
UD10.6 – Laravel

Relaciones entre modelos

2º CFGS
Desarrollo de Aplicaciones Web
2022-23

1.- Relaciones entre modelos

Mediante las **claves ajenas** se plasman las relaciones entre las tablas de una BBDD.

Gracias a Eloquent en Laravel es muy fácil replicar esas relaciones.

Para ello solo hay que **crear métodos en los modelos** que representan esas relaciones.

Como se verá más adelante el **nombre de los métodos** que se van a crear ayudan a entender el funcionamiento.

1.- Relaciones entre modelos

Hay que recordar que a la hora de indicar el nombre de las claves primarias y las claves ajenas se debe seguir las convenciones de Laravel:

- Nombres de tablas: inglés
- Clave primaria: id
- Clave ajena: tablaReferenciadaSingular_id → customer_id, product_id

También es importante tener clara la cardinalidad para añadir dichos métodos.

1.- Relaciones entre modelos

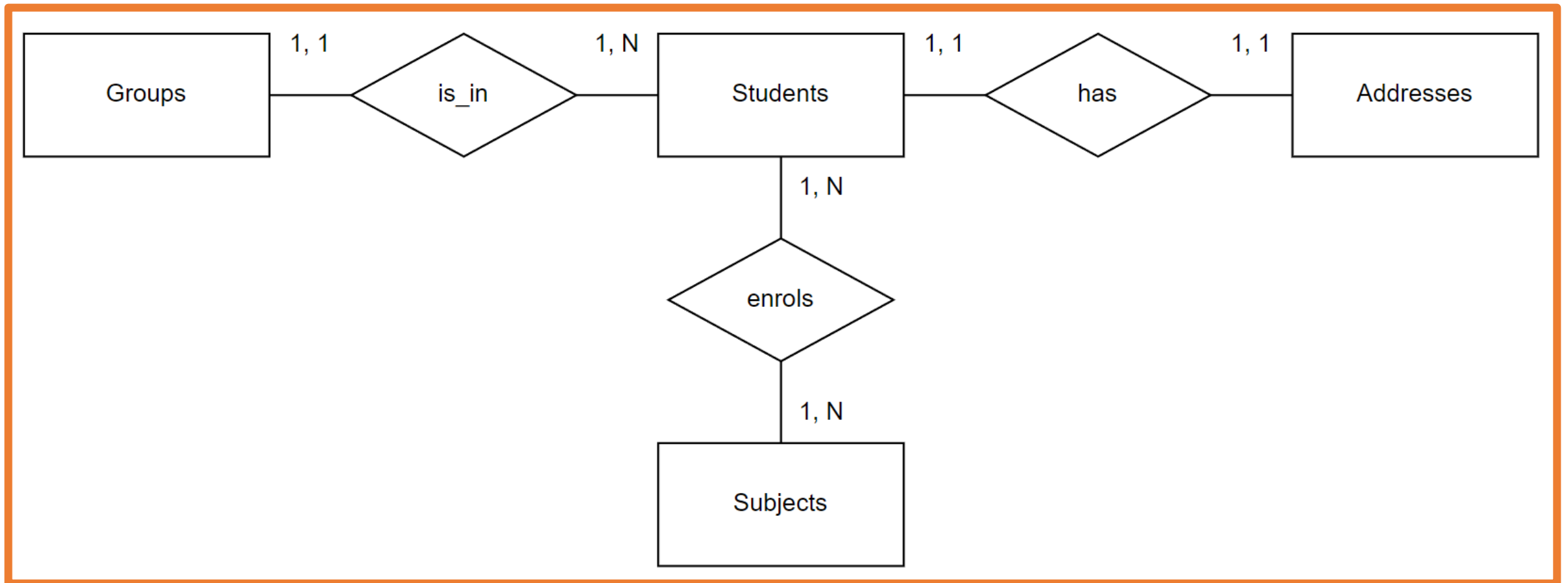
A continuación, se mostrarán las configuraciones más típicas para configurar en los modelos las relaciones entre las tablas.

