

Symfony 6 : Doctrine

Achref El Mouelhi

Docteur de l'université d'Aix-Marseille
Chercheur en Programmation par contrainte (IA)
Ingénieur en Génie logiciel

elmouelhi.achref@gmail.com



Symfony

Plan

- 1 Introduction
- 2 Création et configuration d'une base de données
- 3 Entity
 - Crédit d'une entité
 - Crédit de tables associées aux entités
 - Modification entités/tables
- 4 EntityManager et Repository
 - Insertion
 - Consultation
 - Modification
 - Suppression
 - Autres méthodes d'EntityManager

Plan

5 Repository

- Query Builder
- DQL
- SQL

6 Relation entre entités

- OneToOne
- ManyToOne
- ManyToMany
- Association porteuse de données
- Relation bidirectionnelle
- Inheritance

7 Événement et méthodes callback

8 Génération d'entités à partir d'une base de données existante

Symfony

ORM : Object-Relational Mapping (lien objet-relation)

- Couche d'abstraction à la base de données
- Techniquement : { classes + annotations } permettant l'utilisation de tables d'une base de données relationnelle comme des objets
- Démarche :
 - Associer une ou plusieurs classes à chaque table
 - Associer un attribut de classe à chaque colonne de la table

Symfony

ORM : Object-Relational Mapping (lien objet-relation)

- Couche d'abstraction à la base de données
- Techniquement : { classes + annotations } permettant l'utilisation de tables d'une base de données relationnelle comme des objets
- Démarche :
 - Associer une ou plusieurs classes à chaque table
 - Associer un attribut de classe à chaque colonne de la table

Objectif

Plus de requêtes **SQL** dans l'application.

Symfony

Remarque

Plusieurs **ORM** proposés pour chaque LOO.

Symfony

Remarque

Plusieurs **ORM** proposés pour chaque LOO.

Quel choix pour **PHP** ?

- Doctrine
- pdoMap
- RedBean
- FoxORM
- ...

Symfony

Doctrine ?

- **ORM pour PHP**
- proposé en 2006 par **Konsta Vesterinen** (2.0 fin 2010)
- utilisé par **Symfony** depuis la version 1.3 (et autres comme **Zend Framework**, **CodeIgniter...**)
- inspiré par **Hibernate** : premier **ORM** du langage **Java**

© Achref

Symfony

Doctrine ?

- **ORM pour PHP**
- proposé en 2006 par **Konsta Vesterinen** (2.0 fin 2010)
- utilisé par **Symfony** depuis la version 1.3 (et autres comme **Zend Framework**, **CodeIgniter**...)
- inspiré par **Hibernate** : premier **ORM** du langage **Java**

Doctrine : deux couches

- Doctrine (ORM) qui se base sur Doctrine (DBAL)
- Doctrine (DBAL) (DataBase Abstraction Layer ou couche d'abstraction de base de données) qui se base aussi sur **PDO** pour l'abstraction d'accès aux données

Symfony

Doctrine (DBAL)

- ajoute des fonctionnalités à **PDO**
- permet de manipuler les bases de données avec des fonctions prédéfinies (pas d'utilisation du concept objet)

© Achref EL MOUADJI

Symfony

Doctrine (DBAL)

- ajoute des fonctionnalités à **PDO**
- permet de manipuler les bases de données avec des fonctions prédéfinies (pas d'utilisation du concept objet)

Doctrine (ORM)

- définit le lien entre DBAL et le monde objet
- permet de manipuler les éléments d'une base de données comme des objets

Symfony

Téléchargement (pas besoin avec **Symfony**)

- Aller dans

<http://www.doctrine-project.org/projects/orm.html>

- Télécharger la dernière version stable

Symfony

**Si on ne choisit pas la version complète à la création du projet,
on exécute la commande suivante**

```
composer require symfony/orm-pack
```

Symfony

**Si on ne choisit pas la version complète à la création du projet,
on exécute la commande suivante**

```
composer require symfony/orm-pack
```

Ensuite

```
composer require --dev symfony/maker-bundle
```

Symfony

Préparation de la chaîne de connexion

- Allez dans le fichier `.env`
- Cherchez la ligne `DATABASE_URL="mysql://db_user:db_password@127.0.0.1:3306/db_name?serverVersion=5.7&charset=utf8mb4"`
- Remplacez la par `DATABASE_URL="mysql://root:root@127.0.0.1:3306/courssymfony?serverVersion=8.0.21&charset=utf8mb4"`
- Commentez toute chaîne de connexion concernant **PostgreSQL** ou autre puis enregistrez

Symfony

Préparation de la chaîne de connexion

- Allez dans le fichier `.env`
- Cherchez la ligne `DATABASE_URL="mysql://db_user:db_password@127.0.0.1:3306/db_name?serverVersion=5.7&charset=utf8mb4"`
- Remplacez la par `DATABASE_URL="mysql://root:root@127.0.0.1:3306/courssymfony?serverVersion=8.0.21&charset=utf8mb4"`
- Commentez toute chaîne de connexion concernant **PostgreSQL** ou autre puis enregistrez

Remarque

N'oubliez pas de démarrer **WAMP** (pour démarrer le serveur de données).

Symfony

Pour créer la base de données, exécutez

```
php bin/console doctrine:database:create
```

© Achref EL MOUELHI ©

Symfony

Pour créer la base de données, exécutez

```
php bin/console doctrine:database:create
```

Ou le raccourci

```
php bin/console d:d:c
```

Symfony

Pour créer la base de données, exécutez

```
php bin/console doctrine:database:create
```

Ou le raccourci

```
php bin/console d:d:c
```

Résultat

```
Created database 'courssymfony' for connection named default
```

Symfony

Définition

- correspond à une table d'une base de données relationnelle
- est un objet contenant quelques informations indispensables pour le mapping (faire le lien) avec la base de données

© Achref EL MOKHTARI

Symfony

Définition

- correspond à une table d'une base de données relationnelle
- est un objet contenant quelques informations indispensables pour le mapping (faire le lien) avec la base de données

Informations indispensables : annotations ou attributs

- permettent de décrire les méta-données de l'entité
- sont des commentaires spéciaux (qui peuvent être générés par **Symfony** sans les écrire)

Symfony

3 étapes pour créer ou modifier une table associée à une entité

- créer ou modifier une entité
- créer une migration ⇒ générer le script **SQL**
- appliquer la migration ⇒ exécuter le script

Symfony

Pour créer une entité, exécuter la commande

```
php bin/console make:entity
```

Symfony

Pour créer une entité, exécuter la commande

```
php bin/console make:entity
```

Répondre aux questions suivantes

- Class name of the entity to create or update **par** Personne
- New property name **par** nom
- Field type **par** string
- Can this field be null in the database (nullable) **par** no
- Refaire la même chose pour prenom **et** sexe (string de longueur 1)

Symfony

Remarques

- Les types **Doctrine** sont sensibles à la casse
- La liste complète des types : <https://www.doctrine-project.org/projects/doctrine-orm/en/current/reference/basic-mapping.html#doctrine-mapping-types>

Code généré : l'attribut `$id` sera automatiquement générée

```
namespace App\Entity;

use App\Repository\PersonneRepository;
use Doctrine\ORM\Mapping as ORM;

#[ORM\Entity(repositoryClass: PersonneRepository::class)]
class Personne
{
    #[ORM\Id]
    #[ORM\GeneratedValue]
    #[ORM\Column(type: 'integer')]
    private $id;

    #[ORM\Column(type: 'string', length: 255)]
    private $nom;

    #[ORM\Column(type: 'string', length: 255)]
    private $prenom;

    #[ORM\Column(type: 'string', length: 1)]
    private $sexe;

    // + les getters et setters
}
```

