Strategies heritage avec Doctrine ORM

1. Différentes stratégies

- SINGLE_TABLE
- JOINED

2. Principes

Stratégie SINGLE_TABLE

La stratégie SINGLE_TABLE utilise une seule table pour toute la hiérarchie de classes. Chaque colonne nécessaire pour toutes les classes dans la hiérarchie est ajoutée à cette table unique. Pour différencier les enregistrements de différentes classes, un champ discriminateur est utilisé.

Cette stratégie est simple et offre de bonnes performances, car elle évite les jointures lors des requêtes. Cependant, elle peut conduire à de nombreuses colonnes avec des valeurs NULL si les sous-classes ont des champs différents.

Stratégie JOINED

La stratégie JOINED crée une table séparée pour chaque classe dans la hiérarchie, y compris la classe de base abstraite. Chaque table de sous-classe contient uniquement les champs spécifiques à cette sous-classe et une clé étrangère qui fait référence à la table de la classe de base.

Les requêtes sur ces tables nécessitent des jointures, ce qui peut être moins performant que la stratégie SINGLE_TABLE, mais cette approche est plus normalisée et évite les colonnes inutilisées.

3. Le code PHP/SQL de la hiérarchie de classes

3.1 Stratégie SINGLE_TABLE

```php
#[ORM\Entity]
#[ORM\InheritanceType("SINGLE_TABLE")]
#[ORM\DiscriminatorColumn(name: "type", type: "string")]
#[ORM\DiscriminatorMap(["voiture" => "Voiture", "velo" => "Velo"])]
abstract class Vehicule
{
    #[ORM\Id]
    #[ORM\GeneratedValue]
    #[ORM\Column(type: "integer")]
    private int $id;

    #[ORM\Column(type: "string")]
    private string $marque;
}

#[ORM\Entity]
class Voiture extends Vehicule
{
    #[ORM\Column(type: "string")]
    private string $typeMoteur;
}

#[ORM\Entity]
class Velo extends Vehicule
{
    #[ORM\Column(type: "string")]
    private string $taille;
}
```

Code SQL

```
CREATE TABLE vehicule (
    id INT AUTO_INCREMENT NOT NULL,
    type VARCHAR(255) NOT NULL, -- Colonne discriminante
    marque VARCHAR(255) NOT NULL,
    typeMoteur VARCHAR(255) DEFAULT NULL, -- Spécifique à Voiture
    taille VARCHAR(255) DEFAULT NULL, -- Spécifique à Velo
    PRIMARY KEY(id)
)
```

Schéma de la base de données

Table vehicule

id	marque	Type	typeMoteur	taille
1	Peugeot	**voiture**	Electrique	**NULL**
2	Giant	**velo**	**NULL**	M
3	Renault	**voiture**	Diesel	**NULL**
4	Trek	**velo**	**NULL**	L
5	Citroen	**voiture**	Essence	**NULL**

3.2 Stratégie JOINED

```php
#[ORM\Entity]
#[ORM\InheritanceType("JOINED")]
#[ORM\DiscriminatorColumn(name: "type", type: "string")]
#[ORM\DiscriminatorMap(["voiture" => "Voiture", "velo" => "Velo"])]
abstract class Vehicule
{
    #[ORM\Id]
    #[ORM\GeneratedValue]
    #[ORM\Column(type: "integer")]
    private int $id;

    #[ORM\Column(type: "string")]
    private string $marque;
}

#[ORM\Entity]
class Voiture extends Vehicule
{
    #[ORM\Column(type: "string")]
    private string $typeMoteur;
}

#[ORM\Entity]
class Velo extends Vehicule
{
    #[ORM\Column(type: "string")]
    private string $taille;
}
```

Code SQL

```
CREATE TABLE vehicule (
   id INT AUTO_INCREMENT NOT NULL,
   marque VARCHAR(255) NOT NULL,
   type VARCHAR(255) NOT NULL, -- Colonne discriminante
   PRIMARY KEY(id)
)
```

```
CREATE TABLE voiture (
   id INT NOT NULL,
   typeMoteur VARCHAR(255) NOT NULL,
   PRIMARY KEY(id),
   CONSTRAINT fk_ voiture_vehicule FOREIGN KEY (id) REFERENCES vehicule
(id)
)
```

```
CREATE TABLE velo (
   id INT NOT NULL,
   taille VARCHAR(255) NOT NULL,
   PRIMARY KEY(id),
   CONSTRAINT fk_velo_vehicule FOREIGN KEY (id) REFERENCES vehicule
(id)
)
```

Schéma de la base de données

Table vehicule

id	marque	type
1	Peugeot	voiture
2	Giant	velo
3	Renault	voiture
4	Trek	velo
5	Citroen	voiture

Table voiture

id	typeMoteur
1	Electrique
3	Diesel
5	Essence

Table velo

id	taille
2	M
4	L

4.Exemple de code PHP

Pour récupérer l'ensemble des véhicules en utilisant la stratégie **SINGLE_TABLE** ou **JOINED**, vous pouvez utiliser le **Repository** de l'entité de base (**Vehicule**).

Voici un exemple de code qui illustre comment faire cela :

```
// Récupération du repository de l'entité Vehicule
$vehiculeRepository = $entityManager->getRepository(Vehicule::class);
// Récupération de tous les véhicules
$vehicules = $vehiculeRepository->findAll();
// Affichage des véhicules
foreach ($vehicules as $vehicule) {
    echo $vehicule->getMarque() . "\n";
    // Vous pouvez également vérifier le type de chaque véhicule si nécessaire
    if ($vehicule instanceof Voiture) {
        echo "Type de moteur: " . $vehicule->getTypeMoteur() . "\n";
    } elseif ($vehicule instanceof Velo) {
        echo "Taille: " . $vehicule->getTaille() . "\n";
    }
}
```

Lorsque vous récupérez les entités de la base de données, Doctrine s'occupe de construire les bonnes instances d'entité en fonction des données récupérées. Pour cela il utilise le champs **type**.