.0.0.1'),
    'port' => env('DB_PORT', '3306'),
    'database' => env('DB_DATABASE', 'forge'),
    'username' => env('DB_USERNAME', 'forge'),
    'password' => env('DB_PASSWORD', ''),
    'unix_socket' => env('DB_SOCKET', ''),
    'charset' => 'utf8mb4',
    'collation' => 'utf8mb4_unicode_ci',
    'prefix' => '',
    'prefix_indexes' => true,
    'strict' => true,
    'engine' => null,
    'options' => extension_loaded('pdo_mysql') ? array_filter([
        PDO::MYSQL_ATTR_SSL_CA => env('MYSQL_ATTR_SSL_CA'),
    ]) : [],
],
```

Luego nos dirigimos al controlador y creamos una variable para llamar el modelo y retornar los datos a la vista y se hace de la siguiente manera

```

* @return \Illuminate\Http\Response
*/
public function create()
{
    $marca['marca'] = marca::all();
    return view('producto.create', $marca);
}

/**
 * Store a newly created resource in storage.
 *
 * @param \Illuminate\Http\Request $request
 * @return \Illuminate\Http\Response
 */
public function store(Request $request)
{

```

se hace lo mismo en la función editar

```
public function edit($id)
{
    $marca['marca'] = Marca::all();
    $producto = producto::findOrFail($id);
    return view('producto.edit', compact('producto'), $marca);
}
```

