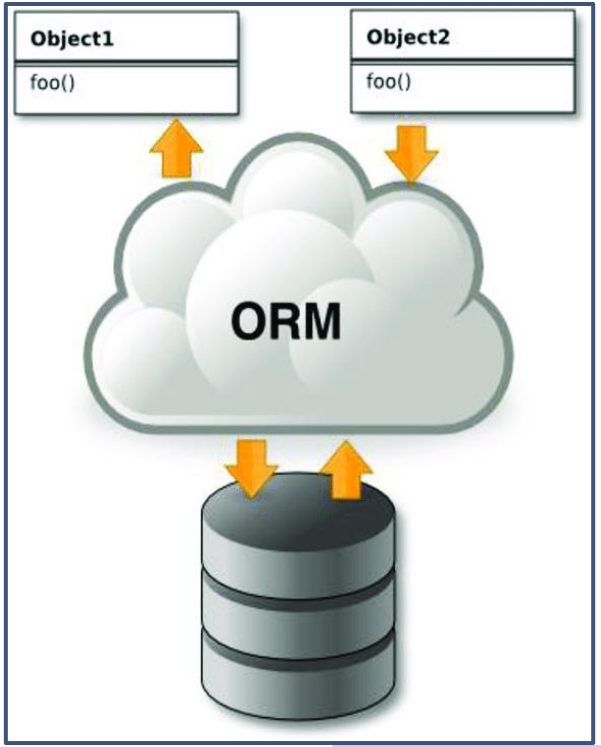
Ch. V

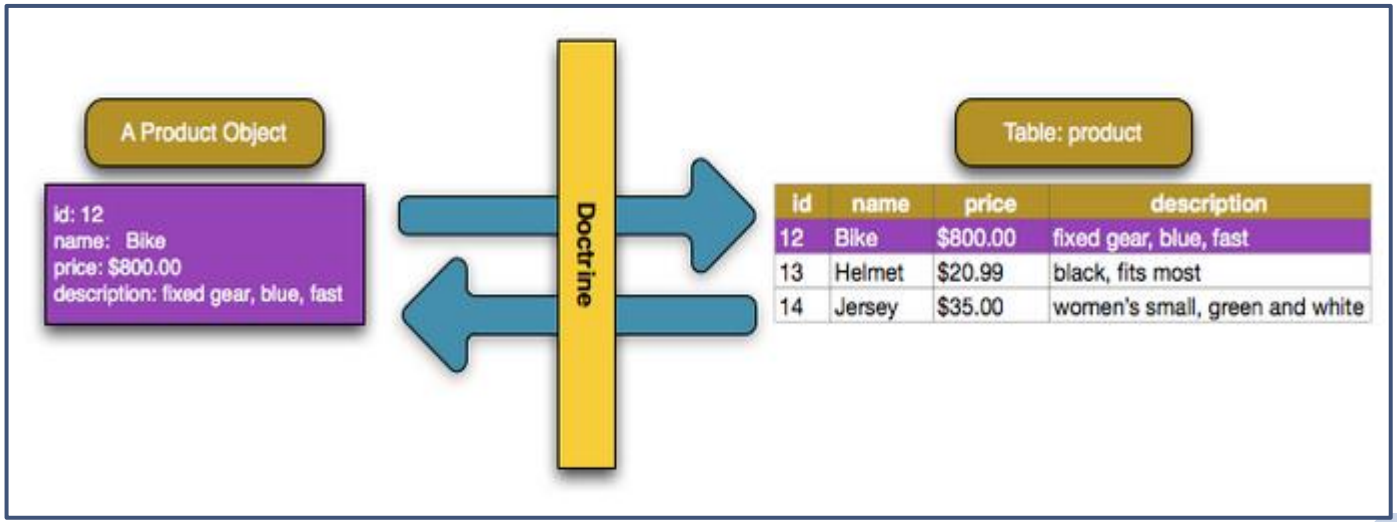
L’ORM Doctrine

Ch. VI: L'ORM Doctrine

- ORM : Object Relation Mapper
- Couche d’abstraction
- Gérer la persistance des données
- Mapper les tables de la base de données relationnelle avec des objets
- Crée l'illusion d'une base de données orientée objet à partir d'une base de données relationnelle en définissant des correspondances entre cette base de données et les objets du langage utilisé.
- Propose des méthodes prédéfinies



