

Base de datos y ORM Eloquent

Modelos en Laravel - Uso de Base de datos (Migration & Eloquent)

Migrations

Requisito previo: Conexión a base de datos

La configuración de los servicios de base de datos de Laravel se encuentra en el archivo ***config/database.php*** de la aplicación.

En este archivo, se pueden **definir todas las conexiones a la base de datos**, así como **especificar cuál de ellas debe usarse por defecto**.

En database.php mismo se proporcionan opciones para la mayoría de los sistemas de bases de datos compatibles con Laravel (incluyendo mysql).

La mayoría de las opciones de configuración dentro de este archivo **están determinadas por** los valores de **las variables de entorno** definidas en el fichero **“.env”** de la aplicación.

Esto significa que, **si se modifican las variables de entorno (.env), no es necesario modificar directamente database.php para poder conectar** con la BBDD.



database.php

```
use Illuminate\Support\Str;
```

```
return [  
    'default' => env('DB_CONNECTION', 'sqlite'),  
    'connections' => [  
        'mysql' => [  
            'driver' => 'mysql',  
            'url' => env('DB_URL'),  
            'host' => env('DB_HOST', '127.0.0.1'),  
            'port' => env('DB_PORT', '3306'),  
            'database' => env('DB_DATABASE', 'laravel'),  
            'username' => env('DB_USERNAME', 'root'),  
            'password' => env('DB_PASSWORD', ''),  
            'unix_socket' => env('DB_SOCKET', ''),  
            'charset' => env('DB_CHARSET', 'utf8mb4'),  
            'collation' => env('DB_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'),  
            ]) : [],  
        ],  
    ],  
];  
  
// ...
```

.env

DB_CONNECTION=mysql

DB_HOST=127.0.0.1

DB_PORT=3306

DB_DATABASE=proyecto1

DB_USERNAME=root

DB_PASSWORD=



Una vez conectada la base de datos: realizar consultas

Una vez que se haya configurado la conexión con la base de datos, se pueden ejecutar consultas utilizando el facade **DB**. Este ofrece métodos para cada tipo de consulta: `select()`, `update()`, `insert()`, `delete()` y `statement()`.


El método `select()` siempre devolverá un array de resultados. Cada resultado dentro del array será un objeto PHP `stdClass` que representa un registro de la base de datos:

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

```
$users = DB::select('select * from users'); // alternativa: DB::table('users')->get();
```

```
foreach ($users as $user) { echo $user->name; }
```

Esta clase DB puede resultar útil para pruebas, pero **NO equivale al modelo**. NO emplea el ORM Eloquent.



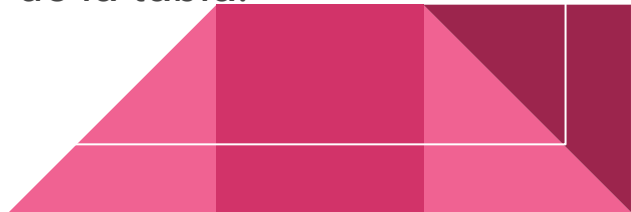
Introducción a Migrations y Eloquent

Con Laravel no hay necesidad de generar manualmente las tablas directamente en base de datos, o construir queries SQL.

Con **Migrations**, es posible **gestionar estructuras (esquemas) de bases de datos**.

Al utilizar **Eloquent**, **cada tabla de la base de datos tiene un "Modelo" correspondiente que se usa para interactuar** con dicha tabla.

Además de seleccionar registros de la tabla de la base de datos, los modelos de Eloquent permiten insertar, actualizar y eliminar registros de la tabla.



Crear tablas en Base de Datos con Laravel - migration

Antes de continuar con Eloquent, es necesario entender el uso del comando de Artisan **migrate**.

“**migrations**” en Laravel es una forma de **versionar y gestionar los cambios en la estructura de la base de datos**.

Permiten definir tablas, columnas, índices y otras características de la base de datos vía script.

Estos scripts se guardan por convenio en ***database/migrations***.



Crear tablas en Base de Datos con Laravel - migration

Estos “migrations” se generan mediante el comando Artisan ***make:migration***:

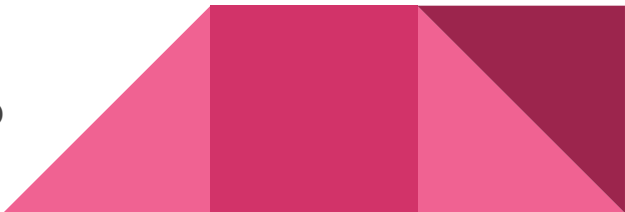
php artisan make:migration nombre_de_la_migracion

Este comando crea un archivo en el directorio database/migrations con una estructura predeterminada, donde se definen los métodos ***up()*** y ***down()***.