En la [documentación oficial](#) se pueden consultar todas las opciones.

También se puede consultar las opciones para cuando no se siguen las convenciones de Laravel.

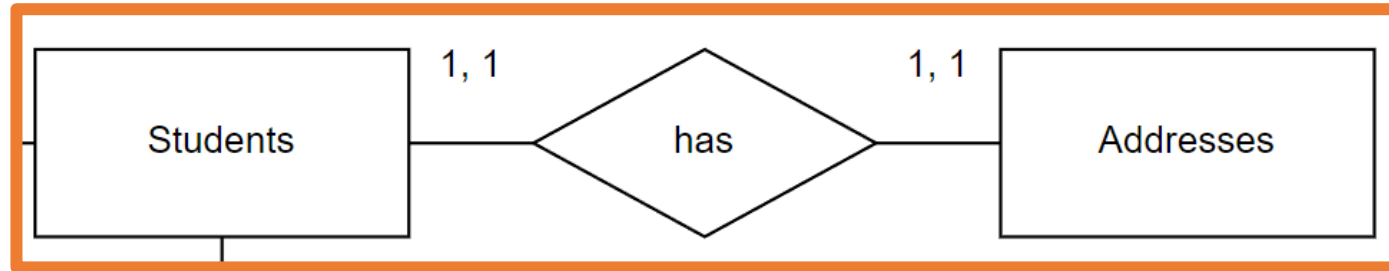
1.- Relaciones entre modelos

Para entenderlo mejor se tomará como ejemplo el siguiente diagrama ER.



2.- Relaciones UNO a UNO

Un estudiante solo tiene una dirección y viceversa.



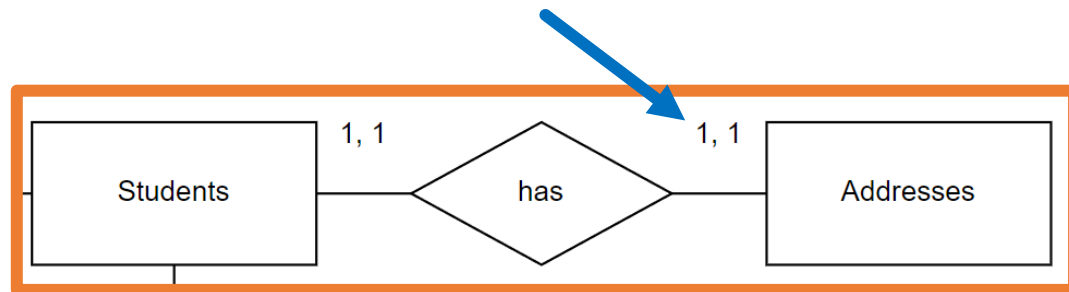
En el **modelo Student** se debe añadir el siguiente método:

```
class Student extends Model
{
    public function address()
    {
        return $this->hasOne(Address::class);
    }
}
```

2.- Relaciones UNO a UNO

El nombre del método va a indicar el tipo de relación.

En este caso un Student tiene una **única dirección**, por ello el nombre del método es el nombre de la tabla de direcciones en singular.

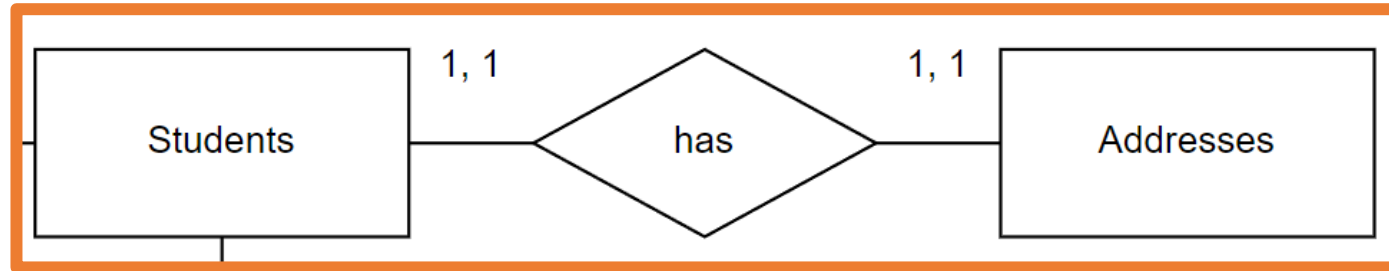


```
class Student extends Model
{
    public function address()
    {
        return $this->hasOne(Address::class);
    }
}
```