- Doctrine : ORM le plus utilisé avec Symfony 6
- Associe des classes PHP avec les tables de votre BD (mapping)



Ch. VI: L'ORM Doctrine

- Objet PHP
- Les entités représente les objets PHP équivalentes à une table de la base de données.
- Une entité est généralement composée par les attributs de la table ainsi que leurs getters et setters
- Manipulable par l'ORM

```php
<?php
// src/OC/PlatformBundle/Entity/Advert.php

namespace OC\PlatformBundle\Entity;

class Advert
{
  protected $id;

  protected $content;

  // Et bien sûr les getters/setters :

  public function setId($id)
  {
    $this->id = $id;
  }
  public function getId()
  {
    return $this->id;
  }

  public function setContent($content)
  {
    $this->content = $content;
  }
  public function getContent()
  {
    return $this->content;
  }
}
```

Configuration des entités

- Configuration Externe : YAML, XML, PHP
- Configuration Interne : annotations
- Choix de la configuration ?

- Deux Visions :
  - Pro-Externe
    - Séparation complète des fonctionnalités spécialement lorsque l'entité est conséquente
  - Pro-Interne
    - Plus facile et agréable de chercher dans un seul fichier l'ensemble des information, plus de visibilité

Annotation des entités

Rôle : Faire le lien entre les entités et les tables de la base de données
Lien à travers les métadonnées
Remarque : Un seul format par bundle (impossibilité de mélanger)

Syntaxe :

```
/**
* les différentes annotations
*/
```

Remarque : Afin d'utiliser les annotations il faut ajouter :
use Doctrine\ORM\Mapping as ORM;

Annotation des entités: Entity

- Permet de définir un objet comme une entité
- Applicable sur une classe
- Placée avant la définition de la classe en PHP

Syntaxe :

```
@ORM\Entity
```

Paramètres :

repositoryClass (facultatif). Permet de préciser le namespace complet du repository qui gère cette entité.

Exemple :

```
@ORM\Entity(repositoryClass="SMF\NameBundle\Entity\carsRepository")
```

Annotation des entités: Table

- Permet de spécifier le nom de la table dans la base de données à associer à l'entité
- Applicable sur une classe et placée avant la définition de la classe en PHP
- Facultative sans cette annotation le nom de la table sera automatiquement le nom de l'entité
- Généralement utilisable pour ajouter des préfixes ou pour forcer la première lettre de la table en minuscule

Syntaxe : @ORM\Table()

Exemple :

```
/**
* @ORM\Table('car')
* @ORM\Entity(repositoryClass="SMF\NameBundle\Entity\carsRepository")
*/
```

Annotation des entités: Column

- Permet de définir les caractéristiques de la colonne concernée
- Applicable sur un attribut de classe juste avant la définition PHP de l'attribut concerné.

Syntaxe : @ORM\Column()

Exemple :

```
/**
*
* @ORM\Column(param1="valParam1", param2="valParam2")
*/
```

Les paramètres de Column

Paramètre	Valeur par défaut	Utilisation
type	string	Définit le type de colonne comme nous venons de le voir.
name	Nom de l'attribut	Définit le nom de la colonne dans la table. Par défaut, le nom de la colonne est le nom de l'attribut de l'objet
length	255	Définit la longueur de la colonne (pour les strings).
unique	false	Définit la colonne comme unique (Exemple : email).
nullable	false	Permet à la colonne de contenir des NULL.
precision	0	Définit la précision d'un nombre à virgule(decimal)
scale	0	le nombre de chiffres après la virgule (decimal)

