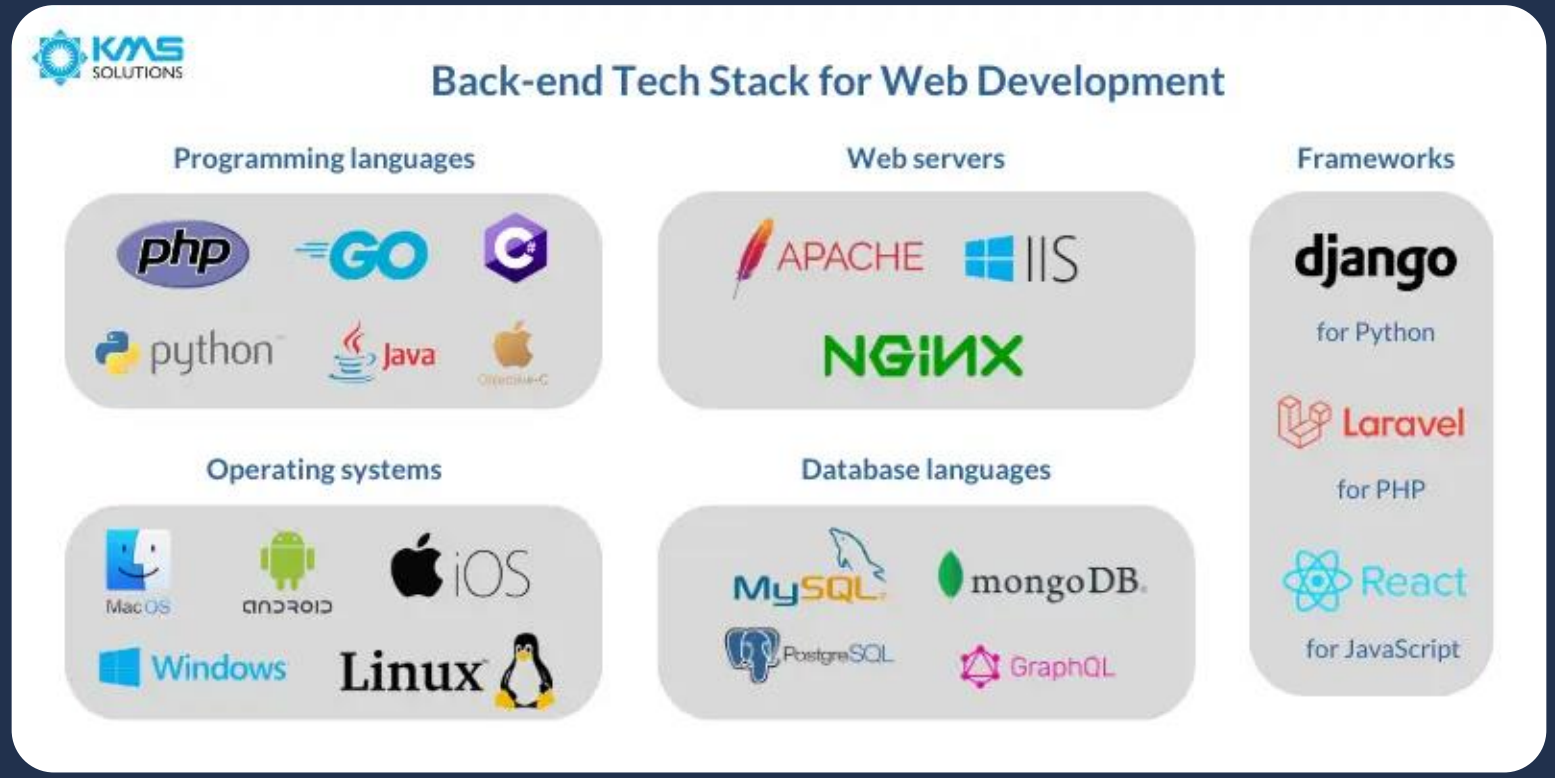
CSE485 – Công nghệ Web

dungkt@tlu.edu.vn



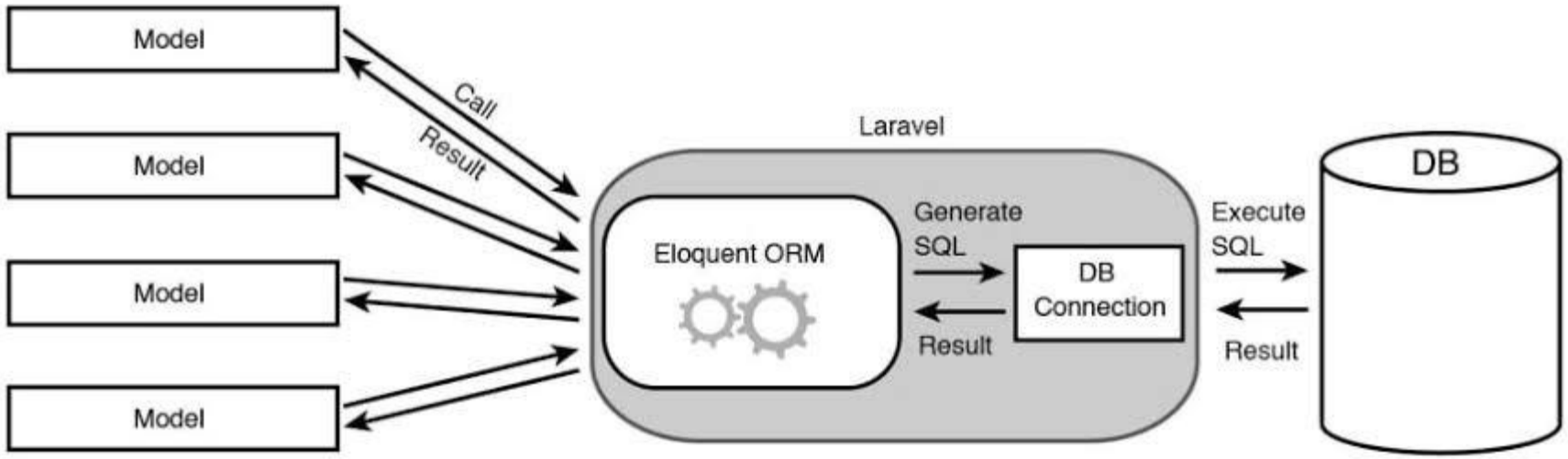
Bài 8. Laravel Framework (Phần 3)

NỘI DUNG

1. Giới thiệu về Eloquent ORM
2. Tạo mô hình và bảng
3. Truy xuất dữ liệu với Eloquent
4. Chèn và cập nhật dữ liệu với Eloquent
5. Truy vấn dữ liệu trên quan hệ
6. Accessors và Mutators

