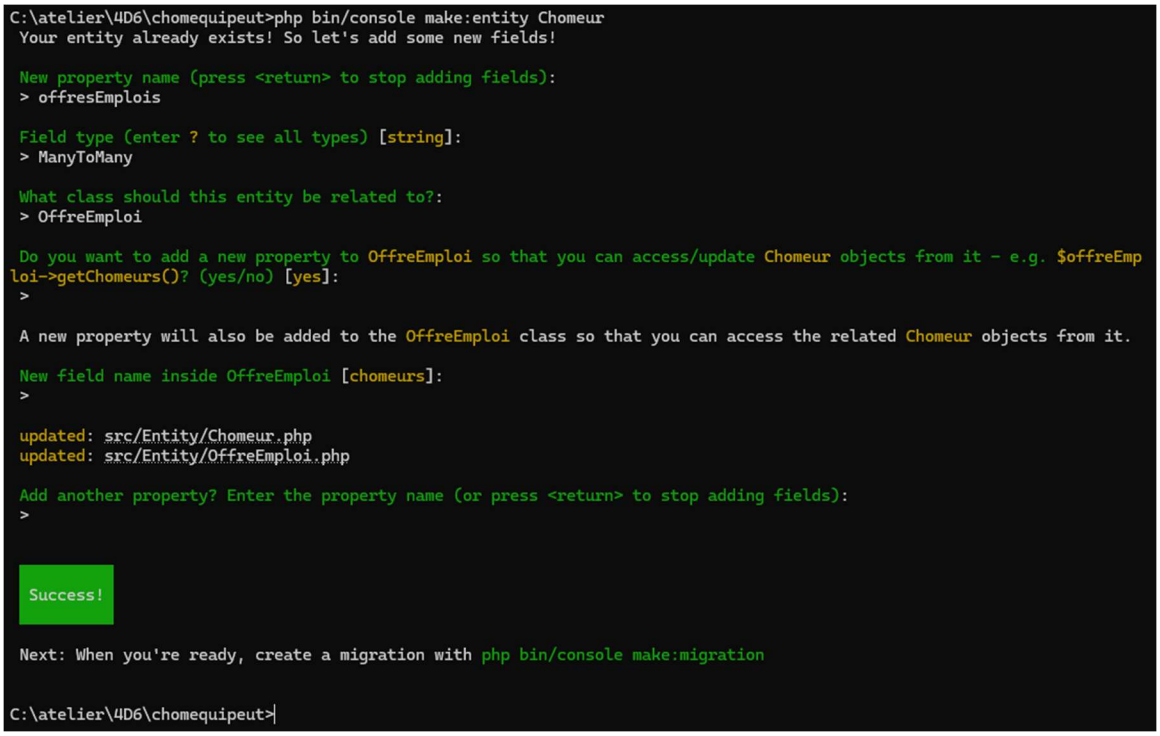
Relations ManyToMany

La relation entre un **Chomeur** et une **OffreEmploi** est **ManyToMany** : un Chomeur peut appliquer sur plusieurs OffreEmploi et une OffreEmploi peut recevoir les applications de plusieurs Chomeur.

Comme dans toutes relations Doctrine, je dois identifier l'entité propriétaire et l'entité inverse. Je pose arbitrairement **Chomeur** comme entité propriétaire, mais ça aurait très bien pu être OffreEmploi, ce n'est pas important. Modifions l'entité **Chomeur** :

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



Voici les changements que l'introduction de l'attribut ManyToMany a provoqué dans l'entity Chomeur :

```
<?php
namespace App\Entity;
. . .
#[ORM\Entity(repositoryClass: ChomeurRepository::class)]
class Chomeur
{
    #[ORM\Id]
    . . .
    #[ORM\ManyToMany(targetEntity: OffreEmploi::class, inversedBy: 'chomeurs')]
    private Collection $offresEmplois;

    public function __construct()
```

```
    {
        $this->offresEmplois = new ArrayCollection();
    }

. . .
    /**
     * @return Collection<int, OffreEmploi>
     */
    public function getOffresEmplois(): Collection
    {
        return $this->offresEmplois;
    }

    public function addOffresEmploi(OffreEmploi $offresEmploi): static
    {
        if (!$this->offresEmplois->contains($offresEmploi)) {
            $this->offresEmplois->add($offresEmploi);
        }
        return $this;
    }
. . .
```