Les types de Column

Type Doctrine	Type SQL	Type PHP	Utilisation
string	VARCHAR	string	Toutes les chaînes de caractères jusqu'à 255 caractères.
integer	INT	integer	Tous les nombres jusqu'à 2 147 483 647.
smallint	SMALLINT	integer	Tous les nombres jusqu'à 32 767.
bigint	BIGINT	string	Tous les nombres jusqu'à 9 223 372 036 854 775 807.
boolean	BOOLEAN	boolean	Les valeurs booléennes true et false.
decimal	DECIMAL	double	Les nombres à virgule.

Les types de Column

Type Doctrine	Type SQL	Type PHP	Utilisation
date ou datetime	DATETIME	objet DateTime	Toutes les dates et heures.
time	TIME	objet DateTime-	Toutes les heures.
text	CLOB	string	Les chaînes de caractères de plus de 255 caractères.
object	CLOB	Type de l'objet stocké	Stocke un objet PHP en utilisant serialize/unserialize.
array	CLOB	array	Stocke un tableau PHP en utilisant serialize/unserialize.
float	FLOAT	double	Tous les nombres à virgule. Attention, fonctionne uniquement sur les serveurs dont la locale utilise un point comme séparateur.

Conventions de Nommage

Même s'il reste facultatif, le champs « name » doit être modifié afin de respecter les conventions de nommage qui diffèrent entre ceux de la base de données et ceux de la programmation OO.

- Les noms de classes sont écrites en « Pascal Case » TheEntity.
- Les attributs de classes sont écrites en « camel Case » oneAttribute
- Les noms des tables et des colonnes en SQL sont écrites en minuscules, les mots sont séparés par « _ » one_table, one_column.

Ch. VI: L'ORM Doctrine

VI.1. Introduction

VI.2. Couche métier: Entity

VI.3. Gestion d'une BD

VI.4. Le service Doctrine

VI.5. Le Repository

VI.6. Le service EntityManager

VI.7. Entities Relations

Configuration

- Afin de configurer la base de données de l'application il faut renseigner les champs dans le fichier .env

```
DATABASE_URL="mysql://root:@127.0.0.1:3306/workshop1?serverVersion=5.7"
```

Création de BD

Afin de créer la base de données du projet, 2 méthodes sont utilisées :

- Manuelle en utilisant le SGBD (le nom de la BD doit être le même que celui mentionné dans le fichier .env)

- En utilisant la ligne de commande avec la commande suivante :
  - php app/console doctrine:database:create
  - Une base de données avec les propriétés mentionnées dans .env sera automatiquement générée

Génération des entités

Deux méthodes pour générer les entités :

- Méthode manuelle (non recommandée)
  - Créer la classe
  - Ajouter le mapping
  - Ajouter les getters et les setters (manuellement ou en utilisant la commande suivante
    php app/console doctrine:generate:entities)

- Méthode en utilisant les commandes
  - Il suffit de lancer la commande suivante :
    php app/console doctrine:generate:entity
  - Ajouter les attributs ainsi que les paramètres qui vont avec
  - Une fois terminé, Doctrine génère l'entité avec toutes les métadonnées de mapping

Création des tables

Afin de créer les tables de la base de données, doctrine se base sur les entités placées dans les différentes dossiers Entity des différents bundles de l'application.

Deux commandes permettent de créer les tables de la BD :

- php app/console doctrine:schema:create
- php app/console doctrine:schema:update –force // utilisable pour la création et pour la mise à jour d'une table

Astuce :

- php app/console doctrine:schema:update –dump-sql // Affiche les requêtes SQL à exécuter pour la BD

Cette astuce permet de vérifier la requête à exécuter avant la mise à jour de la table.

