Desarrollo de aplicaciones web con Laravel 5.1

Clase 7. ORM Eloquent II

Carlos Ruiz Ruso · @micromante

Query builder

Interfaz que permite realizar peticiones a más bajo nivel con Eloquent.

Utilizamos las funciones de PDO pero con una capa de seguridad que evita inyecciones y limpia las cadenas.

Ya no son objetos de tipo Eloquent Collection,

ahora son objetos de tipo PDO.

Búsqueda de elementos con condiciones

```
$users = DB::table('users')
                    ->select('name as Nombre')
                        ->where('name', '=', Carlos)
                            ->get(); o ->first();
```

```
$users = DB::table('users')
                    ->select('name as Nombre')
                        ->where('name', '=', Carlos)
                            ->lists('titulo','descripcion'); //Return array('title','des');
```

Busqueda de elementos con condiciones

```
$users = DB::table('users')
            ->select('name as Nombre')
            ->get();
```

->get() y ->fist()

```
$users = DB::table('users')->get(); //Todos
$users = DB::table('users')->first(); //Primero
```

*Podemos añadirles condiciones a ambos para el filtrado del dato

Insertando elementos

DB::table('usuarios')->insert($data);

$id = DB::table('usuarios')->insertGetId($data); //ID Insertado

$data es una array con los campos y las columnas previamente creadas. Ejemplo array ('name' => …) la column debe existir o lanzará una excepción.

Uniones de tablas

```
$users = DB::table(‘usuarios’)
        ->join(ventas, ‘ventas.id_user', '=', 'usuarios.id')
        ->select(usuarios.nombre', ventas.precio')
        ->get();
```

Actualización de los datos

```
$users = DB::table('usuarios')
        ->where('name','!=','carlos')
        ->update(array('columnname' => 'value', ....))
```

Ejemplo operación avanzada Query builder

```
$users = DB::table('usuarios')
        ->where('id',1)
        ->increment('votos');
```

```
$users = DB::table('usuarios')
        ->where('id',1)
        ->decrement('votos');
```

Tanto a increment como a decrement podeis pasarle otro parametro con el numero a incrementar. Por defecto 1.

Ejemplo operación avanzada Query builder

```
$users = DB::table(‘usuarios’)
        ->leftjoin(ventas, ‘ventas.id_user', '=', 'usuarios.id')
        ->get(array(‘nombre’, ‘precio’));
```

Orden, agrupacion, condición agrupación

```
$users = DB::table('usuarios')
            ->orderBy('name','desc')
            ->groupBy('count')
            ->having('count','>',100)
                ->get();
```

Límite y paginación

```
$users = DB::table('usuarios')
        ->skip(10) //Salta 10 elementos
        ->take(5) //Coge 5 elementos
            ->get();
```

Elementos WHERE

```
$users = DB::table('usuarios')
        ->where('column','operator','value')
        ->whereBetween('votos', array(1,100)
        ->whereNotBetween(...)
        ->whereIn('id', array(1,10,40))
        ->whereNotIn(...)
            ->get();
```

Borrados con condición

```
$users = DB::table('usuarios')
        ->where('nombre', '=', Carlos')
        ->delete();
```

Agregaciones útiles

```
$users = DB::table('usuarios')
        ->count() //total elementos
        ->max('price') //valor máximo
        ->min('price') //valor mínimo
        ->avg('price') //media
        ->sum('votos') //total columna
            ->get()
```

Ejemplo operación avanzada Query builder

```
$users = DB::table('usuarios')->delete(); //OJO con ESTO!

$users = DB::table('usuarios')->truncate();
```

Raw Expressions

```
$users = DB::table('usuarios')
            ->select(DB::raw('count(*) as user_count, status'));
```

Documentación completa de Query Builder

http://laravel.com/docs/5.0/queries

Relaciones entre tablas con Eloquent

Mapeo y Relaciones entre tablas

One to One · Uno a uno

One to Many · Uno a muchos

Many to Many · Muchos a muchos

Algunas convenciones iniciales

- La relación entre ids de tabla deberá ser unsigned
- La tabla pivote deberá tener un nameclass_id y un id auto incremental que identifica cada fila
- Una vez definido todo Eloquent obtendrá la relaciones al hacer las queries
- Podemos ajustar el comportamiento a nuestro gusto.

Estado de tablas

- Users
  - id (auto-incremental)
  - name
  - email (unique)
- Phones
  - id (auto-incremental)
  - user_id (unsigned)
  - phone (unique)

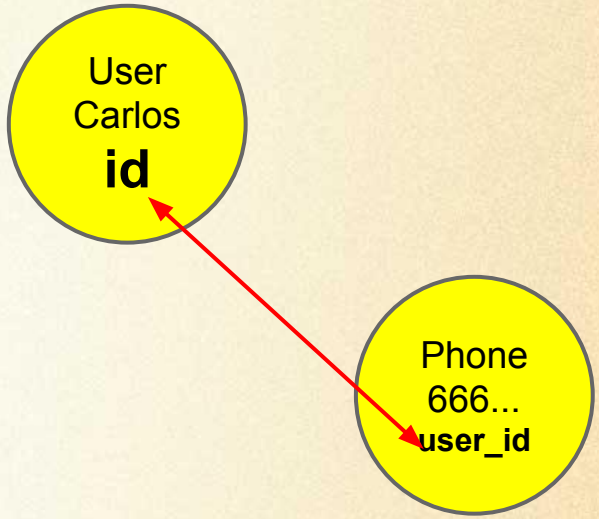
Uno a uno

Un usuario puede tener un telefono y un telefono un usuario.

```
class User extends Eloquent {

    public function phone() { return $this->hasOne('App\Phone');  }

}

class Phone extends Eloquent {

    public function user() { $this->belongsTo('App\User','user_id','id'); }

}
```

