Développement web côté serveur avec Symfony

Module 07 - Relations entre entités

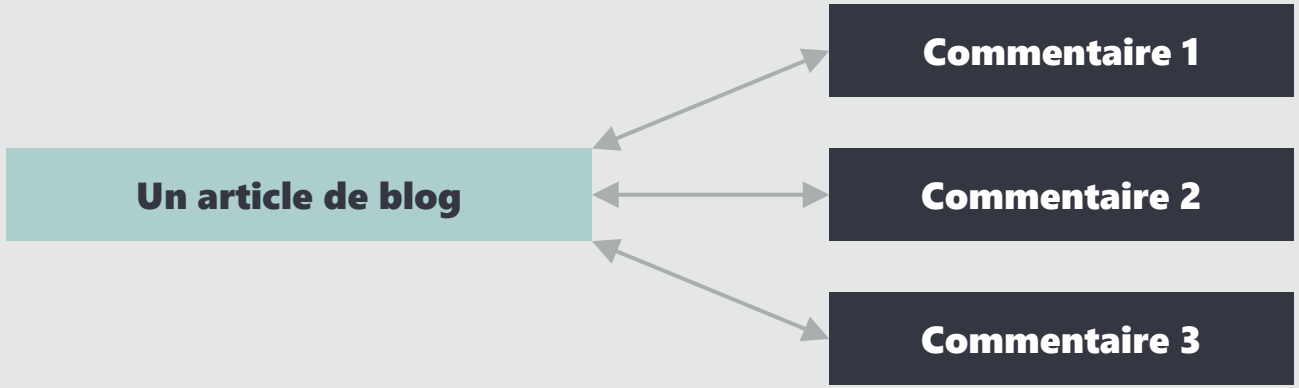
Objectifs

- Comprendre le mécanisme des relations entre entités Doctrine
- Savoir définir des relations entre entités
- Savoir utiliser des entités associées

Relations entre entités

La relation OneToMany / ManyToOne

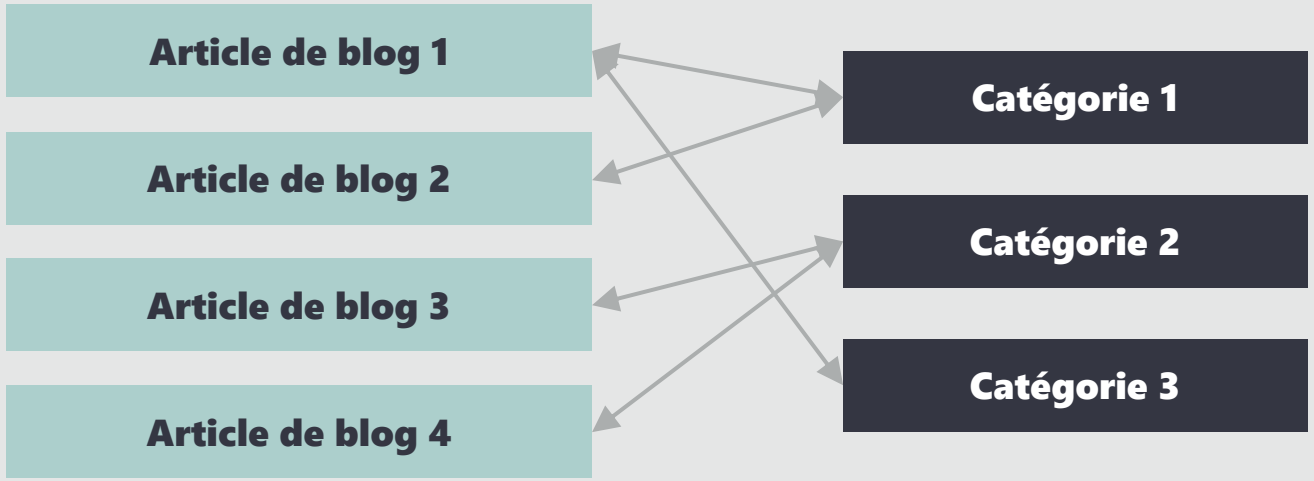
- Un objet peut être associé à plusieurs autres objets
- Relation `OneToMany/ManyToOne`



