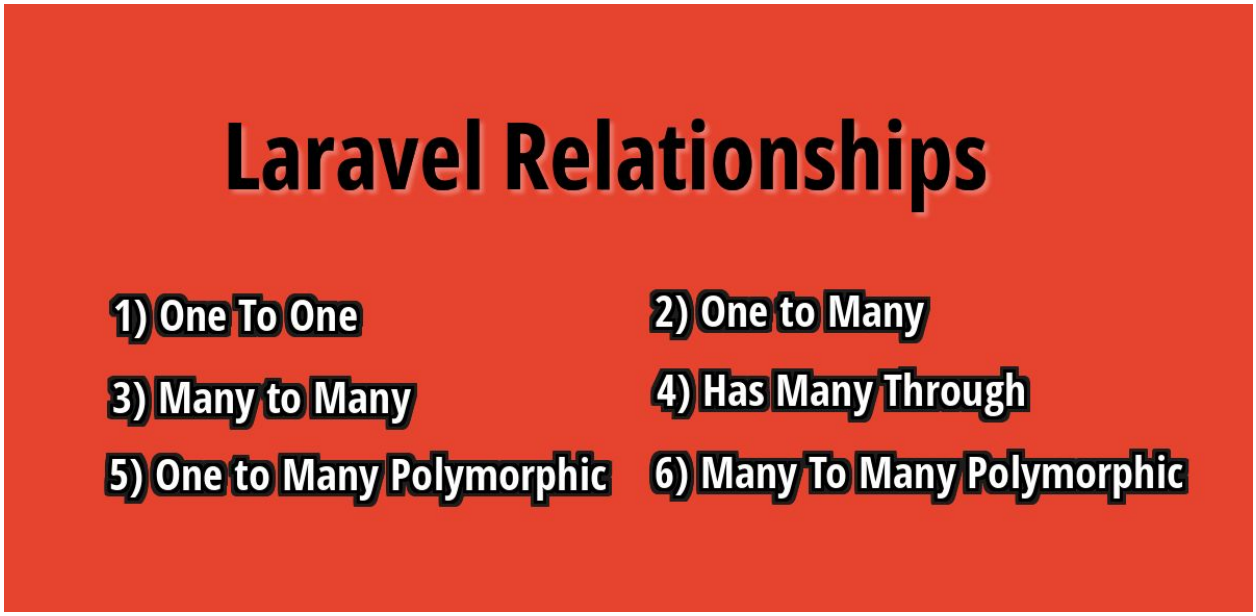
Eloquent Relationships in Laravel Models

—

By Aya Mostafa

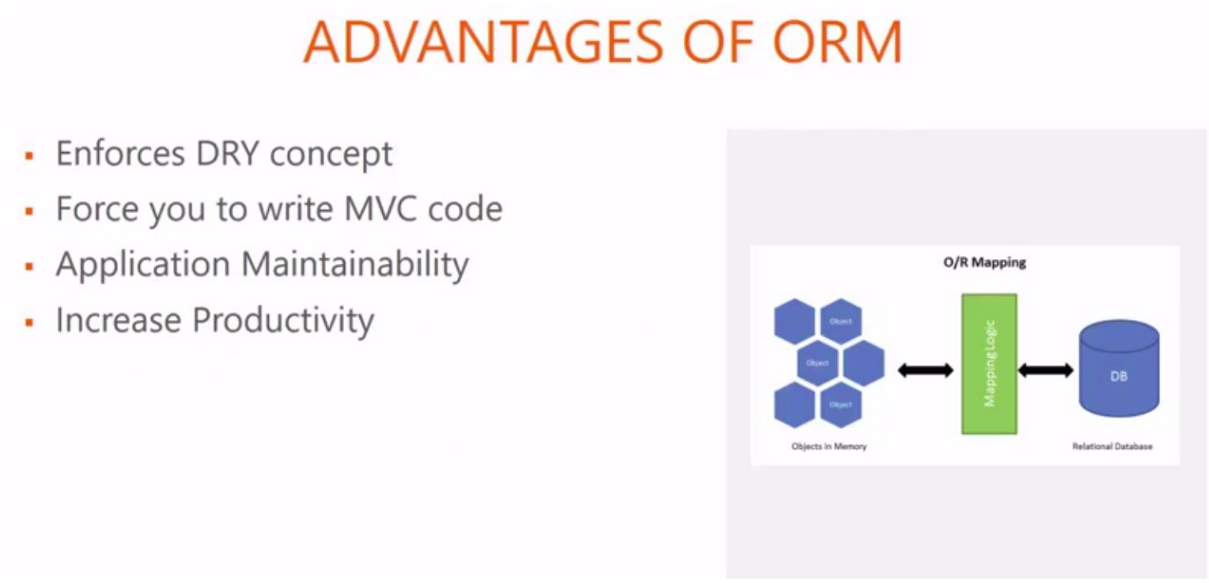


Dear Friends,
Further to my previous topic “database relationships types” which illustrated the database relationships types, in this article we’ll continue talking about eloquent relationships in laravel models.
If you like to revise the previous topic as this topic will depend a lot on the previous one so please check this link:- http://ayamostafa.net/blog/post/database-relationships-types
Also if you're interested in receiving my newsletters to be notified about the new articles, please fill the side subscription form in next link :)
http://ayamostafa.net/blog/web

INTRODUCTION

Laravel provides a great approach for implementing crud operations of tables and relationships between them called “Eloquent ORM”.

