SYMFONY : 8-Les entités et leur relations

Dans ce chapitre, nous allons comprendre comment relier nos articles à des catégories et nous allons aussi permettre de pouvoir ajouter des commentaires pour chaque article. Dans cette partie, nous allons beaucoup nous servir du terminal et passer un certain nombre de lignes de commandes pour simplifier le travail.

Le but va être de créer une nouvelle entité (une nouvelle table) catégorie afin de relier les articles à des catégories. Nous allons donc créer une nouvelle entité.

Lancer dans le terminal :

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

```
Class name of the entity to create or update (e.g. GrumpyGnome):
 > Category
```

Créons un premier champ « title »

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

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

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

 Field length [255]:
 >
 Can this field be null in the database (nullable) (yes/no) [no]:
 >
```

Créons ensuite un champ « description »

```
 Can this field be null in the database (nullable) (yes/no) [no]:
 >

 updated: src/Entity/Category.php

 Add another property? Enter the property name (or press <return> to stop adding fields):
 > description

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

 Can this field be null in the database (nullable) (yes/no) [no]:
 > yes
```

Notre table est maintenant demander à Symfony de créer une relation entre notre table « article » et la table « category »

Nous allons maintenant créer une propriétés (une variables) qui contiendra un ensembles d'articles (un peu à la manière d'un ARRAY) :

```
Add another property? Enter the property name (or press <return> to stop adding fields):
 > articles
```

Taper ensuite un point d'interrogation pour afficher toute les options et notamment les relations :

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

Entrer ensuite « relation »

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

Maintenant à quelle class devrait être relié cette entité ? Et bien aux articles ! Les articles doivent être reliés aux catégories.

```
 What class should this entity be related to?:
 > Article
```

Nous allons maintenant choisir « OneToMany », en effet, un article peut être relié à une catégorie et une catégorie peut posséder plusieurs articles.

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

Il nous demande maintenant d'ajouter une propriété pour les articles, et bien oui chaque articles vont être relié à une catégorie, nous allons garder comme nom « category » donc taper sur « enter »

```
 New field name inside Article [category]:
 >
```

Maintenant le terminal nous demande, est ce qu'une catégorie peut être nulle ? En gros peut-il exister des articles qui n'ont pas de catégorie ? Dans notre cas non.

```
 Is the Article.category property allowed to be null (nullable)? (yes/no) [yes]:
 > no
```

Maintenant le terminal nous demande, si un article perd sa catégorie, devons-nous le supprimer ? Dans notre cas non, taper sur « enter »

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

Nous avons maintenant terminé la création de la table, il reste la migration à faire. Taper sur « enter »

```
Add another property? Enter the property name (or press <return> to stop adding fields):
 >
```

Observer les modifications dans les fichiers suivants :

```
updated: src/Entity/Category.php
updated: src/Entity/Article.php
```

Lancer maintenant la migration :

```
php bin/console make:migration
```

Observer le fichier src->migration->Version.php

On observe le code SQL pour la création de la table « category » ainsi que la modification de la table « article » avec l'ajout d'une clé étrangère category_id + des contraintes d'intégrités.

Attention, pour éviter de casser la migration , effectivement les fausses données (fixtures) dans la table « article » ne pourront être relié à une catégorie donc supprimé toute le contenu de la table « article »

```
ALTER TABLE article ADD category_id INT NOT NULL
```

Lancer maintenant la migration :

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

Confirmer en tapant « y » pour YES, Symfony nous demande si nous souhaitons vraiment migrer.

Nous allons maintenant créer la table commentaire, nous allons créer une nouvelle entité !! C'est partie !!

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

```
Class name of the entity to create or update (e.g. VictoriousKangaroo):
 > Comment
```

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

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

 Field length [255]:
 >
```

```
 Can this field be null in the database (nullable) (yes/no) [no]:
 >
```

```
 Add another property? Enter the property name (or press <return> to stop adding fields):
 > Content

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

```
 Can this field be null in the database (nullable) (yes/no) [no]:
 >
```

```
 Add another property? Enter the property name (or press <return> to stop adding fields):
 > createdAt

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

 Can this field be null in the database (nullable) (yes/no) [no]:
 >
```

Il faut que nos commentaires soient reliés à des articles, nous allons donc créer un champ de relations entre les articles et les commentaires, donc une clé étrangère dans notre tables commentaire

```
Add another property? Enter the property name (or press <return> to stop adding fields):
 > article
```

Taper un point d'interrogation pour voir toute les relations possible

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

```
What class should this entity be related to?:
 > Article
```

Nous allons choisir ManyToOne, en effet chaque commentaire sera relié à un article et des articles pourront avoir plusieurs commentaires

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

Est-ce que la propriété article dans les commentaires peut être nul ? En gros est ce qu'on pourrait avoir un commentaire relié à aucun article ? Et bien non…

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

Maintenant est ce que tu veux qu'on ajoute une propriété dans la classe Article qui fasse référence à nos commentaires ? Par exemple une propriété comments au pluriel comme cela quand on se balade sur un article, on peut accéder facilement aux commentaires de cet article, et bien moi je dis oui !!

