


Benoît Roche
@rocheb83

Benoît Roche / @rocheb83

1

1



LE PRINCIPLE

Benoît Roche / @rocheb83

2

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 une application dans une base de données. Parfois dans plusieurs bases

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

Benoît Roche / @rocheb83

3

3



DOCTRINE : STRUCTURE DE LA BD



Doctrine est totalement découplé de Symfony et son utilisation est optionnelle.

On peut aussi travailler avec les données utiliser

- ✓ un autre ORM comme *Eloquent ORM* (Laravel)
- ✓ ou directement les fonctionnalités PDO.. Pas très conseillé notamment en raison de la sécurité.

Doctrine2 se compose de plusieurs couches :

- ✓ DBAL,
- ✓ ORM et
- ✓ Entité

Benoît Roche / @rocheb83

4

4

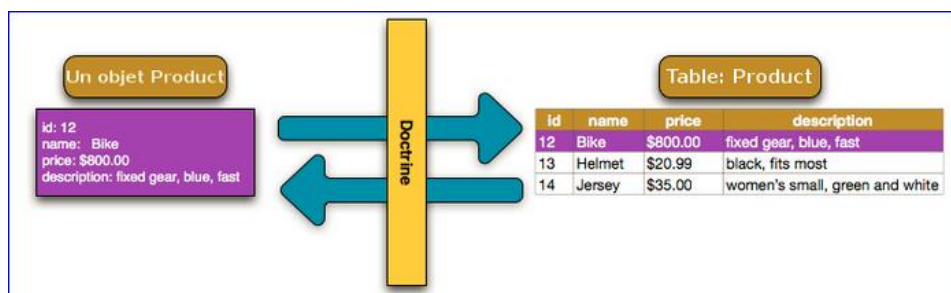


DOCTRINE : STRUCTURE DE LA BD

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

5



DOCTRINE : STRUCTURE DE LA BD

sur un exemple : *Application gestion des employés*

Classe employe \longrightarrow **entité employe** \longrightarrow **table employe**
attributs : id, nom, prenom, date de naissance, salaireM

Classe projet \longrightarrow **entité projet** \longrightarrow **table projet**
attributs : id, nomProjet, dureePrevue

Et le lien entre les deux ???

Benoît Roche / @rocheb83

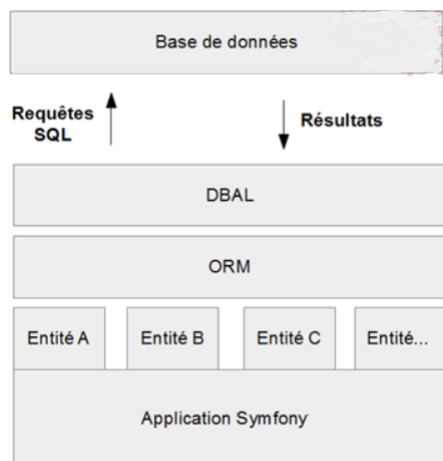
6

6



DOCTRINE : STRUCTURE DE LA BD

Une vue d'ensemble :



DBAL (Database Abstraction Layer)

ORM (Object Relational Mapping)

Entite on Entity

Benoît Roche / @rocheb83

7

7



DOCTRINE : STRUCTURE DE LA BD

Une vue d'ensemble :

La couche DBAL (Database Abstraction Layer)

- ✓ C'est la couche de plus bas niveau située au dessus de PDO
- ✓ 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 bien plus complète.

Benoît Roche / @rocheb83

8

8

**DOCTRINE : STRUCTURE DE LA BD****Une vue d'ensemble :*****La couche Entite***

- ✓ Les entités sont les classes d'une application correspondant à des tables en base de données. Elles sont donc fonctionnelles.
- ✓ L'entité est le reflet applicatif d'une table de la base de données, ses propriétés étant équivalentes aux colonnes de la table en base de données .
- ✓ 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.

Benoît Roche / @rocheb83

9

9

**DOCTRINE : STRUCTURE DE LA BD****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 (objets ou collection d'objets).
- ✓ Elle transforme les interactions avec les différents objets mis à disposition du développeur en requêtes SQL à transmettre au DBAL
- ✓ Elle établit une correspondance entre la base de données relationnelle et la POO.

Benoît Roche / @rocheb83

10

10



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 ?



Souvent les BD ne sont pas ad hoc par rapport aux bonnes pratiques, notamment au niveau des clés étrangères et des contraintes d'intégrité !!!
Doctrine ne sait pas gérer

Benoît Roche / @rocheb83

11

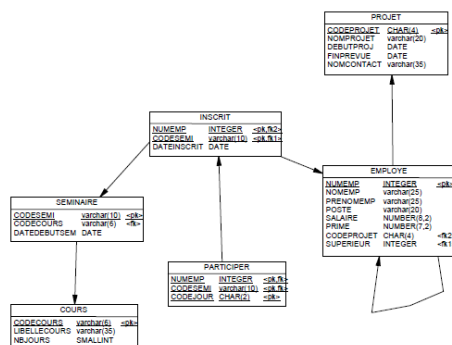
11



DOCTRINE : STRUCTURE DE LA BD



Exemple : tentative de génération d'entités à partir de la base du cours de SQL :



L'importation va échouer à cause de la relation PK/FK entre Participer et Inscrit. Les clés primaires de chacune des tables est composée ...



```
PS T:\Wampsites\CoursSymfony\FromDatabaseCours> php bin/console doctrine:mapping:import "App\Entity" annotation --path=s
rc/Entity

In MappingException.php line 598:

It is not possible to map entity 'Inscrit' with a composite primary key as part of the primary key of another entit
y 'Participer#codesemi'.
```