Code généré : l'attribut `$id` sera automatiquement générée

```
namespace App\Entity;

use App\Repository\PersonneRepository;
use Doctrine\ORM\Mapping as ORM;

#[ORM\Entity(repositoryClass: PersonneRepository::class)]
class Personne
{
    #[ORM\Id]
    #[ORM\GeneratedValue]
    #[ORM\Column(type: 'integer')]
    private $id;

    #[ORM\Column(type: 'string', length: 255)]
    private $nom;

    #[ORM\Column(type: 'string', length: 255)]
    private $prenom;

    #[ORM\Column(type: 'string', length: 1)]
    private $sexe;

    // + les getters et setters
}
```

Allez aussi vérifier la création de `PersonneRepository` dans `src/Repository`

Symfony

Quelques attributs Doctrine

Annotation

[ORM\Entity]
[ORM\Table]
[ORM\Column]
[ORM\Id]
[ORM\GeneratedValue]
[ORM\OneToOne]
[ORM\OneToMany]
[ORM\ManyToMany]

désignation

déclare une classe **PHP** comme entité
décrit la table d'une entité persistante
définit les caractéristiques d'une colonne
marque l'identifiant de l'entité
utilisée pour générer des identifiants
entité en relation avec une seule entité
entité en relation avec plusieurs entités
entités en relation avec plusieurs entités

Symfony

Quelques attributs Doctrine

Annotation

[ORM\Entity]
[ORM\Table]
[ORM\Column]
[ORM\Id]
[ORM\GeneratedValue]
[ORM\OneToOne]
[ORM\OneToMany]
[ORM\ManyToMany]

désignation

déclare une classe **PHP** comme entité
décrit la table d'une entité persistante
définit les caractéristiques d'une colonne
marque l'identifiant de l'entité
utilisée pour générer des identifiants
entité en relation avec une seule entité
entité en relation avec plusieurs entités
entités en relation avec plusieurs entités

Liste complète

<https://www.doctrine-project.org/projects/doctrine-orm/en/current/reference/attributes-reference.html#attributes-reference>

Symfony

Les annotations/attributs **Doctrine 2** peuvent avoir des propriétés.

© Achref EL MOUELHI ©

Symfony

Les annotations/attributs **Doctrine 2** peuvent avoir des propriétés.

```
# [ORM\]Entity]
```

- repositoryClass :
 - il permet de récupérer les entités depuis la base de données
 - il a comme valeur le nom du namespace complet du repository
 - le nom du repository est composé du nom de l'entité + Repository (pour notre exemple : PersonneRepository)
- readonly : précise que cette entité est en lecture seule

Symfony

[ORM\Table]

- name : nom de la table
- indexes : tableau d'annotations @index

Symfony

[ORM\Table]

- name : nom de la table
- indexes : tableau d'annotations @index

[ORM\Column]

- type : nom du type Doctrine (obligatoire)
- name : nom de la colonne
- length : longueur pour les chaînes de caractère
- unique : pour indiquer l'unicité des valeurs de la colonnes
- nullable : pour indiquer si la valeur null est acceptée

Symfony

[ORM\GeneratedValue]

strategy : nom de la stratégie (AUTO, NONE...)

Symfony

[ORM\GeneratedValue]

strategy : nom de la stratégie (AUTO, NONE...)

Liste complète de propriétés

<https://www.doctrine-project.org/projects/doctrine-orm/en/current/reference/basic-mapping.html#property-mapping>

Symfony

Génération d'une table associée à une entité : 3 étapes

- ① Générer le script de création de la table
- ② Vérifier le script **SQL** généré avant exécution
- ③ Exécuter le script pour créer la table

Symfony

1 : Générer le script de création de la table

```
php bin/console make:migration
```

© Achref EL MOUELLI

Symfony

1 : Générer le script de création de la table

```
php bin/console make:migration
```

2 : Vérifier le script **SQL** généré avant exécution

Vérifiez le fichier VersionXXX.php dans migrations (XXX étant la date du jour)

Symfony

3 : Exécuter le script pour créer la table

```
php bin/console doctrine:migrations:migrate
```

© Achref EL MOUELHIDI

Symfony

3 : Exécuter le script pour créer la table

```
php bin/console doctrine:migrations:migrate
```

Ou le raccourci

```
php bin/console d:m:m
```

Symfony

3 : Exécuter le script pour créer la table

```
php bin/console doctrine:migrations:migrate
```

Ou le raccourci

```
php bin/console d:m:m
```

Vérifier la création de la table avec la console **MySQL** ou **phpMyAdmin**

Symfony

Modification d'une entité

- Ajouter un attribut
- Modifier le type d'un attribut
- Supprimer un attribut
- Ajouter/Modifier/Supprimer une/des contrainte(s) sur les attributs

© Archives

Symfony

Modification d'une entité

- Ajouter un attribut
- Modifier le type d'un attribut
- Supprimer un attribut
- Ajouter/Modifier/Supprimer une/des contrainte(s) sur les attributs

Exemple

Supprimons l'attribut `sexe` de la classe `Personne` ainsi que les getter et setter.

Symfony

Pour régénérer la table dans la base de données, exéutez

- `php bin/console make:migration`
- `php bin/console doctrine:migrations:migrate`

© Achref EL
Bouabene

Symfony

Pour régénérer la table dans la base de données, exéutez

- `php bin/console make:migration`
- `php bin/console doctrine:migrations:migrate`

Vérifier les modifications avec la console MySQL ou phpMyAdmin

Symfony

Pour ajouter un nouvel attribut, on peut

- exécuter la commande `php bin/console make:entity`
- préciser le nom d'une entité existante
- préciser les nouveaux attributs (comme dans le cas d'une création)
- Faire la migration

Symfony

Pour générer les getter et setter d'un attribut ajouté manuellement

```
php bin/console make:entity --regenerate
```

© Achref EL MOUADJI

Symfony

Pour générer les getter et setter d'un attribut ajouté manuellement

```
php bin/console make:entity --regenerate
```

Ou

```
php bin/console make:entity --overwrite
```

Symfony

Comment ça marche avec **Doctrine** ?

- Pour la lecture, on utilise le PersonneRepository
- Pour l'écriture, on utilise EntityManager

Symfony

Entity Manager

- un service **Doctrine** (**Doctrine** est un service **Symfony**)
- permettant la manipulation de nos entités

© Achref EL MOUELLI

Symfony

Entity Manager

- un service **Doctrine** (**Doctrine** est un service **Symfony**)
- permettant la manipulation de nos entités

Comment faire pour obtenir le service **Doctrine** ?

- Avant **Symfony 6**
 - \$doctrine = \$this->get('doctrine'); ou
 - \$doctrine = \$this->getDoctrine(); (un raccourci)
- Depuis **Symfony 5** : en injectant ManagerRegistry dans l'action du contrôleur

Symfony

Comment faire pour obtenir le service **Entity Manager** ?

- Avant **Symfony 6**

- \$em = \$this->getDoctrine()->getManager(); ou
- \$em = \$this->get('doctrine.orm.entity_manager');

- Depuis **Symfony 5**

- \$em = \$doctrine->getManager(); ou
- en injectant EntityManagerInterface dans l'action du contrôleur

Symfony

Comment faire pour obtenir le service **Repository** ?

- Avant **Symfony 6** :

```
$em = $this->getDoctrine()->getRepository(EntityName::class);
```

- Depuis **Symfony 5** : en injectant EntityNameRepository dans l'action du contrôleur

Symfony

Pour tester, créons un contrôleur PersonneController

```
php bin/console make:controller Personne
```

Symfony

Contenu de PersonneController.php

```
namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;

class PersonneController extends AbstractController
{
    #[Route('/personne', name: 'app_personne')]
    public function index(): Response
    {
        return $this->render('personne/index.html.twig', [
            'controller_name' => 'PersonneController',
        ]);
    }
}
```

Symfony

Pour personne/index.html.twig, considérons le contenu suivant

```
{% extends 'base.html.twig' %}

{% block title %}Hello PersonneController!{% endblock %}

{% block body %}
    <h1>Hello
        {{ controller_name }}!
    </h1>
    {% if personne is defined %}
        Personne
        {{ adjetif }} :
        {{ personne.id }}
        {{ personne.prenom }}
        {{ personne.nom }}
    {% endif %}
{% endblock %}
```

Symfony

Pour ajouter un tuple dans la table Personne

```
namespace App\Controller;

use App\Entity\Personne;
use Doctrine\Persistence\ManagerRegistry;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;

class PersonneController extends AbstractController
{
    #[Route('/personne/add', name: 'personne_add')]
    public function index(ManagerRegistry $doctrine): Response
    {
        $entityManager = $doctrine->getManager();
        $personne = new Personne();
        $personne->setNom('Wick');
        $personne->setPrenom('John');
        $entityManager->persist($personne);
        $entityManager->flush();
        return $this->render('personne/index.html.twig', [
            'controller_name' => 'PersonneController',
            'personne' => $personne,
            'adjectif' => 'ajoutée'
        ]);
    }
}
```

Explication

- `$entityManager->persist($personne);` : informe **Doctrine** que l'on veut ajouter cet objet dans la base de données.
- `$em->flush();` : permet d'exécuter la requête et d'envoyer tout ce qui a été persisté avant à la base de données.

