



Benoît Roche
@rocheb83

Benoît Roche / @rocheb83

1



LE PRINCIPLE

Benoît Roche / @rocheb83

2



DOCTRINE : STRUCTURE DE LA BD



Doctrine : présentation

Doctrine appartient à la famille des ORM (Object Relational Mapping)

Doctrine assure la persistance des données traitées sous forme de classes dans les bases de données

Doctrine2 est donc une librairie permettant de gérer les interactions entre une application et une (ou plusieurs) base de données.

Benoît Roche / @rocheb83

3



DOCTRINE : STRUCTURE DE LA BD



Doctrine est totalement découplé de Symfony et son utilisation est optionnelle

On peut utiliser un autre ORM comme propel

On peut aussi utiliser des requêtes SQL brutes

Doctrine2 se compose de plusieurs couches :

- ✓ DBAL,
- ✓ ORM et
- ✓ Entité

Benoît Roche / @rocheb83

4



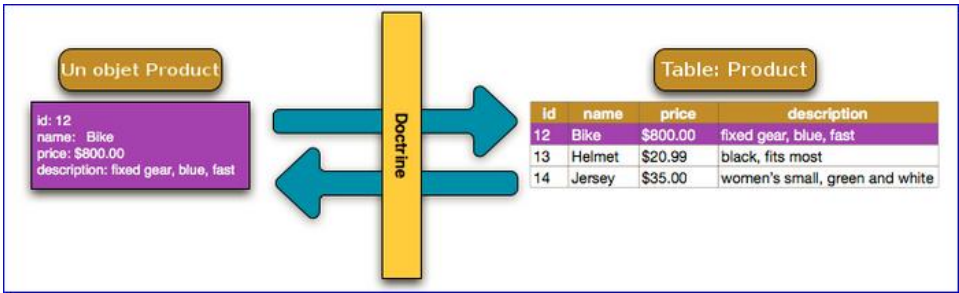
DOCTRINE : STRUCTURE DE LA BD



Un petit exemple

L'objectif de Doctrine est d'associer :

- ✓ des classes PHP avec des tables de la base,
- ✓ des propriétés de ces classes PHP avec des colonnes des tables



Benoît Roche / @rocheb83

5



DOCTRINE : STRUCTURE DE LA BD



sur un exemple : *Application gestion des employés*



Et le lien entre les deux ???

Benoît Roche / @rocheb83

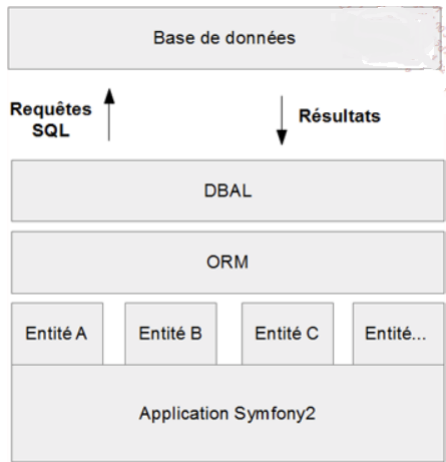
6



DOCTRINE : STRUCTURE DE LA BD



Une vue d'ensemble :



DBAL (Database Abstraction Layer)

ORM (Object Relational Mapping)

Entite on Entity



DOCTRINE : STRUCTURE DE LA BD



Une vue d'ensemble :

La couche DBAL (Database Abstraction Layer)

- ✓ C'est la couche de plus bas niveau.
- ✓ Ne comporte aucune logique applicative et son rôle est d'envoyer des requêtes vers une base de données et de récupérer les résultats.
- ✓ Elle utilise PHP PDO mais elle est plus complète.



Une vue d'ensemble :

La couche Entite

- ✓ Les entités sont les classes d’une application correspondant à des tables en base de données.
- ✓ L’entité est le reflet applicatif d’une table de la base de données, ses propriétés étant équivalentes à des colonnes.
- ✓ On peut donc récupérer, ajouter, modifier et supprimer des données en base sans avoir à écrire une seule ligne de code SQL.
- ✓ Toutes ces actions pouvant être effectuées au travers des objets Entité.



Une vue d'ensemble :

La couche ORM (Object Relational Mapping)

- ✓ Elle est l’intermédiaire entre l’application et la couche DBAL.
- ✓ Son rôle est de convertir les données tabulaires reçues depuis le DBAL en entités,
- ✓ Elle transforme les interactions avec les différents objets mis à disposition du développeur en requêtes SQL à transmettre
- ✓ au DBAL (notamment pour les mises à jour).
- ✓ Elle établit une correspondance entre la base de données relationnelle et la POO.



