

Desarrollo de aplicaciones web con Laravel 5.1

Clase 6. ORM Eloquent I

Carlos Ruiz Ruso • @micromante

¿Por qué Eloquent?

- **Uso sencillo y código limpio**
- Normalmente los modelos estan en la carpeta app pero podrían ponerse en cualquier subcarpeta.
- Todos los modelos extienden de **Illuminate\Database\Eloquent\Model** lo que permite hacer un mapeo activo con el **ActiveRecord**
- Cada tabla “corresponde” con un modelo
- Capas de trabajo
 - **Migrations / Seeding**
 - **Query Builder (Estilo PDO)**
 - **Eloquent ORM**

Primeros pasos con Eloquent

Configuración de los parámetros de la bd en
`config/database.php`



Extendemos nuestros modelos a Eloquent

```
namespace App;
```

```
use Illuminate\Database\Eloquent\Model;
```

```
class User extends Model
```

```
{
```

```
...
```

```
}
```

Convenciones iniciales a tener en cuenta

- El modelo en singular y la tabla en plural. User → users, Product → products, si no queremos que sea así le indicaremos con la variable `$table = 'my_users'`.
- Por defecto busca siempre la primary key “id”, si queremos cambiar esta clave debemos indicarlo en `$primaryKey`.
- Eloquent por defecto usa las columnas `created_at` y `updated_at` para actualizar automáticamente los timestamps. Si no queremos que esto ocurra `$timestamps = false`;. También puedes personalizar el estilo del timestamp con `$dateformat = 'U'`.
- En el controlador le indicamos siempre `use App\User`; y luego ya podemos usarlo como fachada → `User::all()`;
- Eloquent `all` y `get` como veremos siempre devuelven colecciones del tipo `Collection`.

Vale...pero...

Antes de traer los datos de la base de datos con Eloquent, tenemos que crear las tablas.

Vamos entonces!



Migraciones I

- Las migraciones son como un **control de versiones de tablas**.
- Permite a los equipos trabajar facil, compartir y modificar.
- Trabajamos a través de la fachada Schema.
- API unificada para todos los sistemas.
- Abstracción de la creación y modificación.
- Las migraciones se crean en database/migrations, cada migración contiene su timestamp en el nombre.

Migraciones II

- Creamos las migraciones
 - `php artisan make:migration name_migration --table=tablename`
- Ejecutamos la migración
 - `php artisan migrate`
 - Si aparece el error class not found usar “composer dump-autoload”
- Rollback última migración
 - `php artisan migrate:rollback`
- Rollback todas las migraciones en el orden de creación
 - `php artisan migrate:reset`
- Estado de las migraciones
 - `php artisan migrate:status`



Demo Migración

Estructura de la migración up y down

Creacion de tablas

```
Schema::create('usuarios', function($table) {  
    ...  
    //Definición de la tabla y campos  
});
```

\$table es un objeto Blueprint que define la nueva tabla.

Renombrado de tablas

`Schema::rename('nombreactual','nombrenuevo')`

Borrado de tablas

```
Schema::drop('tabla');  
Schema::dropIfExists('tabla');
```

Comprobar si existen tablas o columnas

Schema::hasTable('tabla');

Schema::hasColumn('tabla','columnname');

Multiples bases de datos o sistemas

```
Schema::connection('nombredelaconfig')->create('users', function ($table) {  
    ....  
});
```

```
Schema::create('users', function ($table) {  
    $table->engine = 'InnoDB'; ← tipo de motor  
});
```


Definición de columnas

```
$table->increments('id'); //Valor incremental
```

```
$table->char('name',5); //String
```

```
$table->string('email')->unique(); //String y valor unico
```

```
$table->integer('rol'); //Entero
```

```
$table->longText('description'); //Texto largo
```

```
$table->timestamps(); //Añadimos created_at y updated_at
```

```
..
```

Listado completo <http://laravel.com/docs/5.1/migrations#renaming-and-dropping-tables>

Indices y claves

```
$table->primary('id');  
$table->primary(array('id','email'));  
$table->unique('email');  
$table->index('token');
```

Modificadores

- **nullable(); //Permitir nulos**
- **default(\$valor); //Valores por defecto**
- **unsigned(); //Sin signo**
- **change(); //Cambiar el valor o tamaño de un campo existentes**

Concepto Timestamps

Eloquent utiliza por defecto las columnas **created_at** y **updated_at**

Podemos evitar estas columnas añadiendo en el modelo la variable
protected **\$timestamp = false;**

Es muy útil ya que automáticamente guarda los cambios y creación de un campo sin tener que pensar en esto nosotros.

\$table->timestamps(); //Crea los campos en la migración

\$table->dropTimestamps(); //Elimina los campos de una tabla

Concepto SoftDeletes

Esto nos permite que al borrar un **elemento realmente no se borra**, simplemente se actualiza este campo. Es una forma de hacer eficientes ciertos bases de datos con muchas consultas.

En las migraciones hacemos **`$table->softDeletes();`**
para añadir el campo **`deleted_at`** a la tabla.