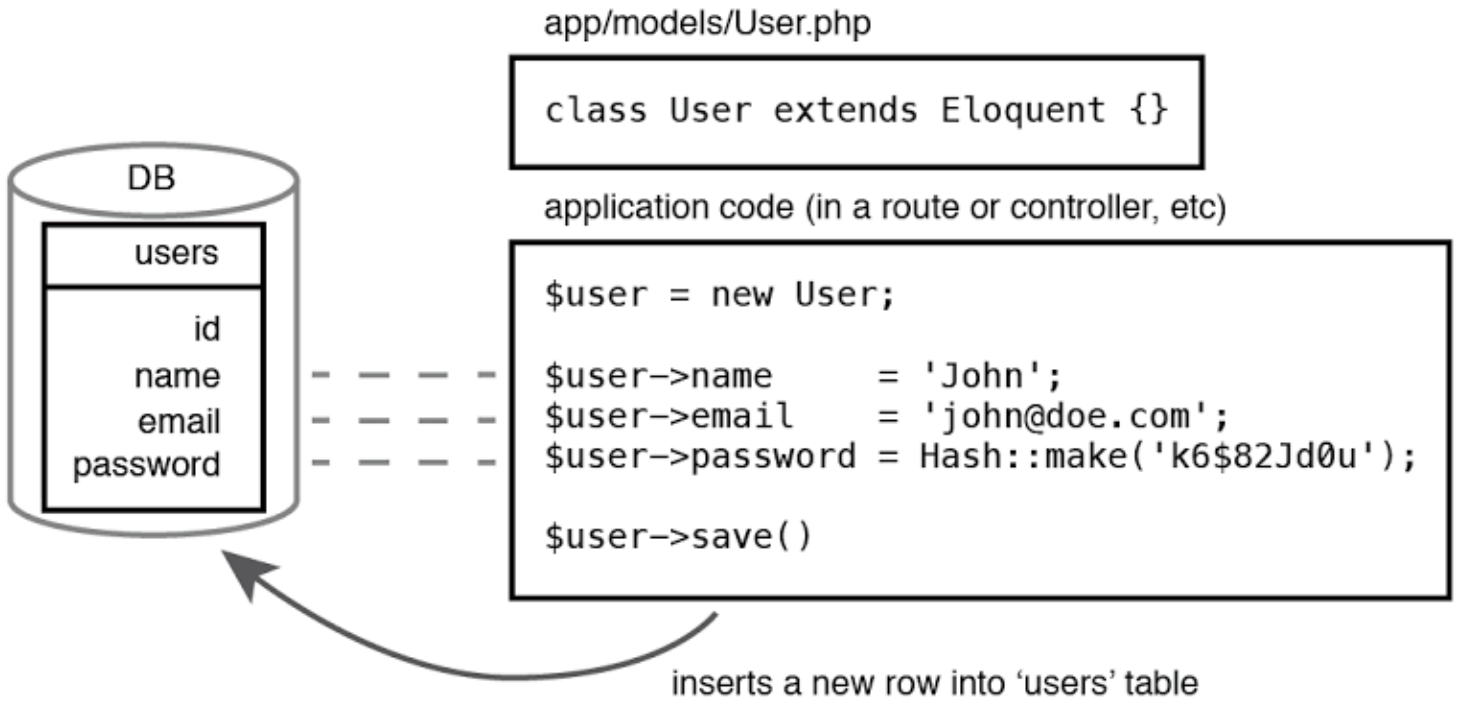
1. Giới thiệu về Eloquent ORM

- **Eloquent ORM** là một tính năng của Laravel, giúp lập trình viên tương tác với cơ sở dữ liệu thông qua các đối tượng PHP thay vì viết truy vấn SQL thuần túy. Nó được thiết kế để làm việc với cơ sở dữ liệu một cách trực quan, dễ dàng và hỗ trợ đầy đủ tính năng của Active Record pattern.
  - Tài liệu gốc chi tiết: [Eloquent: Getting Started - Laravel 10.x - The PHP Framework For Web Artisans]



1. Giới thiệu về Eloquent ORM

- Đặc điểm nổi bật của Eloquent:
  - **Active Record Implementation**: Mỗi bảng trong cơ sở dữ liệu được biểu diễn dưới dạng một "**Model**" trong Laravel, nơi bạn có thể thêm, cập nhật, xóa hoặc truy vấn dữ liệu.