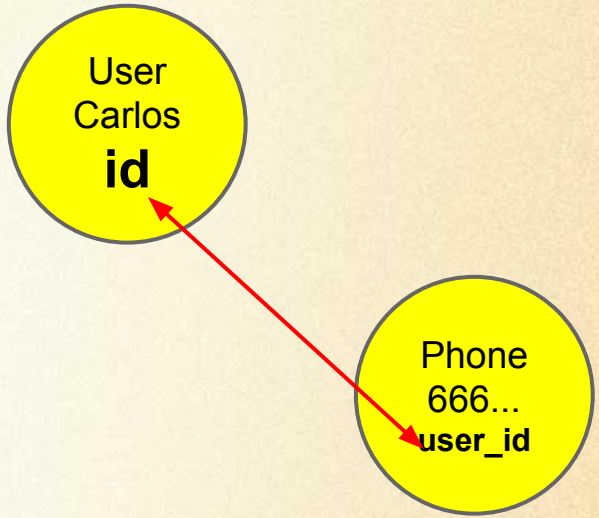
Uno a uno · Creación de relaciones

```
$user = new User;
$user->name = 'Carlos';
$user->email = 'mimm@m.com';
$user->save();

$phone = new Phone;
$phone->phone = '666666666';
//echo $user->id; //id del elemento
$phone->user()->associate($user); // otra forma $phone->user_id = $user->id;
$phone->save();
```



Uno a uno · Obtener datos relación

```
$user = User::find(1);
$user->phone;

$user = User::with('Phone')->get();

//También funciona a la inversa
$phone = Phone::find(1);
$phone->user;

$phone = Phone::with('User')->get();
```



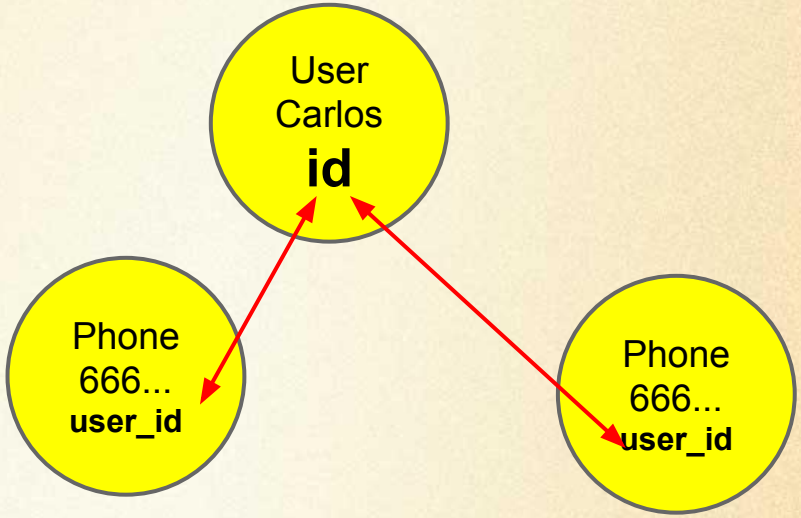
Uno a muchos

Un usuario puede tener un telefono y un telefono un usuario.

```
class User extends Eloquent {

    public function phone() { return $this->hasMany('App\Phone');

}

class Phone extends Eloquent {

    public function $this->belongsTo('App\User','user_id','id');

}
```

Uno a muchos · Obtener datos relación

```
$user = User::find(1);
$user->phone;

$user = User::with('Phone')->get();

//También funciona a la inversa
$phone = Phone::find(1);
$phone->user;

$phone = Phone::with('User')->get();
```



Búsqueda avanzada con filtros

Obtener el telefono pasado como valor, coger el primer resultado y obtener su usuario

```
$respuesta = Phone::where('phone', '=', '666…….')->get();
```

Búsqueda avanzada con filtros y actualización

Obtener el telefono pasado como valor, coger el primer resultado y cambiarlo

```
$respuesta = Phone::where('phone', '=', '666…….')->first();

$respuesta->phone = '77777….';
$respuesta->save();
```

Muchos a muchos

Un usuario puede tener un telefono y un telefono un usuario.

```
class User extends Eloquent {
    public function role() { return $this->belongsToMany('App\Role'); }
}
class Rol extends Eloquent {
    public function user { $this->belongsToMany('App\User','table_name','user_id','id'); }
}
```

*No debemos crear el modelo para la tabla intermedia, solamente creamos la tabla

Documentación oficial de Relaciones

http://laravel.com/docs/5.0/eloquent#relationships

Transacciones

Agrupando transacciones

```
DB::transaction(function()
{
    DB::table('users')->update(array('votes' => 1));

    DB::table('posts')->delete();
});
```

Operaciones Manuales

DB::beginTransaction(); Ejecutamos manualmente

DB::rollback(); Revertimos la operación

DB::commit(); Enviamos los cambios de la transacción

Mutators

http://laravel.com/docs/5.1/eloquent-mutators

Serialization

http://laravel.com/docs/5.1/eloquent-serialization

MUCHAS GRACIAS A TODOS!

Podeis seguirme en la redes sociales como @micromante o Carlos Ruiz Ruso

www.micromante.com