The code snippet shows a **Student** class extending a **Model** class. It includes a **public function address()**. A blue arrow points to the **address()** method name, which corresponds to the relationship name in the UML diagram.

2.- Relaciones UNO a UNO

Un estudiante solo tiene una dirección y viceversa.



En el **modelo Address** se debe añadir el siguiente método:

```
class Address extends Model
{
    public function student()
    {
        return $this->belongsTo(Student::class);
    }
}
```

A blue arrow points to the `student()` method in the code block.

2.- Relaciones UNO a UNO

Al definir los métodos anteriores, Eloquent es capaz de **recuperar todo el registro** referenciado por la clave ajena tanto en los **controladores** como en las **vistas** a partir de un registro dado.

```
public function show(Student $student)
{
    return view('students.show', compact('student'));
}
```

Objeto con todos los datos del estudiante: `$student`
`$student->nombre`

Objeto con todos los datos de la dirección referenciada: `$student->address`
`$student->address->codigopostal`

2.- Relaciones UNO a UNO

Objeto con todos los datos de la dirección referenciada: `$student->address`
`$student->address->codigopostal`

En este ejemplo a través de la variable `$student` se accede a su `address` y a su vez a los campos del registro de la `address`.

En la vista

```
<h1>{{ $student->nombre }} {{ $student->apellido1 }} {{ $student->apellido2 }}</h1>

Dirección:
{{ $student->address->tipo }}
{{ $student->address->nombre }}
{{ $student->address->codigopostal }}
{{ $student->address->localidad }}
```

Navegador

[Principal](#) [Listado de posts](#) [Nuevo post](#) [Lista de ofertas](#)

Rick Sanchez

Dirección: calle del pez 46001 valencia

Álex Torres © 2023

2.- Relaciones UNO a UNO

Con la configuración indicada en los modelos la relación también funciona a la **inversa**:

Obtener el nombre del estudiante al que está asociada una dirección:

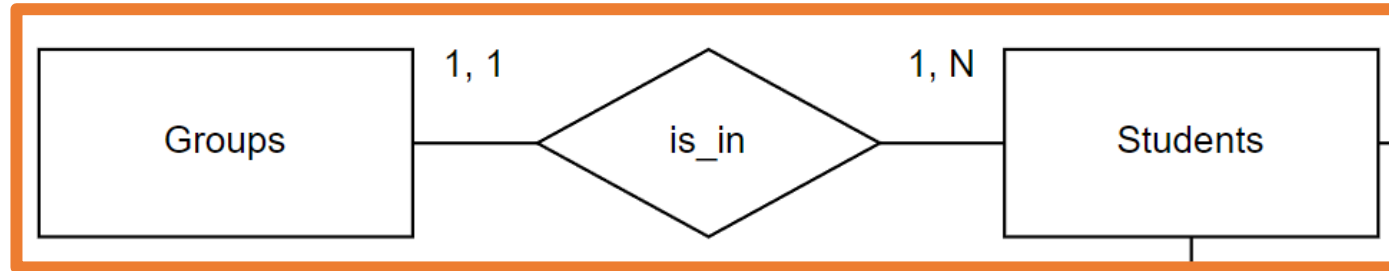
```
$direccion = Address::first();  
$direccion->student->nombre;
```

Hay que tener cuidado porque no hay límite a la hora de recorrer la relación:

```
$direccion->student->address->codigopostal  
$direccion->student->address->codigopostal->nombre
```


3.- Relaciones UNO a MUCHOS

Un grupo tiene muchos estudiantes y un estudiante pertenece a un solo grupo.



