TP N° 10

Manipuler l'ORM Eloquent

ISTA TINGHIR

Module: M205

Développer en Back-End

Realisée par: Groupe 10

le: 26 mai 2023

Introduction

L'ORM (Object Relational Mapper) Eloquent de Laravel est une manière élégante, belle et efficace d'interagir et de gérer les bases de données. L'équipe Laravel a créé Eloquent pour faciliter l'interaction et la communication avec les bases de données et fournir un moyen facile à utiliser pour résoudre tous les problèmes liés aux bases de données et au développement. Laravel Eloquent offre la liberté d'écrire des codes personnalisables, bien formatés, efficaces, faciles à lire, bien documentés et durables en suivant les normes et les meilleures pratiques de l'industrie

Interroger l'existence d'une relation

Lorsque vous récupérez des enregistrements de modèles, vous pouvez souhaiter limiter vos résultats en fonction de l'existence d'une relation. Pour ce faire, vous pouvez passer le nom de la relation aux méthodes has et orHas

```php
use App\Models\Post;

// Retrieve all posts that have at least one comment...

$posts = Post::has('comments')->get();
```

Interroger l'absence d'une relation

Lorsque vous récupérez des enregistrements de modèles, vous pouvez souhaiter limiter vos résultats en fonction de l'absence de relation. Pour ce faire, vous pouvez passer le nom de la

relation aux méthodes doesntHave et orDoesntHave

```
$posts = Post::doesntHave('comments')->get();
```

Requête sur les relations Morph To :

Pour vérifier l'existence de relations "morph to", vous pouvez utiliser les méthodes whereHasMorph et whereDoesntHaveMorph.

```
$comments = Comment::whereHasMorph(
    'commentable'
    ,
    [Post::class, Video::class],
    function (Builder $query) {
    $query->where('title'
    ,
    'like'
    ,
    'code%');
    }
    )->get();
```

Comptage des modèles liés

Vous souhaitez parfois compter le nombre de modèles liés pour une relation donnée sans avoir à charger les modèles. Pour ce faire, vous pouvez utiliser la méthode withCount. La méthode withCount place un attribut {relation}_count sur les modèles résultants.

```


$posts = Post::withCount('comments')->get();
foreach ($posts as $post) {
echo $post->comments_count;
}
```

Comptage des modèles liés

En passant un tableau à la méthode withCount, vous pouvez ajouter les "counts" de plusieurs relations ainsi que des contraintes supplémentaires aux requêtes.

```
$posts = Post::withCount(['votes'
```

```
,
'comments' => function (Builder $query) {
$query->where('content'
,
'like'
,
'code%');
}])->get();
echo $posts[0]->votes_count;
echo $posts[0]->comments_count;
```

Comptage des modèles liés

En utilisant la méthode loadCount, vous pouvez charger un compte de relation après que le modèle parent ait été récupéré.

```
$book = Book::first();
$book->loadCount('genres');
```

Comptage des modèles liés

Si vous devez définir des contraintes de requête supplémentaires pour la requête de comptage, vous pouvez passer un tableau avec les clés des relations que vous souhaitez compter. Les valeurs du tableau doivent être des fermetures qui reçoivent l'instance du constructeur de requêtes

```
$book->loadCount(['reviews' => function ($query) {
    $query->where('rating'
    , 5);
    }])
```

Autres fonctions agrégées

En plus de la méthode withCount, Eloquent fournit les méthodes withMin, withMax, withAvg, withSum et withExists. Ces méthodes placent un attribut {relation}_{fonction}_{column} sur vos modèles résultants.

```
$posts = Post::withSum('comments'
,
```

```
'votes')->get();

foreach ($posts as $post) {

echo $post->comments_sum_votes;

}
```

Comme la méthode loadCount, des versions différées de ces méthodes sont également disponibles. Ces opérations d'agrégation supplémentaires peuvent être effectuées sur les modèles Eloquent qui ont déjà été récupérés.

```
post = Post::first();

$post->loadSum('comments'

,

'votes');
```

Insertion et mise à jour des modèles liés

Eloquent fournit des méthodes pratiques pour ajouter de nouveaux modèles aux relations. Vous pouvez insérer un modèle lié en utilisant la méthode de save:

```
$comment = new Comment(['message' => 'A new comment.']);

$post = Post::find(1);

$post->comments()->save($comment);
```

Si vous devez sauvegarder plusieurs modèles liés, vous pouvez utiliser la méthode saveMany

Les méthodes save et saveMany persistent les instances de modèle données, mais n'ajoutent pas les modèles nouvellement persistés aux relations en mémoire déjà chargées sur le modèle parent. vous pouvez utiliser la méthode refresh pour recharger le modèle et ses relations.

```


$post->comments()->save($comment);

$post->refresh();

// All comments, including the newly saved comment...

$post->comments;
```

Si vous souhaitez sauvegarder votre modèle et toutes les relations qui lui sont associées, vous pouvez utiliser la méthode "push". Dans cet exemple, le modèle Post sera sauvegardé ainsi que ses commentaires et les auteurs des commentaires.