1. Giới thiệu về Eloquent ORM

- Đặc điểm nổi bật của Eloquent:
  - **Tự động Quản lý Timestamps**: Eloquent có khả năng tự động quản lý các cột **created_at** và **updated_at** trong bảng dữ liệu, ghi nhận thời gian tạo và cập nhật bản ghi.
  - **Relationships**: Eloquent hỗ trợ tất cả các loại quan hệ cơ sở dữ liệu phổ biến như *one-to-one*, *one-to-many*, *many-to-many* và cả quan hệ xa hơn (*has-many-through*).
  - **Mass Assignment Protection**: Eloquent cung cấp cơ chế để bảo vệ dữ liệu của bạn khỏi nguy cơ bị gán giá trị hàng loạt (mass assignment vulnerability) thông qua thuộc tính **fillable** hoặc **guarded** trong Model.
  - **Query Scopes**: Eloquent cho phép bạn định nghĩa các query scope để tái sử dụng các phần của truy vấn trên các model, giúp code dễ đọc và tái sử dụng hơn.
  - **Soft Deletes**: Thay vì xóa dữ liệu khỏi cơ sở dữ liệu, Eloquent cho phép bạn "ẩn" các bản ghi thông qua tính năng Soft Deletes.
  - **Mutators** và **Accessors**: Cho phép bạn định nghĩa cách dữ liệu được lưu và truy xuất từ cơ sở dữ liệu, cho phép bạn tùy chỉnh dữ liệu trước khi nó được lưu hoặc hiển thị.
  - **Casting**: Eloquent cho phép bạn chuyển đổi các thuộc tính từ dữ liệu thô của cơ sở dữ liệu sang kiểu dữ liệu PHP mong muốn.
  - **Events**: Eloquent model có thể bắt và phản ứng với các sự kiện khác nhau trong chu kỳ sống của bản ghi, như khi tạo, cập nhật, xóa, và nhiều sự kiện khác.

2. Tạo model và bảng với Eloquent ORM

- Để tạo mô hình và bảng trong cơ sở dữ liệu sử dụng Eloquent ORM của Laravel, bạn sẽ thực hiện hai bước chính: viết migrations để định nghĩa cấu trúc bảng trong cơ sở dữ liệu và tạo mô hình (models) để tương tác với các bảng đó.
- **Bước 1**: Tạo tệp migration
  - **Tạo migration file**: Sử dụng Artisan command line tool để tạo file migration. Đây là một ví dụ về cách tạo migration cho bảng **users**:

```
php artisan make:migration create_users_table --create=users
```

  - **Viết migration**: Mở file migration mới được tạo trong thư mục database/migrations. Bạn sẽ thấy một class với hai phương thức: **up()** và **down()**. Trong up(), bạn định nghĩa cấu trúc của bảng. Trong down(), bạn định nghĩa cách để "rollback" migration, thường là xóa bảng.

2. Tạo model và bảng với Eloquent ORM

- **Bước 1**: Tạo tệp migration
  - **Viết migration**: ví dụ
  - Chạy migration: Để áp dụng migration và tạo bảng trong cơ sở dữ liệu, sử dụng lệnh sau:

```
php artisan migrate
```

```
use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

class CreateUsersTable extends Migration
{
    public function up()
    {
        Schema::create('users', function (Blueprint $table)
{
            $table->id();
            $table->string('name');
            $table->string('email')->unique();
            $table->timestamp('email_verified_at')-
>nullable();
            $table->string('password');
            $table->rememberToken();
            $table->timestamps();
        });
    }

    public function down()
    {
        Schema::dropIfExists('users');
    }
}
```

2. Tạo model và bảng với Eloquent ORM

- **Bước 2**: Tạo model
  - Tạo model: Để tạo một Eloquent model, sử dụng Artisan command. Laravel sẽ tự động tạo một file model trong thư mục app/Models (hoặc app/ nếu bạn đang sử dụng phiên bản Laravel cũ hơn).

```
php artisan make:model User
```

  - Định nghĩa model: Mở file model tương ứng vừa tạo. Laravel đã định nghĩa một số thông tin cơ bản cho bạn. Mỗi model tương ứng với một bảng trong cơ sở dữ liệu. Theo quy ước, *tên model là số ít và tên bảng là số nhiều của tên model*.

```
namespace App\Models;

use Illuminate\Contracts\Auth\MustVerifyEmail;
use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Foundation\Auth\User as Authenticatable;
use Illuminate\Notifications\Notifiable;

class User extends Authenticatable
{
    use HasFactory, Notifiable;

    // Khu vực này có thể được sử dụng để tùy chỉnh model của bạn.
}
```