Relations entre entités

La relation ManyToMany

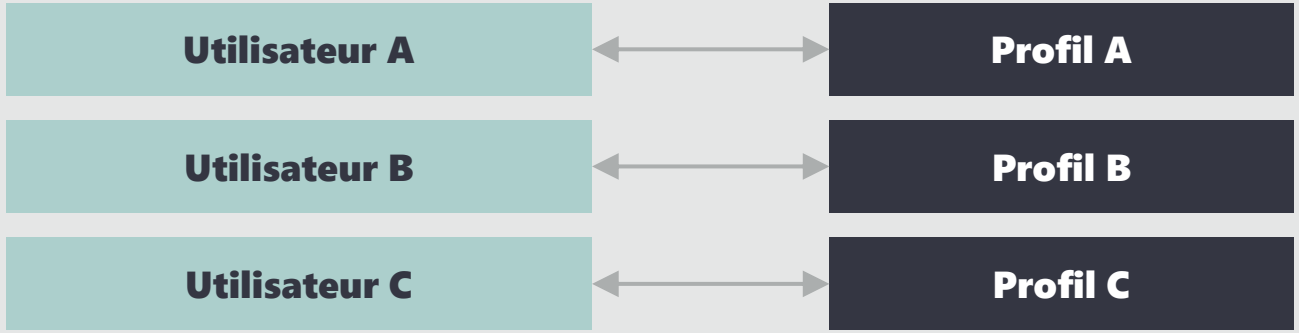
- Plusieurs objets peuvent être associés à plusieurs autres objets
- Relation `ManyToMany`



Relations entre entités

La relation OneToOne

- Un objet peut être associé à un seul autre objet
- Relation `OneToOne`



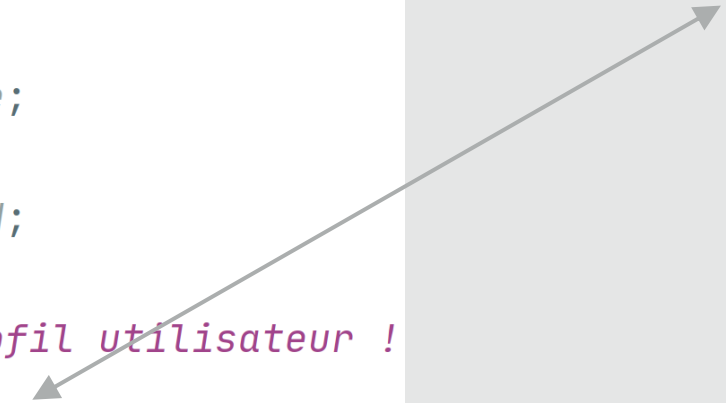
Relations entre entités

Les relations entre objets PHP

- Relation OneToOne :

```php
class User
{
    private $username;
    private $email;
    private $password;

    //contient le profil utilisateur !
    private $profile;
}
```



```php
class Profile
{
    private $firstName;
    private $lastName;
    private $biography;
    private $picture;
}
```

Relations entre entités

Les relations entre objets PHP

- Relation OneToMany :

```php
class Post
{
    private $title;
    private $content;
    private $author;

    //contient les commentaires de cet article !
    private $comments;
}
```

Relations entre entités

Les relations entre objets PHP

- Relation ManyToMany :

```php
class Post
{
    private $title;
    private $content;
    private $author;

    //contient les catégories de cet article !
    private $categories;
}
```

Relations entre entités

Les relations avec Doctrine

- Travailler avec les objets
- Jamais avec les id !
- Les étapes :
    1. Créer d'abord les 2 entités à associer (sans relation) avec `make:entity`
    2. Ajouter maintenant les relations, encore avec `make:entity`
    3. Suivre le wizard
    4. Mettre à jour la base de données avec `doctrine:schema:update --force`

Relations entre entités

Le wizard de relation

```
λ php bin/console make:entity Post

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

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

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

 What class should this entity be related to?:
 > Comment
Comment

What type of relationship is this?
 ------------ ---------------------------------------------------------------------
  Type         Description
 ------------ ---------------------------------------------------------------------
  ManyToOne    Each Post relates to (has) one Comment.
               Each Comment can relate to (can have) many Post objects

  OneToMany    Each Post can relate to (can have) many Comment objects.
               Each Comment relates to (has) one Post

  ManyToMany   Each Post can relate to (can have) many Comment objects.
               Each Comment can also relate to (can also have) many Post objects

  OneToOne     Each Post relates to (has) exactly one Comment.
               Each Comment also relates to (has) exactly one Post.
 ------------ ---------------------------------------------------------------------

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

```
 A new property will also be added to the Comment class so that you can access
and set the related Post object from it.

