esprit
Se former autrement
HONORIS UNITED UNIVERSITIES

L'ORM (Object-Relational Mapping) : Doctrine

UP Web

AU: 2023/2024

Plan

1. Introduction
2. Doctrine2
3. Les entités
4. La Migration
5. Entity Manager: Manipuler les entités avec Doctrine2
6. Les relations entres entités avec Doctrine2

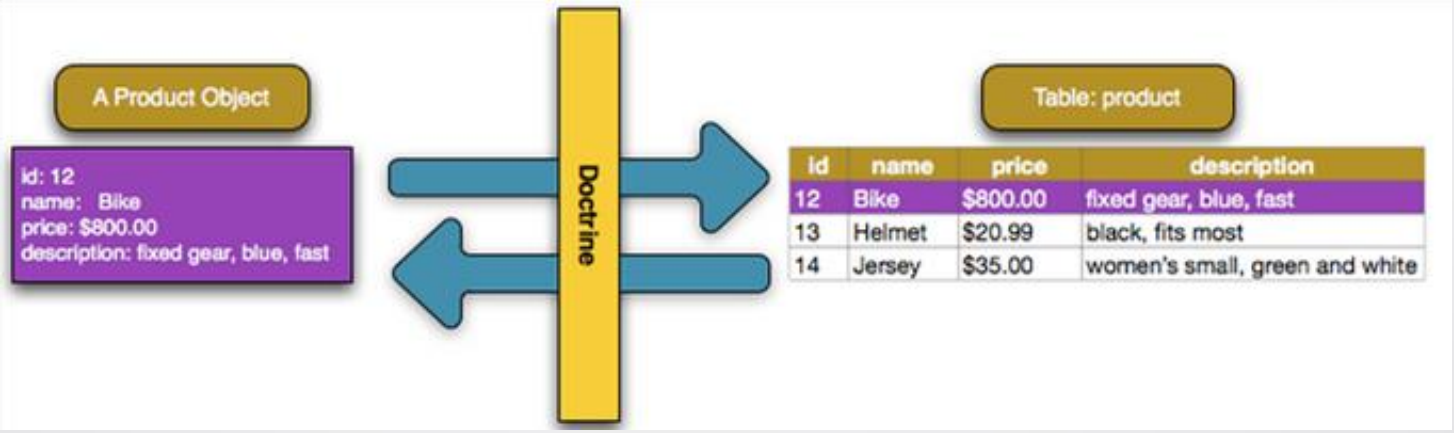
Introduction

- La programmation Orientée Objet, utilisant une base de données relationnelle, nécessite de convertir les données relationnelles en objets et vice-versa.

- Persistance d'objets métiers : les objets modélisés dans les applications sont associées à des données stockées dans les SGBDR

Object-Relational Mapping (ORM)

- C’est une couche d’abstraction à la base de données.
- ORM fait la relation entre les données orientées objet et les données relationnelles.



Object-Relational Mapping (ORM)

Avantages

- Simplifie l'accès aux données
- Facilite le changement de SGBDR
- une indépendance du code vis-à-vis du SGBDR utilisé

Object-Relational Mapping (ORM)

- Les ORM les plus connus:

En Java: - Hibernate

- JPA (Java Persistance API)

- SimpleORM

En .NET: - Nhibernate

- Entity Framework

Quel choix pour PHP:

- **Doctrine**

- Propel

- RedBean

Doctrine

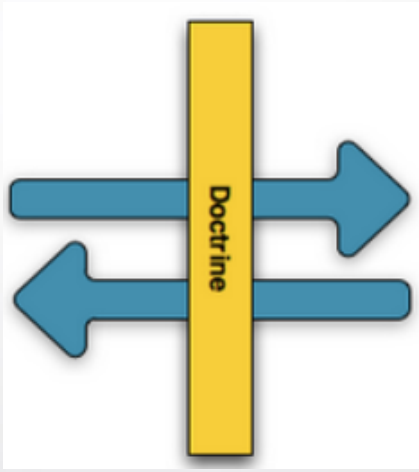
- C’est un ORM pour PHP
- Logiciel open source
- Dernière version stable: **2.13.2**
- Intégré dans différents Frameworks:
  - **Symfony**,
  - Zend Framework
  - CodeIgniter

Doctrine - Caractéristiques

Une classe qui correspond à chaque table

Une classe = une « **Entité** »

```php
class hotel
{

    private $id;


    private $nom;


    private $lieu;


    private $etat;

    public function getId()
    {
        return $this->id;
    }
    // .........
```



