Gestion des relations dans Eloquent

Introduction

Les tables de base de données sont souvent liées les unes aux autres via des relations.Eloquent facilite la gestion et l'utilisation de ces relations et prend en charge une variété de relations communes: one To one, one To many, many To many ...

Définir les relations

Les relations Eloquent sont définies comme des **méthodes** sur vos classes de modèles Eloquent. Étant donné que les relations servent également de puissants *générateurs de requêtes* , la définition de relations en tant que méthodes offre de puissantes capacités de chaînage de méthodes et d'interrogation. Par exemple :

```
$user->posts()->where('active', 1)->get();
```

Types de relations dans ORM Eloquent

L'ORM Eloquent permet de définir plusieurs types de relations entre les modèles. Voici les principaux types de relations :

1. Relation "one-to-one" : une instance du modèle source est liée à une seule instance du modèle de destination, et vice versa. Par exemple, un utilisateur peut avoir une seule adresse, et une adresse peut être associée à un seul utilisateur.
2. Relation "one-to-many" : une instance du modèle source est liée à plusieurs instances du modèle de destination, mais une instance du modèle de destination est liée à une seule instance du modèle source. Par exemple, un utilisateur peut avoir plusieurs posts, mais un post est lié à un seul utilisateur.
3. Relation "many-to-one" : plusieurs instances du modèle source sont liées à une seule instance du modèle de destination, mais une instance du modèle de destination est liée à une seule instance du modèle source. Cette relation est l'inverse de la relation "one-to-many". Par exemple, plusieurs articles peuvent être liés à un seul utilisateur.
4. Relation "many-to-many" : plusieurs instances du modèle source peuvent être liées à plusieurs instances du modèle de destination, et vice versa. Par exemple, un utilisateur peut avoir plusieurs rôles, et un rôle peut être associé à plusieurs utilisateurs.

5. Relation "polymorphique" : une instance du modèle source peut être liée à plusieurs types d'instances de modèles de destination, et vice versa. Par exemple, une image peut être associée à un utilisateur ou à un article.

Eloquent permet de définir chacun de ces types de relations en utilisant des méthodes de relation spécifiques, telles que hasOne, hasMany, belongsTo, belongsToMany, etc. Ces méthodes permettent de définir les clés étrangères et autres détails de la relation entre les modèles, ce qui permet à Eloquent de récupérer les instances liées de manière transparente et efficace.

I- Relation (One To One)

La plus simple relation mais la moins utile. Une relation de type **« Has One »** ou **« One-To-One »**. Cela signifie effectivement qu'un certain enregistrement est lié à un autre enregistrement (mais pas à plusieurs autres enregistrements !).

Exemple :

Chaque *employé* a une seule *fiche* de salaire et chaque fiche de salaire est associée à un seul employé

Donc , les deux tables 'employes' et 'fiches' doivent avoir le schéma suivant :

CREATE TABLE employes (idEmploye PRIMARY KEY AUTO_INCREMENT, nom VARCHAR(50), prénom VARCHAR(50), email VARCHAR(50), dateEmbauche DATE);	CREATE TABLE fiches (idfiche INT PRIMARY KEY AUTO_INCREMENT, salaire FLOAT, dateDébutMois DATE, dateFinMois DATE, idEmployé INT **NOT NULL UNIQUE,** **FOREIGN KEY (idEmploye) REFERENCES Employé(idEmployé)) on delete restrict on update restrict;**

Exercice :

Créer les deux tables en utilisant des migrations

Model : Employe	**Model** : Fiche

class **Employe** extends **Model**
{
use **HasFactory**;
protected **$primaryKey** = 'idEmployé';
public function **fiche**(){
return $this->**hasOne**(**Fiche**::class,'idEmploye');
}}
	class **Fiche** extends **Model**{
	use **HasFactory**;
	protected **$primaryKey** = 'idfiche';
	public function **employe**(){
	return $this->**belongsTo**(**Employe**::class);
	} }

Remarque : La méthode hasOne() prend deux arguments : le nom de la classe du modèle associé et le nom de la clé étrangère dans la table associée.