Dando una pequeña retroalimentación para borrar en cascade se puede configurar desde xampp. Esto para que si se borra en la tabla padre, lo borre también en la tabla hija.

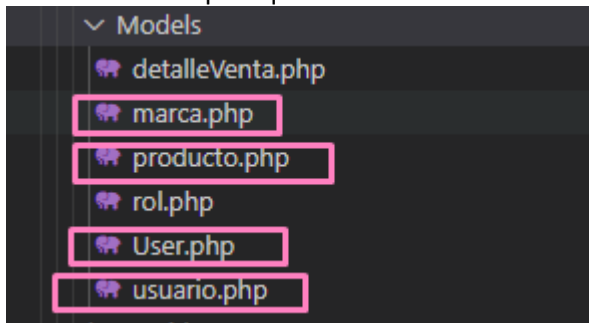
Estructura de tabla Vista de relaciones

Restricciones de clave foránea

Acciones	Propiedades de la restricción	Columna	Restricción de clave foránea (INNODB)	
			Base de datos	Tabla
Eliminar	productos_idmarcafk_foreig	idMarcaFK	crud	marcas
	ON DELETE CASCADE ON UPDATE CASCADE	+ Añadir columna		
	Nombre de la restricción		crud	
	ON DELETE RESTRICT ON UPDATE RESTRICT	+ Añadir columna		

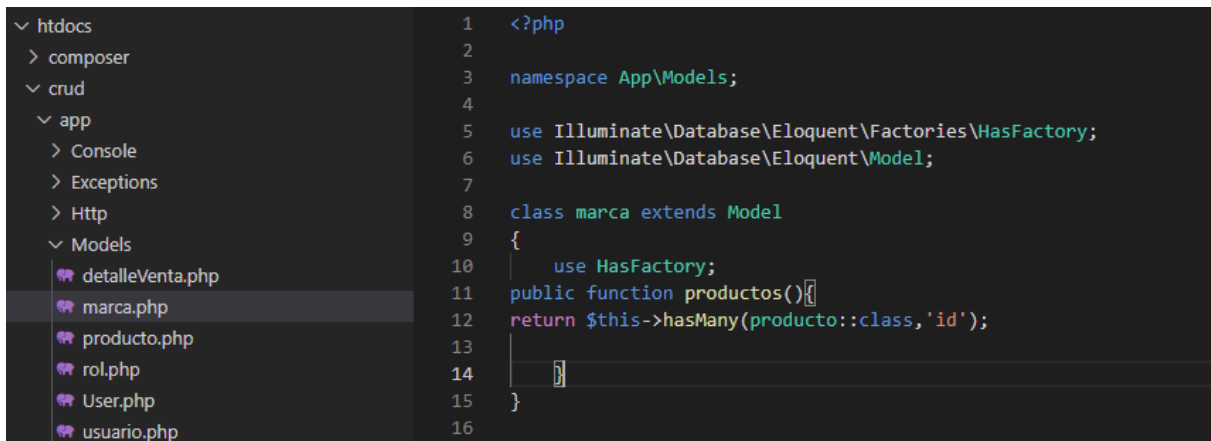
+ Añadir restricción

Ahora nos tenemos que dirigir a los modelos para que no nos muestre el id, sino nos traiga el nombre de lo que queremos traer



En estos modelos realice lo siguiente

MODELO MARCA



MODELO PRODUCTO

En el modelo de producto, comentareando el significado de las funciones:

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

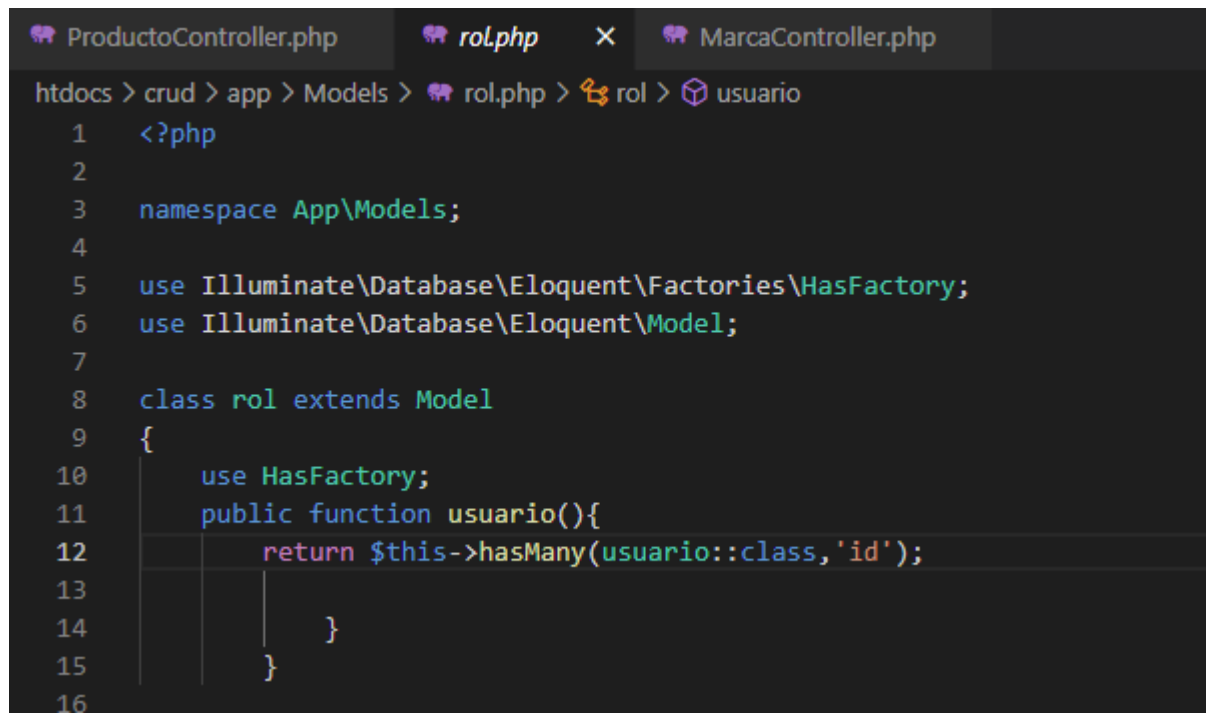
class producto extends Model
{
    use HasFactory;
    public function marca(){
        return $this->belongsTo(Marca::class, 'idMarcaFK');
    }

    //hasmany = muchos
    //belongs= una uno un producto tiene mucgas marcas
    //muchas marcas tienen 1 producto
}
```

