

## Practical 3: Working with Databases and Object-Relational Mapping (Part II)

In this lab, you will use previously developed web application to learn the extensions of concept of database and object-relational mapping (ORM); mass assignment and relationships.

### 1 Mass Assignment

In the previous lab, we manually defined each data collected from a form to be mapped to each object within a model and used “new User” together with “save” method for creating or inserting data into database table. In order to explore the mass assignment feature of Laravel, we will use “create” method to insert data. Let’s do some exercises to get into the exploration.

**Exercise 1:** As shown in Figure 1, create:

1. Another two columns for users table in the web application’s database; “**password**” with **VARCHAR** 40-42 length and “**is\_admin**” with **BOOLEAN** value.
2. A simple user signup form of blade template view and a route to the form
3. A route to bring the data from form to a controller and a controller to process the data passed from the signup form

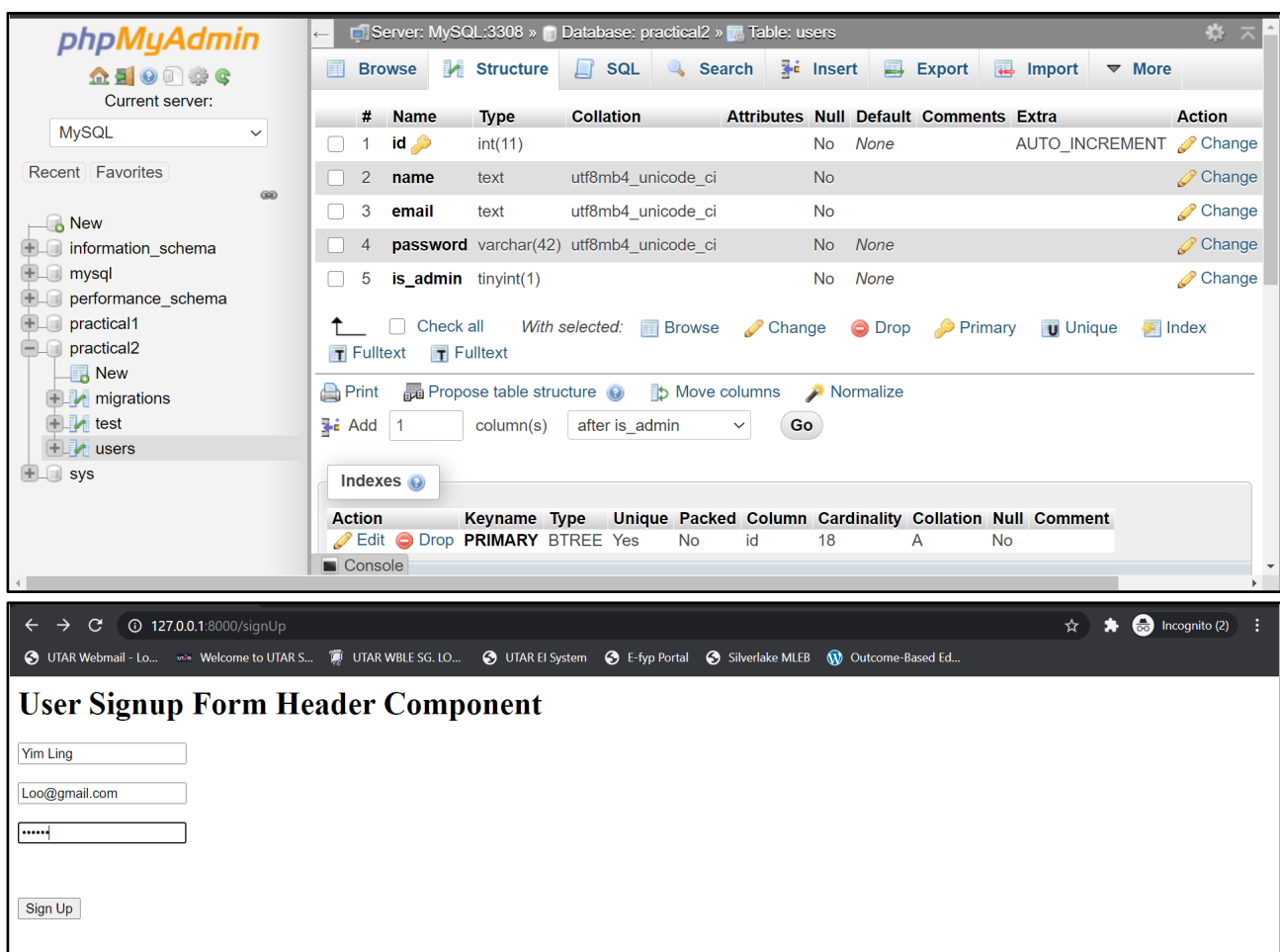


Figure 1: Web Application with User Sign In.

Before using the “**create**” method, fillable or guarded property need to be specified in model class. These properties are required because all Eloquent models are protected against mass assignment vulnerabilities by default.

A mass assignment vulnerability occurs when a user passes an unexpected HTTP request field and that field changes a column in database. For example, a malicious user might send an “**is\_admin**” parameter through an HTTP request, which is then passed to model's create method, allowing the user to escalate themselves to an administrator.

Thus, a protection (default value) is needed to protect sensitive data from being intentionally changed as shown in Figure 2.

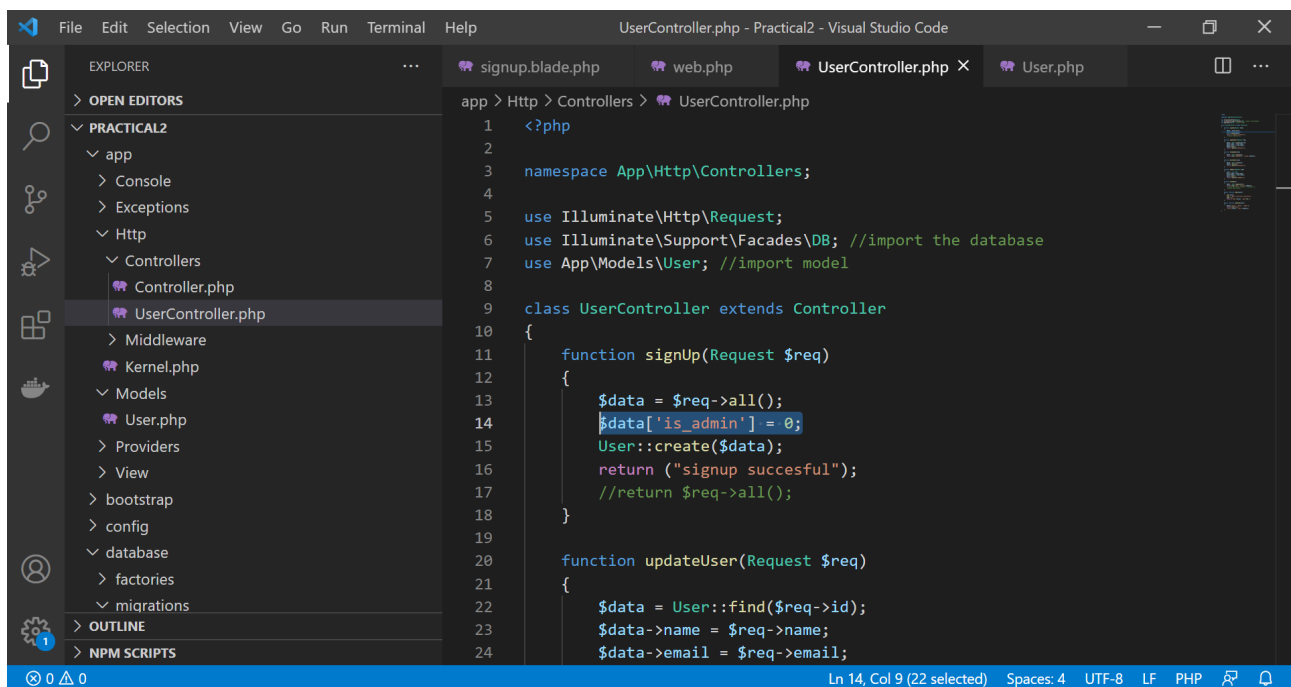


Figure 2: Default values to protect against malicious users.