```
 Do you want to add a new property to Article so that you can access/update Comment object
s from it - e.g. $article->getComments()? (yes/no) [yes]:
 > yes
```

Maintenant il me demande comment je veux appeler cette propriété au sein de la classe Article, et bien comments au pluriel

```
  New field name inside Article [comments]:
 >
```

Maintenant est ce que l'on souhaite supprimer des commentaires qui serait orphelin, c'est-à-dire qu'ils n'appartiennent à aucun articles, et bien oui !!

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

Nous avons maintenant terminé et nous allons validé et migré, taper sur entrer

```
 Add another property? Enter the property name (or press <return> to stop adding fields):
 >
```

Se rendre dans le fichier comment.php dans le dossier Entity et observer la relation entre les tables.

Lancer maintenant la migration :

```
php bin/console make:migration
```

Un fichier de migration a été créé. Se rendre dans le dossier migration pour voir les requêtes SQL

Lancer maintenant la commande suivante pour exécuter le code SQL

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

```
WARNING! You are about to execute a database migration that could result in schema changes
 and data loss. Are you sure you
wish to continue? (y/n)y
```

Nous allons maintenant créer des fausses données (fixtures)

Nous allons maintenant parler de la LIBRAIRIE FAKER (Merveille)

C'est une librairie qui va nous permettre de créer des fausses données un peu plus convaincante que les fausses données que nous avons créées auparavant , nous aurons des phrases, du lorem ipsum, ce n'est pas top mais si nous voulons des prénoms, nous auront des prénoms, des noms de familles, des pays, des villes etc… Nous allons donc utiliser FAKER

Rendez-vous à l'adresse suivante : https://github.com/fzaninotto/Faker

Lancer la commande suivante :

```
composer require fzaninotto/faker –dev
```

Rendez-vous dans le fichier ArticlesFixtures.php dans le dossier DataFixtures

```
$faker = \Faker\Factory::create('FR_fr');
```

On execute la méthode static create() de la classe Factory et on envoie l'argument « FR_fr » pour avoir des fausses données en français

Appeler les classes « Category » et « Comment »

```
use App\Entity\Category;
use App\Entity\Comment;
```

```php
// Créer 3 catégories fakées
        for($i = 1; $i <= 3; $i++)
        {
            $category = new Category;
            $category->setTitle($faker->sentence())
                     ->setDescription($faker->paragraph());

            $manager->persist($category);

            // créer entre 4 et 6 articles
            for($j = 1; $j <= mt_rand(4,6); $j++)
            {
                $article = new Article(); // on instancie la class Article() qui se trouve
 dans le dossier App\Entity

                $content = '<p>' . join($faker->paragraphs(5), '</p><p>') . '</p>';

                // Nous pouvons maintenant faire appel au setteur pour créer des articles
                $article->setTitle($faker->sentence())
                        ->setContent($content)
                        ->setImage($faker->imageUrl())
                        ->setCreatedAt($faker->dateTimeBetween('-
6 months')) // on instancie la classe DateTime() pour formater l'heure
                        ->setCategory($category);

                $manager-
>persist($article); // permet de faire persister l'article dans le temps

                //On donne des commentaires à l'article
                for($k = 1; $k <= mt_rand(4,10); $k++)
                {
                    $comment = new Comment();

                    $content = '<p>' . join($faker->paragraphs(2), '</p><p>') . '</p>';

                    // $now = new \DateTime();
                    // $interval = $now->diff($article->getCreatedAt());
                    // $days = $interval->days;
                    // $minimum = '-' . $days . ' days'; // moins -100 days

                    $days = (new \DateTime())->diff($article->getCreatedAt())->days;


                    $comment->setAuthor($faker->name)
                            ->setContent($content)
                            ->setCreatedAt($faker->dateTimeBetween('-' . $days . ' days'))
                            ->setArticle($article);

                    $manager->persist($comment);

                }
            }
```

Pour voir le résultat, lancer dans le terminal la commande suivante :

```
php bin/console doctrine:fixtures:load
```

```
 Careful, database "blog" will be purged. Do you want to continue? (yes/no) [no]:
 > yes
```

Allez voir la BDD les fausses données sont maintenant crées.

Rendez vous dans le fichier show.html.twig et modifier la ligne suivante :

```
<div class="metadata">Ecrit le {{ article.createdAt | date('d/m/Y') }} à {{ article.create
dAt | date('H:i:s') }} dans la catégorie {{ article.category.title }}</div><hr>
```

Cela permet d'afficher la catégorie dynamiquement

```
<h2 class="text-center mx-auto m-3">Commentaires</h2>

        {# affichage des faux commentaires #}
        <section id="commentaire" class="col-md-10 mx-auto">
            {% for comment in article.comments %}
                <div class="comment">
                    <div class="row">
                        <div class="col-md-3">
                            {{ comment.author }} (<small>{{ comment.createdAt | date('d/m/
Y à H:i:s') }}</small>)
                        </div>
                        <div class="col-md-9">
                            {{ comment.content | raw }}
                        </div>
                    </div>
                </div><hr>
            {% endfor %}
        </section>
```

Il faut maintenant ajouter le champs category au formulaire d'ajout et de modification. Rendez vous dans le dossier Form puis dans le fichier ArticleType.php

Si nous modifions comme cela nous allons faire planter symfony

```
 $builder
            ->add('title')
            ->add('category')
            ->add('content')
            ->add('image')
        ;
```

En effet symfony ne fait pas automatiquement la relation entre la BDD et notre champ category

Ajouter les classes suivante :

```
use Symfony\Bridge\Doctrine\Form\Type\EntityType;
use App\Entity\Category;
```

```
$builder
            ->add('title')
            ->add('category', EntityType::class, [
                'class' => Category::class,
                'choice_label' => 'title'
            ])
            ->add('content')
            ->add('image')
        ;
```

Aller ensuite dans create.html.twig pour faire apparaitre le champ, ajouter cette ligne :

```
{{ form_row(formArticle.category) }}
```