Exécutons **maj** pour modifier notre BD

Une nouvelle table a été créée :

Doctrine a créé une table de liaison entre les deux entités. Une base de données relationnelle a besoin d'une table **chomeur_offre_emploi** pour faire la liaison entre les **chomeur** et les **offre_emploi**. Cette table n'a que deux colonnes : chomeur_id et offre_emploi_id :

Dans votre code PHP, cette table sera transparente: vous ne la verrez jamais. Doctrine se chargera de tout, il faut faire ce saut de confiance : vous utilisez seulement des objets (des instances de classes) et vous laissez Doctrine s'occuper des tables de la BD.

Remarquez la similitude du constructeur et des accesseur/mutateur avec ce que nous avions dans la relation ManyToOne (OffreEmploi et Entreprise).

Dans la méthode **addOffresEmploi()** on ajoute ligne de code permettant la bilatéralité entre les deux entités :

```
    public function addOffresEmploi(OffreEmploi $offresEmploi): static
    {
        if (!$this->offresEmplois->contains($offresEmploi)) {
            $this->offresEmplois->add($offresEmploi);
            $offresEmploi->addChomeur($this);
        }
        return $this;
    }
```

Nous n'avons rien à modifier dans l'entité inverse.

Ajoutons un bouton « Appliquer » à côté de l'offre d'emploi qui intéresse le chômeur connecté dans accueilChomeur.html.twig

```
   . . .
{% for offre in tabOE %}
  <li>
    {{ offre.titre }}, ({{offre.entreprise.nom}}), publié le {{
offre.datePublication|date('d F H:i')}}
    <a class='col-2' href="{{path('appliquer', {'oeid':offre.id}) }}">
       <button class='btn btn-success'>Postuler</button>
    </a>
  </li>
{% endfor %}
```

Dans le contrôleur des chômeurs ajoutons une route pour persister l'application du chômeur sur l'offre d'emploi:

```
   //..\src\controller\ChomeurController.php
. . .
   #[Route('/appliquer/{oeid}', name:'appliquer')]
    public function appliquer(ManagerRegistry $doctrine, Request $req, $oeid ):
Response
    {
       $idChomeur = $req->getSession()->get('chomeurConnecte')->getId();
       $chomeurPostulant = $doctrine->getManager()
             ->getRepository(Chomeur::class)->find($idChomeur);

       $offreEmploi = $doctrine->getManager()
           ->getRepository(OffreEmploi::class)->find($oeid);

       $chomeurPostulant->addOffresEmploi($offreEmploi);
       $doctrine->getManager()->flush();
       $this->addFlash('succes', 'Vous avez postulé sur ' . $offreEmploi-
>getTitre() );

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

Il reste à ajouter un dispositif dans l'accueil du chômeur pour distinguer entre les offres sur lesquelles il a appliqué et celles pour lesquelles il n'a pas encore réagi.

ManyToMany bidirectionnelle

On peut voir sur quelles OffreEmploi les Chomeur ont appliqué. Très bien, mais peut-on voir l'inverse : quel sont les Chomeur qui ont appliqué sur les OffreEmploi? Bien sûr! Notre relation est bidirectionnelle. Nous l'avons configuré ainsi lors de sa création :

```
 Do you want to add a new property to OffreEmploi so that you can access/update Chomeur objects from it - e.g. $offreEmp
loi->getChomeurs()? (yes/no) [yes]:
 >

 A new property will also be added to the OffreEmploi class so that you can access the related Chomeur objects from it.

