Les relations entre entités

Maintenant que les entités sont créées, il nous faut créer les relations. Lorsque les relations seront créées, Doctrine se charge de réaliser le mapping en base de données en créant les champs supplémentaires étant clés étrangères, les nouvelles tables si nécessaire.

Il existe principalement 3 types de relations entre entités :

- la relation many-to-one/one-to-many
- la relation many-to-many
- l'héritage

La relation Many-To-One

Dans le contexte webstudent, il s'agit par exemple de la relation entre l'entité Etudiant et l'entité Maison.

Un étudiant appartient à une et une seule maison (Many-To-One). Cela signifie qu'une propriété supplémentaire de type Maison doit être ajoutée à l'entité Etudiant (avec son getter et son setter). En base de données, la clé étrangère id_maison sera ajoutée dans la table Etudiant et fera référence au champ id de la table Maison.

Une maison comprend plusieurs étudiants (One-To-Many). Cette relation n'est pas toujours nécessairement à implémenter. Il faut se poser la question, si au niveau applicatif, nous aurons besoin, partant d'une maison, de récupérer, lister, consulter des étudiants. Dans notre cas, oui, donc nous implémenterons aussi cette relation. Dans l'entité Maison, une propriété supplémentaire, liste d'étudiants devra être ajoutée. En base de donnée, cette relation “inverse” ne change rien, le lien étant assuré par la clé étrangère créée lors de la relation Many-To-One.

La relation Many-To-One doit donc être implémentée dans tous les cas mais pas forcément la relation inverse One-to-Many.

Pour assurer le mapping avec la base de données, des annotations seront donc ajoutées au dessus de chaque propriété assurant la relation.

Création des relations avec la console :

La version 4 de Symfony permet maintenant de créer ces annotations grâce à la console. Il s'agit de la même commande que le création d'une entité.

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

Liste des informations à fournir :

- Class name of the entity to create or update : Etudiant

- New property name : maison (avec un m minuscule, pour respecter CamelCase, puisqu'il s'agit d'une propriété !)
- Field type : relation
- What class should this entity be related to? : Maison (avec un M majuscule puisqu'il s'agit du nom de l'entité)

- What type of relationship is this?: ManyToOne
- Is the Etudiant.maison property allowed to be null (nullable)?: yes
- Do you want to add a new property to Maison so that you can access/update Etudiant objects from it - e.g. $maison→getEtudiants()? yes (Cette dernière instruction permet de créer la relation inverse OneToMany dans l'entité Maison.)
- New field name inside Maison : etudiants (avec un e minuscule car c'est une propriété et un s car c'est c'est une liste)

Résultat :

- Liste à puceDans l'entité Etudiant, on retrouve la propriété supplémentaire maison annotée de la relation ManyToOne :

```
/**
     * @ORM\ManyToOne(targetEntity="App\Entity\Maison",
inversedBy="etudiants")
     */
    private $maison;
```

. Le getter et le setter de cette propriété ont été également implémentés :

```
public function getMaison(): ?Maison
    {
        return $this->maison;
    }

    public function setMaison(?Maison $maison): self
    {
        $this->maison = $maison;

        return $this;
    }
```

- Dans l'entité Maison, une propriété supplémetaire a été ajoutée :

```
 /**
     * @ORM\OneToMany(targetEntity="App\Entity\Etudiant", mappedBy="maison")
     */
    private $etudiants;
```

Avec les getters et les setters et les méthodes d'ajout/suppression de liste

```
 /**
     * @return Collection|Etudiant[]
```

```
     */
    public function getEtudiants(): Collection
    {
        return $this->etudiants;
    }

    public function addEtudiant(Etudiant $etudiant): self
    {
        if (!$this->etudiants->contains($etudiant)) {
            $this->etudiants[] = $etudiant;
            $etudiant->setMaison($this);
        }

        return $this;
    }

    public function removeEtudiant(Etudiant $etudiant): self
    {
        if ($this->etudiants->contains($etudiant)) {
            $this->etudiants->removeElement($etudiant);
            // set the owning side to null (unless already changed)
            if ($etudiant->getMaison() === $this) {
                $etudiant->setMaison(null);
            }
        }

        return $this;
    }
```

Réaliser le mapping avec le base de données :

Exécuter les instructions de mise à jour de la base de données

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

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

Un nouveau champ nommé maison_id a été créé. Il s'agit en plus d'une clé étrangère faisant référence au champ id de la table Maison.