Benoît Roche / @rocheb83

12

12



LA STRUCTURE DE LA BASE

Benoît Roche / @rocheb83

13

13



DOCTRINE : STRUCTURE DE LA BD

Les étapes de création :



1. Configuration des paramètres :
 - ✓ fichiers .env (racine du projet)
 - ✓ et config/packages/doctrine.yaml

2. Création de la base de données `php bin/console doctrine:database:create`

3. Génération des entités `php bin/console make:entity`

4. Mise à jour du schéma de la BD à l'aide de fichiers migrations

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



Les étapes 3 et 4 peuvent être répétées pour la modification/création d'entités

Benoît Roche / @rocheb83

14

14



DOCTRINE : STRUCTURE DE LA BD



Paramètres de la base

- ✓ Les paramètres seront vus en détail dans le TD
- ✓ 2 fichiers sont à configurer :
 - ✓ Le fichier config/packages/doctrine.yaml
Pour les options de connexion et éventuellement les différentes connexions
 - ✓ Le fichier .env
Pour les paramètres de connexion

```
doctrine50:
    driver: 'pdo_mysql'
    server_version: '8.0'
    charset: utf8mb4
    default_table_options:
        charset: utf8mb4
        collate: utf8mb4_unicode_ci
    dbname: '%env(DATABASE_NAME)%'
    host: '%env(DATABASE_HOST)%'
    user: '%env(DATABASE_USER)%'
    password: '%env(DATABASE_PWD)%'
```

```
DATABASE_URL=mysql://db_user:db_password@127.0.0.1:3306/db_name
### doctrine 50
DATABASE_USER=userdoctrine50
DATABASE_PWD=userdoctrine50
DATABASE_NAME=doctrine50
DATABASE_HOST=localhost
```

Benoît Roche / @rocheb83

15

15



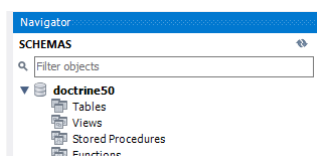
DOCTRINE : STRUCTURE DE LA BD



Création de la BD

- ✓ C'est la commande
- ✓ `doctrine:database:create`

```
PS T:\Wampsites\CoursSymfony\doctrine50> symfony console doctrine:database:create --connection databasecreator
Created database `doctrine50` for connection named databasecreator
PS T:\Wampsites\CoursSymfony\doctrine50> █
```



```
create user 'userdoctrine50'@'localhost' identified by 'userdoctrine50';
grant all privileges on doctrine50.* to 'userdoctrine50'@'localhost';
```



La base de données a été créée avec un utilisateur habilité à créer des BD

Un utilisateur a été créé pour faire toutes les opérations sur la base
uniquement depuis le serveur web (localhost pour l'instant)

Benoît Roche / @rocheb83

16

16



CRÉATION DES ENTITIES

Benoît Roche / @rocheb83
17

17

DOCTRINE : STRUCTURE DE LA BD

Création des entites

- ✓ Les paramètres seront vus en détail dans le TD
- ✓ C'est la commande
 - ✓ ***symfony console make:entity***
- ✓ Il suffit de se laisser guider,
- ✓ 2 classes seront créées :
 - La classe Entity contenant les attributs, getters/setters et constructeur
 - La classe Repository vide, qui pourra contenir des fonctions encapsulant par exemple des requêtes DQL

```

PS T:\Wampsites\CoursSymfony\doctrine50> symfony console make:entity

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

created: src/Entity/Voiture.php
created: src/Repository/VoitureRepository.php
  
```

Benoît Roche / @rocheb83
18

18



DOCTRINE : STRUCTURE DE LA BD

Création des entités

- ✓ Les paramètres seront vus en détail dans le TD
- ✓ Ne jamais saisir l'id, il sera créé automatiquement
- ✓ Pour les types, il sera possible de créer
 - ✓ Un type courant (string, int,..)
 - ✓ Une relation (voir plus loin),
 - ✓ Les types array, json, object...

```
field type (enter ? to see all types) [string].
> ?

Main Types
* string
* text
* boolean
* integer (or smallint, bigint)
* float

Relationships/Associations
* relation (a wizard will help you build the relation)
* ManyToOne
* OneToMany
* ManyToMany
* OneToOne

Array/Object Types
* array (or simple_array)
* json
* object
* binary
* blob

Date/Time Types
* datetime (or datetime_immutable)
* datetimetz (or datetimetz_immutable)
* date (or date_immutable)
* time (or time_immutable)
* dateinterval

Other Types
* ascii_string
* decimal
* guid
```

Benoît Roche / @rocheb83

19

19



LES MIGRATIONS

Benoît Roche / @rocheb83

20

20



DOCTRINE : STRUCTURE DE LA BD



Les migrations

Une Entity étant une classe, la table correspondante en BD n'existe pas encore.

Pour persister les entités et leurs relations, doctrine de faire appel à un système de migrations.