2. Tạo model và bảng với Eloquent ORM

- **Bước 2**: Tạo model
  - Qui ước đặt tên:
    - Nếu tên bảng của bạn không theo quy ước số nhiều của Laravel, bạn có thể xác định tên bảng cụ thể bằng cách thiết lập thuộc tính **$table** trong model.
    - Đối với khóa chính không phải là **id** hoặc có **kiểu không phải là số tự tăng**, bạn cần chỉ định $primaryKey và $incrementing trong model.

```
protected $table = 'my_custom_table'; // Chỉ định nếu tên bảng khác với quy ước.
protected $primaryKey = 'my_custom_id'; // Chỉ định nếu khóa chính không phải là 'id'.
public $incrementing = false; // Chỉ định nếu khóa chính không phải là số tự tăng.
```

3. Truy xuất dữ liệu với Eloquent

- Eloquent ORM cung cấp một cách đơn giản và đẹp mắt để truy xuất dữ liệu từ cơ sở dữ liệu. Dưới đây là ví dụ minh họa.
  - **Truy xuất bản ghi đơn lẻ**: Để lấy một bản ghi duy nhất từ cơ sở dữ liệu, bạn có thể sử dụng phương thức **find** hoặc **findOrFail**, với **findOrFail** sẽ ném ra một ngoại lệ nếu không tìm thấy bản ghi.

```
use App\Models\User;

// Tìm một user với ID cụ thể
$user = User::find(1);

// Tìm một user với ID cụ thể hoặc ném ra một ngoại lệ
$user = User::findOrFail(1);
```

  - **Truy xuất nhiều bản ghi**: Để truy xuất nhiều bản ghi, bạn có thể sử dụng phương thức **get**. Đây là phương thức cơ bản nhất để thực hiện truy vấn và lấy dữ liệu dưới dạng một collection của các models.

```
$users = User::where('active', 1)->get();
```

3. Truy xuất dữ liệu với Eloquent

- Eloquent ORM cung cấp một cách đơn giản và đẹp mắt để truy xuất dữ liệu từ cơ sở dữ liệu. Dưới đây là ví dụ minh họa.
  - **Sử dụng Query Scope**: Query scopes cho phép bạn định nghĩa các ràng buộc truy vấn thông dụng mà có thể tái sử dụng qua nhiều truy vấn.

```
namespace App\Models;

use Illuminate\Database\Eloquent\Model;

class User extends Model
{
    public function scopeActive($query)
    {
        return $query->where('active', 1);
    }
}

// Sử dụng scope đã định nghĩa
$activeUsers = User::active()->get();
```

3. Truy xuất dữ liệu với Eloquent

- Eloquent ORM cung cấp một cách đơn giản và đẹp mắt để truy xuất dữ liệu từ cơ sở dữ liệu. Dưới đây là ví dụ minh họa.
  - **Eager loading**: Eager loading là một phương pháp để truy xuất quan hệ của model ngay lập tức, để tránh vấn đề N+1 query.
    - Nếu mỗi user có nhiều posts, bạn có thể truy xuất tất cả người dùng cùng với tất cả bài viết của họ:

```
$users = User::with('posts')->get();
```

    - Trong mỗi đối tượng user, bạn có thể truy cập posts thông qua thuộc tính posts:

```
foreach ($users as $user) {
    foreach ($user->posts as $post) {
        echo $post->title;
    }
}
```

3. Truy xuất dữ liệu với Eloquent

- Eloquent ORM cung cấp một cách đơn giản và đẹp mắt để truy xuất dữ liệu từ cơ sở dữ liệu. Dưới đây là ví dụ minh họa.
  - **Truy xuất và phân trang**: Khi bạn cần hiển thị dữ liệu trên các trang khác nhau, bạn có thể sử dụng phân trang trong Eloquent.

```
$users = User::paginate(10);

foreach ($users as $user) {
    echo $user->name;
}

// Trong view, để hiển thị các liên kết phân trang:
echo $users->links();
```