 New field name inside Comment [post]:
 >

 Is the Comment.post property allowed to be null (nullable)? (yes/no) [yes]:
 > no

 Do you want to activate orphanRemoval on your relationship?
 A Comment is "orphaned" when it is removed from its related Post.
 e.g. $post->removeComment($comment)

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

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

 updated: src/Entity/Post.php
 updated: src/Entity/Comment.php
```

Relations entre entités

Code généré dans Comment

```php
/**
 * @ORM\ManyToOne(targetEntity=Post::class, inversedBy="comments")
 * @ORM\JoinColumn(nullable=false)
 */
private $post;
```

```php
public function getPost(): ?Post
{
    return $this->post;
}

public function setPost(?Post $post): self
{
    $this->post = $post;

    return $this;
}
```

Relations entre entités

Code généré dans Post

```php
/**
 * @ORM\OneToMany(targetEntity=Comment::class, mappedBy="post", orphanRemoval=true)
 */
private $comments;

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

```php
/**
 * @return Collection|Comment[]
 */
public function getComments(): Collection
{
    return $this->comments;
}

public function addComment(Comment $comment): self
{
    if (!$this->comments->contains($comment)) {
        $this->comments[] = $comment;
        $comment->setPost($this);
    }

    return $this;
}

public function removeComment(Comment $comment): self
{
    if ($this->comments->removeElement($comment)) {
        // set the owning side to null (unless already changed)
        if ($comment->getPost() === $this) {
            $comment->setPost(null);
        }
    }

    return $this;
}
```

Relations entre entités

Les relations bidirectionnelles

- Les deux entités sont "au courant" de la relation
- Un seul côté est propriétaire

```php
//Comment.php

/**
 * @ORM\ManyToOne(targetEntity=Post::class, inversedBy="comments")
 * @ORM\JoinColumn(nullable=false)
 */
private $post;
```

```php
//Post.php

/**
 * @ORM\OneToMany(targetEntity=Comment::class, mappedBy="post", orphanRemoval=true)
 */
private $comments;
```

Relations entre entités

Doctrine et les types de relations

- @ORM\ManyToOne et @ORM\OneToMany
  - la clé étrangère s'ajoute dans la table côté Many
- @ORM\OneToOne()
  - on choisit où s'ajoute la clé étrangère
- @ORM\ManyToMany()
  - Doctrine crée la table de pivot !
  - Le choix de nom de table est personnalisable
  - Impossibilité d'ajouter des colonnes supplémentaires

Démonstration

Relations entre entités

Récupérer une entité associée à une autre

- Doctrine se charge des requêtes à notre place
- *Lazy loading*
- Attention au nombre de requêtes
- Jointure "manuelle"
- Paginator de Doctrine

Démonstration

Relations entre entités

Sauvegarder une entité associée à une autre

- Ne pas se préoccuper des identifiants !

```php
public function savePostWithComment(EntityManagerInterface $entityManager)
{
    $post = new Post();
    $post->setTitle('Symfony rocks!');

    $comment = new Comment();
    $comment->setContent('Excellent article !');
    $comment->setPost($post); // associe ce commentaire à $post !

    $entityManager->persist($post);
    $entityManager->persist($comment);
    $entityManager->flush();

    return $this->redirectToRoute('main_home');
}
```

Relations entre entités

Automatiser la sauvegarde des entités associées

- Avec les opérations de cascade de Doctrine :

```php
//Comment.php

/**
 * @ORM\ManyToOne(targetEntity=Post::class, inversedBy="comments", cascade={"persist"})
 * @ORM\JoinColumn(nullable=false)
 */
private $post;
```

Relations entre entités

Supprimer une entité associée à une autre

- On doit d'abord supprimer les entités propriétaires !

```php
public function removePost(EntityManagerInterface $entityManager, PostRepository $postRepository)
{
    $post = $postRepository->find(5);

    $entityManager->remove($post);
    $entityManager->flush(); // ne marche pas si ce Post a des commentaires associés !

    //...
}
```

Relations entre entités

Automatiser la suppression des entités associées

- Avec les opérations de cascade de Doctrine :

```php
//Post.php

/**
 * @ORM\OneToMany(targetEntity=Comment::class, mappedBy="post", orphanRemoval=true, cascade={"remove"})
 */
private $comments;
```

Démonstration

Conclusion

- Vous savez définir des relations entre entités valides
- Vous savez accéder à des données associées à une entité
- Vous savez sauvegarder et supprimer des entités associées