La mise à jour de la base se fera en 2 temps

La création d'un fichier de mise à jour : ils contiendront les ordres de mise à jour de la base de données

La mise à jour elle-même par l'exécution des fichiers de migration

Benoît Roche / @rocheb83

21

21



DOCTRINE : STRUCTURE DE LA BD



Les migrations

La création d'un fichier de migration :

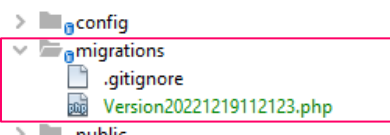
symfony console make:migration

```
PS T:\Wampsites\CoursSymfony\doctrine50> symfony console make:migration
```

Success!

```
Next: Review the new migration "migrations/Version20221219112123.php"
Then: Run the migration with php bin/console doctrine:migrations:migrate
See https://symfony.com/doc/current/bundles/DoctrineMigrationsBundle/index.html
PS T:\Wampsites\CoursSymfony\doctrine50>
```

Un fichier horodaté a été créé, contenant les ordres de mise à jour de la BD



Benoît Roche / @rocheb83

22

22



DOCTRINE : STRUCTURE DE LA BD

Les migrations

Exemple de fichiers de migration :

```
final class Version20221219112123 extends AbstractMigration
{
    public function getDescription(): string
    {
        return '';
    }

    public function up(Schema $schema): void
    {
        // this up() migration is auto-generated, please modify it to your needs
        $this->addSql('CREATE TABLE voiture (id INT AUTO_INCREMENT NOT NULL, immatriculation VARCHAR(10) NOT NULL, puissance INT NOT NULL);');
    }

    public function down(Schema $schema): void
    {
        // this down() migration is auto-generated, please modify it to your needs
        $this->addSql('DROP TABLE voiture');
    }
}
```

Up : mise à jour

Down : annulation des mises à jour

Benoît Roche / @rocheb83

23

23



DOCTRINE : STRUCTURE DE LA BD

Les migrations

Nous remarquons sur la diapositive précédente :

Que chaque fichier de migration est une classe *final*

Que les classes sont horodatées pour préserver leur chronologie

Que les classes contiennent 2 méthodes :

- ✓ up : contient les ordres SQL de mise à jour de la base de données
- ✓ down : contient les ordres qui annulent les ordres de mise à jour

Benoît Roche / @rocheb83

24

24



DOCTRINE : STRUCTURE DE LA BD

Les migrations

Exécution des migrations non encore exécutées :

Symfony console doctrine:migrations:migrate

```
PS T:\Wampsites\CoursSymfony\doctrine50> Symfony console doctrine:migrations:migrate

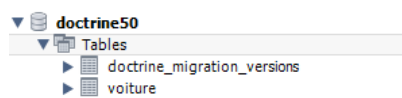
WARNING! You are about to execute a migration in database "doctrine50" that could result in schema changes and data loss.
Are you sure you wish to continue? (yes/no) [yes]:
>

[notice] Migrating up to DoctrineMigrations\Version20221219112123
[notice] finished in 62.5ms, used 20M memory, 1 migrations executed, 1 sql queries

PS T:\Wampsites\CoursSymfony\doctrine50>
```



La base de données a été mise à jour :



Result Grid			Filter Rows:
id	immatriculation	puissance	
*	NULL	NULL	NULL

Benoît Roche / @rocheb83

25

25



LES RELATIONS ENTRE ENTITIES

Benoît Roche / @rocheb83

26

26



DOCTRINE : STRUCTURE DE LA BD



Les relations entre entités

- ✓ Il s'agit ici de créer les liens entre les entités 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

27

27

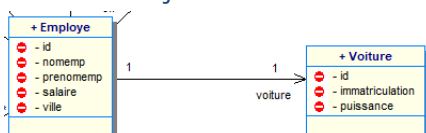


DOCTRINE : STRUCTURE DE LA BD



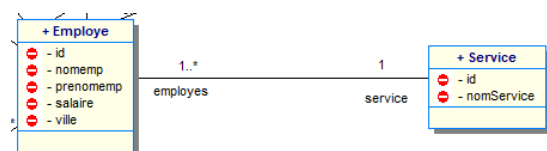
Les relations entre entités

- ✓ De plus, une relation peut-être :
- ✓ **Unidirectionnelle** : la navigation ne se fait que dans un sens. Seule l'entité prioritaire contiendra un objet ou une collection d'objets permettant de relier l'entité non prioritaire



Employee aura un attribut voiture de la classe Voiture

- ✓ **Bidirectionnelle** : la navigation se fera dans les 2 sens : les 2 entités contiendront un objet ou une collection d'objets permettant de "naviguer" d'une entité vers l'autre



Employee aura un attribut service de la classe Service

Service aura un attribut employees qui sera une collection d'objets de la classe Employee

Benoît Roche / @rocheb83

28

28

LES RELATIONS UNIDIRECTIONNELLES

Benoît Roche / @rocheb83

29

29

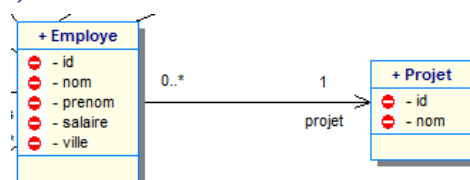
Les relations entre entités

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 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

30

30



DOCTRINE : STRUCTURE DE LA BD



Les relations entre entités

On va créer :

- une annotation #[ORM\ManyToOne] dans l'entité Employe :
- Et l'attribut \$leProjet (qui correspond au rôle UML)

```
#[ORM\ManyToOne]
#[ORM\JoinColumn(nullable: false)]
private ?Projet $projet = null;
```

Employe



\$projet sera un objet de la classe Projet

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

Benoît Roche / @rocheb83

31

31



DOCTRINE : STRUCTURE DE LA BD

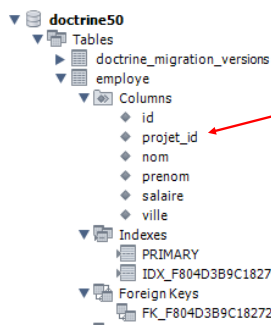


Les relations entre entités

Après mise à jour du schéma de la BD :

