**PHP-Laravel**

**Forms and Gathering Input – Industry**

**1. Explain ORM**

Eloquent is an object relational mapper (ORM) that is included by default within the Laravel framework. An ORM is software that facilitates handling database records by representing data as objects, working as a layer of abstraction on top of the database engine used to store an application's data.

To get started, you’ll need to set up the [Landing Laravel](https://github.com/do-community/landing-laravel) demo project. There are two ways in which you can get the demo application code ready to use with this series:

1. The first way is to follow the guide on [How To Build a Links Landing Page in Laravel](https://www.digitalocean.com/community/tutorial_series/how-to-build-a-links-landing-page-in-php-with-laravel-and-docker-compose). That series explains how to build the demo application that you’ll use as base for the current series from scratch. If you choose this option, you can move on to the first tutorial in this series [How To Create a One-To-Many Relationship in Laravel Eloquent](https://www.digitalocean.com/community/tutorials/how-to-create-a-one-to-many-relationship-in-laravel-eloquent).
2. The second option is to download the complete demo application code and use it as the base that you will build on in this series. On the application [releases page](https://github.com/do-community/landing-laravel/releases), you’ll find separate application versions for each tutorial in the series. You can choose to start from the first tutorial by downloading version 0.1.1, or you can choose to download one of the elo-tutorial releases that are paired with each individual tutorial in the series.

**2. Do Curd using Eloquent Query.**

In Laravel, Eloquent is the ORM (Object-Relational Mapping) included with the framework. To perform CRUD (Create, Read, Update, Delete) operations using Eloquent, you'll typically work with Eloquent models.

**1. Create (Insert):**

// Creating a new record

$curd = new Curd;

$curd->column1 = 'value1';

$curd->column2 = 'value2';

$curd->save();

Curd::create([

'column1' => 'value1',

'column2' => 'value2',

]);

### 3. Update:

$curd = Curd::find(1);

$curd->column1 = 'new value';

$curd->save();

Curd::where('column1', 'old value')->update(['column1' => 'new value']);

### 4. Delete:

// Delete a specific record

$curd = Curd::find(1);

$curd->delete();

// Delete a record by its primary key

Curd::destroy(1);

// Delete multiple records by primary keys

Curd::destroy([1, 2, 3]);

// Delete records based on a query

Curd::where('column1', 'value1')->delete();

**3. Explain - Eloquent Relationships**

In Laravel, Eloquent Relationships are a way to define associations between Eloquent models. These relationships make it easier to perform common database operations, such as retrieving related records, creating and updating relationships, and more. Laravel supports several types of relationships, including:

**1**.**One to One Relationship:**

* + In a one-to-one relationship, each record in the table can be associated with at most one record in another table, and vice versa.
  + Example: A user has one profile.

// User Model

public function profile()

{

return $this->hasOne(Profile::class);

}

// Profile Model

public function user()

{

return $this->belongsTo(User::class);

}

**2**.**One to Many Relationship:**

* In a one-to-many relationship, a record in one table can be associated with multiple records in another table, but each record in the second table is associated with only one record in the first table.
* Example: A user has many posts.

// User Model

public function posts()

{

return $this->hasMany(Post::class);

}

// Post Model

public function user()

{

return $this->belongsTo(User::class);

}

**3. Many to Many Relationship:**

* In a many-to-many relationship, a record in one table can be associated with multiple records in another table, and vice versa.
* Example: Users belong to many roles, and roles can have many users

// User Model

public function roles()

{

return $this->belongsToMany(Role::class);

}

// Role Model

public function users()

{

return $this->belongsToMany(User::class);

}

**4. What is Eager Loading and lazy loading?**

**1. Eager Loading:**

* Eager loading is a mechanism to load specified relationships along with the main model's query. It helps to reduce the number of database queries by fetching the related data in advance.
* When you have a model with relationships (e.g., one-to-one, one-to-many, many-to-many), you can use eager loading to fetch the related data in a more efficient way.
* The **with** method is used for eager loading. For example:

**$posts = Post::with('comments')->get();**

In this example, the **with('comments')** method fetches the comments associated with each post in the result set, resulting in a more optimized query.

**2. Lazy Loading:**

* + Lazy loading, on the other hand, defers the loading of related models until you actually access them. This means that related models are only loaded from the database when you attempt to access a property or method that requires them.
  + Laravel uses dynamic properties to achieve lazy loading. For example:

**$post = Post::find(1);**

**$comments = $post->comments; // Lazy loading, retrieves comments when accessed**