## Working with Database in Laravel

### Preparation

1. Make sure Wamp is started and no two instances of MySQL is started. Create a database in PhpMyAdmin. The database name can be the app name. Change the database name in the .env file. Check if all information is correct in .env and database.php.

|  |
| --- |
| // .env file  DB\_CONNECTION=mysql  DB\_HOST=127.0.0.1  DB\_PORT=3306  DB\_DATABASE=practical2  DB\_USERNAME=root  DB\_PASSWORD= |
| // config/database.php  'mysql' => [              'driver' => 'mysql',              'url' => env('DATABASE\_URL'),              'host' => env('DB\_HOST', '127.0.0.1'),              'port' => env('DB\_PORT', '3306'),              'database' => env('DB\_DATABASE', 'forge'),              'username' => env('DB\_USERNAME', 'forge'),              'password' => env('DB\_PASSWORD', ''),              'unix\_socket' => env('DB\_SOCKET', ''),              'charset' => 'utf8mb4',              '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'),              ]) : [],          ], |

1. Directly interacting with database without using model and utilizing ORM.

|  |
| --- |
| public function index() {      return DB::select('select \* from users');  } |

1. To use ORM, recommended way, you need a model and a table in the database. **The table name must be the plural of the model’s name.** For example, the plural of user is users. To create a model, use this command. **Ensure the columns in the database table are the same as the model’s attributes.** You can add test data manually in PhpMyAdmin.

|  |
| --- |
| php artisan make:model User |

### CRUD (Read)

1. CRUD(R): To read all from table using ORM, first, use the “Model::all” static function from the model. The “Model::all” static function is there by default in any model.

|  |
| --- |
| public function loadAllData() {      return User::all();  } |

1. CRUD(R): To paginate a specific number of data from table using ORM, use Model::paginate(5).

|  |
| --- |
| public function loadData() {    $users = User::paginate(5);      return view('/users', ['users'=>$users]);  } |

1. CRUD(R): An example of how to display paginated data in a table.

|  |
| --- |
| <table border="1">        <thead>              <tr>                  <th>id</th>                  <th>name</th>                  <th>email</th>              </tr>          </thead>          <tbody>              @foreach ($users as $user)                  <tr>                      <td>{{ $user['id'] }}</td>                      <td>{{ $user['name'] }}</td>                      <td>{{ $user['email'] }}</td>                  </tr>              @endforeach          </tbody>  </table>  <span>       {{ $users->links() }}  </span> |

### CRUD(Create)

1. For the create operation, first, create a create page that allows the user to enter the data of the new row that can be accessed through the /addUser route. The action is the same route, but using the post method. Remember to use @csrf.

|  |
| --- |
| <form action="/addUser" method="post">      @csrf      <input type="text" name="id" placeholder="Enter id"><br><br>      <input type="text" name="name" placeholder="Enter name"><br><br>      <input type="text" name="email" placeholder="Enter email"><br><br>      <button type="submit">Add User</button>  </form> |

Add the route with two different methods in the web.php.

|  |
| --- |
| Route::view('/addUser', 'addUser'); |

|  |
| --- |
| Route::post('/addUser', [UsersController::class, "addUser"]); |

1. Then, create a function in controller to process the request and saves the data when the user press the button and redirects to display all data page. The important thing is to have these two things: addUser.blade.php, and addUser function in controller.

|  |
| --- |
| public function addUser(Request $request)  {      $user = new User();      $user->id = $request->id;      $user->name = $request->name;      $user->email = $request->email;      $user->save();      return redirect('/users');  } |

### CRUD Update

1. Setup update user form page, controller function that shows the update form page with user where the id is from URL, controller function that updates the user.

|  |
| --- |
| // 1. updateUser.blade.php  // when the update user button is pressed, the action goes to /updateUser route and use post method is executed.  <form action="/updateUser" method="post">      @csrf      <input type="hidden" name="id" value="{{$user['id']}}">      <input type="text" name="name" value="{{$user['name']}}"> <br><br>      <input type="text" name="email" value="{{$user['email']}}"> <br><br>      <button type="submit">Update User</button>  </form> |

|  |
| --- |
| // 2. Controller function that shows the updateUser view.  // It should take in an id from the route and displays the user info in the view.  public function showUpdatePage($id)  {      $user = User::find($id);      return view('updateUser', ['user'=>$user]);  } |

|  |
| --- |
| // 3. Controller function that updates the user.  // Must take in request in the form of an user and SPECIFY Request as the data type, to perform the update.  public function updateUser(Request $request) {      $userToUpdate = User::find($request->id);      $userToUpdate->name = $request->name;      $userToUpdate->email = $request->email;      $userToUpdate->save();      return redirect('/users');  } |