DOCTRINE : STRUCTURE DE LA BD



De Symfony à la BD ?
Ou
De la BD à Symfony :



Les deux sont possibles

MAIS



Il est préférable d'aller de Symfony à la BD

Pourquoi ?

Pour ne pas perdre en intégrité par une mauvaise interprétation des contraintes

Benoît Roche / @rocheb83

11



DOCTRINE : STRUCTURE DE LA BD



Configuration :

Il est impératif de disposer du bundle Doctrine

Le bundle Doctrine est en standard dans les vendors de symfony



Vendor ??? :

- Répertoire généré par Composer qui contient tous les paquets (librairies ou bundles) dont votre projet dépend.
- Composer est chargé de le générer pour chaque installation/déploiement du projet.
- On ne versionne donc pas ce dossier

Benoît Roche / @rocheb83

12



LA STRUCTURE DE LA BASE



DOCTRINE : STRUCTURE DE LA BD



Les étapes de création :



- 1. Configuration des paramètres
- 2. Création de la base de données
- 3. Générer l'entité employe
- 4. Génération de la table correspondant à l'entité employe



DOCTRINE : STRUCTURE DE LA BD



Les étapes de création :

Configuration des paramètres

Modifier le fichier **app/config/parameters.yml**

```
parameters.yml
Source History
1 # This file is auto-generated during the composer ins
2 parameters:
3     database_host: 127.0.0.1
4     database_port: 3306
5     database_name: testymfony
6     database_user: root
7     database_password: null
8     mailer_transport: smtp
9     mailer_host: 127.0.0.1
10    mailer_user: null
11    mailer_password: null
12    secret: a11b98c79ec3ec50bde691579edbd31794aacab9
```

Benoît Roche / @rocheb83

15



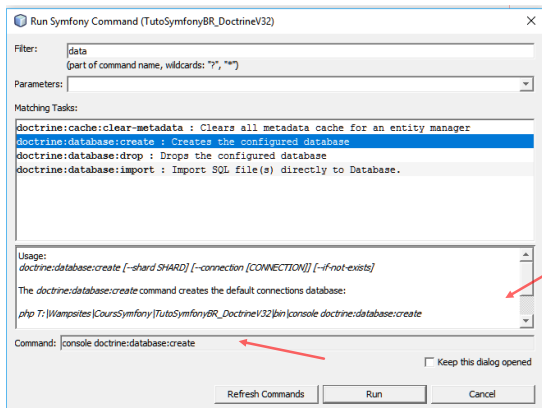
DOCTRINE : STRUCTURE DE LA BD



Les étapes

Création de la base de données:

- ✓ Dans la console de commande Symfony :



run command Doctrine database create



Toujours regarder les options

```
"C:\wamp64\bin\php\php7.0.10\php.exe" "I:\Wampsites\CoursSymfor
Created database 'testymfony' for connection named default
Done.
```

Benoît Roche / @rocheb83

16



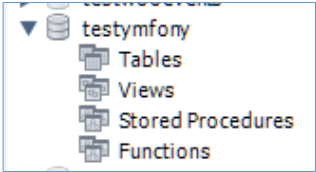
DOCTRINE : STRUCTURE DE LA BD



Les étapes

Création de la base de données

Vérification de l'existence de la bd à l'aide de Mysql Workbench



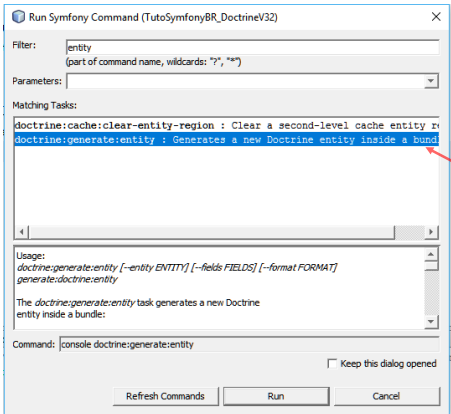
DOCTRINE : STRUCTURE DE LA BD



Les étapes de création :

Générer l'entité Employe

On va gérer une entité (Entity) dans le bundle DoctrineBundle



Il existe une commande symfony ...

Doctrine:generate:entity



A lire !





DOCTRINE : STRUCTURE DE LA BD



Générer l'entité Employe

run command *Doctrine generate entity--entity="AppBundle:Employe"*

Choisir les annotations par défaut pour les Entités

```
Determine the format to use for the mapping information.  
Configuration format (yaml, xml, php, or annotation) [annotation]:
```

les annotations car

- ✓ Leur utilisation est intuitive
- ✓ les configurations sont souvent plus rapides .

