* **Explain ORM**

=> The Eloquent ORM included with Laravel provides a beautiful, simple ActiveRecord implementation for working with your database. Each database table has a corresponding "Model" which is used to interact with that table.

* **Explain - Eloquent Relationships**

=> A **relationship** means that you have two or more tables with records that are related to each other. This 'connection' that allows you to figure out which records belong to each other, is called a relationship.

There are multiple types of Eloquent relationships. Which type you have generally depends on the fact whether you have one or (potentially) multiple items of both the first and the second table.

Eloquent makes managing and working with these relationships easy, and supports a variety of common relationships:

* One To One
* One To Many
* Many To Many
* Has One Through
* Has Many Through
* **One-To-One Eloquent relationship**

A one-to-one relationship is a very basic type of database relationship. For example, a **User** model might be associated with one **Phone** model. To define this relationship, we will place a phone method on the User model. The phone method should call the **hasOne** method and return its result. The hasOne method is available to your model via the model's Illuminate\Database\Eloquent\Model base class:

<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;

use Illuminate\Database\Eloquent\Relations\HasOne;

class User extends Model

{

public function phone()

{

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

}

}

The first argument passed to the hasOne method is the name of the related model class. Once the relationship is defined, we may retrieve the related record using Eloquent's dynamic properties. Dynamic properties allow you to access relationship methods as if they were properties defined on the model:

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

Eloquent determines the foreign key of the relationship based on the parent model name. In this case, the Phone model is automatically assumed to have a user\_id foreign key.

- **Defining The Inverse Of The Relationship**

Define a relationship on the Phone model that will let us access the user that owns the phone. **We can define the inverse of a hasOne relationship using the belongsTo method**:

<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;

use Illuminate\Database\Eloquent\Relations\BelongsTo;

class Phone extends Model

{

public function user()

{

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

}

}

When invoking the user method, Eloquent will attempt to find a User model that has an id which matches the user\_id column on the Phone model.

* **One To Many Eloquent relationship**

A one-to-many relationship is used to define relationships where a single model is the parent to one or more child models. For example, a blog post may have an infinite number of comments. Like all other Eloquent relationships, one-to-many relationships are defined by defining a method on your Eloquent model:

<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;

use Illuminate\Database\Eloquent\Relations\HasMany;

class Post extends Model

{

public function comments()

{

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

}

}

Once the relationship method has been defined, we can access the collection of related comments by accessing the comments property. we can access relationship methods as if they were defined as properties on the model:

use App\Models\Post;

$comments = Post::find(1)->comments;

foreach ($comments as $comment) {

// ...

}

**- One To Many (Inverse) / Belongs To**

Define a relationship to allow a comment to access its parent post. To define the inverse of a hasMany relationship, **define a relationship method on the child model which calls the belongsTo method**:

<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;

use Illuminate\Database\Eloquent\Relations\BelongsTo;

class Comment extends Model

{

public function post()

{

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

}

}

Once the relationship has been defined, we can retrieve a comment's parent post by accessing the post.

use App\Models\Comment;

$comment = Comment::find(1);

return $comment->post->title;

In the example above, Eloquent will attempt to find a Post model that has an id which matches the post\_id column on the Comment model.

* **Many To Many Eloquent relationship**

A many-to-many relationship is a user that has many roles and those roles are also shared by other users in the application. For example, a user may be assigned the role of "Author" and "Editor"; however, those roles may also be assigned to other users as well. So, a user has many roles and a role has many users.

* **Has One Through**

The "has-one-through" relationship defines a one-to-one relationship with another model. However, this relationship indicates that the declaring model can be matched with one instance of another model by proceeding through a third model.

* **Has Many Through**

The "has-many-through" relationship provides a convenient way to access distant relations via an intermediate relation. For example, let's assume we are building a deployment platform like Laravel Vapor. A Project model might access many Deployment models through an intermediate Environment model.

* **What is Eager Loading and lazy loading?**
* **Lazy loading**

Lazy loading is a technique where related models are not loaded at the same time as the main model, but are loaded only when they are accessed for the first time. This means that the related models are only loaded from the database as they are needed.

Lazy loading is beneficial when dealing with a large number of related models, as it can help to reduce the amount of memory used and improve the performance of the application. In Laravel, you can use the **belongsTo** or **hasMany** methods on the main model, followed by the **load()** method, to lazy load a related model.

For example, a Post model and a Comment model, where a post can have many comments. To lazy load the comments for a post, you can use the following code:

$post = Post::find(1);

$comments = $post->comments()->get();

In this example, the **comments()** method returns a relationship object, which is not loaded until the get method is called. This allows you to access the comments for a post without loading them from the database until they are actually needed.

Another way to use lazy loading is using the **whenLoaded()** method, which allows you to access a related model only when it has been loaded.

* **Eager Loading**

Eager loading is a technique where related models are loaded at the same time as the main model, using an additional query to the database. This approach can be beneficial when dealing with a small number of related models, as it can help to eliminate the need for additional queries and improve the performance of the application.

In Laravel, you can use the with method on the main model, followed by the name of the related model, to eager load a related model. The **with** method tells Laravel to include the specified related models in the same query as the main model, so that they are all loaded at the same time.

For example, let's say you have the same **Post** model and **Comment**. To eager load the comments for a post, you can use the following code:

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

* **Do Session for Employee Management System.**

=>

<?php

namespace App\Http\Controllers;

use App\Models\employee;

use Illuminate\Http\Request;

use Hash;

use Session;

class employeeController extends Controller

{

public function employeeLoginAuth(Request $request)

{

$username = $request->username;

$data = employee::where('username', $username)->first();

if ($data) {

if (Hash::check($request->password, $data->password)) {

session()->put('employee\_id', $data->id);

session()->put('employee\_name', $data->name);

session()->put('employee\_uname', $data->username);

} else {

return redirect()->back()->with('login\_fail', 'Password not valid.');

}

} else {

return redirect()->back()->with('login\_fail', 'Username not valid.');

}

}

public function employeeLogout()

{

session()->pull('employee\_id');

session()->pull('employee\_name');

session()->pull('employee\_uname');

return redirect('/')->with('logout\_success', 'Logout successfully.');

}

}

?>