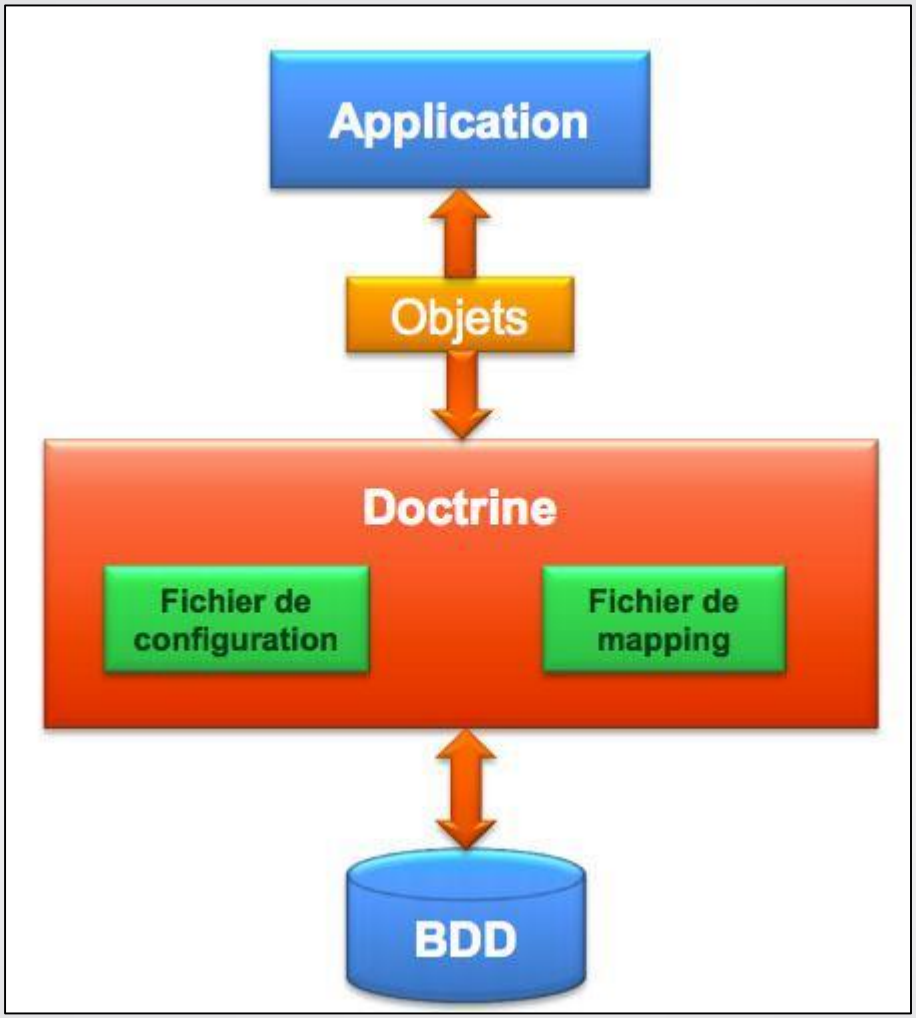
agence hotel
id : int(11)
nom : varchar(255)
lieu : varchar(255)
etat : varchar(7)
prixNuit : double

Doctrine - Caractéristiques

On a deux méthodes pour le Mapping:

- Fichier de mapping YAML, XML.
- Directement dans la classe via des annotations ou des attributes

Doctrine – Architecture Technique



Architecture Technique de Doctrine

Configuration de la base de données

Configurer la base de données de l'application dans le fichier **.env**

```
# DATABASE_URL="sqlite:///%kernel.project_dir%/var/data.db"
DATABASE_URL="mysql://app:!ChangeMe!@127.0.0.1:3306/app?serverVersion=8&charset=utf8mb4"
#DATABASE_URL="postgresql://app:!ChangeMe!@127.0.0.1:5432/app?serverVersion=14&charset=utf8"
###< doctrine/doctrine-bundle ###
```

Nom de l'utilisateur par défaut « root »

Mot de passe de la base de données

Nom de la base de données

Création de la base de données

La commande suivante permet de créer une base de données :

php bin/console doctrine:database:create

Ou

Symfony console doctrine:database:create

=> Une base de données avec les propriétés mentionnées dans **.env** sera automatiquement générée

Les entités

Il existe deux méthodes pour générer les entités :

1. Méthode manuelle (non recommandée)

- ❑ Créer la classe
- ❑ Ajouter le mapping
- ❑ Ajouter les getters et les setters (manuellement ou en utilisant la commande suivante :