Ch. VI: L'ORM Doctrine

VI.1. Introduction

VI.2. Couche métier: Entity

VI.3. Gestion d'une BD

VI.4. Le service Doctrine

VI.5. Le Repository

VI.6. Le service EntityManager

VI.7. Entities Relations

VI.4. Le service Doctrine

Rôle : permet la gestion des données dans la BD : persistance des données et consultation des données.

Méthode :

- $this->get('doctrine');
- $this->getDoctrine(); //helper (raccourcie de la classe Controller)

Le service Doctrine offre deux services pour la gestion d'une base de données :

- Le Repository qui se charge des requêtes Select
- L'EntityManager qui se charge de persister la base de données donc de gérer les requêtes INSERT, UPDATE et DELETE.

Ch. VI: L'ORM Doctrine

VI.1. Introduction

VI.2. Couche métier: Entity

VI.3. Gestion d'une BD

VI.4. Le service Doctrine

VI.5. Le Repository

VI.6. Le service EntityManager

VI.7. Entities Relations

VI.5. Le Repository

- Des classes PHP dont le rôle est de permettre à l'utilisateur de récupérer des entités d'une classe donnée.

Syntaxe :

Pour accéder au repositorie de la classe MaClasse on utilise l'EntityManager

- $repo = $EntityManager->getReository(« Bundle:MAClasse »);

Quelque méthodes offertes par le repository :

- $repository->findAll(); // récupère tous les entités (enregistrements) relatifs à l'entité associé au repository
- $repository->find($id); // requête sur la clé primaire
- $repository->findBy(); //retourne un ensemble d'entités avec un filtrage sur plusieurs critères (nbre donné)
- $repository->findOneBy(); //même principe que findBy mais une seule entité
- $repository->findByNomPropriété();
- $repository->findOneByNomPropriété();

findAll()

Rôle : retourne l'ensemble des entités qui correspondent à l'entité associé au repository. Le format du retour est un Array

Exemple :

```
//On récupère le repository de l'entity manager correspondant à l'entité Etudiant
$repository = $this
->getDoctrine()
->getRepository('OCPlatformBundle:Advert') ;

//On récupère la liste des étudiants
$listAdverts = $repository->findAll();
```

Find($id)

Rôle : retourne l'entité qui correspond à la clé primaire passé en argument. Généralement cette clé est l'id.

Exemple :

```
//On récupére le repository de l'entity manager correspondnat à l'entité Etudiant
$repo = $this
->getDoctrine()
->getRepository('Rt4AsBundle:Etudiant');

//on lance la requéte sur l'étudiant d'id 2
$etudiant = $repository->find(2);
```

findBy()

Rôle : retourne l'ensemble des entités qui correspondent à l'entité associé au repository comme findAll sauf qu'elle permet d'effectuer un filtrage sur un ensemble de critère passés dans un Array .

Elle offre la possibilité de trier les entités sélectionnées et facilite la pagination en offrant un nombre de valeur de retour.

Synatxe:

```
$repository->findBy( array $criteria, array $orderBy = null, $limit = null, $offset = null);
```

Exemple :

```
$repository = $this->getDoctrine()->getRepository('EtudiantBundle:Etudiant');
$listeEtudiants = $repository->findBy(array('section' => 'GL', 'nom' => 'Mohamed'),
array('date' => 'desc'),10, 0);
```

findOneBy()

Rôle : Même principe que FindBy mais en retournant une seule entité ce qui élimine automatiquement les paramètres d'ordre de limite et d'offset

Exemple :

```
$repository = $this->getDoctrine()->getRepository('EtudiantBundle:Etudiant');
$Etud = $repository->findOneBy(array('section' => 'GL','nom' => 'Mohamed'));
```

findByPropriété()

Rôle : En remplaçant le mot Propriété par le nom d'une des propriété de l'entité, la fonction va faire le même rôle que findBy mais avec un seul critère qui est le nom de la propriété et sans les options.

Exemple :

```
$repository = $this->getDoctrine()->getRepository('EtudiantBundle:Etudiant');
$listeEtudiants = $repository->findByNom('Aymen');
```

findOneByPropriété()

Rôle : En remplaçant le mot Propriété par le nom d'une des propriété de l'entité, la fonction va faire le même rôle que findOneBy mais avec un seul critère.