Eloquent ORM (Object Relational Mapper) is a code library that automates the transfer of data stored in relational databases tables into objects. Each table has a corresponding "Model" which is used to interact with that table.it makes managing and working with relationships easy, and supports several different types of relationships.

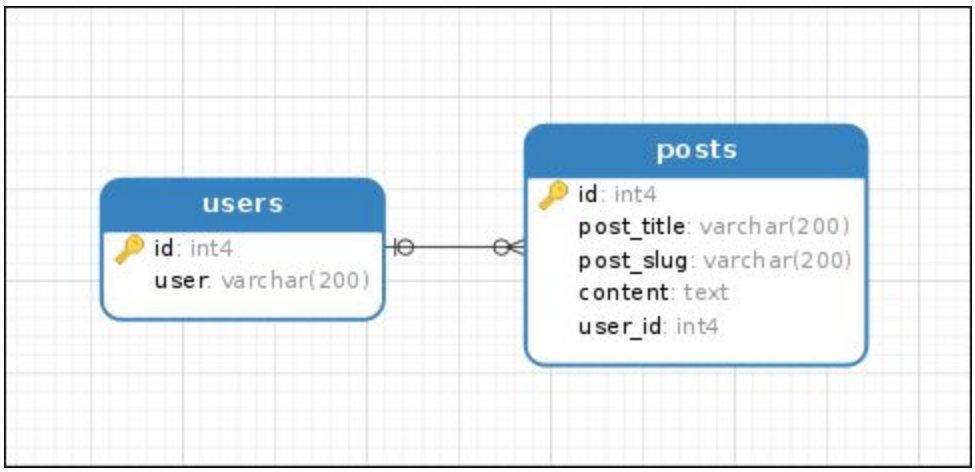


In this article, we’ll start demonstrating the first 3 types of Laravel Eloquent Relationships:-

1. **hasMany (one to many)**
2. **belongsTo (inverse one to many - many to one)**
3. **belongsToMany (many to many)**
4. **hasOne if table with no foreign key (one to one)**
5. **belongsTo if table with foreign key (Inverse one to one)**

let’s talk about each one with examples in more detail, so be ready :)

1. hasMany (One To Many)



In this example, each user **-has many-** posts and each post is created by only one user. *Notice, in the diagram the fork notation is on posts table which means the foreign key of the relation will be in posts table or in other words, posts represent the **many** in this relationship.
To implement this relationship in Laravel, **In User Model:-**

```
public function posts(){
        return $this->hasMany('App\Post');
        //or if you like to define the foreign key and local key
        return $this->hasMany('App\Post','user_id','id');
}
```

Once the relationship has been defined, we can access the collection of posts that the user has been created like that:-

```
$posts = App\User::findOrFail(1)->posts; // this will retrieve all posts that user 1 has created
```

2. belongsTo (Many To One) or (Inverse One To Many)

In the previous example, we said that each user **-has Many-** posts and every post **-belongs To-** only one user. So how can we define the inverse of the previous relation?

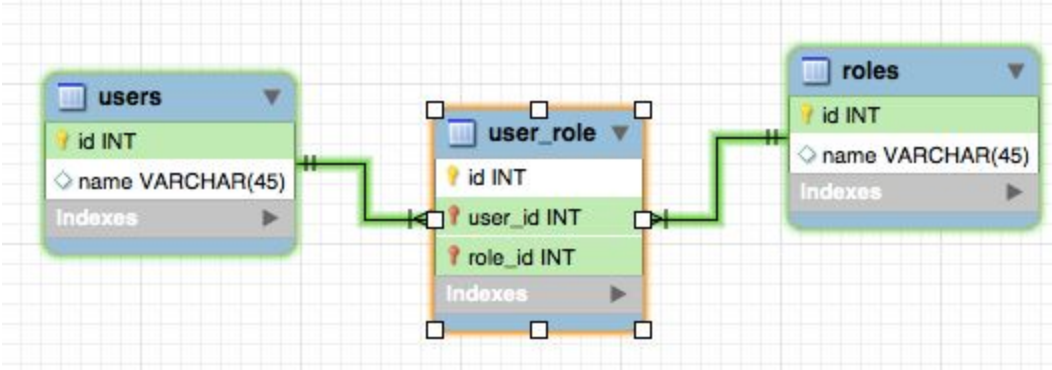
In Post Model:-

```
public function user()
 {
        return $this->belongsTo('App\User');
        //or if you like to define the foreign key and parent key
        return $this->belongsTo('App\User','user_id','id');  // note that id is the
        primary key of parent model(User)
 }
```