Saisir le nom des propriétés

```
date, time, decimal, float, binary, blob, guid.
```

Sauf le champ id !!!

```
nomemp  
Field type [string]:  
Field length [255]: 70  
Is nullable [false]:  
Unique [false]:  
New field name (press <return> to stop adding fields):
```

Pour chaque champ, on saisit le type (string,integer,...), la longueur éventuellement, l'acceptation du null et l'unicité des valeurs

Benoît Roche / @rocheb83

19



DOCTRINE : STRUCTURE DE LA BD



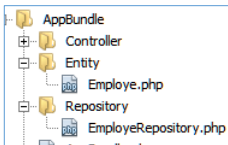
Générer l'entité Employe

Une fois l'entité créée, on voit deux nouveaux dossiers dans le bundle :

- ✓ Entity
- ✓ Repository

Le fichier correspondant à la classe Employe, employe.php est créé dans

Le dossier Entity



```
/**  
 * Employe  
 */  
 * @ORM\Table(name="employee")  
 * @ORM\Entity(repositoryClass="AppBundle\Repository\EmployeeRepository")  
 */  
class Employee  
{  
    /**  
     * @var int  
     */  
    * @ORM\Column(name="id", type="integer")  
    * @ORM\Id  
    * @ORM\GeneratedValue(strategy="AUTO")  
    */  
    private $id;
```

Mais RIEN n'a encore été généré dans la base...

Benoît Roche / @rocheb83

20



DOCTRINE : STRUCTURE DE LA BD



Générer l'entité Employe

On voit dans l'annotation que le colonne Id a été déclarée clé primaire et qu'elle est en numérotation auto:

```
/**
 * @var integer
 *
 * @ORM\Column(name="id", type="integer")
 * @ORM\Id
 * @ORM\GeneratedValue(strategy="AUTO")
 */
private $id;
```



- @ORM\Id : clé primaire
- @ORM\GeneratedValue(strategy="AUTO") : il s'agit de autoincrément

Le contrôle se fait à l'insertion dans la BD

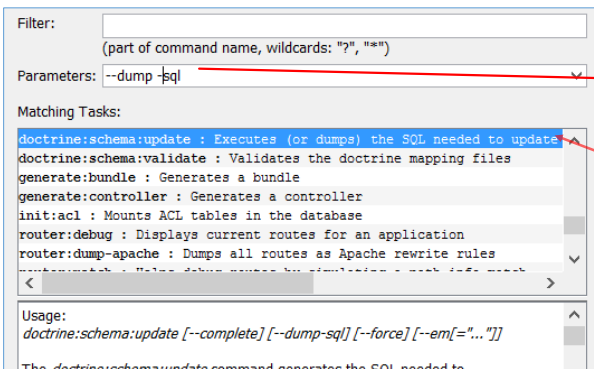


DOCTRINE : STRUCTURE DE LA BD



Générer l'entité Employe

Pour que Doctrine crée la table dans la base, il va falloir mettre à jour le schéma de la BD : C'est la commande **doctrine:schema:update** qui va comparer l'état actuel de notre bdd avec nos entités et générer les ordres SQL de créations des tables afin qu'elles correspondent !



- Permet
- ✓ de générer les ordres sql de mise à jour de la BD
 - ✓ d'afficher les ordres sql générés

```
"C:\wamp64\bin\php\php7.0.10\php.exe" "I:\Wampsites\CoursSymfony\TutoSymfonyBR_D
CREATE TABLE employe (id INT AUTO_INCREMENT NOT NULL, nomemp VARCHAR(70) NOT NUL
Done.
```

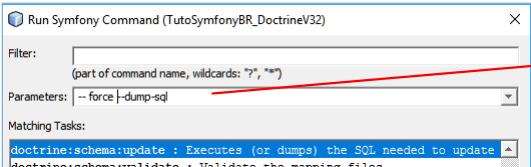


DOCTRINE : STRUCTURE DE LA BD



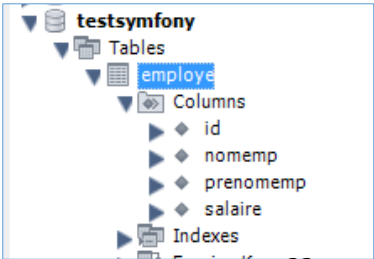
Générer l'entité Employe

Pour que Doctrine crée la table dans la base, il va falloir mettre à jour le schéma de la BD : Ici on n'a généré que le sql, on va utiliser l'option force pour créer réellement la table dans la BD :