Exemple :

```
$repository = $this->getDoctrine()->getRepository('EtudiantBundle:Etudiant');
$listeEtudiants = $repository->findOneByNom('Aymen');
```

Création d'une requête

- Les requêtes de doctrine sont écrites en utilisant le langage de doctrine le Doctrine Query Language DQL
- Ou en utilisant un Objet créateur de requêtes le CreateQueryBuilder
  - createQuery : Méthode de l'Entity Manager
  - CreateQueryBuilder : Méthode du repository

CreateQuery /DQL

- Le DQL peut être défini comme une adaptation du SQL adapté à l'orienté objet et donc à DOCTRINE
- La requête est défini sous forme d'une chaine de caractère

- Afin de créer une requête DQL il faut utiliser la méthode createQuery() de l'EntityManager
- La méthode setParameter('label','valeur') permet de définir un paramètre de la requête
- Pour définir plusieurs paramètres ou bien utiliser setParameter plusieurs fois ou bien la méthode setParameters(array('label1','valeur1', 'label2','valeur2'),.. 'labelN','valeurN'))
-
- Une fois la requête créée, la méthode getResult() permet de récupérer un tableau de résultat

Le langage DQL est explicité dans le lien suivant :
http://docs.doctrine-project.org/projects/doctrine-orm/en/latest/reference/dql-doctrine-querylanguage.html

QueryBuilder

- Constructeur de requête DOCTRINE Alternative au DQL
- Accessible via le Repository
- Le résultat fourni par la méthode getQuery du QueryBuilder permet de générer la requête en DQL
- De même que le createQuery , une fois la requête créée la méthode getResult() permet de récupérer un tableau de résultat.

QueryBuilder

- Afin de récupérer le QueryBuilder dans notre Repository , on utilise la méthode createQueryBuilder.
- Cette méthode récupère en paramètre l'allias de l'entité cible et offre la requête « select from » de l'entité en question.
- Généralement l'allias est la première lettre en minuscule du nom de l'entité
- Si aucun paramètre n'est passé a createQueryBuilder alors on aura une requête vide et il faudra tout construire.

```
$qb=$this->_em->createQueryBuilder()
              ->select('t')
              ->from($this->_entityName,'alias');
```

QueryBuilder: Méthodes

- from(‘entityName’,’entityAlias’)
  - from($this->_entityName,’t’)
- where(‘condition’) permet d’ajouter le where dans la requête
  - where(‘t.destination= :dest‘)
- setParameter(‘nomParam’,param) permet d’ajouter la définition d’un des paramètres définis dans le where
  - setParameter(‘dest’,$dest)
- andWhere(‘condition’) permet d’ajouter d’autres conditions
  - andWhere(‘t.statut = :status)
- orderBy(‘nomChamp’,’ordre’) permet d’ajouter un orderBy et prend en paramètre le champ à ordonner et l’ordre DESC ou ASC.
  - orderBy(‘t.dateTransfert’,’DESC’)
- setParameters(array(1=>’param1’,2=>’param2’))

VI.5. Le Repository

Externalisation des requêtes dans un dépôt personnalisé pour chaque entité

But :

- Isoler la couche modèle
- Réutilisabilité

Faisabilité :

Ajouter le dépôt dans le mapping de l'entité

(@ORM\Entity(RepositoryClass=« NotreRepository »)

Ch. VI: L'ORM Doctrine

VI.1. Introduction

VI.2. Couche métier: Entity

VI.3. Gestion d'une BD

VI.4. Le service Doctrine

VI.5. Le Repository

VI.6. Le service EntityManager

VI.7. Entities Relations

Rôle :

L'interface ORM proposée par doctrine offrant des méthodes prédéfinies pour persister dans la base ou pour mettre à jour ou supprimer une entité.