Explication

- `$entityManager->persist ($personne) ;` : informe **Doctrine** que l'on veut ajouter cet objet dans la base de données.
- `$em->flush () ;` : permet d'exécuter la requête et d'envoyer tout ce qui a été persisté avant à la base de données.

Utilisez le profiler pour mieux comprendre le fonctionnement

- Allez à la page http://localhost:8000/_profiler/14d964?panel=db
- Vérifiez la présence de 3 Queries

Explication

- `$entityManager->persist ($personne) ;` : informe **Doctrine** que l'on veut ajouter cet objet dans la base de données.
- `$em->flush () ;` : permet d'exécuter la requête et d'envoyer tout ce qui a été persisté avant à la base de données.

Utilisez le profiler pour mieux comprendre le fonctionnement

- Allez à la page http://localhost:8000/_profiler/14d964?panel=db
- Vérifiez la présence de 3 Queries

Symfony utilise les transactions pour les opérations sur une base de données

- START TRANSACTION
- INSERT INTO ...
- COMMIT

Symfony

Pour vérifier l'insertion de données depuis PowerShell

```
php bin/console dbal:run-sql 'SELECT * FROM personne'
```

© Achref EL MOUADJI

Symfony

Pour vérifier l'insertion de données depuis PowerShell

```
php bin/console dbal:run-sql 'SELECT * FROM personne'
```

Pour vérifier l'insertion de données depuis Invite de commandes

```
php bin/console dbal:run-sql "SELECT * FROM personne"
```

Symfony

On peut aussi injecter le gestionnaire d'entité dans l'action

```
namespace App\Controller;

use App\Entity\Personne;
use Doctrine\ORM\EntityManagerInterface;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;

class PersonneController extends AbstractController
{
    #[Route('/personne/add', name: 'personne_add')]
    public function index(EntityManagerInterface $entityManager): Response
    {
        $personne = new Personne();
        $personne->setNom('Dalton');
        $personne->setPrenom('Jack');
        $entityManager->persist($personne);
        $entityManager->flush();
        return $this->render('personne/index.html.twig', [
            'controller_name' => 'PersonneController',
            'personne' => $personne,
            'adjectif' => 'ajoutée'
        ]);
    }
}
```

Pour vérifier si les valeurs sont valides avant insertion, on peut utiliser ValidatorInterface

```
namespace App\Controller;

use App\Entity\Personne;
use Doctrine\ORM\EntityManagerInterface;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;
use Symfony\Component\Validator\Validator\ValidatorInterface;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;

class PersonneController extends AbstractController
{
    #[Route('/personne/add', name: 'personne_add')]
    public function index(EntityManagerInterface $entityManager, ValidatorInterface
        $validator): Response
    {
        $personne = new Personne();
        $personne->setNom('Wick');
        $personne->setPrenom('John');
        $errors = $validator->validate($personne);
        if (count($errors) > 0) {
            return new Response((string) $errors, 400);
        }
        $entityManager->persist($personne);
        $entityManager->flush();
        return $this->render('personne/index.html.twig', [
            'controller_name' => 'PersonneController',
            'personne' => $personne,
            'adjectif' => 'ajoutée'
        ]);
    }
}
```

Symfony

**Si on ne choisit pas la version complète à la création du projet
composer require symfony/validator**

Symfony

Contraintes vérifiées par **Symfony**

- Type
- NotNull
- UniqueEntity
- Length

Symfony

Quatre méthodes prédéfinies pour la recherche

- `find` : cherche et retourne un seul tuple selon la clé primaire
- `findOneBy` : cherche et retourne un seul tuple selon les colonnes données en paramètre dans un tableau associatif
- `findBy` : cherche et retourne plusieurs tuples selon les colonnes données en paramètre dans un tableau associatif
- `findAll` : retourne tous les tuples de la table.

Symfony

Exemple avec `find` (action à placer après `index`)

```
#Route('/personne/{id}', name: 'personne_show')
public function showPersonne(int $id, PersonneRepository
    $personneRepository)
{
    $personne = $personneRepository->find($id);

    if (!$personne) {
        throw $this->createNotFoundException(
            'Personne non trouvée avec l\'id ' . $id
        );
    }

    return $this->render('personne/index.html.twig', [
        'controller_name' => 'PersonneController',
        'personne' => $personne,
        'adjectif' => 'recherchée'
    ]);
}
```

Symfony

Rien à changer dans personne/index.html.twig

```
{% extends 'base.html.twig' %}

{% block title %}Hello PersonneController!{% endblock %}

{% block body %}
    <h1>Hello
        {{ controller_name }}!
    </h1>
    {% if personne is defined %}
        Personne
        {{ adjetif }} :
        {{ personne.id }}
        {{ personne.prenom }}
        {{ personne.nom }}
    {% endif %}
{% endblock %}
```

Symfony

Exemple avec **findOneBy** (action à placer après showPersonne)

```
# [Route('/personne/{nom}/{prenom}', name: 'personne_show_one')]
public function showPersonneByNomAndPrenom(string $nom, string $prenom,
    PersonneRepository $personneRepository)
{
    $personne = $personneRepository->findOneBy([
        "nom" => $nom,
        "prenom" => $prenom
    ]);

    if (!$personne) {
        throw $this->createNotFoundException('Personne non trouvée');
    }

    return $this->render('personne/index.html.twig', [
        'controller_name' => 'PersonneController',
        'personne' => $personne,
        'adjectif' => 'recherchée'
    ]);
}
```

Symfony

Exemple avec `findAll` (action à placer après `index`)

```
# [Route('/personne/show', name: 'personne_show_all')]
public function showAllPersonne(PersonneRepository $personneRepository)
{
    $personnes = $personneRepository->findAll();

    if (!$personnes) {
        throw $this->createNotFoundException('La table est vide');
    }

    return $this->render('personne/show.html.twig', [
        'controller_name' => 'PersonneController',
        'personnes' => $personnes,
    ]);
}
```

Symfony

Contenu de show.html.twig

```
{% extends 'base.html.twig' %}

{% block title %}Hello PersonneController!
{% endblock %}

{% block body %}

<h1>Hello {{ controller_name }}!</h1>
<ul>
    {% for personne in personnes %}
        <li>{{ personne.prenom }} {{ personne.nom }}</li>
    {% endfor %}
</ul>

{% endblock %}
```

Symfony

Avant de modifier une personne, il faut la récupérer avec personneRepository

```
# [Route('/personne/edit/{id}', name: 'personne_update')]
public function updatePersonne(int $id, EntityManagerInterface
    $entityManager)
{
    $personne = $entityManager->getRepository(Personne::class)->find(
        $id);
    if (!$personne) {
        throw $this->createNotFoundException(
            'Personne non trouvée avec l\'id ' . $id
        );
    }
    $personne->setNom('Travolta');
    $entityManager->flush();
    return $this->render('personne/index.html.twig', [
        'controller_name' => 'PersonneController',
        'personne' => $personne,
        'adjectif' => 'modifiée'
    ]);
}
```