Ordres sql générés

Table Employe :



```
public function up(Schema $schema): void
{
    // this up() migration is auto-generated, please modify it to your needs
    $this->addSql('CREATE TABLE employe (id INT AUTO_INCREMENT NOT NULL, projet_id INT NOT NULL, nom VARCHAR(100) NOT NULL, prenom VARCHAR(100) NOT NULL, salaire INT NOT NULL, ville VARCHAR(100) NOT NULL)');
    $this->addSql('CREATE TABLE projet (id INT AUTO_INCREMENT NOT NULL, nom VARCHAR(100) NOT NULL, prenom VARCHAR(100) NOT NULL, salaire INT NOT NULL, ville VARCHAR(100) NOT NULL)');
    $this->addSql('ALTER TABLE employe ADD CONSTRAINT FK_F804D3B9C18272 FOREIGN KEY (projet_id) REFERENCES projet (id)');
}
```

La clé étrangère a été créée dans la table projet.

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



Voir documentation pour choisir le nom de la clé étrangère

Benoît Roche / @rocheb83

32

32



DOCTRINE : STRUCTURE DE LA BD



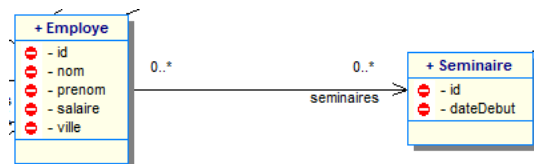
Les relations entre entités

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

Benoît Roche / @rocheb83

33

33



DOCTRINE : STRUCTURE DE LA BD



Les relations entre entités

On va créer :

- une annotation #[ORM\ManyToMany] dans l'entité Employee :
- Et l'attribut \$lesSeminaires (qui correspond au rôle UML)

```

#[ORM\ManyToMany(targetEntity: Seminaire::class)]
private Collection $seminaires;

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

Employee



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

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

Benoît Roche / @rocheb83

34

34



DOCTRINE : STRUCTURE DE LA BD

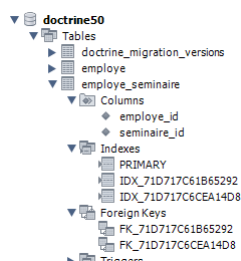
Les relations entre entites

Après mise à jour du schéma de la BD :

Ordres sql générés

```
$this->addSql('CREATE TABLE employe_seminaire (employe_id INT NOT NULL, seminaire_id INT
$this->addSql('ALTER TABLE employe_seminaire ADD CONSTRAINT FK_71D717C61B65292 FOREIGN K
$this->addSql('ALTER TABLE employe_seminaire ADD CONSTRAINT FK_71D717C6CEA14D8 FOREIGN K
```

Table employe_seminaire :



Il y a eu création d'une table de jointure : *employe_seminaire*
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

35

35



DOCTRINE : STRUCTURE DE LA BD

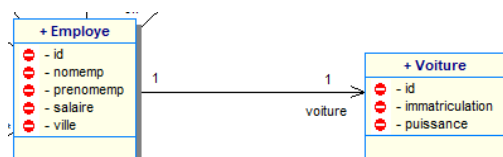
Les relations entre entites

La relation **OneToOne**

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

Exemple : UN Employe possède Une Voiture (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

Benoît Roche / @rocheb83

36

36



DOCTRINE : STRUCTURE DE LA BD



Les relations entre entités

On va dans ce cas créer :

- une annotation #[ORM\OneToOne] dans l'entité Employe :
- Et l'attribut \$laVoiture (qui correspond au rôle UML)

```
#[ORM\OneToOne(cascade: ['persist', 'remove'])]
#[ORM\JoinColumn(nullable: false)]
private ?Voiture $voiture = null;
```

Employe



\$voiture sera un objet de la classe Voiture

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

Benoît Roche / @rocheb83

37

37



DOCTRINE : STRUCTURE DE LA BD



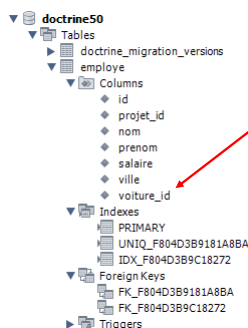
Les relations entre entités

Après mise à jour du schéma de la BD :

```
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);
```

Ordres sql générés

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 pas être affectée plus d'une fois !

Index: UNIQ_F804D3B9181A8BA

Definition:

Type	BTREE
Unique	Yes
Visible	Yes
Columns	voiture_id



Voir documentation pour choisir le nom de la clé étrangère

Benoît Roche / @rocheb83

38

38



LES RELATIONS BIDIRECTIONNELLES

Benoît Roche / @rocheb83

39

39



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 pour répondre à un besoin fonctionnel.

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.

Benoît Roche / @rocheb83

40

40



DOCTRINE : STRUCTURE DE LA BD



Les relations bidirectionnelles

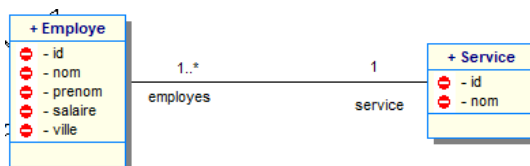
Cas des relations ManyToOne

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

On doit pouvoir naviguer dans les 2 sens :

- ✓ Pour trouver le service d'un employé,
- ✓ Pour trouver les employés d'un service

Représentation UML :



Benoît Roche / @rocheb83

41

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

On va dans ce cas :

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

```
#[ORM\ManyToOne(inversedBy: 'employes')]
#[ORM\JoinColumn(nullable: false)]
private ?Service $service = null;
```

Employe