MODELO USUARIO

```
ProductoController.php  usuario.php X  MarcaController.php
tdocs > crud > app > Models > usuario.php > usuario > rol
1  <?php
2
3  namespace App\Models;
4
5  use Illuminate\Database\Eloquent\Factories\HasFactory;
6  use Illuminate\Database\Eloquent\Model;
7
8  class usuario extends Model
9  {
10     use HasFactory;
11     public function rol(){
12         return $this->belongsTo(Rol::class, 'idRolFK');
13     }
14 }
15
```

MODELO ROL



The screenshot shows a code editor with three tabs: `ProductoController.php`, `rol.php` (active), and `MarcaController.php`. The breadcrumb navigation indicates the file path: `htdocs > crud > app > Models > rol.php > rol > usuario`. The code defines a `rol` class that extends `Model` and includes a `usuario` method.

```
1  <?php
2
3  namespace App\Models;
4
5  use Illuminate\Database\Eloquent\Factories\HasFactory;
6  use Illuminate\Database\Eloquent\Model;
7
8  class rol extends Model
9  {
10     use HasFactory;
11     public function usuario(){
12         return $this->hasMany(usuario::class,'id');
13     }
14 }
15
16
```


Ahora nos dirigimos a la vista form.blade y creamos un select option para poder llamar los datos de nuestra relación, cabe resaltar que hice este proceso con 2 tablas la primera con marca y producto y la segunda con rol y usuario

```
</div>
</div>
<div class="row mb-3">
  <label class="col-md-4 col-form-label text-md-end" for="idMarcaFK">Marca: </label>
  <div class="col-md-6">
    <select class="form-control" value="{{isset($producto->idMarcaFK)?$producto->idMarcaFK:''}}" name="idMarcaFK">
      @foreach ($marca as $marca)
        <option value="{{ $marca['id'] }}">{{ $marca['nombreMarca'] }}</option>
      @endforeach
    </select>
  </div>
</div>
```

El cual estamos trayendo en el producto EL NOMBRE de las marcas.

```
<div class="col-md-6">
  <select class="form-control" value="{{isset($producto->idMarcaFK)?$producto->idMarcaFK:''}}" name="idMarcaFK">
    @foreach ($marca as $marca)
      <option value="{{ $marca['id'] }}">{{ $marca['nombreMarca'] }}</option>
    @endforeach
  </select>
</div>
```


Y así se vería en la vista

Cantidad:

Marca:

Foto:

Esta es la foto actual:

 Agregue una foto por favor

Y así quedaría nuestra relación con el usuario rol

Registro Usuario

Email:

Password:

Nombre:

Rol:

Estado:

Enviar

Ahora en el editar

localhost/crud/public/usuario/10/edit

Correo:


Contraseña:

Nombre:

Rol:

Estado:

Volver Enviar

 **Administracion de Mercancia**

[Productos](#) [Marcas](#) [Usuarios](#)

[Iniciar Sesión](#) [Registrar](#)

Veamos el estado

Estado:

Activo

Activo

Activo

Inactivo

Mostrare a continuación el borrado en cascada , si borro una marca se borran todos los productos que tenían esa marca , y si borro un producto solo se borra el producto















Bienvenido

ID	Nombre	descripcion	foto	acciones
8	airpods	electrodomesticos		 
9	tablets	NUEVO		 



Bienvenido

ID	Nombre del producto	costo	precio	Cantidad	Marca	Foto	acciones
6	gferytyooooooo	12	12	1	airpods		 
7	PRUEBA45	12	12	1	airpods		 
9	ghyh	12	1	1	tablets		 
11	gferger	12	12	1	airpods		 






Bienvenido

ID	Nombre	descripcion	foto	acciones
8	airpods	electrodomesticos		 



Bienvenido

ID	Nombre del producto	costo	precio	Cantidad	Marca	Foto	
9	ghyh	12	1	1	tablets		 



localhost dice






Desea eliminar el registro?

Aceptar Cancelar

ID	Correo	Contraseña	Nombre	Usuario	Estado	acciones
2	jmorantes32@gmail.com	4121534	gfrgre	usuario	1	 Borrar 
4	admiiiiin@gmail.com	123456	valeryyyyy	Administrador	1	 Borrar 
5	admin@gmail.com	123456	valeryyyyy	Administrador	1	 Borrar 
10	zhixtin1207@icloud.com	123456	valeryyyyy	Administrador	Activo	 Borrar 
11	zhixtin1207@icloud.com	123456	valeryyyyy	Administrador	Activo	 Borrar 
12	PRUEBA1@GMAIL.COMM	123456	nose	usuario	Inactivo	 Borrar 
17	admin@gmail.com	123456	nose	Administrador	Activo	 Borrar 



Tabla usuarios





iD	Correo	Contraseña	Nombre	Usuario	Estado	acciones
2	jmorantes32@gmail.com	4121534	gfrgre	usuario	1	 Borrar
10	zhixtin1207@icloud.com	123456	valeryyyyy	Administrador	Activo	 Borrar
11	zhixtin1207@icloud.com	123456	valeryyyyy	Administrador	Activo	 Borrar
12	PRUEBA1@GMAIL.COMM	123456	nose	usuario	Inactivo	 Borrar
17	admin@gmail.com	123456	nose	Administrador	Activo	 Borrar

Borre algunos usuarios cabe recalcar que antes tenia el estado como 1 y 2 1 como activo 2 como inactivo y fui implementando cada vez más Lo deje string y con la tabla relacionada

Roles que tengo



Tabla Roles

iD	Nombre Rol	Estado Rol	Acciones
1	Administrador		 Borrar 
3	usuario		 Borrar 



Administracion de Mercancia

[Productos](#) [Marcas](#) [Usuarios](#)

[Iniciar Sesion](#) [Registrar](#)

GRACIAS POR SU ATENCION