esprit
Se former autrement

L'ORM (Object-Relational Mapping) : Doctrine

UP Web

AU: 2019/2020

United Nations Educational, Scientific and Cultural Organization
UNESCO Chair "Project-based learning" ESPRIT School of engineering, Tunisia
CONFÉRENCE DES GRANDES ÉCOLES
EXCELLENCE FOR A COMPLEX WORLD
European Accreditation of Engineering Programmes EUR-ACE®
délivrée par CTI Commission des Titres d'Ingénieur
cdio
CONCEIVE DESIGN IMPLEMENT OPERATE

Plan

1. Qu'est ce qu'un ORM?
2. Pourquoi utiliser Doctrine2?
3. La couche métier: les entités
4. Manager: Manipuler les entités avec Doctrine2
5. Repository: Récupérer les entités avec Doctrine2
6. Les relations entre entités avec Doctrine2
    a. Many to One
    b. One to One
    c. One to Many
    d. Many to Many

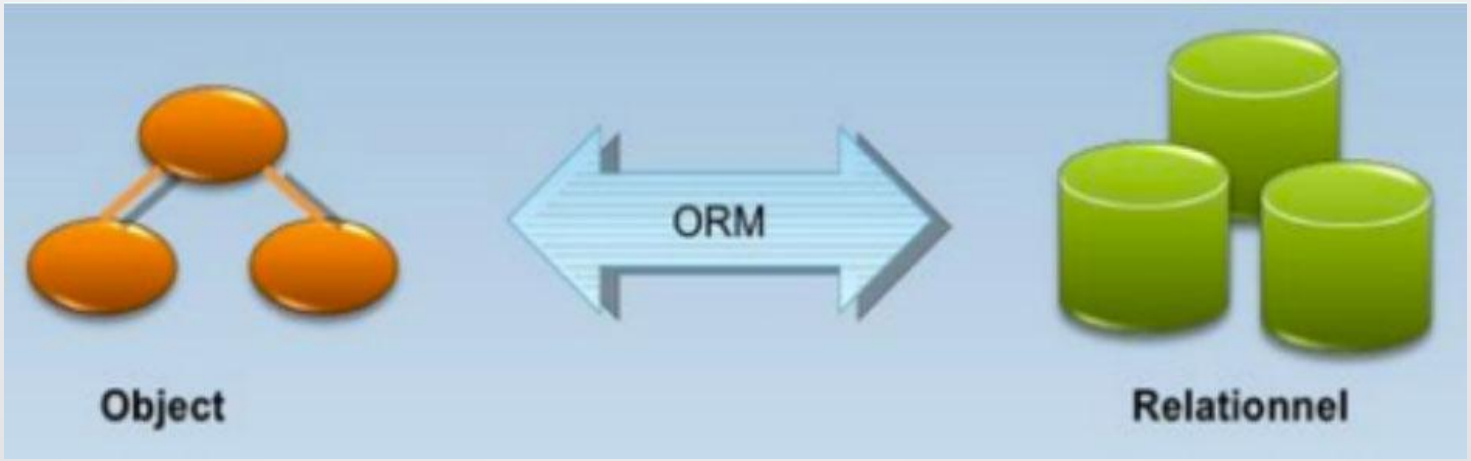
Introduction

- 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 donnée.
- Fait la relation entre les données orientées objet et les données relationnelles.



Object-Relational Mapping

- 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

Object-Relational Mapping

- Avantages:
  - L'ORM masque les communications avec la base de données
  - Simplifie l'accès aux données
  - Facilite le changement de SGBDR
  - Permet une indépendance du code vis-à-vis du SGBDR utilisé

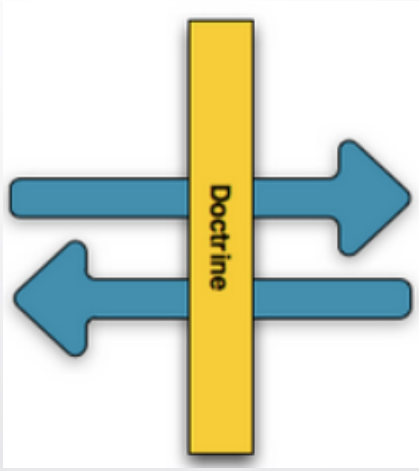
Doctrine

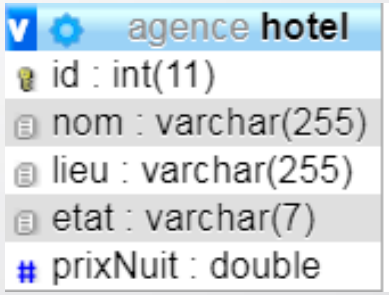
- C’est un ORM pour PHP
- Logiciel open source
- Dernière version stable: 2.6.3
- Intégré dans différents Frameworks:
  - Symfony,
  - Zend Framework,
  - CodeIgniter.