- Permet
- D'afficher les ordres SQL ET
 - de mettre à jour la BD

```
"C:\wamp64\bin\php\php7.0.10\php.exe" "T:\Wampsites\CoursSymfony\T
CREATE TABLE employe (id INT AUTO_INCREMENT NOT NULL, nomemp VARCH
Updating database schema...
Database schema updated successfully! "1" query was executed
Done.
```



DOCTRINE : STRUCTURE DE LA BD



Modifier l'entité Employe : rajouter une colonne

en fonction du *mapping* que Doctrine connaît :

- Modifier la classe employé en ajoutant un attribut par exemple ville avec comme valeur par défaut Toulon

```
/**
 * @var string
 *
 * @ORM\Column(name="ville", type="string", length=35 ,options={"default"="Toulon"})
 */
private $ville;
```

- Enregistrer maintenant dans la bdd :

doctrine:schema:update --dump-sql --force

```
"C:\wamp64\bin\php\php7.0.10\php.exe" "T:\Wampsites\CoursSymfony\TutoS
ALTER TABLE employe ADD ville VARCHAR(70) DEFAULT 'Toulon' NOT NULL;
Updating database schema...
Database schema updated successfully! "1" query was executed
Done.
```



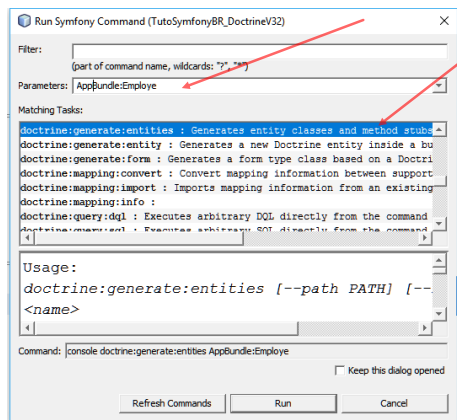
DOCTRINE : STRUCTURE DE LA BD



Modifier l'entité Employe : rajouter une colonne

Lorsque l'on crée un attribut directement dans la classe, il existe une commande symfony pour générer les accesseurs et mutateurs :

doctrine:generate:entities



```
"C:\wamp64\bin\php\php7.0.10\php.exe" "T:\Wampsit
Generating entity "AppBundle\Entity\Employee"
> backing up Employee.php to Employee.php-
> generating AppBundle\Entity\Employee
Done.
```



Voir les options

```
public function setVille($ville)
{
    $this->ville = $ville;

    return $this;
}

/**
 * Get ville
 *
 * @return string
 */
public function getVille()
```

Ce qui a été généré

Benoît Roche / @rocheb83

25



DOCTRINE : STRUCTURE DE LA BD



Modifier l'entité Employe : rajout de 2 index à la table mappée

Dans les annotations de la table, rajouter une annotation correspondant aux index que l'on veut créer :

index sur la colonne ville,

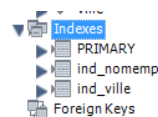
Index sur la colonne nomemp

```
* @ORM\Table(name="employee", indexes={@ORM\Index(name="ind_nomemp", columns={"nomemp"}),@ORM\Index(name="ind_ville", columns={"ville"})})
* * @ORM\Entity(repositoryClass="AppBundle\Repository\EmployeeRepository")
```

Enregistrer maintenant dans la bdd :

doctrine:schema:update --dump-sql --force

On vérifie dans la structure de la table Employee :



Benoît Roche / @rocheb83

26



LES RELATIONS ENTRE ENTITIES

Benoît Roche / @rocheb83

27



DOCTRINE : STRUCTURE DE LA BD



Les relations entre entities

- ✓ Il s'agit ici de créer les liens entre les entities afin de pouvoir récupérer des informations à partir d'un objet d'une classe.
- ✓ Créer des relations consiste à créer des objets ou des collections d'objets comme attributs d'une classe.
- ✓ Exemple : un attribut \$projet (classe Projet) dans la classe (Entity) Employe.
- ✓ On trouvera des liens de type :
 - ✓ **OneToOne**
 - ✓ **ManyToOne**
 - ✓ **ManyToMany**
 - ✓ **OneToMany**