- ***up()***: Se utiliza para definir los **cambios que deben aplicarse** a la base de datos, como la creación de tablas, columnas, índices, etc.
- ***down()***: Este método **revierte los cambios** definidos en el método ***up()***. Es útil cuando se desea deshacer una migration.

Al hacer un ***php artisan migrate***, se efectúa un ***up()*** de todas las “migrations” definidas.

Por el contrario, ***php artisan migrate:rollback***, que ejecuta el método ***down()*** de las “migrations”.



Crear tablas en Base de Datos con Laravel - migration

Dentro de estos métodos **up()** es donde se puede usar el facade “**Schema**” (*Illuminate\Support\Facades\Schema*) para definir o modificar tablas.

Por ejemplo, para crear una tabla “users” y sus columnas:

Schema::create('users', function (Blueprint **\$table**) { //Para crear una nueva tabla.

\$table->id(); // Para añadir una clave id primaria. Alternativa: **->primary('id');**

\$table->string('name'); // Para añadir una columna de tipo string.

\$table->string('email')->unique(); // Para añadir valor único

\$table->integer('edad')->nullable(); // Para añadir numérico que puede ser nulo

\$table->timestamps(); //Añade las columnas created_at y updated_at

});

Para ver todos los métodos de Schema (tipos de columnas):

<https://laravel.com/docs/11.x/migrations#creating-columns>

Crear tablas en Base de Datos con Laravel - migration

Por otro lado, en los métodos ***down()*** se usa el facade “**Schema**” para realizar la **acción opuesta a la definida en *up()***.

Por ejemplo, para compaginar el *up()* anterior se debe hacer **drop de la tabla “users”**:

```
Schema::dropIfExists('users');
```



Modificar tablas en Base de Datos con Laravel - migration

Migrate rehace la estructura de tablas, lo que puede conllevar la **pérdida de datos**.

Si se desea **modificar una tabla existente**, para evitar la pérdida de datos, se debe crear un migrate específico. Si se emplea el convenio de snake_case indicando con “to” el nombre de la tabla, Laravel es capaz de preparar el método dentro del nuevo *up()*.

Por ejemplo, añadir “surname” a la tabla user:

```
php artisan make:migration add_surname_to_users_table
```



Modificar tablas en Base de Datos con Laravel - migration

Si se revisa el documento *FECHA_HORA_add_surname_to_users_table.php* generado, dentro del método *up()* habrá:

Schema::table('users', function (Blueprint \$table) { //Trabaja con tabla existente.

\$table->text('surname')->after('name'); // Para añadir una columna de tipo texto.

});

En el método down(), dentro del Schema, habrá que definir lo opuesto:

\$table->dropColumn('surname');

Nota: Todos estos migrations también se pueden crear manualmente.



Eloquent - Modelos

Generar una clase Modelo

Los modelos suelen estar en el directorio **app\Models** y extienden la clase ***Illuminate\Database\Eloquent\Model***.

Se puede usar el comando ***make:model*** de Artisan para generar un nuevo modelo (el nombre, por convención se define en PascalCase - primera letra mayúscula):

```
php artisan make:model Vehiculo
```

Si se desea generar una migración de base de datos al mismo tiempo que se crea el modelo, se puede usar la opción ***--migration*** o ***-m***. Esto permite **recrear las tablas** al ejecutar ***php artisan migrate***.

Además, es posible **generar otros tipos de clases junto con el modelo**, como **controladores de recursos (con *--controller --resource*)** y form requests. Estas opciones pueden combinarse para crear múltiples clases a la vez:

```
# Genera un modelo, un controlador VehiculoController tipo resource y su correspondiente migration  
php artisan make:model Vehiculo --controller --resource --migration
```

Generar una clase Modelo

El resultado de generar el modelo “Vehiculo” será un fichero php en **app/Models**. Por convención, el modelo se crea con el nombre singular de la tabla. Para especificar el nombre real de la tabla vinculada al modelo, se debe añadir dicho nombre como variable ***\$table*** protegida:

```
<?php
```

```
namespace App\Models;  
use Illuminate\Database\Eloquent\Model;
```

```
class Vehiculo extends Model {  
    protected $table = 'vehiculos';
```

```
    // ...
```

```
}
```

*Si no se especifica el nombre, Eloquent buscará la tabla en BBDD como el plural en “snake_case”: con un Modelo **AirTrafficController** busca la tabla **air_traffic_controllers***


Claves primarias del Modelo

Eloquent también asume que la clave primaria del modelo se llama **id**.

Por convenio, esta nomenclatura es suficiente, pero se puede especificar un nombre de id diferente añadiendo el atributo protegido ***\$primaryKey***:

```
namespace App\Models;
use Illuminate\Database\Eloquent\Model;

class Vehiculo extends Model {
    protected $primaryKey = 'id_vehiculo'; // clave primaria será id_vehiculo en lugar de id
    // ...
}
```




Una vez creada la clase Modelo de la tabla...