Once the relationship has been defined, we can retrieve the User has been created the post like that:-

```
$user = App\Post::find(1)->user;   // this will retrieve the user that created post 1
```

3. belongsToMany (Many To Many)



In this example, users can has one or many roles and role also can has one or many users so this is **(many to many)** relationship, in this type of relationship we have to create a third table that is called **"join table"** or **"pivot table".** This table will relate users and roles with each other. And inside this table will put the user id and role id as foreign keys.
We can define this relation like that(users **belongsToMany** roles and roles **belongsToMany** users)

To implement this relationship in Laravel, **In User Model:-**

```
public function roles(){
    return $this->belongsToMany('App\Role');
      //or if you like to determine the table name of the relationship's joining table,
    Eloquent  will join the two related model names in alphabetical order. However,
    you are free to override this convention. You may do so by passing a second
    argument to the belongsToMany method
      return $this->belongsToMany('App\Role', 'user_role');
```

```
/// or In addition to customizing the name of the joining table, you may also
customize the column names of the keys on the table by passing additional
arguments to the belongsToMany method. The third argument is the foreign key
name of the model on which you are defining the relationship-User-, while the
fourth argument is the foreign key name of the model that you are joining
to-Role-
belongsToMany(
   'ModelName',
   'pivot_table',
   'foreign_key_defined_model',
   'foreign_key_joined_model'
);
Like that:
  return $this->belongsToMany('App\Role', 'user_role', 'user_id', 'role_id');
}
```

Once the relationship has been defined, we can retrieve all roles collections belongs to user 1 like that:-

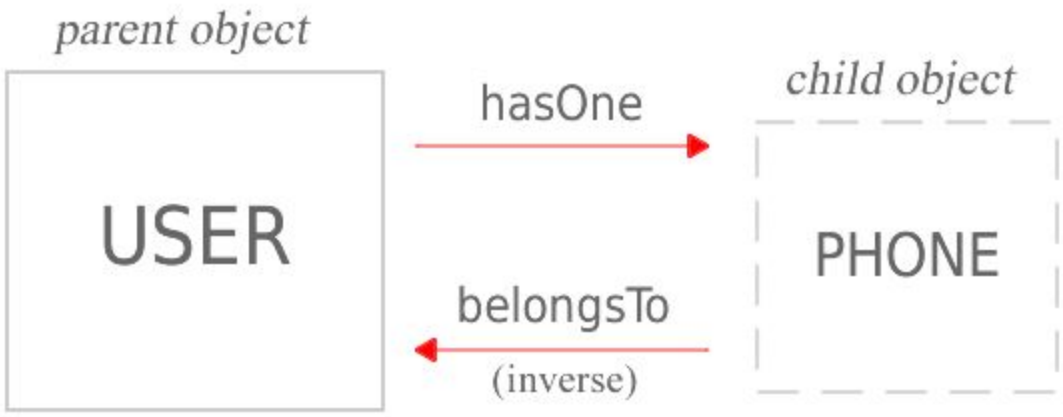
```
$roles= App\User::find(1)->roles
```

*Notice that the inverse of many to many is the same for the second model but consider foreign key names for the second model, as per the previous example the inverse of many to many will be like that :-

In Role Model:-

```
public function users(){
   return $this->belongsToMany('App\User, ‘‘user_role’’, 'role_id',’user_id',);
}
```

4. hasOne (One To One)



In this example, each user **-has one-** phone and each phone **-belongs to-** only one user. This is **(one to one)** relationship so to implement this relationship in Laravel, **In User Model:**

```
public function phone()
{
   return $this->hasOne('App\Phone');
 // notice that  Eloquent determines the foreign key of the relationship based on the model name. In this case, the Phone model is automatically assumed to have a user_id foreign key. If you wish to override this convention, you may pass a second argument to the hasOne method:
return $this->hasOne('App\Phone', user_id'); // foreign key here is user_id

//Additionally, Eloquent assumes that the foreign key should have a value matching the id (or the custom $primaryKey) column of the parent. In other words, Eloquent will look for the value of the user's id column in the user_id column of the Phone record. If you would like the relationship to use a value other than id, you may pass a third argument to the hasOne method specifying your custom key:

return $this->hasOne('App\Phone', 'foreign_key', 'local_key');
}
```

Once the relationship is defined, we can retrieve the phone of any user like that:

```
$phone = User::find(1)->phone; // this will retrieve phone data that user 1 has
```

5. belongsTo (Inverse One To One)

So, we can access the phone model from our User. Now, let's define the inverse relationship on the phone model that will let us access the user that owns the phone.

In Phone Model

```
public function user()
  {
     return $this->belongsTo('App\User');
        //or if you like to define the foreign key and parent key
 return $this->belongsTo('App\User', 'user_id','id');

  }
```

Once the relationship is defined, we can retrieve the user from any phone like that:

```
$user = Phone::find(1)->user;  // this will retrieve user data that phone 1 belongsto
```

Conclusion

By the end of this article, hope you enjoyed reading it and if you have any participation about this topic, please share it with us :)
In the next article we'll discuss the rest of types isa.