Benoît Roche / @rocheb83

28



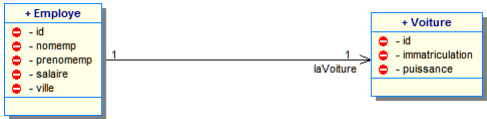
DOCTRINE : STRUCTURE DE LA BD



Les relations entre entities

La relation **OneToOne**
Elle signifie qu'à UN objet d'une Entity correspond UN objet d'une autre Entity.
Exemple : UN Employe possède ou as Une Voiture UN (et une voiture n'est affectée QUE sur un employe)

Représentation UML :



- La relation est unidirectionnelle.
- On a choisi de privilégier le lien de Employee vers Vehicule...**Il s'agit d'un choix de conception**



DOCTRINE : STRUCTURE DE LA BD



Les relations entre entities

- On va créer :
- une annotation `@ORM\OneToOne` dans l'entité Employe :
 - Pour un attribut `$laVoiture` (qui correspond au rôle UML)

```
/**
 * @ORM\OneToOne(targetEntity="AppBundle\Entity\Voiture")
 */
private $laVoiture;
```

Employe



`$laVoiture` sera un objet de la classe Voiture
On n'aura rien à faire dans l'entité Voiture



DOCTRINE : STRUCTURE DE LA BD



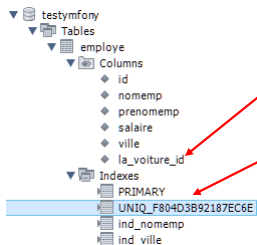
Les relations entre entités

Après mise à jour du schéma de la BD :

Ordres sql générés

```
ALTER TABLE employe ADD la_voiture_id INT DEFAULT NULL;  
ALTER TABLE employe ADD CONSTRAINT FK_F804D3B92187EC6E FOREIGN KEY (la_voiture_id) REFERENCES voiture (id);  
CREATE UNIQUE INDEX UNIQ_F804D3B92187EC6E ON employe (la_voiture_id);
```

Table Employe :



La clé étrangère a été créée dans la table projet.

Mais aussi un index unique sur la clé étrangère pour garantir qu'une voiture ne peut être affectée plus d'une fois !

Index: **UNIQ_F804D3B92187EC6E**

Definition:
Type BTREE
Unique Yes
Columns la_voiture_id



Voir documentation pour choisir le nom de la clé étrangère

Benoît Roche / @rocheb83

31



DOCTRINE : STRUCTURE DE LA BD



Les relations entre entités

La relation **ManyToOne**

Elle signifie qu'à Plusieurs objets d'une Entity correspond UN objet d'une autre Entity

Exemple : PLUSIEURS Employes Travaillent sur UN projet (Un employé travaille sur un projet et que sur un projet travaillent plusieurs employés)

Représentation UML :



- La relation est unidirectionnelle.
- On a choisi de privilégier le lien de Employee vers Projet...**Il s'agit d'un choix de conception**

Benoît Roche / @rocheb83

32



DOCTRINE : STRUCTURE DE LA BD



Les relations entre entités

On va créer :

- une annotation @ORM\ManyToOne dans l'entité Employe :
- Pour un attribut \$leProjet (qui correspond au rôle UML)

```
/**
 * @ORM\ManyToOne(targetEntity="AppBundle\Entity\Projet")
 */
private $leProjet;
```

Employe



\$leProjet sera un objet de la classe Projet

On n'aura rien à faire dans l'entité Projet

Benoît Roche / @rocheb83

33



DOCTRINE : STRUCTURE DE LA BD



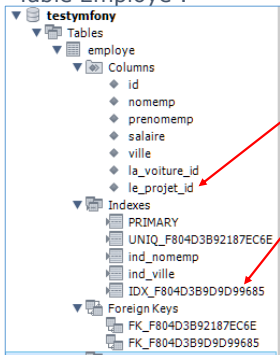
Les relations entre entités

Après mise à jour du schéma de la BD :

Ordres sql générés

```
C:\wamp\bin\php\php7.0.10\php.exe -f C:\wamp\bin\php\php7.0.10\php.exe --a
ALTER TABLE employe ADD le_projet_id INT DEFAULT NULL;
ALTER TABLE employe ADD CONSTRAINT FK_F804D3B9D9D99685 FOREIGN KEY (le_projet_id) REFERENCES projet (id);
CREATE INDEX IDX_F804D3B9D9D99685 ON employe (le_projet_id);
```

Table Employe :



La clé étrangère a été créée dans la table projet.

Mais aussi un index sur la clé étrangère .. Pas unique cette fois !