3. Truy xuất dữ liệu với Eloquent

- Eloquent ORM cung cấp một cách đơn giản và đẹp mắt để truy xuất dữ liệu từ cơ sở dữ liệu. Dưới đây là ví dụ minh họa.
  - **Truy xuất sử dụng Raw Query**: Đôi khi bạn cần viết truy vấn raw SQL, Eloquent cũng hỗ trợ việc này thông qua query builder.

```
$users = User::whereRaw('age > ? and votes = 100', [25])->get();
```

3. Truy xuất dữ liệu với Eloquent

- Khi nào sử dụng Elequent ORM, Query Builder và Raw Data:
  - **Eloquent ORM**:
    - Cần đặc trưng Hướng đối tượng: Eloquent cho phép bạn làm việc với dữ liệu dưới dạng các đối tượng và mối quan hệ giữa chúng, giúp mã nguồn dễ đọc và dễ bảo trì hơn.
    - Cần tận dụng các Relationships: Khi bạn cần tương tác với mô hình dữ liệu có các mối quan hệ phức tạp như one-to-one, one-to-many, hoặc many-to-many.
    - Cần tự động hóa: Eloquent tự động xử lý timestamps, soft deletes, và nhiều tính năng tiện ích khác.
    - Đang Thực Hiện CRUD: Eloquent rất hữu ích cho các thao tác cơ bản như tạo, đọc, cập nhật và xóa bản ghi (CRUD).

```
$user = User::where('email', 'example@example.com')->first();
```

3. Truy xuất dữ liệu với Eloquent

- Khi nào sử dụng Elequent ORM, Query Builder và Raw Data:
  - **Query Buidder**:
    - Cần viết truy vấn linh hoạt: Query Builder cung cấp một cách truy vấn dữ liệu linh hoạt và dễ kiểm soát hơn, đặc biệt khi truy vấn phức tạp không thể được biểu diễn dễ dàng bằng Eloquent.
    - Cần hiệu suất tốt hơn: Trong một số trường hợp, Query Builder có thể nhanh hơn Eloquent do nó không cần phải tạo ra các đối tượng model.
    - Không cần sử dụng Model: Khi bạn không cần hoặc không muốn sử dụng các models của Eloquent.
    - Cần truy vấn dữ liệu đơn giản: Đối với các truy vấn không liên quan đến logic của ứng dụng và chỉ cần lấy dữ liệu ra mà không cần sự phức tạp của một ORM.

```
$users = DB::table('users')->where('votes', '>', 100)->get();
```

3. Truy xuất dữ liệu với Eloquent

- Khi nào sử dụng Elequent ORM, Query Builder và Raw Data:
  - **Raw Data**:
    - Cần tối ưu cao nhất: Khi truy vấn cực kỳ phức tạp và việc tối ưu hóa cần được kiểm soát chặt chẽ, viết raw SQL cho phép bạn tận dụng tối đa khả năng của cơ sở dữ liệu.
    - Cần sử dụng đặc trưng đặc thù của DBMS: Khi bạn cần sử dụng các tính năng đặc biệt của hệ quản trị cơ sở dữ liệu mà không được hỗ trợ trực tiếp qua Eloquent hoặc Query Builder.
    - Di chuyển từ hệ thống cũ: Khi bạn đang chuyển đổi từ một hệ thống cũ mà trong đó các truy vấn đã được viết bằng SQL thuần túy và không có thời gian hoặc nguồn lực để chuyển đổi chúng sang Eloquent hay Query Builder.

```
$users = DB::select('SELECT * FROM users WHERE votes = ?', [100]);
```

Bảo mật: Khi sử dụng raw SQL queries, bạn cần phải tự mình đảm bảo rằng truy vấn của mình không dễ bị SQL injection. Trong khi đó, Eloquent và Query Builder tự động thoát dữ liệu và cung cấp một lớp bảo mật khỏi các mối đe dọa này.
Bảo trì: Eloquent và Query Builder giúp mã nguồn dễ đọc và bảo trì hơn là sử dụng raw SQL.