 New field name inside OffreEmploi [chomeurs]:
 >
```

Pour tester ça, on modifie l'accueil d'une entreprise pour que celle-ci en plus des emplis qu'elle offre affiche aussi les noms des chômeurs ayant postulé sur ces postes :

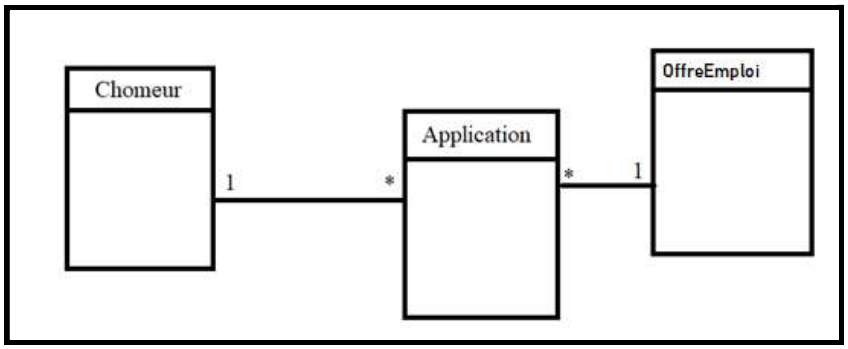
```
{# accueilEntreprise.html.twig #}
. . .
<section class="col-11">
       <ul>
        {% for oe in entrepriseConnectee.offreEmplois %}
        <li>{{oe.Titre}} , {{oe.salaireAnnuel}}$CAN

          {% if oe.chomeurs|length > 0 %}
           Appliquants:
            {% for c in oe.chomeurs %}
              {{c.nom}}  
            {% endfor %}

          {% endif %}
        {% endfor %}
        </li>
    </ul>
    <a href="{{ path('rte_creer_offre_emploi') }}" class='btn btn-info'>Ajouter
une offre d'emploi</a>
</section>
. . .
```

Relations ManyToMany avec attributs

La relation ManyToMany est parfois satisfaisante, mais il arrive fréquemment qu'on doive ajouter des attributs à une relation ManyToMany. Par exemple comment faire pour conserver la date à laquelle le Chomeur a appliqué sur l'OffreEmploi? Ou encore comment avoir un « flag » qui indique si le Chomeur a été convoqué à la suite de son application? Ces attributs n'appartiennent ni au Chomeur ni à l'OffreEmploi, mais bien à la relation entre les deux.

Que faire?

Il faut créer une nouvelle entité! Je propose de l'appeler **Application**. Elle déploiera deux relations ManyToOne : une avec Chomeur et une avec OffreEmploi. On aura donc le schéma suivant :



N'est-ce pas? Un chômeur peut faire plusieurs applications, mais une application n'est faite que par un seul Chomeur, de même une OffreEmploi peut recevoir plusieurs applications mais une application ne concerne qu'une seule OffreEmploi. Ce sont bien deux relations ManyToOne.

La beauté de la chose c'est que vous savez déjà créer des entités et créer des relations ManyToOne! Il n'y a rien à ajouter de spécial ici, les apprentissages que vous avez fait lors des ManyToOne simples sont applicables en totalité dans ce contexte. Alors retroussez vos manches et créez l'entité **Application** qui aura les attributs :

$date : dateTime

$convoque : boolean

$chomeur : relations ManyToOne avec targetEntity= « Chomeur »

$offreEmploi : relation ManyToOne avec targetEntity= « OffreEmploi ».

L'entité **Application**, vous l'aurez compris, sera l'entité propriétaire de ces deux nouvelles relations.

Vous avez toutes les connaissances requises pour relever ce défi. Je vous encourage à essayer de créer cette relation many to many avec attributs. Dans le TP4 vous aurez ce type de relation à développer entre un produit et une commande : une commande contient plusieurs produits et un produit peut être présent dans plusieurs commandes.