Symfony

Symfony nous permet de récupérer l'objet personne dont l'identifiant est passé en paramètre dans la barre d'adresse

```
# [Route('/personne/edit/{id}', name: 'personne_update')]
public function updatePersonne(Personne $personne,
    EntityManagerInterface $entityManager)
{
    if (!$personne) {
        throw $this->createNotFoundException(
            'Personne non trouvée avec l\'id ' . $personne->$personne
                ->getId()
        );
    }
    $personne->setNom('Abruzzi');
    $entityManager->flush();
    return $this->render('personne/index.html.twig', [
        'controller_name' => 'PersonneController',
        'personne' => $personne,
        'adjectif' => 'modifiée'
    ]);
}
```

Symfony

Pour supprimer une personne, il faut aussi la récupérer avant avec personneRepository

```
# [Route('/personne/delete/{id}', name: 'personne_delete')]
public function deletePersonne(int $id, EntityManagerInterface
    $entityManager)
{
    $personne = $entityManager->getRepository(Personne::class)
        ->find($id);
    if (!$personne) {
        throw $this->createNotFoundException(
            'Personne non trouvée avec l\'id ' . $id
        );
    }
    $entityManager->remove($personne);
    $entityManager->flush();
    return $this->redirectToRoute("personne_show_all");
}
```

Symfony

Ou aussi

```
# [Route('/personne/delete/{id}', name: 'personne_delete')]
public function deletePersonne(Personne $personne,
    EntityManagerInterface $entityManager)
{
    if (!$personne) {
        throw $this->createNotFoundException(
            'Personne non trouvée avec l\'id ' . $personne->
                getId()
        );
    }
    $entityManager->remove($personne);
    $entityManager->flush();
    return $this->redirectToRoute("personne_show_all");
}
```

Symfony

Autres méthodes

- `clear()` : annule tous les persist effectués par l'EntityManager
- `detach($entity)` : annule le persist effectué par l'EntityManager sur \$entity
- `refresh($entity)` : remet à jour l'entité en argument par les valeurs de la base de données. Les nouvelles modifications sur cette entité seront perdues.
- `contains($entity)` : retourne true si \$entity est gérée par EntityManager.

Symfony

Repository

- classe **PHP**
- contenant les méthodes de récupération de données relatives à nos entités
- pouvant être utilisé pour
 - définir des nouvelles méthodes
 - personnaliser des méthodes existantes

Symfony

La méthode findBy()

Elle peut prendre plusieurs paramètres

- un tableau de contraintes pour le `where`, obligatoire
- un tableau de contraintes pour le `orderBy`, par défaut
- une valeur pour `limit`, par défaut
- une valeur pour `offset`, par défaut

Exemple avec findBy()

```
# [Route('/personne/show/{nom}/{prenom}/{number}', name: 'personne_show_some')]
public function showSomePersonne(string $nom, string $prenom, int
    $number, PersonneRepository $personneRepository)
{
    $personnes = $personneRepository->findBy(
        [
            "nom" => $nom,
            "prenom" => $prenom
        ],
        ["nom" => "ASC"],
        $number,
        1
    );
    if (!$personnes) {
        throw $this->createNotFoundException('Aucun résultat trouvé');
    }
    return $this->render('personne/show.html.twig', [
        'controller_name' => 'PersonneController',
        'personnes' => $personnes,
    ]);
}
```

Symfony

Les méthodes magiques

- `findByAttribut ($valeur)` : retourne un tableau de tous les tuples dont Attribut **a** comme valeur `$valeur`)
- `findOneByAttribut ($valeur)` : retourne un seul tuple dont Attribut **a** comme valeur `$valeur`

© Achref EL MOUADJI

Symfony

Les méthodes magiques

- `findByAttribut ($valeur)` : retourne un tableau de tous les tuples dont Attribut **a** comme valeur `$valeur`)
- `findOneByAttribut ($valeur)` : retourne un seul tuple dont Attribut **a** comme valeur `$valeur`

Pour notre entité Personne, on peut avoir

- `findByNom ()`
- `findByPrenom ()`
- `findOneByNom ()`
- `findOneByPrenom ()`

Symfony

Query Builder

- Dans la classe `PersonneRepository`, on définit une méthode qui
 - utilise un objet `QueryBuilder` : on l'obtient avec la méthode `createQueryBuilder()` de l'`EntityManager` et on l'utilise pour construire la requête
 - récupère l'objet `Query` de `QueryBuilder`
 - récupère les résultats de la `Query`
- retourne le résultat

Symfony

Définissons notre méthode dans la classe PersonneRepository

```
public function findOneByNomAndPrenom(string $nom,  
    string $prenom)  
{  
    $queryBuilder = $this->createQueryBuilder('p')  
        ->where('p.nom = :nom')  
        ->setParameter('nom', $nom)  
        ->andWhere('p.prenom = :prenom')  
        ->setParameter('prenom', $prenom);  
    $query = $queryBuilder->getQuery();  
    $result = $query->setMaxResults(1)->  
        getOneOrNullResult();  
    return $result;  
}
```

Symfony

Modifications showPersonneByNomAndPrenom **dans** PersonneController

```
# [Route('/personne/{nom}/{prenom}', name: 'personne_show_one')]
public function showPersonneByNomAndPrenom(string $nom, string $prenom,
    PersonneRepository $personneRepository)
{
    $personne = $personneRepository->findOneByNomAndPrenom($nom,
        $prenom);

    if (!$personne) {
        throw $this->createNotFoundException('Personne non trouvée');
    }

    return $this->render('personne/index.html.twig', [
        'controller_name' => 'PersonneController',
        'personne' => $personne,
        'adjectif' => 'recherchée'
    ]);
}
```

Symfony

Autres méthodes de Query

- `getResult()` : Exécute la requête et retourne le résultat sous forme d'un tableau d'objets (même quand il s'agit d'un seul objet)
- `getArrayResult()` : Exécute la requête et retourne le résultat sous forme d'un tableau de tableaux
- `getScalarResult()` : Exécute la requête et retourne le résultat sous forme d'une valeur (à utiliser lorsque la requête retourne une unique valeur)
- `getOneOrNullResult()` : Exécute la requête et retourne un seul objet ou une valeur null
- **Plusieurs autres** : `getSingleResult()`, `getSingleScalarResult()` ...
- `execute()` : à utiliser pour exécuter des requêtes insert, update, delete ou select

Symfony

DQL

- Langage de requêtes adapté à **Doctrine**
- Contrairement à `QueryBuilder`, **DQL** permet d'écrire des requêtes sous forme de chaînes de caractères

© Achref EL MOUADJI

Symfony

DQL

- Langage de requêtes adapté à **Doctrine**
- Contrairement à `QueryBuilder`, **DQL** permet d'écrire des requêtes sous forme de chaînes de caractères

Remarques

- Pas besoin de `QueryBuilder` pour construire les requêtes
- Par contre, on doit passer par la méthode `createQuery()`
- Et on a toujours besoin de `Query` pour récupérer les résultats

Symfony

Modifions la méthode `findOneByNomAndPrenom` de la PersonneRepository et utilisons DQL

```
public function findOneByNomAndPrenom(string $nom, string
    $prenom)
{
    $entityManager = $this->getEntityManager();

    $query = $entityManager->createQuery(
        'SELECT p
        FROM App\Entity\Personne p
        WHERE p.nom = :nom
        and p.prenom = :prenom'
    )->setParameter('nom', $nom)
        ->setParameter('prenom', $prenom);
    $result = $query->setMaxResults(1)->getOneOrNullResult();
    return $result;
}
```

Symfony

La méthode `showPersonneByNomAndPrenom` de `PersonneController` reste inchangée

```
# [Route('/personne/{nom}/{prenom}', name: 'personne_show_one')]
public function showPersonneByNomAndPrenom(string $nom, string $prenom,
    PersonneRepository $personneRepository)
{
    $personne = $personneRepository->findOneByNomAndPrenom($nom,
        $prenom);

    if (!$personne) {
        throw $this->createNotFoundException('Personne non trouvée');
    }

    return $this->render('personne/index.html.twig', [
        'controller_name' => 'PersonneController',
        'personne' => $personne,
        'adjectif' => 'recherchée'
    ]);
}
```