4. Chèn và cập nhật dữ liệu với Eloquent

- **Chèn dữ liệu**

```
use App\Models\User;

// Tạo một thể hiện mới của model User
$user = new User;

// Thiết lập các thuộc tính
$user->name = 'Jane Doe';
$user->email = 'jane.doe@example.com';
$user->password = bcrypt('password');

// Lưu người dùng vào cơ sở dữ liệu
$user->save();
```

```
$user = User::create([
    'name' => 'Jane Doe',
    'email' => 'jane.doe@example.com',
    'password' => bcrypt('password'),
]);
```

4. Chèn và cập nhật dữ liệu với Eloquent

- **Cập nhật dữ liệu**

```
// Tìm người dùng theo ID
$user = User::find(1);

// Cập nhật thuộc tính
$user->email = 'new.email@example.com';

// Lưu thay đổi vào cơ sở dữ liệu
$user->save();
```

```
// Cập nhật tất cả người dùng có trạng thái hoạt động
User::where('active', 1)
    ->update(['status' => 'inactive']);
```

4. Chèn và cập nhật dữ liệu với Eloquent

- **Lưu ý về bảo mật**
  - Khi sử dụng phương thức create hoặc update của Eloquent, luôn cần lưu ý đến lỗ hổng bảo mật gán giá trị hàng loạt. Bạn nên chỉ định hoặc là thuộc tính fillable hoặc guarded trên model của mình, định nghĩa các thuộc tính nào được phép gán hàng loạt.

```
class User extends Model
{
    // Sử dụng fillable để cho phép gán giá trị hàng loạt
    protected $fillable = ['name', 'email', 'password'];

    // Hoặc sử dụng guarded để ngăn chặn gán giá trị hàng loạt
    protected $guarded = ['id'];
}
```

4. Chèn và cập nhật dữ liệu với Eloquent

- **Xóa bản ghi**

```
// Xóa một bản ghi cụ thể
$user = User::find($id);
$user->delete();

// Xóa dựa trên điều kiện
User::where('active', 0)->delete();
```

```
// Sử dụng Soft Delete
use Illuminate\Database\Eloquent\SoftDeletes;

class User extends Model {
    use SoftDeletes;
}

// Truy xuất bản ghi không bị Soft Delete
$users = User::all();

// Truy xuất bao gồm cả bản ghi Soft Deleted
$users = User::withTrashed()->get();
```

5. Truy vấn dữ liệu trên quan hệ

- **Quan hệ 1 - 1**

```
// Trong Model User
public function profile() {
    return $this->hasOne(Profile::class);
}

// Truy xuất profile từ user
$profile = $user->profile;
```

5. Truy vấn dữ liệu trên quan hệ

- **Quan hệ 1 - n**

```
// Trong Model Post
public function comments() {
    return $this->hasMany(Comment::class);
}

// Truy xuất comments từ post
$comments = $post->comments;
```

5. Truy vấn dữ liệu trên quan hệ

- **Quan hệ n – n**

```
// Trong Model User
public function roles() {
    return $this->belongsToMany(Role::class);
}

// Truy xuất roles từ user
$roles = $user->roles;
```

- **Eager Loading**

```
// Eager load "profile" khi truy xuất User
$user = User::with('profile')->find($id);

// Eager load nhiều quan hệ
$users = User::with(['posts', 'roles'])->get();
```

6. ACCESSORS VÀ MUTATORS

- **Accessors và Mutators** trong Laravel là các phương thức trong Eloquent ORM cho phép bạn định nghĩa cách các thuộc tính của model được lưu vào và truy xuất từ cơ sở dữ liệu.

```
// Accessor trong Model User
// thay đổi giá trị đó trước khi nó được trả về.
public function getFirstNameAttribute($value) {
    return ucfirst($value);
}

// Mutator trong Model User
// xử lý hoặc chuẩn hóa dữ liệu trước khi nó được lưu
public function setPasswordAttribute($value) {
    $this->attributes['password'] = bcrypt($value);
}
```

“Câu hỏi & Thảo luận”

THE END!