➔ **php bin/console make:entity --regenerate App**

➔ **symfony console make:entity --regenerate**

Les entités

2. Méthode en utilisant la dépendance Maker Bundle

- Ajouter une entité en lançant la commande suivante :

➔ **php bin/console make:entity**

➔ **symfony console make:entity**

- Ajouter les attributs et les paramètres

Les entités

Configuration de l'entité:

- **#[ORM\Id]:** spécifie la clé primaire
- **#[ORM\GeneratedValue]:** auto-incrémentée l'ID
- **#[ORM\Column]:** s'applique sur un attribut de la classe et permet de définir les caractéristiques de la colonne concernée (nom , taille , types, etc.)

```
#[ORM\Id]
#[ORM\GeneratedValue]
#[ORM\Column]
private ?int $id = null;
```

La migration

La migration permet de traquer les différents changements et évolutions de votre base de données

Une image = Une évolution de la base de donnée

La commande ci-dessous nous permet de créer un fichier de migration:

- ➔ **php bin/console make:migration**
- ➔ **symfony console make:migration**

La commande ci-dessous nous permet de lancer une migration:

- ➔ **php bin/console doctrine:migrations:migrate**
- ➔ **symfony console doctrine:migrations:migrate**

Le fichier de migration est générée en se basant sur la date et l'heure actuelle sous le nom « **YYYYMMDDHHMMSS »**

Le nom de fichier de migration est un **timestamp** (La valeur représentant la date et l'heure) qui sera stocké dans la base dans une table **doctrine_migration_versions.**

La migration

Chaque fichier de migration possède trois méthodes:

- La méthode **getDescription()** :permet de décrire la migration
- La méthode **up():** est exécutée lorsqu'on applique la migration en utilisant la commande **php bin/console doctrine:migrations:migrate**
  symfony console doctrine:migrations:migrate

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

 WARNING! You are about to execute a database migration that could result in schema changes
 and data loss. Are you sure you wish to continue? (yes/no) [yes]:
 >

[notice] Migrating up to DoctrineMigrations\Version20210122210730
[notice] finished in 217.6ms, used 18M memory, 1 migrations executed, 1 sql queries
```

- La méthode **down()** :est exécutée lorsqu'on annule la migration en utilisant la commande
  php bin/console doctrine:migrations:migrate prev
  symfony console doctrine:migrations:migrate prev

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

 WARNING! You are about to execute a database migration that could result in schema changes
 and data loss. Are you sure you wish to continue? (yes/no) [yes]:
 >

[notice] Migrating down to DoctrineMigrations\Version20210117184800
[notice] finished in 447.7ms, used 18M memory, 1 migrations executed, 1 sql queries
```

La migration

doctrine:migrations:current	Afficher la version actuelle
doctrine:migrations:execute version	Exécuter une seule version de migration
doctrine:migrations:generate	Créer un fichier de migration vide
doctrine:migrations:latest	Afficher la dernière version migration
doctrine:migrations:migrate	Exécuter toutes les versions de migrations non exécutées
doctrine:migrations:status	Afficher l'état d'un ensemble de migrations
doctrine:migrations:up-to-date	Nous indiquer si le schéma est à jour
doctrine:migrations:version version -- add/delete	Ajouter ou supprimer manuellement les versions de migration de la table des versions.
doctrine:migrations:sync-metadata-storage	Vérifier si le stockage des métadonnées est à jour
doctrine:migrations:list	Afficher la liste de toutes les migrations disponibles et leur état
doctrine:migrations:migrate next	Executer la méthode up de la premiere migration générée et non exécutée (une seule)
doctrine:migrations:migrate prev	Exécuter la méthode down de la dernière migration

(*) version: namespace\version
Exemple: DoctrineMigrations\Version20201013084414

Le repository

- Pour chaque entité, il existe un Repository (Exemple: StudentRepository est associé à l'entité Student)
- Un Repository centralise tout ce qui touche à la récupération des entités.
- C'est une classe PHP qui contient les méthodes de récupération de données relatives aux entités.
- Un Repository utilise plusieurs types d'entités, dans le cas d'une jointure par exemple.
- L'appel de la classe Repository se fait dans la classe Entity:

```
#[ORM\Entity(repositoryClass: StudentRepository::class)]
class Student
```

Le repository

Il existe 2 méhodes pour utiliser le repository:

1ère méthode: **Utiliser le service ManagerRegistry :**

```
public function listStudent(ManagerRegistry $doctrine){
$repository=$doctrine->getRepository(NomClasse::class);
 $list= $repository->findAll();
...
}
```