Otras funciones de Schema Builder

```
$table->dropColumn('votos');
```

```
$table->dropColumn(array('votos','nombre'));
```

```
$table->renameColumn('este','otro');
```

```
$table->dropbPrimary('id');
```

```
$table->dropUnique('email');
```

```
$table->dropIndex('name');
```


Foreign Key y relación con tablas

```
Schema::table('posts', function ($table) {
```

```
$table->integer('user_id')->unsigned(); //Siempre unsigned
```

```
$table->foreign('user_id')->references('id')->on('users');
```

```
//user_id → campo en nuestra tabla
```

```
//id → nombre del campo en la tabla ajena
```

```
//users → nombre de la tabla ajena
```

```
});
```

Modificadores de relaciones

```
Schema::table('posts', function ($table) {
```

```
    $table->integer('user_id')->unsigned(); //Siempre unsigned
```

```
    $table->foreign('user_id')->references('id')->on('users')->onDelete('cascade');
```

```
});
```

Podemos también eliminar la relación con `$table->dropForeign('nombre_foreign');`

Datos de prueba con Seeding

- Clase especial dentro de database/seed
- Definimos siempre con el formato
 - NameClassSeeder
- **Extends Seeder**
- Podemos invocar otras clases con `$this->call('NameClassSeeder');`
- Ejecutamos inserción
 - **php artisan db:seed**
 - `php artisan db:seed --class=MiTabelaSeeder`
 - `php artisan migrate:refresh --seed` //Regenera por completo las tablas y seeds

Ejemplo de Seed

```
use DB;  
use Illuminate\Database\Seeder;  
use Illuminate\Database\Eloquent\Model;  
  
class DatabaseSeeder extends Seeder  
{  
  
    public function run()  
    {  
        //Inserción de datos en la bd con DB o Eloquent  
    }  
}
```

ELOQUENT ORM

Crear un elemento desde la Fachada

```
User::create(array(
```

```
    'nombre'      => 'Carlos',
```

```
    'apellidos'   => 'Ruiz Ruso',
```

```
    'email'       => 'micromante@gmail.com'
```

```
));
```


Creando un objeto y guardandolo

```
$user = new User;
```

```
$user->nombre = 'Carlos';
```

```
$user->apellidos = 'Ruiz Ruso';
```

```
$user->email = 'micromante@gmail.com'
```

```
$user->save();
```

Buscar elementos • Parte 1

Obtener todos los elementos

```
$users = User::all();
```

Buscar una ID en concreto

```
$user = User::find(1);
```

Buscar elementos • Parte 2

Busqueda por campo o campos

```
$users = User::where('nombre','=','Carlos')->get();
```

```
$users = User::where('edad','>',25)->get(['nombre']);
```

Obtener la primera fila de los resultados

```
$users = User::where('edad','>',25)->first();
```


Buscar o Crear

Búsqueda de un elemento y si no existe lo creamos

```
User::firstOrCreate(['nombre' => 'Carlos']);
```

Podemos lanzar la instancia a un objeto

```
$user = User::firstOrCreate(['nombre' => 'Carlos']);
```

```
$user->save();
```

Último registro Actualizado o Creado

Cuando guardamos o creamos un elemento podemos saber su id

```
$user = User::firstOrCreate(['nombre' => 'Carlos']);
```

```
$user->save();
```

```
$id_insertada = $user->id;
```

Actualizar

Búsqueda de un elemento y luego lo modificamos

```
$user = User::where('nombre','=','Carlos')->first();
```

Ahora hacemos los cambios en la variable objeto

```
$user->edad = 26;
```

```
$user->save();
```


Eliminar • Parte 1

Búsqueda de un elemento y luego lo modificamos

```
$user = User::where('nombre','=','Carlos')->first();
```

```
$user = User::find(1);
```

```
$user->delete();
```

```
$user = User::where('edad','<',24)->delete();
```

```
User::destroy([1,2,3]);
```

Eligiendo la configuración de la conexión

```
$user = User::on('connection-name')->find(1);
```

Extra

`$user->touch()` → Actualizar el timestamp

`$user->touch()` → igual que `->save()` pero guardando las relaciones entre tablas


Capturando excepciones de los Models

```
use Illuminate\Database\Eloquent\ModelNotFoundException;

class Handler extends ExceptionHandler {

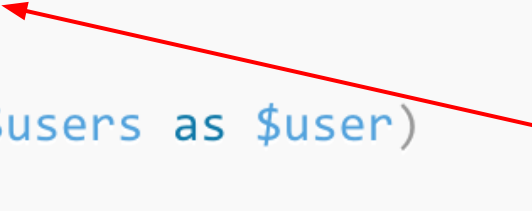
    public function render($request, Exception $e)
    {
        if ($e instanceof ModelNotFoundException)
        {
            // Custom logic for model not found...
        }

        return parent::render($request, $e);
    }
}
```



Procesando millones de registros sin romper la memoria

```
User::chunk(200, function($users)
{
    foreach ($users as $user)
    {
        //
    }
});
```



Próximo día Parte II

Relaciones de tablas
Consultas al estilo PDO con Query Builder
Cacheo y Transacciones

MUCHAS GRACIAS A TODOS!

Podéis seguirme en la redes sociales como @micromante o Carlos Ruiz Ruso
www.micromante.com