Mass assignment will only be used when “**create**”, “**fill**” and “**update**” model methods are called. By default, the best practice will be manually assigning model’s object to the different data passed from HTTP request.

## 2 Relationship Assignment

Database tables are often related to one another. For example, a blog post may have many comments or an order could be related to the user who placed it. Eloquent makes managing and working with these relationships easy, and supports a variety of common relationships. The following session explore the assignment of different relationships that can be declared in models, and use of queries to fetch data inline with the relationship. Let’s do some exercises to get into the exploration.

**Exercise 2:** As shown in Figure 3, create:

1. A table named “Companies” with “id”, “name” and “user\_id”.
2. A model for Companies table.
3. A controller to fetch model and show output.
4. A route to the controller.

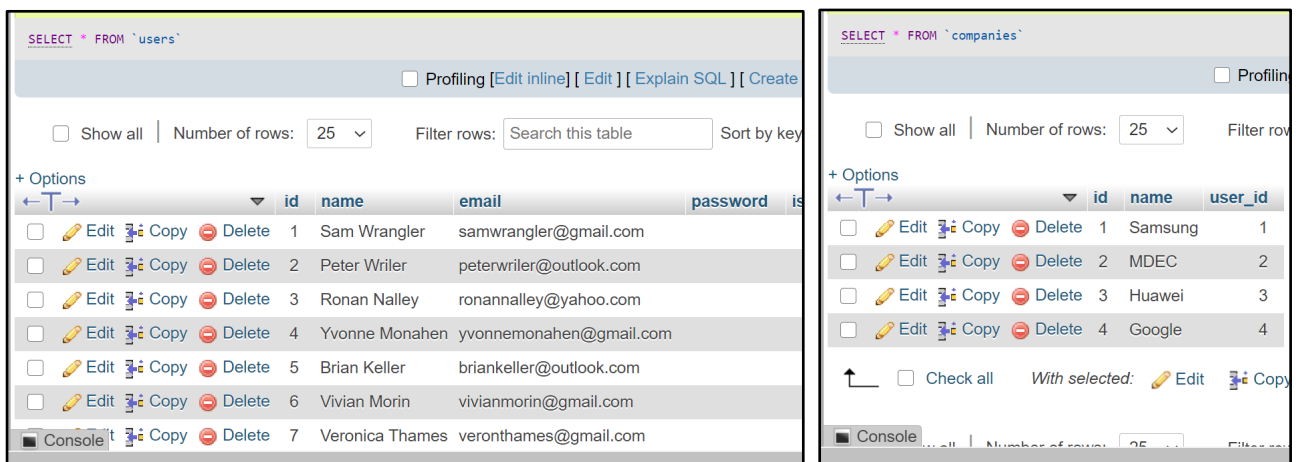
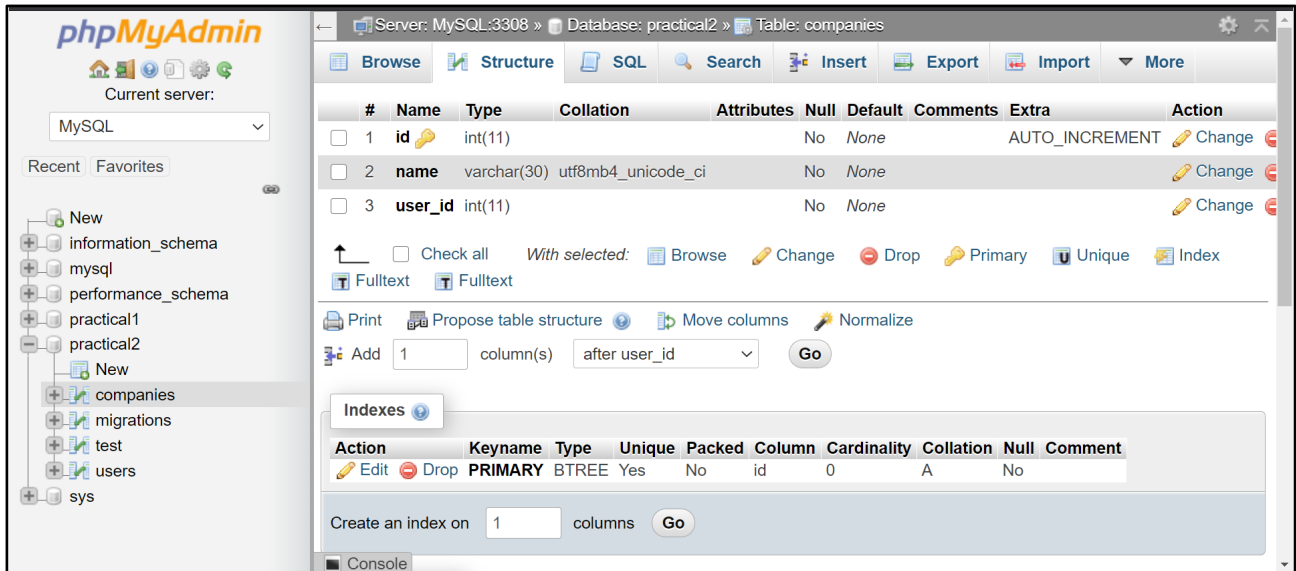
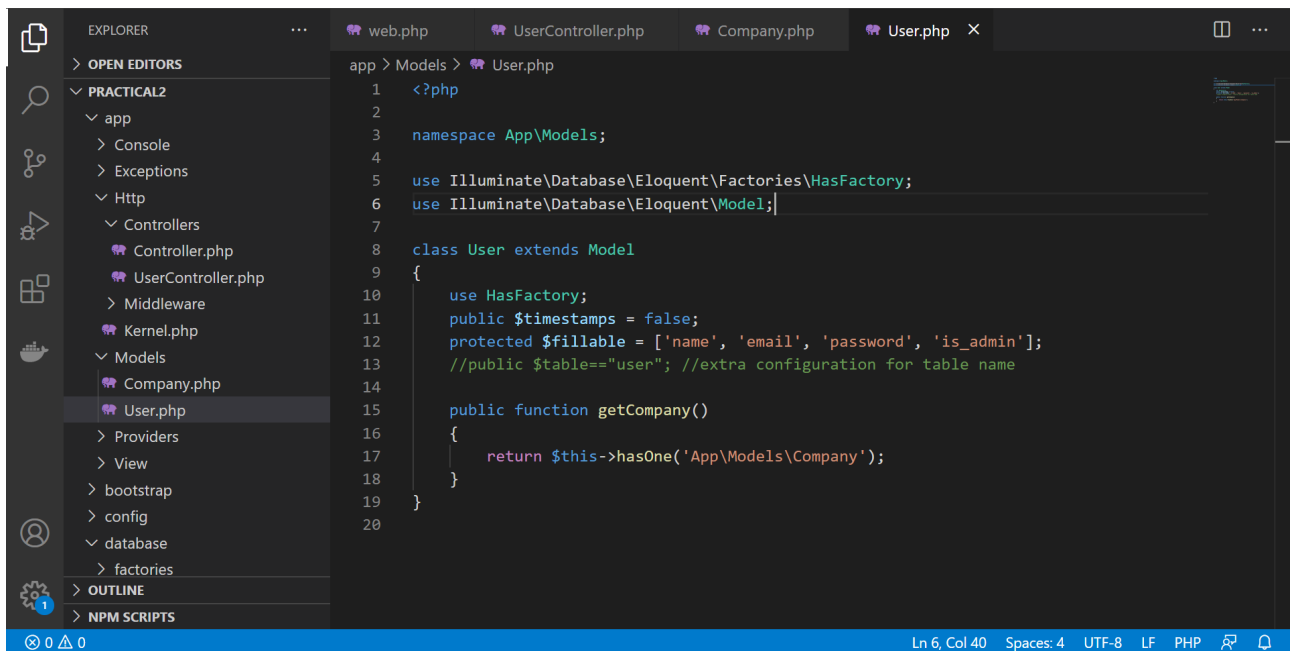