2ème méthode: **Injecter le repository**

```
public function listStudent(StudentRepository $repository){
  $list= $repository->findAll();
   ...
}
```

Les méthodes de récupération de base

Il existe 3 façons pour récupérer les *objets* :

- les méthodes de récupération de base: findAll(), findBy(), find($id)
- les méthodes magiques: findByX(), findOneByX()
- les méthodes de récupération personnalisées: DQL/QueryBuilder

Méthode	Description
find($id)	Trouver un objet à partir son ***id***
findAll()	Trouver tous les objets
findBy()	Trouver plusieurs objets à partir d'un ou plusieurs d'attributs
findOneBy()	même principe que findBy mais elle retourne un seul objet

Les méthodes de récupération magiques

Méthode	Description
findByX($valeur) En remplaçant X par un attribut de l'objet **Exemple:** findBynsc('05896542') findByNom('Club Info')	Similaire à findBy() avec un seul critère, celui du nom de la méthode
findOneByX($valeur) En remplaçant X par un attribut de l'objet **Exemple:** findOneBynsc('05896542')	Similaire à findOneBy() mais en retournant un seul objet

Entity Manager

- Entity Manager (EM) est un gestionnaire d'entités
- C'est le chef d'orchestre de l'ORM Doctrine
- EM permet l'insertion, la mise à jour et la suppression des données de la base de données

```
use Doctrine\Persistence\ManagerRegistry;
…
public function add (ManagerRegistry $doctrine){
 $em= $doctrine->getManager();
….
}
```

Manipulation des entités avec Doctrine 2

- La méthode **persist()** :Utilisée pour l'ajout d'un nouvel objet afin d'informer Doctrine qu'on veut ajouter cet objet dans la base de données.
  - **$em->persist($object)**
- La méthode **flush()** permet d'envoyer tout ce qui a été persisté avant à la base de données afin d'exécuter la requête. Cette méthode est utilisée pour l'ajout, la modification et la suppression
  - **$em->flush()**

Manipulation des entités avec Doctrine 2

- La méthode **remove()** indique à Doctrine d'exécuter la requête de suppression de l'entité en argument de la base de données
  - **$em->remove($object)**
- La méthode **clear()** permet d'annuler tous les persist
  - **$em->clear()**
- La méthode **detach()** permet d'annuler le persist effectué sur l'entité en argument
  - **$em->detach($object)**

Les relations entre les entités

Les types de relation possibles

Une relation (ou une association) peut être:

- ***Unidirectionnelle :*** Seules les instances de l'une des entités de l'association peuvent retrouver les instances de l'entité partenaire.

⇒ **Par exemple :** un utilisateur peut obtenir la liste de ses adresses connues, par contre il n'est pas possible de retrouver un utilisateur à partir d'une adresse.

Les relations entre les entités

Limites:

- ❑ Les relations sont unidirectionnelles
- ❑ On peut faire $student->getProjects()
- ❑ Mais on ne peut pas faire $project->getStudent()

Solution:

- ❑ Rendre les relations bidirectionnelles

Les relations entre les entités

Les types de relation possibles

- ***Bidirectionnelle:*** Les instances de l'une ou de l'autre des entités de l'association peuvent retrouver les instances de l'entité partenaire.

⇒ **Par exemple :** un utilisateur peut obtenir la liste des commandes qu'il a effectué et on peut retrouver un utilisateur à partir d'une commande.

Les relations entre les entités

One To One ,bidirectionnel

Exemple : un étudiant possède sa propre carte d'étudiant.

⇒ ***mappedBy*** fait référence à l'attribut ***card*** dans la classe ***Student***

```php
class StudentCard
{
    #[ORM\OneToOne(mappedBy: 'card', cascade: ['persist', 'remove'])]
    private ?Student $student = null;
```

⇒ ***inversedBy*** fait référence à l'attribut ***student*** dans la classe ***Studentcard***

```php
class Student
{
    #[ORM\OneToOne(inversedBy: 'student', cascade: ['persist', 'remove'])]
    #[ORM\JoinColumn(nullable: false)]
    private ?StudentCard $card = null;
```

Les relations entre les entités

One To Many ,bidirectionnel

Exemple : Une classe contient plusieurs étudiants.

```php
#[ORM\Entity(repositoryClass: ClasseroomRepository::class)]
class Classeroom
{
    #[ORM\OneToMany(mappedBy: 'classrooms', targetEntity: Student::class)]
    private Collection $studentsClass;
```

- ***targetEntity :*** namespace complet vers l'entité liée.
- ***mappedBy :*** il s'agit de l'attribut de l'entité cible qui illustre la relation entre les deux entités

```php
class Student
{
    #[ORM\ManyToOne(inversedBy: 'studentsClass')]
    private ?Classeroom $classrooms = null;
```

NB : Obligatoirement dans l'entité *target* il faut avoir une définition d'attribut avec le mot clé *ManyToOne*

Les relations entre les entités

Many To Many ,bidirectionnelle avec table de jointure

- On peut générer la relation entre les deux entités automatiquement:

  Prenons l'exemple d'une relation ***ManyToMany***

  ⇒ Plusieurs étudiants peuvent appartenir à plusieurs clubs.