Index: IDX_F804D3B9D9D99685

Definition:
Type BTREE
Unique No
Columns le_projet_id



Voir documentation pour choisir le nom de la clé étrangère

Benoît Roche / @rocheb83

34



DOCTRINE : STRUCTURE DE LA BD



Les relations entre entites

La relation **ManyToMany**

Elle signifie qu'à Plusieurs objets d'une Entity Correspondent PLUSIEURS objets d'une autre Entity.

Exemple : PLUSIEURS Employés s'inscrivent à PLUSIEURS Séminaires

Représentation UML :



- La relation est unidirectionnelle.
- On a choisi de privilégier le lien de Employee vers Seminaire...**Il s'agit d'un choix de conception**



DOCTRINE : STRUCTURE DE LA BD



Les relations entre entites

On va créer :

- une annotation @ORM\ManyToMany dans l'entité Employee :


```
/**
 * @ORM\ManyToMany(targetEntity="AppBundle\Entity\Seminaire")
 */
private $lesSeminaires;
```

Employee




\$lesSeminaires sera une collection d'objets de la classe Seminaire

On n'aura rien à faire dans l'entité Seminaire



DOCTRINE : STRUCTURE DE LA BD



Les relations entre entités

Après mise à jour du schéma de la BD :

Ordres sql générés

```
CREATE TABLE employe_seminaire (employe_id INT NOT NULL, seminaire_id INT NOT NULL, dateDebutSeminai
CREATE TABLE seminaire (id INT AUTO_INCREMENT NOT NULL, dateDebutSeminai
ALTER TABLE employe_seminaire ADD CONSTRAINT FK_71D717C61B65292 FOREIGN
ALTER TABLE employe_seminaire ADD CONSTRAINT FK_71D717C6CEA14D8 FOREIGN
```

Table Employe :

testsymfony

Tables

employe

employe_seminaire

Columns

employe_id

seminaire_id

Indexes

Foreign Keys

FK_71D717C61B65292

FK_71D717C6CEA14D8


Triggers

projet

Il y a eu création d'une table de jointure

Avec ses 2 clés étrangères



Cette table sera transparente pour notre application Symfony



Voir documentation pour choisir le nom de la table de jointure et de ses champs

Benoît Roche / @rocheb83

37



LES RELATIONS BIDIRECTIONNELLES

Benoît Roche / @rocheb83

38

Slam4

19



DOCTRINE : STRUCTURE DE LA BD



Les relations bi-directionnelles

Jusqu'à maintenant, nous avons vu uniquement des relations unidirectionnelles. Dans certains cas, la navigabilité entre les Entities (classes) doit se faire de manière symétrique.

Exemples :

UN employé est affecté dans UN service. UN service a PLUSIEURS employés

Un employé a PLUSIEURS compétences pour UNE compétence on a PLUSIEURS employés



Dans tous les cas, une Entity sera propriétaire et l'autre sera inverse.



DOCTRINE : STRUCTURE DE LA BD

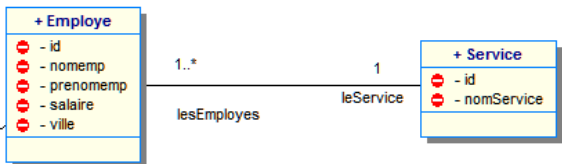


Les relations bidirectionnelles

Cas des relations ManyToOne
Exemple : UN employé est affecté dans UN service. UN service a PLUSIEURS employés

- On doit pouvoir naviguer dans les 2 sens :
- ✓ Trouver le service d'un employé,
 - ✓ Trouver les employés d'un service

Représentation UML :





DOCTRINE : STRUCTURE DE LA BD



Les relations bidirectionnelles

Cas des relations ManyToOne
Exemple : UN employé est affecté dans UN service. UN service a PLUSIEURS employés

On va dans ce cas :

- Créer une relation ManyToOne dans l'entity Employe (propriétaire)
- Créer une relation OneToMany INVERSE dans Service (inverse)

Nouvelles options

```
/**
 * @ORM\ManyToOne(targetEntity="AppBundle\Entity\Service", inversedBy="lesEmployes")
 * @ORM\JoinColumn(nullable=false)
 */
private $leService;

/**
 * @ORM\OneToMany(targetEntity="AppBundle\Entity\Employe", mappedBy="leService")
 */
private $lesEmployes;
```