```
#[ORM\OneToMany(mappedBy: 'service', targetEntity: Employe::class)]
private Collection $employes;
```

Service

Benoît Roche / @rocheb83

42

42



DOCTRINE : STRUCTURE DE LA BD



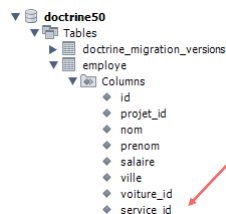
Les relations bidirectionnelles

Cas des relations ManyToOne

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

Au niveau de la base de données :

```
public function up(Schema $schema): void
{
    // this up() migration is auto-generated, please modify it to your needs
    $this->addSql('ALTER TABLE employe ADD service_id INT NOT NULL');
    $this->addSql('ALTER TABLE employe ADD CONSTRAINT FK_F804D3B9ED5CA9E6 FOREIGN KEY (service_id)');
    $this->addSql('CREATE INDEX IDX_F804D3B9ED5CA9E6 ON employe (service_id)');
```



Il n'y a QUE la clé étrangère le_service_id qui a été créée avec son index.

Doctrine gèrera le tout !!!



Benoît Roche / @rocheb83

43

43



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 :

- ✓ Pour trouver les compétences d'un employé. Un employé peut ne pas avoir de compétences.
- ✓ Pour trouver les employés ayant une compétence donnée.

Représentation UML :



Benoît Roche / @rocheb83

44

44



DOCTRINE : STRUCTURE DE LA BD



Les relations bidirectionnelles

Cas des relations ManyToMany

Exemple : UN employé a PLUSIEURS compétences. UNE compétences 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: Competence::class, inversedBy: 'employees')]  
private Collection $competences;
```

Employe

```
#[ORM\ManyToMany(targetEntity: Employe::class, mappedBy: 'competences')]  
private Collection $employees;
```

Competence

Benoît Roche / @rocheb83

45

45



DOCTRINE : STRUCTURE DE LA BD



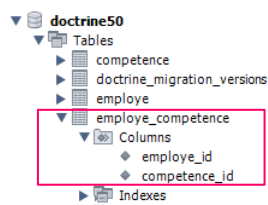
Les relations bidirectionnelles

Cas des relations ManyToMany

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

Au niveau de la base de données :

```
$this->addSql('CREATE TABLE employe_competence (employe_id INT NOT NULL, competence_id INT NOT NULL, INDEX IDX_1  
$this->addSql('ALTER TABLE employe_competence ADD CONSTRAINT FK_161DB2B81B65292 FOREIGN KEY (employe_id) REFEREN  
$this->addSql('ALTER TABLE employe_competence ADD CONSTRAINT FK_161DB2B815761DAB FOREIGN KEY (competence_id) REF
```



La table de jointure employe_competence a été créée

Doctrine gèrera le tout !!!

Benoît Roche / @rocheb83

46

46

LES ASSOCIATIONS PORTEUSES DE DONNÉES

Benoît Roche / @rocheb83

47

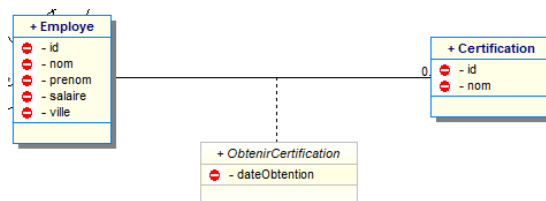
47

Les associations porteuses de données

Cas des associations porteuses de données

Exemple : Un employé obtient une certification à une date donnée

Représentation UML:



- ✓ Dans ce cas, on peut créer une Entité *ObtenirCertification*
- et
- ✓ y rajouter 2 relations ManyToOne vers les Entités *Employee* et *Certification*

Benoît Roche / @rocheb83

48

48

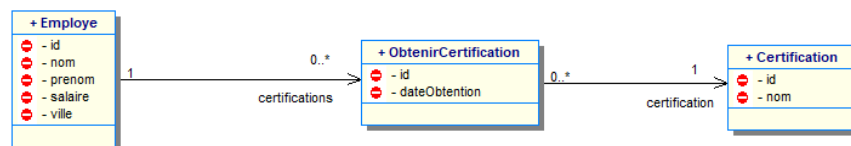


DOCTRINE : STRUCTURE DE LA BD

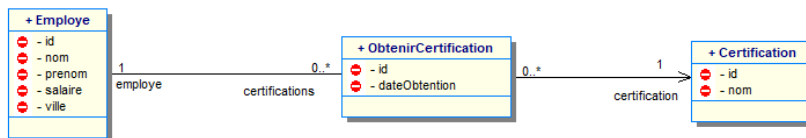


Les associations porteuses de données

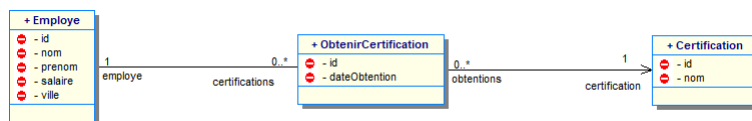
Nouvelle modélisation :



Ou :



OU



Benoît Roche / @rocheb83

49

49



DOCTRINE : STRUCTURE DE LA BD



Les associations porteuses de données

Dans ce cas, on a fait le choix de liens bidirectionnels.