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

$post->comments[0]->message =
```

```
'Message';

$post->comments[0]->author->name =

'Author Name';

$post->push();
```

Vous pouvez utiliser la méthode create, qui accepte un tableau d'attributs pour crée un modèle et l'insère dans la base de données.

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

$comment = $post->comments()->create([

'message' => 'A new comment.'

,

]);
```

Si vous souhaitez attribuer un modèle enfant à un nouveau modèle parent, vous pouvez utiliser la méthode associate.

```
$account = Account::find(10);

$user->account()->associate($account);

$user->save();
```

Pour supprimer un modèle parent d'un modèle enfant, vous pouvez utiliser la méthode dissociate. Cette méthode donne la valeur null à la clé étrangère de la relation.

```
$user->account()->dissociate();

$user->save();
```

Eloquent fournit également des méthodes pour rendre plus pratique le travail avec les relations many-to-many. Par exemple, imaginons qu'un utilisateur puisse avoir plusieurs rôles et qu'un rôle puisse avoir plusieurs utilisateurs. Vous pouvez utiliser la méthode attach pour attacher un rôle à un utilisateur en insérant un enregistrement dans la table intermédiaire de la relation.

```
$user = User::find(1);

$user->roles()->attach($roleId);
```

Lorsque vous attachez une relation à un modèle, vous pouvez également transmettre un tableau de données supplémentaires à insérer dans la table intermédiaire.

```
$user->roles()->attach($roleId, ['expires' => $expires]);
```

Pour supprimer un enregistrement de relation n:n, utilisez la méthode detach. Cette méthode supprime l'enregistrement correspondant de la table intermédiaire, mais les deux modèles restent dans la base de données.

```
// Détacher un rôle unique de l'utilisateur...
```

```php
$user->roles()->detach($roleId);

// Détacher tous les rôles de l'utilisateur...

$user->roles()->detach();
```

Vous pouvez également utiliser la méthode sync pour construire des associations de plusieurs à plusieurs. La méthode sync accepte un tableau d'identifiants à placer dans la table intermédiaire. Tous les ID qui ne sont pas dans le tableau donné seront supprimés du tableau intermédiaire. Ainsi, une fois cette opération terminée, seuls les identifiants du tableau donné existeront dans la table intermédiaire :

```php
$user->roles()->sync([1, 2, 3]);
```

Si vous devez mettre à jour une ligne existante dans le tableau intermédiaire de votre relation, vous pouvez utiliser la méthode updateExistingPivot. Cette méthode accepte la clé étrangère de l'enregistrement intermédiaire et un tableau d'attributs à mettre à jour

```php
$user = User::find(1);

$user->roles()->updateExistingPivot($roleId, [

'active' => false,

]);
```

MANIPULATION DES COLLECTIONS

Introduction

Toutes les méthodes Eloquent qui renvoient plus d'un résultat de modèle renverront des instances de la classe Illuminate\Database\Eloquent\Collection, y compris les résultats récupérés via la méthode get ou accessibles via une relation. L'objet Eloquent collection étend la collection de base de Laravel, il hérite donc naturellement de dizaines de méthodes utilisées pour travailler de manière fluide avec le tableau sous-jacent des modèles Eloquent. Toutes les collections servent également d'itérateurs, ce qui vous permet de les parcourir en boucle comme s'il s'agissait de simples tableaux PHP.

Les collections sont beaucoup plus puissantes que les tableaux et exposent une variété d'opérations de map / reduce qui peuvent être enchaînées à l'aide d'une interface intuitive.

```php
$names = User::all()->reject(function (User $user) {

    return $user->active === false;

    })->map(function (User $user) {

    return $user->name;

    });
```

Liste des méthodes

Toutes les collections Eloquent étendent l'objet base collection de Laravel ; par conséquent, elles héritent de toutes les méthodes puissantes fournies par la classe base collection. En outre, la classe Illuminate\Database\Eloquent\Collection fournit un super ensemble de méthodes pour faciliter la gestion des collections de votre modèle.

la listes des méthodes dans la décumentation