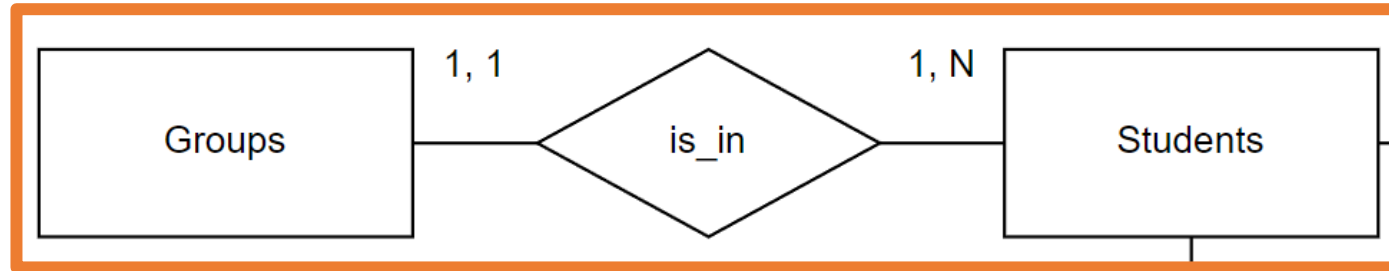
En el **modelo Group** se debe añadir el siguiente método:

```
class Group extends Model
{
    public function students()
    {
        return $this->hasMany(Student::class);
    }
}
```



3.- Relaciones UNO a MUCHOS

Un grupo tiene muchos estudiantes y un estudiante pertenece a un solo grupo.



En el **modelo Student** se debe añadir el siguiente método:

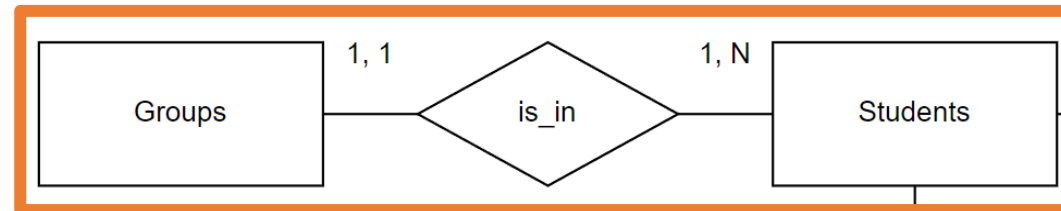
```
class Student extends Model
{
    public function group()
    {
        return $this->belongsTo(Group::class);
    }
}
```

3.- Relaciones UNO a MUCHOS

Como se ha indicado antes, el nombre del método va a indicar el tipo de relación.

En este caso un Student tiene una **única grupo**, por ello el nombre del método es el nombre de la tabla de grupos en singular.

En este caso un Grupo tiene **muchos Students**, por ello el nombre del método es el nombre de la tabla de students en plural.



```
class Group extends Model
{
    public function students()
    {
        return $this->hasMany(Student::class);
    }
}
```

A blue arrow points from the `students()` method name to the `hasMany` method call in the code.

```
class Student extends Model
{
    public function group()
    {
        return $this->belongsTo(Group::class);
    }
}
```

A blue arrow points from the `group()` method name to the `belongsTo` method call in the code.

3.- Relaciones UNO a MUCHOS

Gracias a los métodos anteriores Laravel es capaz de recuperar todo el registro referenciado tanto en los **controladores** como en la **vistas**.