Méthode :

- $EntityManager = $this->get('doctrineorm.entity_manager')
- $this->getDoctrine()->getManager(); //helper (raccourcie de la classe Controller)

Insertion des données

- Etant un ORM, Doctrine traite les objets PHP
- Pour enregistrer des données dans la BD il faut préparer les objets contenant ces données
- La méthode persist() de l'enityManager permet d'associer les objets à persister avec Doctrine
- Afin d'exécuter les requêtes sur la BD (enregistrer les données dans la base) il faut utiliser la méthode flush()
- L'utilisation de flush permet de profiter de la force de Doctrine qui utilise les Transactions
- La persistance agit de la même façon avec l'ajout (insert) ou la mise à jour (update)

Add, Update d'une entité

```php
public function AddUpdateAction($id)
{
    // on récupére notre entity manager
    $em = $this->getDoctrine()->getManager();
    $repo = $em->getRepository('Rt4AsBundle:Etudiant');
    // On prépare un nouvel étudiant
    $etudiant = new Etudiant();
    $etudiant->setNom('nouvel etudiant')
             ->setCin('12345678')
             ->setDateNaissance(new \DateTime())
             ->setPrenom('mon prenom')
             ->setNumEtudiant(1234);
    // On l'associe à Doctrine en le persistant
    $em->persist($etudiant);
    //on récupére une entité qui est automatiquement associé à Doctrine plus besoin de la persister
    $etud = $repo->find($id);
    if ($etud)
        $etud->setCin(2782);
    $em->flush();
    //on recupére l'ensemble des entités et on le transmet à la page d'affichage
    $etudiants = $em->getRepository("Rt4AsBundle:Etudiant")->findAll();
    return $this->render('Rt4AsBundle:Default:lister.html.twig', array(
        'etudiants' => $etudiants,
    ));
}
```

Suppression d'une entité

- La méthode remove() permet de supprimer une entité

```
public function DeleteAction($id)
{
    // on récupére notre entity manager et notre repository
    $em = $this->getDoctrine()->getManager();
    $repo = $em->getRepository('Rt4AsBundle:Etudiant');
    //on récupére une entité qui est automatiquement associé à Doctrine plus besoin de la persister
    $etud = $repo->find($id);
    if ($etud)
    {
        $em->remove($etud);
        $em->flush();
    }
    //on recupére l'ensemble des entités et on le transmet à la page d'affichage
    $etudiants = $em->getRepository("Rt4AsBundle:Etudiant")->findAll();
    return $this->render('Rt4AsBundle:Default:lister.html.twig', array(
        'etudiants' => $etudiants,
    ));
}
```

Ch. VI: L'ORM Doctrine

VI.1. Introduction

VI.2. Couche métier: Entity

VI.3. Gestion d'une BD

VI.4. Le service Doctrine

VI.5. Le Repository

VI.6. Le service EntityManager

VI.7. Entities Relations

Types de relation

Les entités de la BD présentent des relations d'association :

- A OneToOne B : à une entité A on associe une entité de B et inversement
- A ManyToOne B : à une entité B on associe plusieurs entité de A et à une entité de A on associe une entité de B
- A ManyToMany B : à une entité de A on associe plusieurs entité de B et inversement

Types de relation

- La notion de navigabilité de UML est la source de la notion de relation unidirectionnelle ou Bidirectionnelle

- Une relation est dite navigable dans les deux sens si les deux entités doivent avoir une trace de la relation.

- ***Exemple :*** Supposons que nous avons les deux classes CandidatPresidentielle et Electeur. L'électeur doit savoir à qui il a voté donc il doit sauvegarder cette information par contre le candidat pour cause d'anonymat de vote ne doit pas connaitre les personnes qui ont voté pour lui.
- On aura donc un attribut Candidat dans la table Electeur mais pas de collection ou tableau nommé électeur dans la table CandidatPresidentielle. Ici on a une relation unidirectionnelle.