Tester une requête DQL avec la Console

```
php bin/console doctrine:query:dql "requêteDQL"
```

© Achref EL MOUADJI

Symfony

Tester une requête DQL avec la Console

```
php bin/console doctrine:query:dql "requêteDQL"
```

Exemple

```
php bin/console doctrine:query:dql "SELECT p FROM  
App\Entity\Personne p"
```

Symfony

Remarques

- La requête **DQL** précédente permet de sélectionner un objet
- Il est tout de même possible de sélectionner seulement quelques attributs d'un objet
- Dans ce cas, le résultat est un tableau contenant les champs sélectionnés
- Cependant, il est impossible de modifier (ou supprimer) les valeurs de ces attributs sélectionnées

Symfony

Modifions la méthode findOneByNomAndPrenom **de la** PersonneRepository **et utilisons** SQL

```
public function findOneByNomAndPrenom(string $nom, string $prenom)
{
    $entityManager = $this->getEntityManager();

    $query = $entityManager->getConnection()->prepare(
        'SELECT *
         FROM personne
         WHERE nom = :nom
           and prenom = :prenom'
    );
    $result = $query->executeQuery([
        'nom' => $nom,
        "prenom" => $prenom
    ]);
    return $result->fetchAllAssociative()[0];
}
```

Symfony

Quatre (ou trois) relations possibles

- OneToOne : chaque objet d'une première classe est en relation avec un seul objet de la deuxième classe
- OneToMany : chaque objet d'une première classe peut être en relation avec plusieurs objets de la deuxième classe (la réciproque est ManyToOne)
- ManyToMany : chaque objet d'une première classe peut être en relation avec plusieurs objets de la deuxième classe et inversement

Symfony

Pour la suite

- Créons une entité **Adresse** avec la commande `php bin/console make:entity`
- Cette entité a trois attributs :
 - `rue` (`string de taille 30`),
 - `codePostal` (`string de taille 5`) et
 - `ville` (`string de taille 30`)

Symfony

Contenu de l'entité Adresse

```
namespace App\Entity;

use App\Repository\AdresseRepository;
use Doctrine\ORM\Mapping as ORM;

#[ORM\Entity(repositoryClass: AdresseRepository::class)]
class Adresse
{
    #[ORM\Id]
    #[ORM\GeneratedValue]
    #[ORM\Column(type: 'integer')]
    private $id;

    #[ORM\Column(type: 'string', length: 255)]
    private $rue;

    #[ORM\Column(type: 'string', length: 255)]
    private $codePostal;

    #[ORM\Column(type: 'string', length: 255)]
    private $ville;
```

Symfony

Pour ajouter Adresse dans Personne

- exédez la commande `php bin/console make:entity`
- répondez à Class name of the entity to create or update par Personne
- répondez à New property name par adresse
- répondez à Field type par OneToOne
- répondez à What class should this entity be related to? par Adresse
- répondez à Is the Personne.adresse property allowed to be null (nullable)? par yes
- répondez à Do you want to add a new property to Adresse so that you can access/update Personne par no
- cliquez sur entrée pour répondre à Add another property?

Nouveau contenu de Personne

```
namespace App\Entity;

use App\Repository\PersonneRepository;
use Doctrine\ORM\Mapping as ORM;

#[ORM\Entity(repositoryClass: PersonneRepository::class)]
class Personne
{
    ...

    #[ORM\OneToOne(targetEntity: Adresse::class, cascade: ['persist', 'remove'])]
    #[ORM\JoinColumn(nullable: true)]
    private $adresse;
```

Nouveau contenu de Personne

```
namespace App\Entity;

use App\Repository\PersonneRepository;
use Doctrine\ORM\Mapping as ORM;

#[ORM\Entity(repositoryClass: PersonneRepository::class)]
class Personne
{
    ...

    #[ORM\OneToOne(targetEntity: Adresse::class, cascade: ['persist', 'remove'])]
    #[ORM\JoinColumn(nullable: true)]
    private $adresse;
```



Notation

- Personne : entité propriétaire
- Adresse : entité inverse

Symfony

```
# [ORM\OneToOne(targetEntity: Adresse::class, cascade: ['persist', 'remove'])]
```

© Achref EL MOUELHI ©

Symfony

```
# [ORM\OneToOne(targetEntity: Adresse::class, cascade: ['persist', 'remove'])]
```

Explication

- targetEntity : namespace complet vers l'entité liée.
- cascade : permet de cascader les opérations comme persist, remove qu'on peut faire de l'entité propriétaire à l'entité inverse.

Symfony

```
# [ORM\OneToOne(targetEntity: Adresse::class, cascade: ['persist', 'remove'])]
```

Explication

- targetEntity : namespace complet vers l'entité liée.
- cascade : permet de cascader les opérations comme persist, remove qu'on peut faire de l'entité propriétaire à l'entité inverse.

Pour indiquer que chaque personne doit avoir une adresse

```
# [ORM\JoinColumn(nullable: false)]
```

Symfony

Avant de réaliser la migration, supprimons la valeur persist de la propriété cascade dans l'entité Personne

```
namespace App\Entity;

use App\Repository\PersonneRepository;
use Doctrine\ORM\Mapping as ORM;

#[ORM\Entity(repositoryClass: PersonneRepository::class)]
class Personne
{
    ...

    #[ORM\OneToOne(targetEntity: Adresse::class, cascade: ['remove'])]
    #[ORM\JoinColumn(nullable: true)]
    private $adresse;

    ...
}
```

Symfony

Pour régénérer la table dans la base de données, exécutez

```
php bin/console make:migration
```

Symfony

Pour régénérer la table dans la base de données, exéutez

```
php bin/console make:migration
```

Ensuite

```
php bin/console doctrine:migrations:migrate
```

Symfony

Pour régénérer la table dans la base de données, exéutez

```
php bin/console make:migration
```

Ensuite

```
php bin/console doctrine:migrations:migrate
```

Vérifier les modifications avec la console MySQL ou phpMyAdmin

Modifions addPersonne pour ajouter une personne avec une adresse

```
# [Route('/personne/add', name: 'personne_add')]
public function index(EntityManagerInterface $entityManager,
    ValidatorInterface $validator): Response
{
    $adresse = new Adresse();
    $adresse->setRue('paradis');
    $adresse->setVille('Marseille');
    $adresse->setCodePostal('13015');
    $entityManager->persist($adresse);

    $personne = new Personne();
    $personne->setNom('Wick');
    $personne->setPrenom('John');
    $personne->setAdresse($adresse);
    $entityManager->persist($personne);
    $entityManager->flush();

    return $this->render('personne/index.html.twig', [
        'controller_name' => 'PersonneController',
        'personne' => $personne,
        'adjectif' => 'ajoutée'
    ]);
}
```

Symfony

Ajoutons la valeur persist à la propriété cascade dans l'entité Personne pour éviter de persister chaque fois l'adresse avant la personne

```
namespace App\Entity;

use App\Repository\PersonneRepository;
use Doctrine\ORM\Mapping as ORM;

#[ORM\Entity(repositoryClass: PersonneRepository::class)]
class Personne
{
    ...

    #[ORM\OneToOne(targetEntity: Adresse::class, cascade: ['persist', 'remove'])]
    #[ORM\JoinColumn(nullable: true)]
    private $adresse;

    ...
}
```