Une fois que vous avez créé ces modèles, vous pouvez utiliser les méthodes Eloquent pour interagir avec les tables Employé et fiche , ainsi que pour récupérer les enregistrements correspondants en utilisant la relation One-to-One définie entre les deux tables

Exemple 1:

Récupérer un employé et sa fiche de salaire associée :	$employé = Employe::find(1); $ficheDeSalaire = $employé->fiche;
Récupérer une fiche de salaire et l'employé associé :	$ficheDeSalaire = Fiche::find(1); $employé = $ficheDeSalaire->employe;

Exemple 2 : Créer un nouvel employé avec une fiche de salaire associée :

```
$employé = new Employe;
$employé->nom = 'Alaoui';
$employé->prénom = 'Aya';
$employé->email = 'Alaoui.aya@gmail.com';
$employé->dateEmbauche = '2022-02-01';
$employé->save();

$ficheDeSalaire = new Fiche;
$ficheDeSalaire->salaire = 4500;
$ficheDeSalaire->dateDébutMois = '2022-02-01';
$ficheDeSalaire->dateFinMois = '2022-02-28';

$employé->fiche()->save($ficheDeSalaire);
```

Exemple 3 : Mettre à jour la fiche de salaire d'un employé :

```
$employé = Employe::find(1);
$ficheDeSalaire = $employé->fiche;
$ficheDeSalaire->salaire = 5000;
$ficheDeSalaire->save();
```

Exemple 4 : Supprimer un employé et sa fiche de salaire associée

```
$employé = Employe::find(1);
$employé->fiche->delete();
$employé->delete();
```

II- Relation (One To Many)

Il s'agit d'une relation très importante, peut-être même la plus importante. Une relation telle que « **Has Many** » ou « **One-To-Many** ». Cela signifie en fait qu'un certain enregistrement est lié à plusieurs autres enregistrements.

Exemple :

un client peut avoir plusieurs commandes .Alors qu'une commande ne concerne qu'un seul client.

la relation directe:

hasMany

l'inverse de la relation :

belongsTo

Exercice : Créer les deux tables dans la base de donnéesvia des migrations

Model : Client

```php
class Client extends Model {
    use HasFactory;
    public function commandes(){
    return $this->hasMany(Commande::class,'client_id');
    }
}
```

Model : Commande

```php
class Commande extends Model {
    use HasFactory;
    public function clients(){
        return $this->belongsTo(Client::class,'client_id');
    }
}
```

Une fois que vous avez créé ces modèles, vous pouvez utiliser les méthodes Eloquent pour interagir avec les tables clients et commandes

Exemples d'utilisation :

Récupérer toutes les commandes passées par un client donné :	`$client = Client::find(1);` `$commandes = $client->commandes;`

Ajouter une nouvelle commande pour un client donné	`$client = Client::find(1);`
	`$commande = new Commande([`
	`'dateCommande' => '2022-06-01',`
	`'dateLivraison' => '2022-06-15'`
	`]);`
	`$client->commandes()->save($commande);`

Supprimer une commande pour un client donné	`$client = Client::find(1);`
	`$commande = $client->commandes()->where('id', 1)->first();`
	`$commande→delete();`

Mettre à jour la date de livraison d'une commande pour un client donné :	`$client = Client::find(1);`
	`$commande = $client->commandes()->where('id', 1)->first();`
	`$commande->dateLivraison = '2022-07-01';`
	`$commande→save();`

Récupérer tous les clients avec leurs commandes triées par date de commande :	`$clients =Client::with(['commandes' =>`
	`function ($query) {`
	`$query->orderBy('dateCommande', 'asc');`
	`}])→get();`

Récupérer tous les clients qui ont passé au moins une commande :	`$clients = Client::has('commandes')→get();`

Récupérer le nombre total de commandes pour chaque client :	`$clients = Client::withCount('commandes')->get();`
	`foreach ($clients as $client) {`
	`echo "Le client " . $client->nom . " a passé " . $client->commandes_count . " commandes.";`
	`}`

III- Relation (Many To Many)

Les relations **n..n (Many To Many)** permettent de définir une relation entre plusieurs objets d'un côté et plusieurs objets de l'autre.