Doctrine - Caractéristiques

- Utilisation d'objets
- 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;
    }
    // ........
```





Doctrine - Caractéristiques

- Un **mapping** sous la forme de métadonnées:
  - ***Fichier de mapping*** YAML, XML.
  - Directement dans la classe via des ***annotations***.

    - Les annotations sont des blocs de commentaires qui permettent de rajouter un ensemble de métadonnées à du code.

Doctrine - Caractéristiques

Instruction, qui importe le préfixe d'annotations ORM

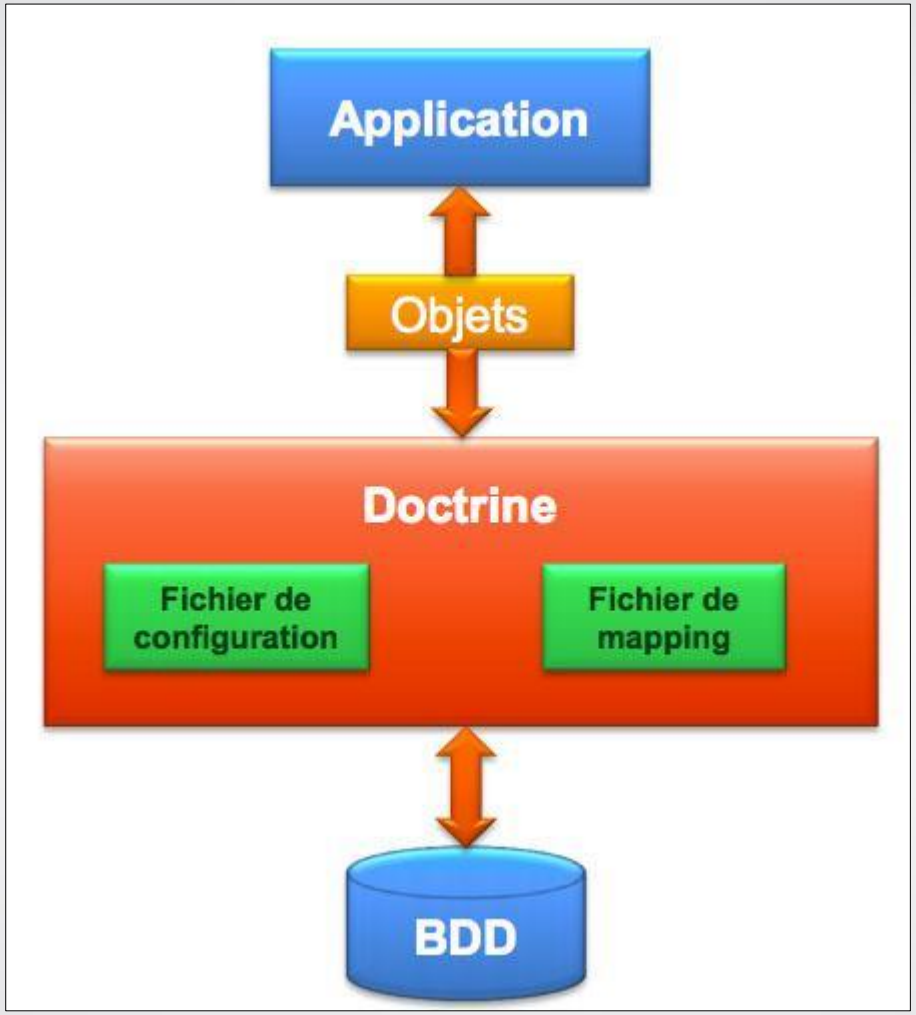
```php
use Doctrine\ORM\Mapping as ORM;

/**
 * hotel
 *
 * @ORM\Table(name="hotel")
 * @ORM\Entity(repositoryClass="reservationBundle\Repository\hotelRepository")
 */
class hotel
{
    /**
     * @var int
     *
     * @ORM\Column(name="id", type="integer")
     * @ORM\Id
     * @ORM\GeneratedValue(strategy="AUTO")
     */
    private $id;

    /**
     * @var string
     *
     * @ORM\Column(name="nom", type="string", length=255)
     */
    private $nom;