Array con todos los estudiantes del grupo: `$group->students`

```
foreach($group->students as $estudiante) {  
    $estudiante->nombre;  
}
```

Objeto con el grupo al que pertenece el estudiante: `$student->group`

```
$student->group->nombre;
```

4.- Relaciones MUCHOS a MUCHOS

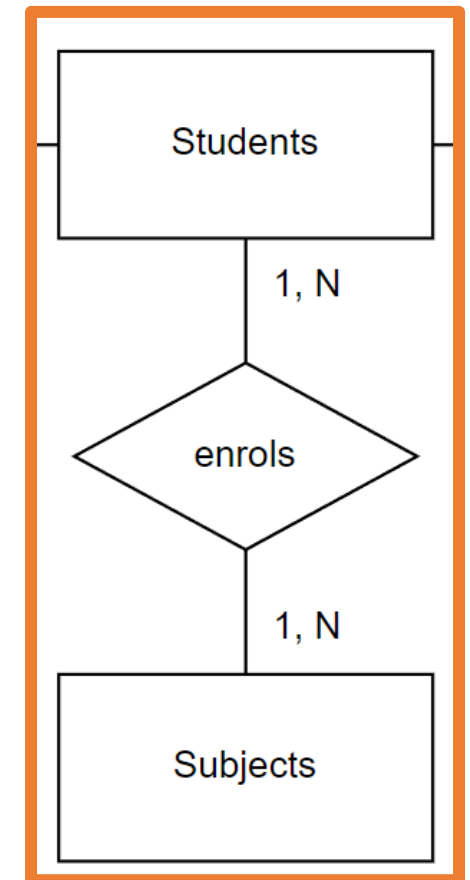
Un estudiante está matriculado en muchas asignaturas y en una asignatura están matriculados muchos estudiantes.

En este caso es necesario crear una tabla nueva para plasmar la relación N:M.

El **nombre de la tabla** intermedia tiene que ser **los nombres de las tablas en singular unidas con el carácter _ y en orden alfabético**.

Se deberá crear una migración para una tabla que se llame **subject_student** y debe tener al menos los siguientes atributos:

```
Schema::create('subject_student', function (Blueprint $table) {  
    $table->foreignId('student_id')->constrained();  
    $table->foreignId('subject_id')->constrained();  
    $table->unique(['student_id', 'subject_id'], 'claves_ajenas');  
});
```

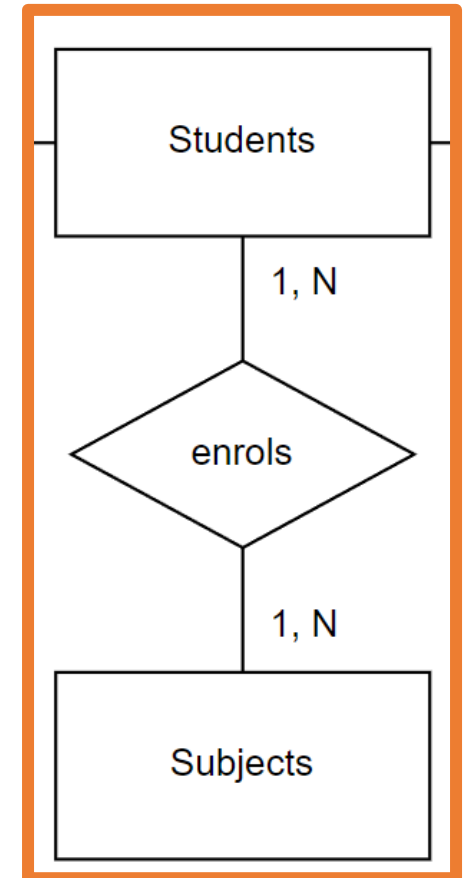



4.- Relaciones MUCHOS a MUCHOS

Un estudiante está matriculado en muchas asignaturas y en una asignatura están matriculados muchos estudiantes.

En el **modelo Student** se debe añadir el siguiente método:

```
class Student extends Model
{
    public function subjects()
    {
        return $this->belongsToMany(Subject::class);
    }
}
```