Tests de la relation ManyToOne :

Pour tester la relation ManyToOne, nous allons récupérer une maison et l'ajouter à un étudiant. Nous allons créer la nouvelle route permettant de modifier un étudiant : Dans le fichier routes.yaml :

```
etudiantModifier:
    path: /etudiant/modifier/{id}
    controller: App\Controller\EtudiantController::modifierEtudiant
```

Dans le fichier EtudiantController, nous ajoutons la nouvelle méthode :

```
public function modifierEtudiant(ManagerRegistry $doctrine, $id){

        //récupération de l'étudiant dont l'id est passé en paramètre
        $etudiant= $doctrine->getRepository(Etudiant::class)->find($id);

        if (!$etudiant) {
            throw $this->createNotFoundException(
            'Aucun etudiant trouvé avec le numéro '.$id
            );
        }
        else
        {


        // récupération de la maison des griffondor à partir du code de la
maison
                $maison=
$doctrine->getRepository(Maison::class)->findOneBy(['code' => 'SPT']);


        if (!$maison) {
            throw $this->createNotFoundException(
            'Aucune maison trouvé avec ce nom'
            );
        }
        else
        {

        //Affectation de la maison à l'étudiant
        $etudiant->setMaison($maison);

        // persistence de l'objet modifié
                $entityManager = $this->getDoctrine()->getManager();
        $entityManager->persist($etudiant);
        $entityManager->flush();



        //return new Response('Etudiant : '.$etudiant->getNom());
        return $this->render('etudiant/consulter.html.twig', [
            'etudiant' => $etudiant,]);
        }
        }
    }
```

http://localhost/webstudent/public/etudiant/modifier/4

Tests de la relation OneToMany:

Pour tester la relation OneToMany, nous allons récupérer une maison à partir du code qui sera passé en paramètre de l'url et lister tous les étudiants de cette maison.

Ajout de la route :

```
maisonConsulter:
    path: /maison/consulter/{code}
    controller: App\Controller\MaisonController::consulterMaison
```

Ajout du contrôleur et de la méthode consulterMaison :

```
class MaisonController extends AbstractController
{
    /*
     * @Route("/maison", name="maison")
     */

    public function consulterMaison(ManagerRegistry $doctrine, $code){
        $repository = $this->getDoctrine()->getRepository(Maison::class);
        $maison= $doctrine->getRepository(Maison::class)->findOneBy(['code'
=> 'SPT']);

        return $this->render('maison/consulter.html.twig', ['pMaison' =>
$maison,]);
    }
}
```

Création de la vue dans un nouveau dossier maison

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Maison-consulter</title>
    </head>
    <body>

        <h5>BIENVENUE DANS LA MAISON DES {{ pMaison.nom | upper}}</h5>
 <p>
  <table >
<tr><td>id</td><td>Nom</td><td>Prénom</td><td>datenaissance</td><td>ville</t
d></tr>
    {% for e in pMaison.etudiants %}
    <tr>
            <td><a href="{{ path('etudiantConsulter', { 'id': e.id }) }}">{{
e.id }}</a></td>
            <td><a href="{{ path('etudiantConsulter', { 'id': e.id }) }}">{{
e.nom }}</a></td>
```

```
            <td><a href="{{ path('etudiantConsulter', { 'id': e.id }) }}">{{
e.prenom }}</a></td>
            <td>{{ e.dateNaiss|date('d/m/Y') }}</td>
            <td>{{ e.ville}}</td>
            </tr>
       {% endfor %}

 </table
    </body>
</html>
```

http://localhost/webstudent/public/maison/consulter/GFD

Relation Many-To-Many

Dans le contexte webstudent, il s'agit de la relation entre Competence et Professeur.

Création de la relation avec la console :

Avec la commande permettant de générer les entités, nous pouvons créer la relation en renseignant les informations ci-dessous :

- Class name of the entity to create or update : Competence
- New property name : professeurs (avec un s car c'est une collection)
- Field type : relation
- What class should this entity be related to?: Professeur
- Relation type? : ManyToMany
- Do you want to add a new property to Professeur so that you can access/update Competence objects from it - e.g. $professeur→getCompetences()? yes (permet de créer la relation inverse, c'est-à-dire dans Professeur, créer une collection de professeurs.

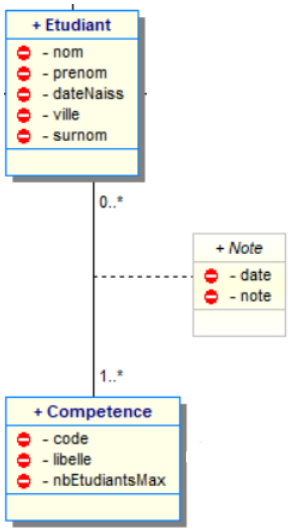
Dans l'entité Compétence, une propriété de type collection de professeurs a été ajoutée. Dans l'entité Professeur, une propriété de type collection de compétences a été ajoutée.

mapping des relations en base de données :

Après avoir exécuté les commandes de mapping, une nouvelle table a été créée en base de données nommée `competence_professeur`. Cette table comporte deux champs composant la clé primaire : competence_id + professeur_id. Chaque champ est aussi clé étrangère en référence respectivement à la table compétence et professeur.

Relation de type Classe-association

Dans le contexte webstudent, ce type de relation est représenté par les entités et relation entre Etudiant et Competence.



Il faut transformer les relations manyToMany en relations ManyToOne comme ci-dessous :

Il faut donc créer une nouvelle entité, Note avec les deux propriétés primitives dateNote et note puis créer les deux relations ManyToOne vers Etudiant et vers Competence.

Après exécution des commandes de mapping de base de données, la table Note contiendra id en clé primaire ainsi que deux clés étrangères etudiant_id et competences_id faisant référence respectivement à Etudiant et Competence.