- On doit tout d'abord modifier notre entité "**Student**" en tapant la commande suivante:

```
C:\wamp65\www\myproject-course>php bin/console make:entity

 Class name of the entity to create or update (e.g. TinyPopsicle):
 > Student

 Your entity already exists! So let's add some new fields!
```

Les relations entre les entités

Many To Many ,bidirectionnelle avec table de jointure

- Maintenant il faut ajouter l'attribut, dans notre cas "**clubs**":

```
New property name (press <return> to stop adding fields):
> clubs
```

- Spécifier le nom de la classe avec laquelle est reliée qui "***Club***":

```
What class should this entity be related to?:
> Club
```

- Spécifier le type de cet attribut, tapez "**relation**" :

```
Field type (enter ? to see all types) [string]:
> relation
```

Les relations entre les entités

Many To Many ,bidirectionnelle avec table de jointure

- Par la suite , veuillez indiquer le type de relation : **ManyToMany**

```
Relation type? [ManyToOne, OneToMany, ManyToMany, OneToOne]:
> ManyToMany
```

Les relations entre les entités

Many To Many ,bidirectionnelle avec table de jointure

Student.php

```
#[ORM\ManyToMany(targetEntity: Club::class, inversedBy: 'studentsClubs')]
#[ORM\JoinTable(name:'student_club')]
#[ORM\JoinColumn(name: "student_id", referencedColumnName: "nsc")]
#[ORM\InverseJoinColumn(name: "club_id", referencedColumnName: "ref")]
private Collection $clubs;
```

Club.php

```
#[ORM\ManyToMany(targetEntity: Student::class, mappedBy: 'clubs')]

private Collection $studentsClubs;
```

Les relations entre les entités

Cascade

Dans doctrine2, toutes les opérations de cascade sont par défaut désactivées, c'est-à-dire qu'aucune opération ne sera cascadée.

Si une entité est supprimée ou modifiée, les entités avec lesquelles elle était en relation ne seront pas supprimées ou modifiées dans la base de données.

Pour réaliser la suppression ou la modification en cascade, il faut rajouter l'option **cascade={"action"}** dans le mapping de l'entité.

Les relations entre les entités

Les actions en cascade

Persist	Si l'entité est sauvegardée, faire de même avec les entités associées
Remove	Si l'entité est supprimée, faire de même avec les entités associées.
Merge	Cascades fusionne les opérations avec les entités associées.
Detach	Cascades détache les opérations aux entités associées.
Refresh	Les opérations d'actualisation des cascades vers les entités associées.
all	Les cascades persistent, suppriment, fusionnent, actualisent et détachent les opérations aux entités associée

Les relations entre les entités

Exemple

```
class Classeroom
{
    #[ORM\OneToMany(mappedBy: 'classrooms', targetEntity: Student::class,
     cascade: ["persist", "remove", "merge"],
    orphanRemoval: true
    )]
    private Collection $studentsClass;
```

Les relations entre les entités

Cascade

Une style de cascade spécial, **delete-orphan**, s'applique seulement aux associations un-vers-plusieurs si vous répondez par oui à la dernière question

```
Do you want to activate orphanRemoval on your relationship?
A Agences is "orphaned" when it is removed from its related Villes.
e.g. $villes->removeAgences($agences)

NOTE: If a Agences may *change* from one Villes to another, answer "no".

Do you want to automatically delete orphaned App\Entity\Agences objects (orphanRemoval ? (yes/no) [no]:
> yes
```

```php
class Classeroom
{
    #[ORM\OneToMany(mappedBy: 'classrooms', targetEntity: Student::class,
     cascade: ["persist", "remove", "merge"],
    orphanRemoval: true
    )]
    private Collection $studentsClass;
```

Références

- https://symfony.com/doc/master/bundles/DoctrineMigrationsBundle/index.htmlhttp://php.net/manual/fr/language.oop5.magic.php
- https://symfony.com/doc/current/doctrine.html#doctrine-queries