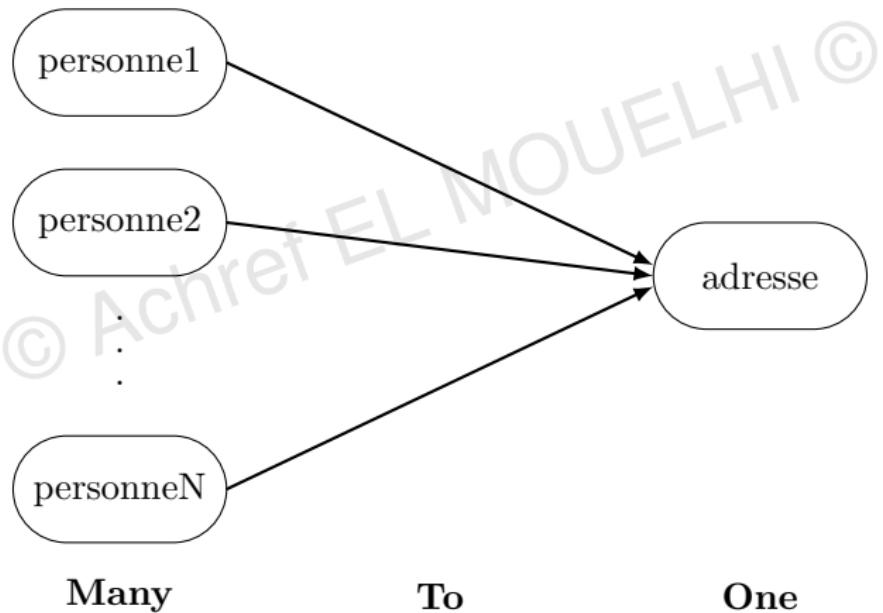
Ainsi pour ajouter une personne

```
# [Route('/personne/add', name: 'personne_add')]
function addPersonne(EntityManagerInterface $entityManager)
{
    $adresse = new Adresse();
    $adresse->setRue('paradis');
    $adresse->setVille('Marseille');
    $adresse->setCodePostal('13015');
    $personne = new Personne();
    $personne->setNom('Wick');
    $personne->setPrenom('John');
    $personne->setAdresse($adresse);
    $entityManager->persist($personne);
    $entityManager->flush();
    return $this->render('personne/index.html.twig', [
        'controller_name' => 'PersonneController',
        'personne' => $personne,
        'adjectif' => 'ajoutée'
    ]);
}
```

Symfony

Exemple

Si plusieurs personnes pouvaient avoir la même adresse.



Symfony

Il suffit de changer

```
# [ORM\ManyToOne(targetEntity: Adresse::class, cascade: ['persist', 'remove'])]
# [ORM\JoinColumn(nullable: true)]
```

Symfony

Pour régénérer la table dans la base de données, exécutez

```
php bin/console make:migration
```

Symfony

Pour régénérer la table dans la base de données, exéutez

```
php bin/console make:migration
```

Ensuite

```
php bin/console doctrine:migrations:migrate
```

Symfony

Pour régénérer la table dans la base de données, exéutez

```
php bin/console make:migration
```

Ensuite

```
php bin/console doctrine:migrations:migrate
```

Vérifier les modifications avec la console MySQL ou phpMyAdmin

Pour tester

```
/* adresse */
$adresse = new Adresse();
$adresse->setRue('paradis');
$adresse->setVille('Marseille');
$adresse->setCodePostal('13015');

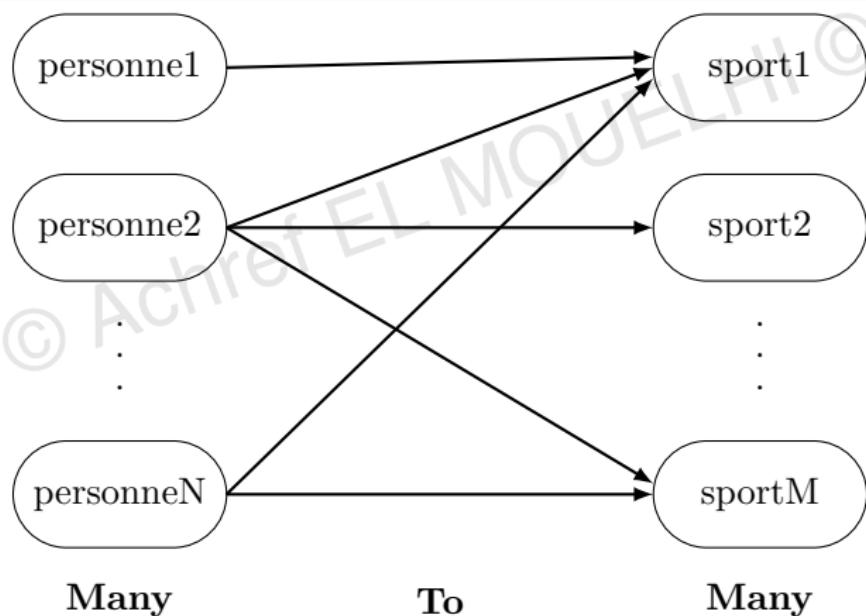
/* première personne */
$personne = new Personne();
$personne->setNom('Cohen');
$personne->setPrenom('Sophie');
$personne->setAdresse($adresse);

/* deuxième personne */
$personne2 = new Personne();
$personne2->setNom('Wolf');
$personne2->setPrenom('Bob');
$personne2->setAdresse($adresse);

/* persistance de données */
$entityManager->persist($personne);
$entityManager->persist($personne2);
$entityManager->flush();
```

Exemple

- Une personne peut pratiquer plusieurs sports
- Un sport peut être pratiqué par plusieurs personnes



Symfony

Démarche

- On commence par créer une entité Sport avec un seul attribut name
- On définit la relation ManyToMany (exactement comme pour les deux relations précédentes) soit dans Personne soit dans Sport

Symfony

Contenu de l'entité Sport

```
namespace App\Entity;

use App\Repository\SportRepository;
use Doctrine\ORM\Mapping as ORM;

#[ORM\Entity(repositoryClass: SportRepository::class)]
class Sport
{
    #[ORM\Id]
    #[ORM\GeneratedValue]
    #[ORM\Column(type: 'integer')]
    private $id;

    #[ORM\Column(type: 'string', length: 255)]
    private $name;

    // + getters/setters...
}
```

Symfony

Pour ajouter Sport dans Personne

- exécutez la commande `php bin/console make:entity`
- répondez à Class name of the entity to create or update **par** Personne
- répondez à New property name **par** sports
- répondez à Field type **par** ManyToMany
- répondez à What class should this entity be related to? **par** Sport
- répondez à Is the Personne.sports property allowed to be null (nullable)? **par** yes
- répondez à Do you want to add a new property to Sport so that you can access/update Personne **par** no
- cliquez sur entrée pour répondre à Add another property?

Symfony

Nouveau contenu de l'entité Personne

```
namespace App\Entity;

use App\Repository\PersonneRepository;
use Doctrine\Common\Collections\ArrayCollection;
use Doctrine\Common\Collections\Collection;
use Doctrine\ORM\Mapping as ORM;

#[ORM\Entity(repositoryClass: PersonneRepository::class)]
class Personne
{
    // les attributs précédents

    #[ORM\ManyToOne(targetEntity: Adresse::class, cascade: ['persist', 'remove'])]
    #[ORM\JoinColumn(nullable: true)]
    private $adresse;

    #[ORM\ManyToMany(targetEntity: Sport::class)]
    private $sports;

    public function __construct()
    {
        $this->sports = new ArrayCollection();
    }

    // + getters/setters...
}
```

Symfony

Ajoutons la propriété cascade pour sports

```
namespace App\Entity;

use App\Repository\PersonneRepository;
use Doctrine\Common\Collections\ArrayCollection;
use Doctrine\Common\Collections\Collection;
use Doctrine\ORM\Mapping as ORM;

#[ORM\Entity(repositoryClass: PersonneRepository::class)]
class Personne
{
    // les attributs précédents

    #[ORM\ManyToOne(targetEntity: Adresse::class, cascade: ['persist', 'remove'])]
    #[ORM\JoinColumn(nullable: true)]
    private $adresse;

    #[ORM\ManyToMany(targetEntity: Sport::class, cascade: ['persist', 'remove'])]
    private $sports;

    public function __construct()
    {
        $this->sports = new ArrayCollection();
    }

    // + getters/setters...
}
```

Symfony

Pour régénérer les tables dans la base de données, exécutez

```
php bin/console make:migration
```

© Achref EL MOUELHID

Symfony

Pour régénérer les tables dans la base de données, exéutez

```
php bin/console make:migration
```

Ensuite

```
php bin/console doctrine:migrations:migrate
```

Symfony

Pour régénérer les tables dans la base de données, exéutez

```
php bin/console make:migration
```

Ensuite

```
php bin/console doctrine:migrations:migrate
```

Vérifier les modifications avec la console MySQL ou phpMyAdmin

Symfony

Pour tester

```
$sport = new Sport();
$sport->setName('Football');

$sport2 = new Sport();
$sport2->setName('Tennis');

$personne = new Personne();
$personne->setNom('Dalton');
$personne->setPrenom('Jack');
$personne->addSport($sport);
$personne->addSport($sport2);

$personne2 = new Personne();
$personne2->setNom('Benamar');
$personne2->setPrenom('Karim');
$personne2->addSport($sport);

$entityManager->persist($personne);
$entityManager->persist($personne2);
$entityManager->flush();
```

Symfony

Si l'association est porteuse de données

- Par exemple : la relation (`ArticleCommande`) entre `Commande` et `Article`
- Pour chaque article d'une commande, il faut préciser la quantité commandée.