    /**
     * @var string
     *
     * @ORM\Column(name="lieu", type="string", length=255)
     */
    private $lieu;
```

esprit
Se former autrement

Doctrine - Caractéristiques

- Plusieurs façons de « faire des requêtes »
  - Très simplement, grâce à **EntityManager** pour les requêtes **CRUD** de base
  - A l'aide du langage **DQL** (Doctrine Query Language)
  - Grâce à l'API SQL **QueryBuilder**

Doctrine – Architecture Technique



Architecture Technique de Doctrine

Configuration de la base de données

- Configurer les informations de connexion de votre BD.
- Dans le fichier app/config/parameters.yml:

```
parameters:
    database_host: 127.0.0.1
    database_port: null
    database_name: agence
    database_user: root
    database_password: null

# ....
```

Les entités

- Des classes métiers qui décrivent les objets de l'application.
- Une classe avec un ensemble d'attributs liés à des colonnes d'une table de la base de données via Doctrine.
- Placer les entités ➔ src/Entity.
- Toutes les entités doivent avoir l'annotation **Entity** : lien entre une classe PHP et Doctrine.

Les entités

Configuration de l'entité:

- **@Column:** s'applique sur un attribut de la classe et permet de définir les caractéristiques de la colonne concernée (nom , taille , types, etc.)
- **@Id:** spécifie la clé primaire,par défaut auto-incrémentée avec l'annotation `GeneratedValue`.

```
<?php
/**
 * @Id
 * @Column(type="integer")
 * @GeneratedValue
 */
protected $id;
```

Les entités

Configuration de l'entité:

- Pour générer le schéma de la BD il faut appeler avec la console la fonction suivante:
  doctrine:schema:create
- La commande qui permet de mettre à jour la BD ou de récupérer les requêtes SQL que Doctrine va exécuter pour faire la mise à jour:
  doctrine:schema:update --force

Manager: Manipuler les entités avec Doctrine2

Sauvegarder les entités:

- Utiliser l'ORM pour manipuler les entités.
- Le service Doctrine permet de gérer la persistance de l'objet accessible depuis le contrôleur getDoctrine().
- getManager() :récupérer le gestionnaire d'entité (Entity Manager).
- L'appel persist($hotel) indique à Doctrine de gérer le nouveau objet $hotel.
- flush() permet d'envoyer tout ce qui a été persisté avant à la base de données.

```
$hotel = new Hotel(); //Créer une instance de l'objet hotel
$em = $this->getDoctrine()->getManager();
$em->persist($hotel);
$em->flush();
```

esprit
Se former autrement

Manager: Manipuler les entités avec Doctrine2

Supprimer un objet:

- La suppression nécessite un appel à la méthode remove() du gestionnaire d'entités.
- La méthode remove() indique à Doctrine de supprimer l'objet spécifié de la base de données (appeler la requête DELETE).

```
$em = $this->getDoctrine()->getManager();
$em->remove($hotel);
$em->flush();
```

Repository: Récupérer les entités avec Doctrine2

Récupérer les entités:

Doctrine propose deux méthodes pour récupérer des données:

- ❑ Récupérer une entité avec la clé primaire
- ❑ Récupérer une ou plusieurs entités selon des critères différents

Depuis un repository, il existe deux façons de récupérer les entités :

- ❑ En utilisant du DQL
- ❑ En utilisant le QueryBuilder

Repository: Récupérer les entités avec Doctrine2

Récupérer les entités:

Les repositories (entrepôts) sont des classes spécialisées qui nous permettent de récupérer nos entités.

Méthode	Description
find($id)	Récupère une entité grâce à sa clé primaire
findAll	Récupère toutes les entités
findBy	Récupère une liste d'entités selon un ensemble de critères
findOneBy	Récupère une entité selon un ensemble de critères

Récupérer une entité avec la clé primaire :

```
$hotel = $this->getDoctrine()
        ->getRepository(Hotel::class)
        ->find($id);
```

Les relations entre entités avec Doctrine2

Many To One ,unidirectionnel

Exemple : un hôtel héberge plusieurs personnes

```
/** @Entity */
class Personne
{
    // ...
    /**
     * @ManyToOne(targetEntity="Hotel")
     * @JoinColumn(name="hotel_id",
       referencedColumnName="id")
     */
    private $hotel;
}
/** @Entity */
class Hotel
{ // ...
}
```

Les relations entre entités avec Doctrine2

One To One ,unidirectionnel

Exemple : un seul manager dirige un hôtel

```php
<?php
/** @Entity */
class Manager
{
    // ...
    /**
     * @OneToOne(targetEntity="Hotel")
     * @JoinColumn(name="hotel_id",
       referencedColumnName="id")
     */
    private $hotel;
    // ...
}
/** @Entity */
class Hotel
{ // ...
}
```

Les relations entre entités avec Doctrine2

One To One ,bidirectionnel

Exemple : un client et un panier.

Le panier a une référence au client, il est donc bidirectionnel.