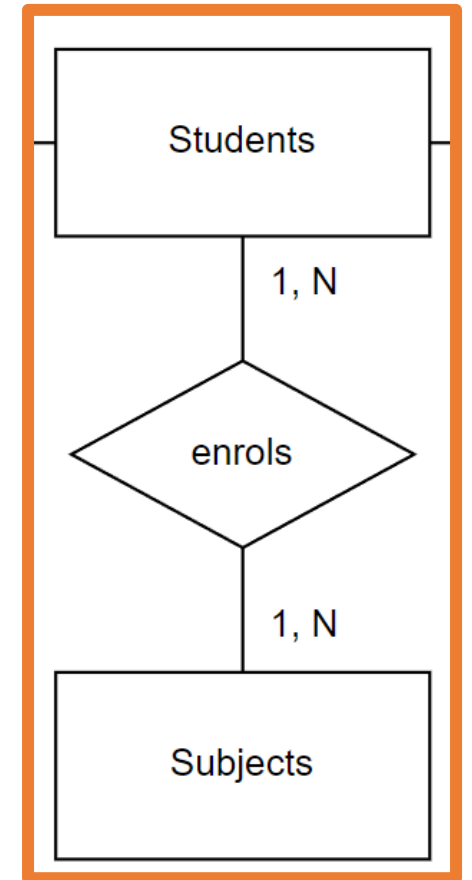



4.- Relaciones MUCHOS a MUCHOS

Un estudiante está matriculado en muchas asignaturas y en una asignatura están matriculados muchos estudiantes.

En el **modelo Subject** se debe añadir el siguiente método:

```
class Subject extends Model
{
    public function students()
    {
        return $this->belongsToMany(Student::class);
    }
}
```



4.- Relaciones MUCHOS a MUCHOS

Gracias a los métodos anteriores Laravel es capaz de recuperar todo el registro referenciado tanto en los **controladores** como en la **vistas**.

Array con todas las asignaturas de un estudiante está matriculado: `$student->subjects`

```
foreach($student->subjects as $asignatura) {  
    $asignatura->nombre;  
}
```

Array con todos los estudiantes matriculados en una asignatura: `$subject->students`

```
foreach($subject->students as $estudiantes) {  
    $estudiante->nombre;  
}
```

5.- Otras acciones con Eloquent

En los ejemplos vistos en esta unidad y la anterior se utiliza Eloquent para acceder a la base de datos.

Se han visto algunas funciones que se pueden utilizar para filtrar los datos pero existen muchas más.

En la documentación se pueden consultar [todas](#).

- # Select Statements
- # Raw Expressions
- # Joins
- # Unions
- # Basic Where Clauses
 - # Where Clauses
 - # Or Where Clauses
 - # Where Not Clauses
 - # JSON Where Clauses
 - # Additional Where Clauses
 - # Logical Grouping
- # Advanced Where Clauses
 - # Where Exists Clauses
 - # Subquery Where Clauses
 - # Full Text Where Clauses
- # Ordering, Grouping, Limit & Offset
 - # Ordering
 - # Grouping
 - # Limit & Offset
- # Conditional Clauses

Or Where Clauses

When chaining together calls to the query builder's `where` method, the "where" clauses will be joined together using the `and` operator. However, you may use the `orWhere` method to join a clause to the query using the `or` operator. The `orWhere` method accepts the same arguments as the `where` method:

```
$users = DB::table('users')
    ->where('votes', '>', 100)
    ->orWhere('name', 'John')
    ->get();
```

If you need to group an "or" condition within parentheses, you may pass a closure as the first argument to the `orWhere` method:

```
$users = DB::table('users')
    ->where('votes', '>', 100)
    ->orWhere(function($query) {
        $query->where('name', 'Abigail')
        ->where('votes', '>', 50);
    })
    ->get();
```

The example above will produce the following SQL:

```
select * from users where votes > 100 or (name = 'Abigail' and votes > 50)
```

6.- Consultas directas a la base de datos

En ocasiones se necesita realizar consultas directamente a la base de datos.

Para ello se usa la clase **DB**.

Se debe añadir al archivo en la parte superior la siguiente instrucción:

```
use Illuminate\Support\Facades\DB;
```

Posteriormente en el archivo:

```
$clientes = DB::table('customers')  
    ->select('nombre', 'dni')  
    ->where('vip', 1)  
    ->get();
```

Se pueden consultar todas las opciones en [la documentación](#).

Práctica

Actividad 11:

Relacionando modelos.