Figure 3: Web Application with relational database

## 2.1 One-To-One Relationship

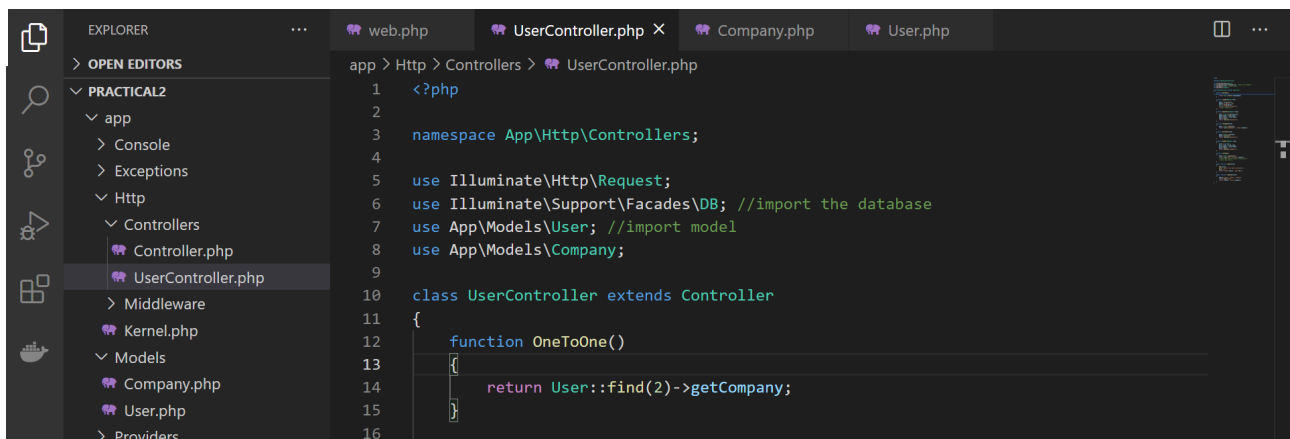
In order to declare that a user has only one company, define the one-to-one relationship in user model as shown in Figure 4.



```
1  <?php
2
3  namespace App\Models;
4
5  use Illuminate\Database\Eloquent\Factories\HasFactory;
6  use Illuminate\Database\Eloquent\Model;
7
8  class User extends Model
9  {
10     use HasFactory;
11     public $timestamps = false;
12     protected $fillable = ['name', 'email', 'password', 'is_admin'];
13     //public $table="user"; //extra configuration for table name
14
15     public function getCompany()
16     {
17         return $this->hasOne('App\Models\Company');
18     }
19 }
20
```

Figure 4: One-to-One Relationship for User-Company models.

In order to fetch the company that is linked to the user, the controller could be defined as shown Figure 5.



```
1  <?php
2
3  namespace App\Http\Controllers;
4
5  use Illuminate\Http\Request;
6  use Illuminate\Support\Facades\DB; //import the database
7  use App\Models\User; //import model
8  use App\Models\Company;
9
10 class UserController extends Controller
11 {
12     function OneToOne()
13     {
14         return User::find(2)->getCompany();
15     }
16 }
```

Figure 5: Getting data linked between relational table.

## 2.2 One-To-Many Relationship

As a pre-requisite to exploration of one-to-many relationship between database tables, add in data into “companies” table in order to have a few companies relating to one user. Then, using the similar functions in model and controller, define the one-to-many relationship in user model and fetching of user that has many companies respectively. The result should be as shown in Figure 6.

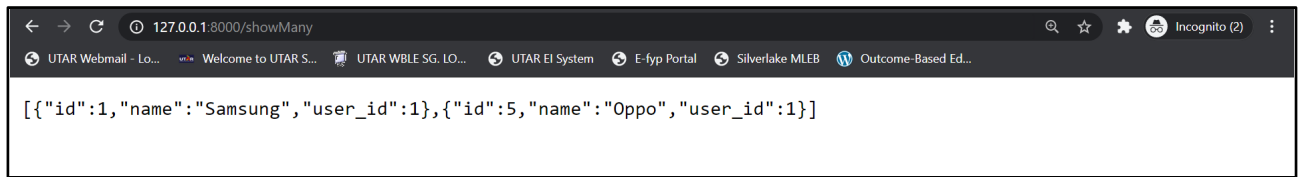


Figure 6: Fetching of data from one-to-many relationship user-company models.