```
/** @Entity */
class Client
{
    // ...
    /**
     * @OneToOne(targetEntity="Panier",
       mappedBy="client")
     */
    private $panier;
    // ...
}
```

```
/** @Entity */
class Panier
{
    // ...
    /**
     * @OneToOne(targetEntity="Client",
       inversedBy="panier")
     * @JoinColumn(name="client_id",
       referencedColumnName="id")
     */
    private $client;
    // ...
}
```

Les relations entre entités avec Doctrine2

One To Many ,bidirectionnel

- Une association OneToMany est bidirectionnelle, sauf si il y'a une table de jointure,car le côté « Many »d'une association OneToMany détient la clé étrangère il est le propriétaire:
- Ce mappage bidirectionnel nécessite l'attribut mappedBy du côté « One » et l'attribut inversedBy du côté « Many ».
- Il n'y a pas de différence entre une association OneToMany bidirectionnel et ManyToOne bidirectionnel.

➢ **En ORM,les tableaux(array) PHP, manquent de fonctionnalités qui les rendent aptes au chargement et mise a jour. C'est pourquoi, dans tous les exemples d'associations à valeurs multiples on utilisent une interface « Collection » et son implémentation par défaut « ArrayCollection », qui sont tous deux définis dans l'espace de noms Doctrine \ Common \ Collections.**

Les relations entre entités avec Doctrine2

One To Many ,bidirectionnel

```php
/** @Entity */
class Produit
{
    // ...
    /**
     * Un produit a plusieurs propriétés. C'est
      l'inverse
     * @OneToMany(targetEntity="Propriete",
       mappedBy="produit")
     */
    private $proprietes;
    // ...
    public function __construct() {
        $this->proprietes = new ArrayCollection();
    }
}
```

```php
/** @Entity */
class Propriete
{
    // ...
    /**
     * plusieurs propriétés ont un seul produit.
       C'est le côté propriétaire.
     * @ManyToOne(targetEntity="Produit",
       inversedBy="proprietes")
     * @JoinColumn(name="produit_id",
       referencedColumnName="id")
     */
    private $produit;
    // ...
}
```

Les relations entre entités avec Doctrine2

One To Many ,unidirectionnel

```
/** @Entity */
class Utilisateur
{
     /**
      * Plusieurs utilisateurs possedent plusieurs
email .
      * @ManyToMany(targetEntity="Email")
      * @JoinTable(name="utilisateurs_emails",
      *  joinColumns={@JoinColumn
         (name="utilisateur_id",
         referencedColumnName="id")},
      *  inverseJoinColumns={@JoinColumn
         (name="email_id",
         referencedColumnName="id",
         unique=true)}
      *    )
      */
 private $emails;
     public function __construct()
     {$this->emails = new \Doctrine\Common
              \Collections\ArrayCollection();}}
```

```
/** @Entity */
class Email
{
     // ...
}
```

Les relations entre entités avec Doctrine2

Many To Many ,unidirectionnel

```
/** @Entity */
class Ingenieur
{
    /**
     * Plusieur ingenieurs ont plusieurs projets
     * @ManyToMany(targetEntity="Projet")
     * @JoinTable(name="ingenieurs_projets",
     *
joinColumns={@JoinColumn(name="ingenieur_id",
referencedColumnName="id")},
     *
inverseJoinColumns={@JoinColumn(name="projet_id",
referencedColumnName="id")}
     *      )
     */
    private $projets;

    public function __construct() {
        $this->projets = new \Doctrine\Common\Collections
\ArrayCollection();
    }
}
```

```
/** @Entity */
class Projet
{
    // ...
}
```

- **Atelier: ….**

Référence

- https://www.doctrine-project.org/projects/orm.html
- http://developpement-informatique.com/formation/Framework-Symfony/1/Initiation-%C3%A0-la-prise-en-main-de-Doctrine2
- https://fr.slideshare.net/arhouati/symfony-2-chapitre-3-les-modles-en-twig