Employe

Service

Benoît Roche / @rocheb83

41



DOCTRINE : STRUCTURE DE LA BD

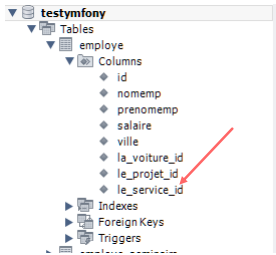


Les relations bidirectionnelles

Cas des relations ManyToOne
Exemple : UN employé est affecté dans UN service. UN service a PLUSIEURS employés

Au niveau de la base de données :

```
ALTER TABLE employe ADD le_service_id INT NOT NULL;
ALTER TABLE employe ADD CONSTRAINT FK_F804D3B9CE5CD64D FOREIGN KEY (le_service_id) REFERENCES service (id);
CREATE INDEX IDX_F804D3B9CE5CD64D ON employe (le_service_id);
```



Il n'y a QUE la clé étrangère le_service_id qui a été créée
Doctrine gèrera le tout !!!

Benoît Roche / @rocheb83

42



DOCTRINE : STRUCTURE DE LA BD



Les relations bidirectionnelles

Cas des relations ManyToMany
Exemple : Un employé a PLUSIEURS compétences pour UNE compétence on a PLUSIEURS employés

- On doit pouvoir naviguer dans les 2 sens :
- ✓ Trouver les compétences d'un employé. Un employé peut ne pas avoir de compétences.
 - ✓ Trouver les employés ayant une compétence donnée.

Représentation UML :



DOCTRINE : STRUCTURE DE LA BD



Les relations bidirectionnelles

Cas des relations ManyToOne
Exemple : UN employé est affecté dans UN service. UN service a PLUSIEURS employés

- On va dans ce cas :
- Créer une relation ManyToMany dans l'entity Employe (propriétaire)
 - Créer une relation ManyToMany INVERSE dans Competence (inverse)

```
/**
 * @ORM\ManyToMany(targetEntity="AppBundle\Entity\Competence", inversedBy="lesEmployes")
 */
private $lesCompetences;
```

```
/**
 * @ORM\ManyToMany(targetEntity="AppBundle\Entity\Employee", mappedBy="lesCompetences")
 * @ORM\JoinColumn(nullable=false)
 */
private $lesEmployes;
```

Employee

Competence

Au moins un employé a la compétence



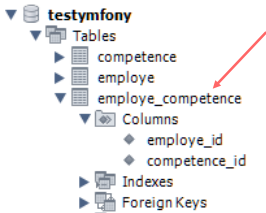
DOCTRINE : STRUCTURE DE LA BD



Les relations bidirectionnelles

Cas des relations ManyToMany
Exemple : Un employé a PLUSIEURS compétences pour UNE compétence on a PLUSIEURS employés
Au niveau de la base de données :

```
CREATE TABLE employee_competence (employee_id INT NOT NULL, competence_id INT NOT NULL, INDEX IDX_161DB2B81B65292 (employee_id), INDEX IDX_161DB2B81B65292 (competence_id));
ALTER TABLE employee_competence ADD CONSTRAINT FK_161DB2B81B65292 FOREIGN KEY (employee_id) REFERENCES employee (id) ON DELETE CASCADE;
ALTER TABLE employee_competence ADD CONSTRAINT FK_161DB2B815761DAB FOREIGN KEY (competence_id) REFERENCES competence (id) ON DELETE CASCADE;
```



La table de jointure employee_competence a été créée
Doctrine gèrera le tout !!!



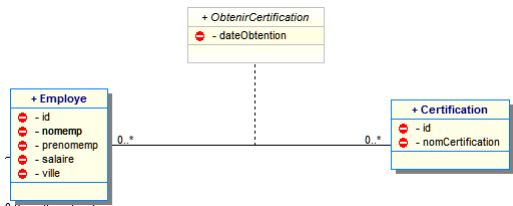
DOCTRINE : STRUCTURE DE LA BD



Les relations bidirectionnelles

Cas des associations porteuses de données
Exemple : Un employé obtient une certification à une date donnée

Représentation UML:



- ✓ Dans ce cas, il faut faire une Entity ObtenirCertification et
- ✓ y rajouter 2 relations ManyToOne vers les Entities Employee et Certification



DOCTRINE : STRUCTURE DE LA BD



Les relations entre entités

```
private $id;

/**
 * @var Employee
 *
 * @ORM\ManyToOne(targetEntity="AppBundle\Entity\Employee")
 * @ORM\JoinColumn(nullable=false)
 */
private $leEmploye;

/**
 * @var Certification
 *
 * @ORM\ManyToOne(targetEntity="AppBundle\Entity\Certification")
 * @ORM\JoinColumn(nullable=false)
 */
private $laCertification;

/**
 * @var \DateTime
 *
 * @ORM\Column(name="dateObtention", type="date")
 */
private $dateObtention;
```