Exemple : un étudiant peut suivre plusieurs cours et un cours peut être suivi par plusieurs étudiants, d'où une relation Many-to-Many entre les deux tables.

Etudiant(idE,nom,prenom)

Cours(idC,nom)

EtudiantCours(id,#idE,#idC)

Exercice :

Créer les trois tables via des migrations

Remarque : Il faut donc au préalable créer également une migration pour la table **cours_etudiant**. le nom de la table est construit en joignant les noms des tables ***(tous deux au singulier !)*** dans l'ordre alphabétique. Cela permet à Laravel de faire directement le lien entre les tables sans avoir à préciser le nom de la table de relation.

Comme la relation est symétrique on a une seule méthode : belongsToMany

```
// Modèle Etudiant
class Etudiant extends Model
{
 protected $table = 'etudiant';
public function cours() {
 return $this->belongsToMany(Cours::class, 'etudiant_cours', 'id_etudiant', 'id_cours');
} }
```

```
// Modèle Cours
class Cours extends Model
{
    protected $table = 'cours';
   public function etudiants()
    {
  return $this->belongsToMany(Etudiant::class, 'etudiant_cours', 'id_cours', 'id_etudiant');
    }
}
```

Remarque :

Si vous choisissez une autre table que le nom par défaut, vous pouvez ajouter votre propre nom de table en tant que paramètre à la méthode BelongsToMany.

Voici quelques exemples de méthodes pour interagir avec ces deux tables en utilisant ces modèles :

Récupérer tous les étudiants qui suivent un cours donné :	$cours = Cours::find(1); $etudiants = $cours->etudiants;
Récupérer tous les cours suivis par un étudiant donné :	$etudiant =Etudiant::find(1); $cours = $etudiant->cours;
Ajouter un cours à un étudiant :	$etudiant = Etudiant::find(1); $cours = new Cours(['nom' => 'Chimie']); $etudiant→cours()→save($cours);
Supprimer un cours d'un étudiant :	$etudiant = Etudiant::find(1); $etudiant->cours()->detach(1);
Supprimer tous les cours d'un étudiant :	$etudiant = Etudiant::find(1); $etudiant->cours()->detach();

TP :

Soit le schéma relationnel suivant de la base de données GFormations:

Etudiant (codeE, nom,prenom,adresse,dateNaissance,#idclasse)

Classe(idc,libelle,#idformation,NombreMax)

Formation(idf,Titre,NbreHeure)

Avis(#idE,#idf ,points)

1. Créer une nouvelle application Laravel
2. Configurer la base de données dans le fichier .env.
3. Créer les migrations pour les tables .Éditer les migrations pour ajouter les champs nécessaires aux tables. Exécuter les migrations à l'aide de la commande php artisan migrate.
4. Créer les modèles necessaires.
5. Créer les seeders et factorys pour pouvoir remplir les tables
6. Éditer les modèles pour ajouter les relations entre les tables.
7. Gérer les opérations suivantes : Afficher les iformations des etudiants, afficher le détails (informations de la classe de l'étudiant ainsi que la formation

concernée),modifier la classe d'un etudiant, afficher si la classe de l'étudiant est pleine ou non) . Créer les contrôleurs et les routes nécessaires .

8. Créer les vues pour le test.
9. Proposer les options suivantes : Rechercher les informations (titre , nombre d'heures, nombre de classes, nombre des étudiants inscrits à cette formation) d'une formation.
10. On veut afficher le nombre de points d'une formation donnée.
11. Ajouter des fonctionnalités supplémentaires à l'application, telles que la possibilité de supprimer des inscriptions à des formations, la pagination des formations, la recherche des formations d'un etudiant donné, etc.