1. Setup the route for these the update functions in the controller.

|  |
| --- |
| Route::get('/updateUser/{id}', [UsersController::class, "showUpdatePage"]);  Route::post('/updateUser', [UsersController::class, "updateUser"]); |

1. Example of table

|  |
| --- |
| <table border="1">          <thead>              <tr>                  <th>id</th>                  <th>name</th>                  <th>email</th>                  <th>Operation</th>                  <th>Operation</th>              </tr>          </thead>          <tbody>              @foreach ($users as $user)                  <tr>                      <td>{{ $user['id'] }}</td>                      <td>{{ $user['name'] }}</td>                      <td>{{ $user['email'] }}</td>                      <td><a href="/updateUser/{{$user['id']}}">Update</a></td>                      <td><a href="/deleteUser/{{$user['id']}}">Delete</a></td>                  </tr>              @endforeach          </tbody>      </table> |

### CRUD Delete

|  |
| --- |
| public function deleteUser($id) {      $userToDelete = User::find($id);      $userToDelete->delete();      return redirect('/users');  } |

1. Setup the delete function in controller. It should take in an id and deletes the user corresponding to that id.
2. Specify the route of the function.

|  |
| --- |
| Route::get('/deleteUser/{id}', [UsersController::class, "deleteUser"]); |

### Migrations

1. Migrations are test database tables that developers should work with instead of touching the real database. First step of creating migrations is to create the migration class that specifies the database table and its attributes.

The command to create a migration class is as follows. A file like 2024\_02\_24\_021632\_create\_test\_table.php will be created in database/migrations.

|  |
| --- |
| php artisan make:migration create\_test\_table |

Edit the migration file to the table name that you want, and then add the add the table attributes in the schema create function.

|  |
| --- |
| public function up()      {          Schema::create('testMigrateTable', function (Blueprint $table) {              $table->id('id');              $table->string('name');              $table->string('email');          });  }      public function down()      {          Schema::dropIfExists('testMigrateTable');      } |

1. Then, you need to execute the migration class to create the actual table in the database.

The command to execute the migration class is as follows. The example of migration\_name is 2024\_02\_24\_021632\_create\_test\_table.php.

|  |
| --- |
| php artisan migrate --path=/database/migrations/migration\_name.php |

1. Finally, you need to seed the database with data. To do so, create a class that specifies the data to seed it.

The command to create a seeder class is as follows:

|  |
| --- |
| php artisan make:seeder TestSeeder |

A TestSeeder class will be created located in database/seeders. Edit the run function to specify how you want to seed the data.

|  |
| --- |
| public function run()      {          for($i=0; $i<10; $i++)          {              DB::table('testMigrateTable')->insert([                  'id' => $i+1,                  'name' => Str::random(10),                  'email' => Str::random(10).'gmail.com'              ]);          }      } |

To execute the TestSeeder class run function that seeds the migration database, execute the command as follows.

|  |
| --- |
| php artisan db:seed --class=TestSeeder |

### Mass Assignments

1. Mass Assignments is a quick way to map a request to its model. It is convenient but is susceptible to mass assignments attack if not guarded properly. To start with mass assignment, add fillable or guarded property in the model.

|  |
| --- |
| protected $fillable = [      'name',      'email',      'password'  ];  Or  protected $guarded = [      'is\_admin',  ]; |

In this example, the is\_admin is not fillable. The field has to be manually assigned in the controller. If error is\_admin default value does not exist error exists, add default is null in the sql table.

|  |
| --- |
| public function signup(Request $request) {          $data = $request->all();          $data['is\_admin'] = 0;          User::create($data);          return ("Signup successful.");      } |

### Relationships (One-to-one, One-to-many)

11. Define the relationship in the model. Example, one user has one company.

|  |
| --- |
| public function getCompany() {      return $this->hasOne('App\Models\Company');  } |

public function getManyCompanies() {

    return $this->hasMany('App\Models\Company');

}

}

12. To get the company of a user in controller,

|  |
| --- |
| function OneToOne() {      return User::find(1)->getCompany;  } |

function OneToMany() {

    return User::find(1)->getManyCompanies;

}