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

    #[ORM\Column(type: Types::DATE_MUTABLE)]
    private ?\DateTimeInterface $dateObtention = null;

    #[ORM\ManyToOne(inversedBy: 'certifications')]
    #[ORM\JoinColumn(nullable: false)]
    private ?Employee $employe = null;

    #[ORM\ManyToOne(inversedBy: 'obtentions')]
    #[ORM\JoinColumn(nullable: false)]
    private ?Certification $certification = null;
}
```



On remarque que les objets \$employe et \$certification ne peuvent pas être null

Benoît Roche / @rocheb83

50

50



HYDRATATION DES OBJETS

Benoît Roche / @rocheb83

51

51



DOCTRINE : STRUCTURE DE LA BD

Hydratation des objets

Lors du chargement d'un objet (son hydratation), seuls les attributs de l'objet lui-même sont hydratés.



Les objets des entités liées par une relation ne sont pas hydratés

Par exemple dans l'Entity Employe il y a un attribut \$voiture de l'entity Voiture. Lors du chargement d'un objet employe, l'attribut \$voiture n'est pas hydraté.

C'est lorsque l'on fait appel à l'objet que Doctrine va faire une deuxième requête (*silently request*) à la BD pour retrouver et hydrater l'attribut voiture.

Par exemple l'instruction \$employe->getVoiture() déclenchera la requête sur la BD dont le résultat hydratera l'attribut \$voiture.



On dit que c'est un chargement paresseux (lazily loaded)

Benoît Roche / @rocheb83

52

52

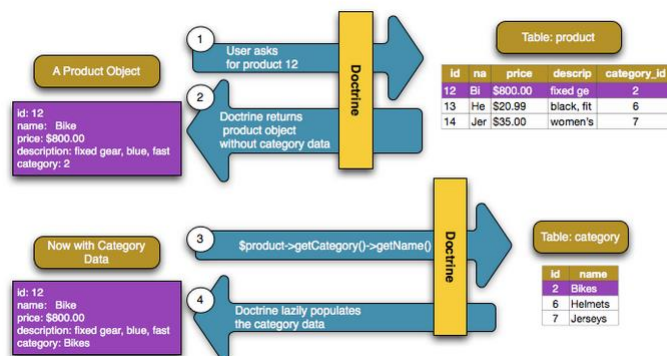


DOCTRINE : STRUCTURE DE LA BD



Hydratation des objets

Ce schéma de la documentation symfony illustre bien ce mode lazily loaded



<https://symfony.com/doc/5.4/doctrine/associations.html>

Benoît Roche / @rocheb83

53

53



**ATTRIBUER UN NOM AUX TABLES DE JOINTURE ET
AUX COLONNES DE JOINTURE**

Benoît Roche / @rocheb83

54

54



DOCTRINE : STRUCTURE DE LA BD

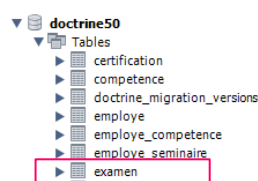
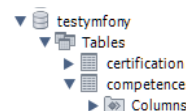


Attribuer un nom aux tables de jointure et aux colonnes de jointure

Donner un nom à une table

Par défaut la table portera le nom de l'entité mappée
On peut spécifier explicitement un nom à la table :

```
# [ORM\Table(name: 'examen')]
#[ORM\Entity(repositoryClass: ObtenirCertificationRepository::class)]
class ObtenirCertification
```



```
$this->addSql('CREATE TABLE examen (id INT AUTO_INCREMENT NOT NULL, employe_id INT NOT NULL,
$this->addSql('ALTER TABLE examen ADD CONSTRAINT FK_514C8FEC1B65292 FOREIGN KEY (employe_id)
$this->addSql('ALTER TABLE examen ADD CONSTRAINT FK_514C8FECCB47068A FOREIGN KEY (certificat
$this->addSql('ALTER TABLE obtenir_certification DROP FOREIGN KEY FK_7393B21E1B65292');
$this->addSql('ALTER TABLE obtenir_certification DROP FOREIGN KEY FK_7393B21ECB47068A');
$this->addSql('DROP TABLE obtenir_certification');
```



Il vaut mieux prévoir le nom ... avant la migration

Benoît Roche / @rocheb83

55

55



DOCTRINE : STRUCTURE DE LA BD

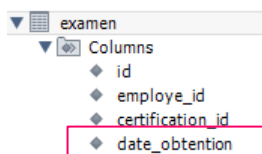


Attribuer un nom aux tables de jointure et aux colonnes de jointure

Donner un nom à une colonne

Par défaut la colonne portera le nom de la colonne mappée
On peut spécifier explicitement un nom à une colonne :

```
# [ORM\Column(name: 'dateobtention', type: Types::DATE_MUTABLE)]
private ?\DateTimeInterface $dateObtention = null;
```



Il vaut mieux prévoir le nom ... avant la migration

Benoît Roche / @rocheb83

56

56



DOCTRINE : STRUCTURE DE LA BD

Attribuer un nom aux tables de jointure et aux colonnes de jointure

En ce qui concerne les tables de jointures la relation bidirectionnelle suivante :



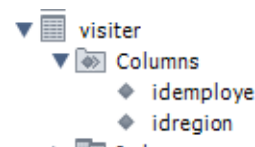
On peut enrichir les annotations en nommant :

La table de jointure

Les colonnes clé étrangères de la table de jointure

