Laravel - Eloquent

1 - Eloquent

Laravel utilise Eloquent [1] pour modéliser les tables et interroger la base de données.

> [1] Eloquent ORM (object-relational mappe) inclus avec Laravel fournit une belle et simple implémentation Active Record pour travailler avec votre base de données. Chaque table de base de données a un "modèle" correspondant qui est utilisé pour interagir avec cette table. Les modèles vous permettent de rechercher des données dans vos tables, ainsi que d'insérer de nouveaux enregistrements dans la table.
> Référence: https://laravel.com/docs/8.x/eloquent#introduction

Laravel enregistre toutes les tables en tant que modèle sous l'application Directory

---

2 - Chargement des données de la base de données

Chargement des données de la base de données

Pour ce faire, utilisez la commande suivante en haut du fichier du contrôleur après la commande namespace.

```
use App\Models\BlogPost;
```

Choisissez la fonction où les données doivent être chargées dans la classe de contrôleur et, à l'aide d'eloquent, écrivez l'instruction de requête pour sélectionner les données.

```
$blog = BlogPost::select()->get();
```

ou

```
$blog = BlogPost::all();
```

Le résultat est un tableau JSON, pour l'imprimer, tapez :

```
return $blog;
```

Pour transmettre ce résultat à un panneau d'affichage, utilisez le code suivant :

```
return view ('/queries', ['blog' => $blog]);
```

---

3 - Requêtes

Comment créer des requêtes standard avec eloquent.

SELECT

```
$blog = BlogPost::select('title', 'body')
->get();
```

WHERE

```
$blog = BlogPost::select()
->where('user_id', '=', 1)
->get();
```

WHERE Clé Primaire

```
$blog = BlogPost::find(1);
```

AND

```
$blog = BlogPost::select()
->where('user_id', '=', 1)
->where('id', '=', 1)
->get();
```

OR

```
$blog = BlogPost::select()
->where('user_id', '=', 1)
->orWhere('id', '=', 1)
->get();
```

ORDER BY

```
$blog = BlogPost::select()
->orderBy('title', 'DESC')
->get();
```

INNER JOIN

```
$blog = BlogPost::select()
->join('users', 'blog_posts.user_id', '=', 'users.id')
->get();
```

LEFT / RIGHT OUTER JOIN

```
$blog = BlogPost::select()
->rightJoin('users', 'blog_posts.user_id', '=', 'users.id')
->get();
```

Aggregates Function : Max, Min, Avg, Sum, Count

```
$blog = BlogPost::max('id');
```

ou avec select

```
$blog = BlogPost::where('user_id', 1)
->count('id');
```

Références: https://laravel.com/docs/8.x/queries

---

4 - Expressions brutes

Parfois, vous devrez peut-être insérer une chaîne arbitraire dans une requête. Pour créer une expression de chaîne brute, vous pouvez utiliser la méthode brute fournie par la façade DB :

Les instructions brutes seront injectées dans la requête sous forme de chaînes, vous devez donc être extrêmement prudent pour éviter de créer des vulnérabilités d'injection SQL.

En haut du fichier chargez la façade DB :

```
use Illuminate\Support\Facades\DB;
```

Écrivez la requête brute :

```
$blog = BlogPost::select(DB::raw('count(*) as blogs, user_id '))
->groupBy('user_id')
->get();
```

---

5 - Modèles liés

Parfois, un modèle peut avoir de nombreux modèles liés, mais vous souhaitez récupérer facilement la jointure entre les tables.

Ouvrez le modèle BlogPost et créez la fonction blogHasUser à l'aide de la méthode hasOne :

```
public function blogHasuser(){
    return $this->hasOne('App\Models\User', 'id', 'user_id');
}
```

Utilisez le contrôleur pour rechercher un enregistrement de blog.

```
$blog = BlogPost::find(1);
return view('blog/queires',  ['blog' =>$blog]);
```

À l'aide de la vue, afficher l'utilisateur qui a écrit ce message, en appelant la méthode créée.

```
{{ $blog->blogHasUser}}
```

Référence: https://laravel.com/docs/8.x/eloquent-relationships#has-one-of-many

---