1. Explain ORM

ORM, or Object-Relational Mapping, in Laravel refers to the way the framework handles database interactions through an abstraction layer, allowing developers to interact with their database using PHP objects instead of writing raw SQL queries. Laravel's ORM is called Eloquent. Here are the key features and concepts of Eloquent ORM in Laravel:

* Key Concepts of Eloquent ORM

1. Models:

- Each database table has a corresponding "Model" in Laravel, which is used to interact with that table.

- Models are PHP classes that extend `Illuminate\Database\Eloquent\Model`.

- A model typically represents a single table in the database.

2. Conventions:

- Table names: Eloquent assumes the table name is the plural form of the model name. For example, a `User` model corresponds to a `users` table.

- Primary keys: Eloquent assumes the primary key is a column named `id`.

- Timestamps: Eloquent expects `created\_at` and `updated\_at` columns to exist on your tables. These are managed automatically by Eloquent.

3. Retrieving Data:

- You can retrieve data using methods like `all()`, `find()`, `where()`, and more.

4. Inserting & Updating Data:

- You can create new records and update existing records using Eloquent.

5. Deleting Data:

- Eloquent provides methods to delete records from the database.

6. Relationships:

- Eloquent makes it easy to define relationships between different models, such as one-to-one, one-to-many, and many-to-many relationships.

7. Eager Loading:

- Eloquent supports eager loading to reduce the number of queries executed when retrieving related models.

8. Accessors and Mutators:

- Accessors allow you to format Eloquent attribute values when you retrieve them.

- Mutators allow you to alter attribute values before they are saved to the database.

1. **Do Curd using Eloquent Query**

use App\Models\User;

* // Create

$newUser = User::create([

'name' => 'John Doe',

'email' => 'john@example.com',

'password' => bcrypt('password'),

]);

* // Read

$allUsers = User::all();

$singleUser = User::find(1);

$activeUsers = User::where('status', 'active')->get();

* // Update

$updateUser = User::find(1);

$updateUser->name = 'Jane Doe';

$updateUser->email = 'jane@example.com';

$updateUser->save();

User::where('email', 'john@example.com')->update(['name' => 'Jane Doe']);

* // Delete

$deleteUser = User::find(1);

$deleteUser->delete();

User::where('status', 'inactive')->delete();

1. **Eloquent relationships**

* Eloquent relationships in Laravel are powerful features that allow you to define and work with relationships between different models in your application. Eloquent supports various types of relationships, making it easier to handle and query related data.

Types of Eloquent Relationships

1. One-to-One
2. One-to-Many
3. Many-to-Many
4. Has One Through
5. Has Many Through
6. Polymorphic Relationships
7. Many-to-Many Polymorphic Relationships
8. **What is Eager Loading and lazy loading?**

**Eager Loading**

Eager loading is a technique where related data is loaded at the same time as the main data query, typically through a single SQL query with joins. This can reduce the number of database queries executed and improve performance, especially when you need to access related data frequently.

**Example of Eager Loading:**

Consider you have a Post model that has many Comment models related to it. Without eager loading, if you retrieve posts and then access comments for each post, Eloquent will run an additional query for each post to fetch its comments.

**Lazy Loading**

Lazy loading is the default behavior in Eloquent where related data is loaded only when it is explicitly accessed. This means that Eloquent will execute a separate query to fetch the related data each time it is accessed.

**Example of Lazy Loading:**

Using the same Post and Comment example, when you access the comments of a post, Eloquent will run a separate query for each post.