ObtenirCertification



\$On remarque que les objets \$leEmploye et \$laCertification ne peuvent être null

Benoît Roche / @rocheb83

47



ENRICHIr LES RELATIONS

Benoît Roche / @rocheb83

48



DOCTRINE : STRUCTURE DE LA BD



Les relations bidirectionnelles

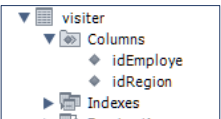
Donner des noms à la table de jointure et aux colonnes de cette table :
Soit la relation bidirectionnelle suivante :

```
/**
 * @ORM\ManyToMany(targetEntity="AppBundle\Entity\Region", mappedBy="lesEmployes")
 * @ORM\JoinTable(name="Visiter",
 *    joinColumns={@ORM\JoinColumn(name="idEmploye", referencedColumnName="id")},
 *    inverseJoinColumns={@ORM\JoinColumn(name="idRegion", referencedColumnName="id")}
 * )
 */
private $lesRegions;

/**
 *
 * @ORM\ManyToMany(targetEntity="AppBundle\Entity\Employee", inversedBy="lesRegions")
 * @ORM\JoinTable(name="Visiter",
 *    joinColumns={@ORM\JoinColumn(name="idEmploye", referencedColumnName="id")},
 *    inverseJoinColumns={@ORM\JoinColumn(name="idRegion", referencedColumnName="id")}
 * )
 */
private $lesEmployes;
```

On redonne les noms des tables et colonnes dans l'entité inverse

Résultat :



Benoît Roche / @rocheb83

49



DOCTRINE : STRUCTURE DE LA BD



Les relations bidirectionnelles

Les suppressions en cascade :

Au moment de la création de la relation, il existe un moyen de générer au niveau de la BD des suppressions en cascade : le on delete cascade sql :
Reprenons l'Entité Seminaire :



Le code suivant générera l'option On delete cascade sur la FK idCours de tla table seminaire :

```
/**
 * @ORM\ManyToOne(targetEntity="AppBundle\Entity\Cours")
 * @ORM\JoinColumn(onDelete="CASCADE")
 */
private $leCours;
```

Foreign Key: FK_B0F91190B0815359

Definition:

| | |
|-----------|--------------------------|
| Target | cours (le_cours_id → id) |
| On Update | RESTRICT |
| On Delete | CASCADE |

ALTER TABLE seminaire ADD CONSTRAINT FK_B0F91190B0815359 FOREIGN KEY (le_cours_id) REFERENCES cours (id) ON DELETE CASCADE;

Benoît Roche / @rocheb83

50



DOCTRINE : STRUCTURE DE LA BD



Les relations bidirectionnelles

Les suppressions en cascade :

Application :

| Table cours | |
|-------------|------------|
| id | nomCours |
| 1 | Le cours 1 |
| 2 | Le cours 2 |
| 3 | Le cours 3 |
| * NULL | NULL |

| Table seminaire | | |
|-----------------|-------------|--------------------|
| id | le_cours_id | dateDebutSeminaire |
| 13 | 1 | 2017-09-12 |
| 14 | 1 | 2017-09-12 |
| 15 | 2 | 2017-09-12 |
| * NULL | NULL | NULL |

En exécutant ce code, on a :

```
/**
 * @Route("/SupprimerCoursOnDeleteCascade", name="DeleteCoursODC")
 */
public function coursOnDeleteCascadeeAction()
{
    $em=$this->getDoctrine()->getManager();
    $repoitory= $em->getRepository('AppBundle:Cours');
    $unCours=$repoitory->find(1);
    $em->remove($unCours);
    $em->flush();

    return new \Symfony\Component\HttpFoundation\Response('suppressionde cours OK');
}
```

| id | nomCours |
|--------|------------|
| 2 | Le cours 2 |
| 3 | Le cours 3 |
| * NULL | NULL |

| id | le_cours_id | dateDebutSeminaire |
|--------|-------------|--------------------|
| 15 | 2 | 2017-09-12 |
| * NULL | NULL | NULL |

Table le cours d'id 1 et ses séminaires ont été supprimés

Benoît Roche / @rocheb83



DOCTRINE : STRUCTURE DE LA BD



Exercice:

Il est temps de mettre en pratique tout ceci....

[Voir : exercices Doctrine](#)

Benoît Roche / @rocheb83



Fin du cours, merci

Question/Réponses

Benoît Roche
@rocheb83