© Achref EL MOUELLI

Symfony

Si l'association est porteuse de données

- Par exemple : la relation (`ArticleCommande`) entre `Commande` et `Article`
- Pour chaque article d'une commande, il faut préciser la quantité commandée.

Solution

- Créer trois entités `Article`, `Commande` et `ArticleCommande`
- Définir la relation `OneToMany` entre `Article` et `ArticleCommande`
- Définir la relation `ManyToOne` entre `ArticleCommande` et `Commande`
- La relation `OneToMany` est l'inverse de `ManyToOne`

Symfony

Remarques

- Les relations, qu'on a étudiées, sont unidirectionnelles
- C'est à dire on peut faire `$personne->getAdresse ()`
- Mais on ne peut faire `$adresse->getPersonne ()`

© Achref EL MOUADJI

Symfony

Remarques

- Les relations, qu'on a étudiées, sont unidirectionnelles
- C'est à dire on peut faire `$personne->getAdresse ()`
- Mais on ne peut faire `$adresse->getPersonne ()`

Solution

Rendre les relations bidirectionnelles

Symfony

Remarques

- Les relations, qu'on a étudiées, sont unidirectionnelles
- C'est à dire on peut faire `$personne->getAdresse ()`
- Mais on ne peut faire `$adresse->getPersonne ()`

Solution

Rendre les relations bidirectionnelles

Avant de commencer

Supprimons tout ce qui concerne `Adresse` dans `Personne`.

Symfony

Démarche

- Exécutez la commande `php bin/console make:entity`
- Class name of the entity to create or update : **Personne**
- New property name : **adresse**
- Field type : **ManyToOne**
- What class should this entity be related to : **Adresse**
- Is the Personne.adresse property allowed to be null (nullable) ?
(yes/no) [yes] : : **yes**
- Do you want to add a new property to Adresse so that you can access/update Personne objects from it - e.g.
`$adresse->getPersonnes() m?` (yes/no) [yes] : : **yes**
- New field name inside Adresse [personnes] : : **personnes**
- Cliquez sur **entrez**

Contenu d'Adresse.php

```
namespace App\Entity;

use App\Repository\AdresseRepository;
use Doctrine\Common\Collections\ArrayCollection;
use Doctrine\Common\Collections\Collection;
use Doctrine\ORM\Mapping as ORM;

#[ORM\Entity(repositoryClass: AdresseRepository::class)]
class Adresse
{
    ...

    #[ORM\OneToMany(mappedBy: 'adresse', targetEntity: Personne::class)]
    private $personnes;

    public function __construct()
    {
        $this->personnes = new ArrayCollection();
    }

    ...
}
```

mappedBy fait référence à l'attribut adresse dans la classe Personne

Contenu de Personne.php

```
namespace App\Entity;

use App\Repository\PersonneRepository;
use Doctrine\Common\Collections\ArrayCollection;
use Doctrine\Common\Collections\Collection;
use Doctrine\ORM\Mapping as ORM;

#[ORM\Entity(repositoryClass: PersonneRepository::class)]
class Personne
{
    ...

    #[ORM\ManyToOne(targetEntity: Adresse::class, inversedBy: 'personnes')]
    private $adresse;

    public function __construct()
    {
        $this->sports = new ArrayCollection();
    }

    ...
}
```

inversedBy fait référence à l'attribut personnes dans la classe Adresse

Symfony

Ainsi, on peut faire :

```
$personne = new Personne();
$personne->setNom('Wick');
$personne->setPrenom('John');
$adresse = new Adresse();
$adresse->setRue('10 rue de Lyon');
$adresse->setVille('Marseille');
$adresse->setCodePostal(13015);
$adresse->addPersonne($personne);
dd($personne->getAdresse());
```

Symfony

Exercice

Écrire un code qui permet d'insérer une adresse dans la base de données appartenant à deux personnes

- la première étant une personne qui est déjà dans la base de données.
- la deuxième est une nouvelle qui n'existe pas.

Symfony

Deux possibilités avec l'héritage

- SINGLE_TABLE
- JOINED

© Achref EL MOUADJI

Symfony

Deux possibilités avec l'héritage

- SINGLE_TABLE
- JOINED

Exemple

- Une classe mère Personne
- Deux classes filles Etudiant et Enseignant

Symfony

Démarche

- On commence par créer deux entités Etudiant et Enseignant
- L'entité Etudiant a un seul attribut niveau de type string (de longueur 30)
- L'entité Enseignant a un seul attribut salaire de type integer

Symfony

Pour indiquer comment transformer les classes mère et filles en tables

Il faut utiliser # [ORM\InheritanceType]

© Achref EL MOUSSA

Symfony

Pour indiquer comment transformer les classes mère et filles en tables

Il faut utiliser `# [ORM\InheritanceType]`

Il faut aussi indiquer la solution choisie pour l'héritage

Dans la classe mère on ajoute

`# [ORM\InheritanceType ("SINGLE_TABLE")]`

Symfony

Exemple

Et pour distinguer étudiant, enseignant et personne

- `# [ORM\DiscriminatorColumn(name: 'type', type: 'string')]
dans la classe mère,`
- `# [ORM\DiscriminatorMap(['personne' => 'Personne',
'etudiant' => 'Etudiant', 'enseignant' => 'Enseignant'])]`



Symfony

Exemple

Et pour distinguer étudiant, enseignant et personne

- `# [ORM\DiscriminatorColumn(name: 'type', type: 'string')]`
dans la classe mère,
- `# [ORM\DiscriminatorMap(['personne' => 'Personne', 'etudiant' => 'Etudiant', 'enseignant' => 'Enseignant'])]`



Dans la table personne, on aura une colonne type qui aura comme valeur soit personne, soit etudiant soit enseignant.