```

#[ORM\ManyToMany(targetEntity: Region::class, inversedBy: 'employes')]
#[ORM\JoinTable(name: 'visiter')]
#[ORM\JoinColumn(name: 'idemploye', referencedColumnName: 'id')]
#[ORM\InverseJoinColumn(name: 'idregion', referencedColumnName: 'id')]
private Collection $regions;
  
```



Benoît Roche / @rocheb83

57

57



MISES À JOUR EN CASCADE

Benoît Roche / @rocheb83

58

58



DOCTRINE : STRUCTURE DE LA BD



Mises à jour en cascade

Par défaut, Doctrine ne répercute pas en cascade les mises à jour faites dans l'application.

Ainsi,

- ✓ Si on crée un objet d'une entité mère qui contient lui-même un objet d'une entité fille, au moment de la mise à jour de la BD, seul l'enregistrement correspondant à l'entité mère sera créé Ce qui provoquera une erreur d'intégrité référentielle si l'enregistrement correspondant de l'entité fille n'existe pas.
- ✓ De même pour les suppressions. Si on supprime un objet d'une entité contenant un ou plusieurs objets d'une entité fille, il n'y aura pas de suppression en cascade dans la BD

Benoît Roche / @rocheb83

59

59

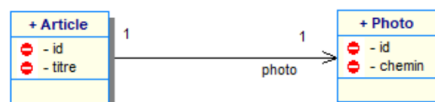


DOCTRINE : STRUCTURE DE LA BD



Mises à jour en cascade

Exemple



On ne peut créer l'article **QUE** si la photo existe

La photo ne peut pas exister **SANS** l'article

La solution :

```

#[ORM\OneToOne(cascade: ['persist'])]
#[ORM\JoinColumn(nullable: false)]
private ?Photo $photo = null;
  
```

Benoît Roche / @rocheb83

60

60



DOCTRINE : STRUCTURE DE LA BD



Mises à jour en cascade

```
# [ORM\OneToOne(cascade: ['persist'])]
# [ORM\JoinColumn(nullable: false)]
private ?Photo $photo = null;
```

En créant une migration, on ne voit aucune option SQL ...

```
$this->addSql('ALTER TABLE article ADD photo_id INT NOT NULL');
$this->addSql('ALTER TABLE article ADD CONSTRAINT FK_23A0E667E9E4C8C FOREIGN KEY (photo_id) REFERENCES photo (id)');
$this->addSql('CREATE UNIQUE INDEX UNIQ_23A0E667E9E4C8C ON article (photo_id)');
```

En fait c'est Doctrine qui va se charger de générer les ordres sql de persistance avec les données issues d'un formulaire

Benoît Roche / @rocheb83

61

61



DOCTRINE : STRUCTURE DE LA BD



Mises à jour en cascade

Doctrine qui va se charger de générer les ordres sql de persistance avec les données issues d'un formulaire

Nouvel Article

Titre: Mon bateau
Photo
Chemin: /photos/monbateau.jpg
Submit

```
public function creer(EntityManagerInterface $em, Request $request): Response{
    $article = new Article();
    $form = $this->createForm(ArticleType::class, $article);
    $form->handleRequest($request);
    if($form->isSubmitted() && $form->isValid()){
        $em->persist($article);
        $em->flush();
        return new Response('Création article OK');
    }
    return $this->render('test/creer.html.twig',
        ['formArticle' => $form->createView()]);
}
```

id	titre	photo_id
1	Mon bateau	1
* NULL	NULL	NULL

id	chemin
1	d:/photos/monbateau.jpg
* NULL	NULL



La base à été mise à jour

<http://doctrine50.sym/creer>

Benoît Roche / @rocheb83

62

62



DOCTRINE : STRUCTURE DE LA BD



Mises à jour en cascade

On peut essayer sans l'option cascade persist :

```
//#[ORM\OneToOne(cascade: ['persist', 'remove'])] ← Commentaire !!!
#[ORM\JoinColumn(nullable: false)]
private ?Photo $photo = null;
```

Nouvel Article

Titre Mer méditerranée
Photo
Chemin d:\photos\merMed.jpg

PDOException > Exception > ForeignKeyConstraintViolationException

HTTP/500 Internal Server Error

An exception occurred while executing a query: SQLSTATE[23000]: Integrity constraint violation: 1452 Cannot add or update a child row: a foreign key constraint fails ('doctrine50': 'article', CONSTRAINT 'FK_23A0E667E9E4C8C' FOREIGN KEY ('photo_id') REFERENCES 'photo' ('id'))



La base n'a pas été mise à jour

Benoît Roche / @rocheb83

63

63



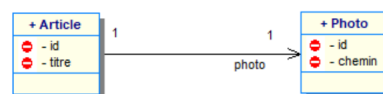
DOCTRINE : STRUCTURE DE LA BD



Mises à jour en cascade



Il en est de même pour la suppression en cascade :



```
#[ORM\OneToOne(cascade: ['persist', 'remove'])]
#[ORM\JoinColumn(nullable: false)]
private ?Photo $photo = null;
```

Doctrine qui va se charger de générer les ordres sql de persistance, ici il va gérer la suppression d'un article **ET** d'une photo

Benoît Roche / @rocheb83

64

64



DOCTRINE : STRUCTURE DE LA BD



Mises à jour en cascade



Il en est de même pour la suppression en cascade :



Doctrine qui va se charger de générer les ordres sql de persistance, ici il va gérer la suppression d'un article ET d'une photo

```
#[Route('/supprimer/{id}', name: 'supprimer')]
public function supprimer(EntityManagerInterface $em, Article $article): Response{
    $em->remove($article);
    $em->flush();
    return new Response('article supprimé');
}
```

<http://doctrine50.sym/supprimer/1>



La base à été mise à jour

id	titre	photo_id
1		

id	chemin
1	

Benoît Roche / @rocheb83

65

65



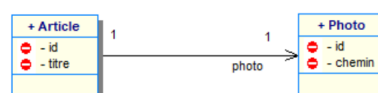
DOCTRINE : STRUCTURE DE LA BD



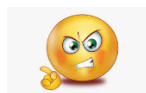
Mises à jour en cascade



Si on n'avait pas l'option *cascade remove* :