Una vez creada la clase modelo de una tabla **ya es posible trabajar con la tabla** equivalente de la base de datos.

Existen varias formas para trabajar con registros de base de datos. Por ejemplo, se puede **referenciar directamente la clase Modelo**. Con un modelo de User, para añadir un usuario:

```
// crear e insertar un nuevo registro en la base de datos en una sola operación  
User::create(['name' => 'Juan', 'email' => 'juan@example.com']);
```

Requiere que se haya habilitado la propiedad \$fillable o \$guarded en el modelo para especificar qué campos pueden ser asignados de forma masiva.



Trabajar con instancias del Modelo

Sin embargo, la forma más sencilla y fácil de trabajar con registros de base de datos es instanciando la clase Modelo.

Replicando el ejemplo previo, para crear un nuevo registro en la tabla “users”:

```
use App\Models\User
```

```
$newUser = new User();    // instancia de clase User
```

```
$newUser->name = 'Juan';  // declarar valor para columna name
```

```
$newUser->email = 'juan@example.com'; // columna email
```

```
$newUser->save();        // guarda en base de datos
```



Trabajar con instancias del Modelo - Métodos comunes

Listado de métodos básicos de eloquent para trabajar con registros de tablas en bases de datos vía instancias de clase Modelo correspondientes:

- **get()** - realizar una consulta a la base de datos.
\$these_users = User::where('name', 'Juan')->get(); // Usuarios de nombre Juan
- **all()** - obtiene todos los registros de la tabla
\$users = User::all(); // Todos los usuarios
- **find(\$id)** - obtener un registro por su id (clave primaria)
\$this_user = User::find(1); // Usuario con ID=1



Trabajar con instancias del Modelo - Métodos comunes

- **save()** - Actualizar o guardar el registro en base de datos
`$this_user->save();`
- **delete()** - Eliminar el registro de base de datos
`$this_user->delete();`

`get()`, `save()` y `delete()` son suficientes para poder desarrollar **CRUD**.



Trabajar con instancias del Modelo - Select

Para **realizar una consulta (select)** a base de datos vía una clase de Modelo correspondiente, se puede **concatenar las características de la consulta** a la instancia, **finalizando con el método `get()`**.

Por ejemplo:

```
$usuarios = User::where('name', 'Juan')    // FROM Users WHERE name = 'Juan'  
    ->orderBy('id')    // ORDER BY id  
    ->take(10)    // LIMIT 10  
    ->get();    // SELECT *
```

Listado de métodos que se pueden emplear:

<https://laravel.com/docs/11.x/queries>



Trabajar con instancias del Modelo - Update

Empleando una instancia del Modelo es posible seleccionar las columnas como atributos y darles valor. Cuando todo esté listo, ¡guarda los cambios!

Siguiendo el ejemplo de la tabla users:

```
$this_user = User::find(1); // Suponiendo que el usuario tiene el ID 1
```

```
$this_user->surname = 'Perico'; // Asignando el nuevo valor a columna 'surname'
```

```
$this_user->save(); // Guarda el cambio en la base de datos
```



Trabajar con instancias del Modelo - Más Métodos

Los siguiente métodos también pueden ser de mucha utilidad:

- **create()** - Crea el registro directamente en base de datos (no requiere save())
`User::create(['name' => 'Juan', 'email' => 'juan@example.com']);`
- **update()** - Actualiza uno o varios registros.
`$this_user->update(['name' => 'John Lewis']);`
- **count()** - Obtiene el número de registros que cumplen con la condición.
`$count = User::where('name', 'Juan')->count();`
- **exists()** - Verifica si existe un registro que cumpla con las condiciones. Devuelve booleano.
`$exists = User::where('email', 'juan@example.com')->exists();`
- **findOrFail()** - Encuentra un registro por id o lanza una excepción si no se encuentra.
`$user = User::findOrFail(1);`
- **pluck()** - Extraer una lista de valores de una sola columna de los registros. Este metodo puede devolver un array asociativo si se añade como segundo parámetro una segunda columna (actuará como clave).
`$emails = User::pluck('email', 'name'); // devuelve: ['Juan' => 'juan@example.com']`

Modelo - Controlador - Vista

Así como desde el Controlador se incluyen las redirecciones a rutas o cargas a vistas, también **se puede incluir la clase modelo y aplicar los métodos para trabajar con registros de bases de datos en los métodos del controlador.**

De esta forma, el controlador pasa a ser el elemento central que gestiona o, más bien, compone el núcleo de las funcionalidades de la aplicación web, cumpliendo el patrón de desarrollo de **Modelo-Vista-Controlador.**