Symfony

La classe Personne

```
# [ORM\InheritanceType('SINGLE_TABLE')]  
#[ORM\Entity(repositoryClass: PersonneRepository::class)]  
#[ORM\DiscriminatorColumn(name: 'type', type: 'string')]  
#[ORM\DiscriminatorMap(['personne' => 'Personne', 'etudiant' => 'Etudiant', 'enseignant' => 'Enseignant'])]  
class Personne {  
  
    // + tout le code précédent  
  
}
```

La classe Etudiant

```
#[ORM\Entity(repositoryClass:  
    EtudiantRepository::class)]  
class Etudiant extends Personne  
{  
    // le contenu ne change pas  
}
```

La classe Enseignant

```
#[ORM\Entity(repositoryClass:  
    EnseignantRepository::class)]  
class Enseignant extends Personne  
{  
    // le contenu ne change pas  
}
```

Symfony

Pour régénérer la table dans la base de données, exécutez

```
php bin/console make:migration
```

Symfony

Pour régénérer la table dans la base de données, exéutez

```
php bin/console make:migration
```

Ensuite

```
php bin/console doctrine:migrations:migrate
```

Symfony

Pour régénérer la table dans la base de données, exéutez

```
php bin/console make:migration
```

Ensuite

```
php bin/console doctrine:migrations:migrate
```

Vérifier les modifications avec la console MySQL ou phpMyAdmin

Symfony

Ainsi, on peut faire :

```
$personne = new Personne();
$personne->setNom('Wick');
$personne->setPrenom('John');

$etudiant = new Etudiant();
$etudiant->setNom('Maggio');
$etudiant->setPrenom('Carol');
$etudiant->setNiveau('master');

$enseignant = new Enseignant();
$enseignant->setNom('Baggio');
$enseignant->setPrenom('Roberto');
$enseignant->setSalaire(2000);

$entityManager->persist($personne);
$entityManager->persist($etudiant);
$entityManager->persist($enseignant);
$entityManager->flush();
```

Symfony

Allons voir la base de données

- une seule table Personne a été créée
- cette table a les colonnes id, nom, prenom, salaire, niveau et type
- la personne Wick John a la valeur null dans salaire et niveau et la valeur personne dans type
- l'étudiant Maggio Carol a la valeur null dans salaire et la valeur etudiant dans type
- l'enseignant Baggio Roberto a la valeur null dans niveau et la valeur enseignant dans type

Symfony

Remplaçons SINGLE_TABLE par JOINED dans la classe Personne

```
# [ORM\InheritanceType('JOINED')]
#[ORM\Entity(repositoryClass: PersonneRepository::class)]
#[ORM\DiscriminatorColumn(name: 'type', type: 'string')]
#[ORM\DiscriminatorMap(['personne' => 'Personne', 'etudiant' => 'Etudiant', 'enseignant' => 'Enseignant'])]
class Personne
{
    // + tout le code précédent
}
```

La classe Etudiant

```
#[ORM\Entity(repositoryClass:
    EtudiantRepository::class)]
class Etudiant extends Personne
{
    // le contenu ne change pas
}
```

La classe Enseignant

```
#[ORM\Entity(repositoryClass:
    EnseignantRepository::class)]
class Enseignant extends Personne
{
    // le contenu ne change pas
}
```

Symfony

Pour régénérer la table dans la base de données, exécutez

```
php bin/console make:migration
```

© Achref EL MOUELHID

Symfony

Pour régénérer la table dans la base de données, exécutez

```
php bin/console make:migration
```

Ensuite

```
php bin/console doctrine:migrations:migrate
```

Symfony

Pour régénérer la table dans la base de données, exécutez

```
php bin/console make:migration
```

Ensuite

```
php bin/console doctrine:migrations:migrate
```

Vérifier les modifications avec la console MySQL ou phpMyAdmin

Symfony

Allons voir la base de données

- trois tables : créées Personne, Etudiant et Enseignant
- une table Personne avec les colonnes id, nom, prenom et type
- une table Etudiant avec les colonnes id et niveau
- une table Enseignant avec les colonnes id et salaire

Symfony

Ainsi, on peut faire

```
$personne = new Personne();
$personne->setNom('Wick');
$personne->setPrenom('John');

$etudiant = new Etudiant();
$etudiant->setNom('Maggio');
$etudiant->setPrenom('Carol');
$etudiant->setNiveau('master');

$enseignant = new Enseignant();
$enseignant->setNom('Baggio');
$enseignant->setPrenom('Roberto');
$enseignant->setSalaire(2000);

$entityManager->persist($personne);
$entityManager->persist($etudiant);
$entityManager->persist($enseignant);
$entityManager->flush();
```

Symfony

Cycle de vie d'une entité

Le cycle de vie de chaque objet d'une entité passe par trois événements principaux

- création (avec `persist()`)
- mise à jour (avec `flush()`)
- suppression (avec `remove()`)

Symfony

Une méthode callback

- Une méthode callback est une méthode qui sera appelée avant ou après un évènement survenu sur une entité
- On utilise les annotations pour spécifier quand la méthode callback sera appelée



Symfony

Une méthode callback

- Une méthode callback est une méthode qui sera appelée avant ou après un évènement survenu sur une entité
- On utilise les annotations pour spécifier quand la méthode callback sera appelée



C'est comme les triggers en **SQL**

Symfony

Les méthodes callback

- `@PrePersist` : avant qu'une nouvelle entité soit persistée.
- `@PostPersist` : après l'enregistrement de l'entité dans la base de données.
- `@PostLoad` : après le chargement d'une entité de la base de données.
- `@PreUpdate` : avant que la modification d'une entité soit enregistrée en base de données.
- `@PostUpdate` : après que la modification d'une entité est enregistrée en base de données.
- `@PreRemove` : avant qu'une entité soit supprimée de la base de donnée.
- `@PostRemove` : après qu'une entité est supprimée de la base de donnée.

Symfony

La classe Personne

```
class Personne
{
    ...
#[ORM\Column(name: "nbrMAJ", type: "integer")]
private $nbrMAJ = 0;
...
public function setNbrMAJ($nbrMAJ)
{
    $this->nbrMAJ = $nbrMAJ;
    return $this;
}
public function getNbrMAJ()
{
    return $this->nbrMAJ;
}
}
```

Symfony

La classe Personne

```
class Personne
{
    ...
#[ORM\Column(name: "nbrMAJ", type: "integer")]
private $nbrMAJ = 0;

...
public function setNbrMAJ($nbrMAJ)
{
    $this->nbrMAJ = $nbrMAJ;
    return $this;
}
public function getNbrMAJ()
{
    return $this->nbrMAJ;
}
}
```

On utilise l'attribut nbrMAJ pour compter le nombre de modifications d'une entité

Symfony

Démarche

- Tout d'abord, on doit indiquer à Doctrine que notre entité utilise une fonction callback avec l'annotation/attribut HasLifecycleCallbacks
- Ensuite, on va créer une méthode qui sera appelée avant chaque modification
- Cette méthode doit incrémenter chaque fois le nombre de mise-à-jour (nbrMAJ)

Symfony

Dans la classe Personne

```
# [ORM\InheritanceType('JOINED')]
# [ORM\Entity(repositoryClass: PersonneRepository::class)]
# [ORM\DiscriminatorColumn(name: 'type', type: 'string')]
# [ORM\DiscriminatorMap(['personne' => 'Personne', 'etudiant' => 'Etudiant', 'enseignant' => 'Enseignant'])]
# [ORM\HasLifecycleCallbacks]
class Personne
{
    ...
    #[ORM\PreUpdate]
    public function updateNbrMAJ()
    {
        $this->setNbrMAJ($this->getNbrMAJ() + 1);
    }
    ...
}
```

Dans le contrôleur

```
# [Route("/personne/event", name: 'personne_event')]
function event(EntityManagerInterface $entityManager)
{
    $personne = new Personne();
    $personne->setNom('Wick');
    $personne->setPrenom('John');

    $entityManager->persist($personne);
    $entityManager->flush();

    $personne->setNom('Travolta');
    $entityManager->flush();

    $personne->setNom('Abruzzi');
    $entityManager->flush();
    return $this->render('personne/index.html.twig', [
        'controller_name' => 'PersonneController',
        'personne' => $personne,
        'adjectif' => $personne->getNbrMAJ()
    ]);
}
```

Symfony

Pour générer les entités à partir d'une base de données existante, exécutez la commande

```
php bin/console doctrine:mapping:import "App\Entity" annotation  
--path=src/Entity
```

Symfony

Pour générer les entités à partir d'une base de données existante, exécutez la commande

```
php bin/console doctrine:mapping:import "App\Entity" annotation  
--path=src/Entity
```

Remarque

Les attributs des entités générées n'ont pas de getters/setters.

Symfony

Pour générer les entités à partir d'une base de données existante, exécutez la commande

```
php bin/console doctrine:mapping:import "App\Entity" annotation  
--path=src/Entity
```

Remarque

Les attributs des entités générées n'ont pas de getters/setters.

Pour générer les getters/setters, exécutez la commande

```
php bin/console make:entity --regenerate App
```