```
//#[ORM\OneToOne(cascade: ['persist', 'remove'])]
#[ORM\JoinColumn(nullable: false)]
private ?Photo $photo = null;
```



L'article a bien été supprimé
La photo n'a pas été supprimée !!!

id	titre	photo_id
1		

id	chemin
1	d:\photos\monbateau.jpg

<http://doctrine50.sym/supprimer/1>

Benoît Roche / @rocheb83

66

66



DOCTRINE : STRUCTURE DE LA BD



Mises à jour en cascade

Dans le cas de mise à jour en cascade de collections d'objets :

On peut laisser l'application gérer la mise à jour en base de données en cascade avec l'option **orphanRemoval**.

Par exemple :



La composition indique que si on supprime un bâtiment, on doit supprimer en cascade toutes les salles de ce bâtiment.

Or l'attribut lesSalles de la classe Batiment est une collection d'objets, donc la mise à jour en cascade par l'application ne peut se faire qu'avec l'option **orphanRemoval**

Benoît Roche / @rocheb83

67

67



DOCTRINE : STRUCTURE DE LA BD



Mises à jour en cascade



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

```
#[ORM\ManyToOne(inversedBy: 'salles')]
#[ORM\JoinColumn(nullable: false)]
private ?Batiment $batiment = null;
```

Entity Salle

```
#[ORM\OneToMany(mappedBy: 'batiment', targetEntity: Salle::class, orphanRemoval: true)]
private Collection $salles;
```

Entity Batiment

Benoît Roche / @rocheb83

68

68



DOCTRINE : STRUCTURE DE LA BD



Mises à jour en cascade

On peut tester :

	id	nom
▶	1	Batiment 1
	2	Batiment 2

	id	nom	capacite	batiment_id
▶	1	salle10	100	1
	2	salle11	130	1
	3	salle12	60	1
	4	salle13	100	1
	5	salle20	100	2
	6	salle21	190	2
*	NULL	NULL	NULL	NULL

<http://doctrine50.sym/supprimer/batiment/1>

```
#[Route('/supprimer/batiment/{id}', name: 'supprimer_batiment')]
public function supprimerBatiment(EntityManagerInterface $em, Batiment $batiment): Response{
    $em->remove($batiment);
    $em->flush();
    return new Response('Batiment supprimé');
}
```



Le Bâtiment ET LES Salles ont été supprimées

	id	nom
▶	2	Batiment 2
*	NULL	NULL

	id	nom	capacite	batiment_id
▶	5	salle20	100	2
	6	salle21	190	2

Benoît Roche / @rocheb83

69

69



DOCTRINE : STRUCTURE DE LA BD



Mises à jour en cascade

On peut voir les ordres générés du fichier migration:

```
$this->addSql('ALTER TABLE salle ADD CONSTRAINT FK_4E977E5CD6F6891B FOREIGN KEY (batiment_id) REFERENCES batiment (id)');
```

On ne voit pas d'ordre la clause on delete cascade sur la clé étrangère, cela signifie bien que c'est l'application qui se charge :

- En premier de supprimer les enregistrements enfants (ici els salles)
- En deuxième l'enregistrement parent (ici le bâtiment)



Il sera intéressant parfois de prévoir l'option on delete cascade dans la relation pour assure la mise à jour en cascade dans tous les cas dans la BD.
(Voir diapos suivantes).

Benoît Roche / @rocheb83

70

70

MISES À JOUR EN CASCADE PAR LA BASE ET NON PAR L'APPLICATION (ENTITÉS)

Benoît Roche / @rocheb83

71

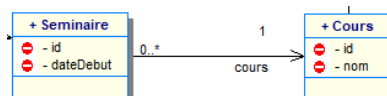
71

Mises à jour en cascade

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 `onDelete cascade` sur la FK `idCours` de la table `seminaire` :

```
#[ORM\ManyToOne]
#[ORM\JoinColumn(name:'idcours',onDelete:"CASCADE",nullable: false)]

$this->addSql('ALTER TABLE seminaire ADD CONSTRAINT FK_B0F911902BCFE085 FOREIGN KEY (idcours) '
    . 'REFERENCES cours (id) ON DELETE CASCADE');
```

Benoît Roche / @rocheb83



La gestion des suppressions se fait ici via les ordres SQL

72

72

DOCTRINE : STRUCTURE DE LA BD

Les relations bidirectionnelles

<http://doctrine50.sym/supprimer/cours/1>

Les suppressions en cascade :

Application :

Table cours

id	nom
1	Cours 1
2	Cours 2

id	date_debut	idcours
1	2024-09-12	1
2	2024-09-24	1
3	2024-10-15	1
4	2024-11-23	1
5	2024-09-12	2
6	2024-10-14	2

Table seminaire

En exécutant ce code, on a :

```
#[Route('/supprimer/cours/{id}', name: 'supprimer_batiment')]
public function supprimerCours(EntityManagerInterface $em, Cours $cours): Response{
    $em->remove($cours);
    $em->flush();
    return new Response('Cours supprimé');
}
```

Après la suppression

id	nom
2	Cours 2

id	date_debut	idcours
5	2024-09-12	2
6	2024-10-14	2

Benoît Roche / @rocheb83



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

73

73

DOCTRINE : STRUCTURE DE LA BD



Exercice:

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

[Voir : exercices Doctrine](#)

Benoît Roche / @rocheb83

74

74



Fin du cours, merci

Question/Réponses

Benoît Roche

@rocheb83

Benoît Roche / @rocheb83

75

75