6. Les repositories et le langage DQL

Les repositories vont permettre de récupérer des données de la base de données de différentes manières :

- via des méthodes spécifiques de Doctrine 2 ;
- en utilisant le langage DQL ;
- en exécutant des requêtes SQL.

Pour récupérer une entité, on va utiliser le repository de l'entité correspondante. Dans le cas d'une jointure, le repository utilisera plusieurs types d'entité à travers l'EntityManager.

Les repositories héritent de la classe Doctrine\ORM\Entity\Repository qui dispose de méthodes permettant de récupérer des entités.

Si les repositories ne sont pas créées, il suffit de renseigner dans les entités l'annotation Entity et rajouter la classe repositoryClass :

```
/**
 * Coach
 *
 * @ORM\Table(name="Coach")
 * @ORM\Entity(repositoryClass="App\Repository\CoachRepository")
 */
```

Ensuite relancer la commande :

php bin/console make:entity --regenerate

Si on ne précise pas de nom d'entité, il va créer les repositories pour l'ensemble des entités créées.

1. Les méthodes de base du repository

1.1. Les méthodes normales

Il existe 4 méthodes normales dans le repository d'une entité.

Exemple : repository CoachRepository

```
namespace App\Repository;

use App\Entity\Coach;
use Doctrine\Bundle\DoctrineBundle\Repository\ServiceEntityRepository;
use Doctrine\Persistence\ManagerRegistry;

/**
 * @method Coach|null find($id, $lockMode = null, $lockVersion = null)
 * @method Coach|null findOneBy(array $criteria, array $orderBy = null)
 * @method Coach[]    findAll()
 * @method Coach[]    findBy(array $criteria, array $orderBy = null, $limit = null,
$offset = null)
 */
```

Pour utiliser les méthodes normales, il convient de récupérer au préalable le repository de l'entité concernée :

$repository = $this->getDoctrine()->getManager()->**getRepository('NomEntite::class')** ;

Ou

$repository = $this->get('doctrine')->getManager()->**getRepository('NomEntite::class')** ;

Méthode	Rôle
find($id) find(array('cle_primaire_1' =>$id1, 'cle_primaire_2' =>$id2)) ;	Récupère l'entité correspondant à l'id fourni en paramètre : $id, donc une instance de l'entité NomEntite (ce qui correspond à un tuple de la table correspondante à l'entité) Dans le cas où la clé primaire est constituée de plusieurs colonnes.
findAll()	Retourne un tableau contenant toutes les instances de l'entité (tous les tuples de la table). On peut parcourir ce tableau d'entités depuis le contrôleur ou dans une vue Twig.
findBy(array $criteres, array $orderBy, $limits, $offset)	Retourne une liste d'instances d'entité qui correspondent à un ou plusieurs critères. Le nombre d'instances à retourner peut être indiqué par $limits, à partir de $offset (par rapport au jeu de résultat). On peut également indiquer l'ordre de tri ('DESC' ou 'ASC').
findOneBy(array $criteres).	Retourne une seule instance d'entité qui répond aux critères fournis sous forme de tableau.

Exemples d'extraction de données à l'aide des méthodes normales :

Récupérer tous les enregistrements :

```
$coaches = $this->getDoctrine()
            ->getRepository(Coach::class)
            ->findAll();
```

Récupérer un enregistrement :

```
$coach = $this->getDoctrine()
            ->getRepository(Coach::class)
            ->find(3);
```

1.2. Les méthodes personnalisées du repository

Les méthodes précédentes ne sont pas adéquates dans le cas de requêtes plus élaborées comme les jointures. Pour cela, on va manipuler : le QueryBuilder. La méthode getQuery() récupère la requête et la méthode getResult() les résultats.

1.2.1. Le QueryBuilder

Il permet de construire une requête. L'EntityManager (EM) dispose de la méthode createQueryBuilder() qui retournera une instance de QueryBuilder.

Syntaxe :

```
// Dans le repository
public function findByExampleField($value)
{     return $this->createQueryBuilder('c')
            ->andWhere('c.exampleField = :val')
            ->setParameter('val', $value)
            ->orderBy('c.id', 'ASC')
            ->setMaxResults(10)
            ->getQuery()
            ->getResult() ;
}
```

Exemple 1 :

```
// Dans le repository
public function findByDiscipline($value)
    {
        return $this->createQueryBuilder('c')
            ->andWhere('c.discipline = :val')
            ->setParameter('val', $value)
            ->orderBy('c.discipline', 'DESC')
            ->setMaxResults(2)
            ->getQuery()
            ->getResult()
        ;
    }
```

Appel de la méthode depuis le contrôleur :

```
/**
     * @Route("/getcoachsbydiscipline/{discipline}", name="coachs_by_discipline",
     * methods={"GET","POST")
     */

    public function getCoachsByDiscipline($discipline)
    {
         $rp = $this->get('doctrine')->getRepository(Coach::class);
         $coachs = $rp->findByDiscipline($discipline);

         return $this->render('coach/index.html.twig', [
             'coachs' => $coach,
         ]);
    }
```

Exemple 2

```
// Dans le repository
public function findByDisciplinenAndPalmares($discipline, $palmares)
    {
        return $this->createQueryBuilder('c')
            ->where('c.discipline = :discipline)
            ->setParameter(discipline, $discipline)
            ->andWhere('c.palmares = :palmares)
            ->setParameter(palmares, $palmares)
            ->getQuery()
            ->getResult() ;
           // $this->createQueryBuilder('c')
           // ->where(c.discipline = :discipline)
           // ->andWhere('c.palmares = :palmares)
           // ->setParameters(array(discipline=> $discipline, palmares => $palmares)
           // ->getQuery()
           // ->getResult() ;

    }
```

1.2.2. La requête

Le queryBuilder est l'objet à partir duquel on extrait les résultats. Il existe différentes manières de récupérer les résultats :

GetResult()
`$liste = $queryBuilder ->getQuery()`
`->getResult() ;`
	Exécute la requête et retourne un tableau contenant les résultats sous forme d'objets

GetArrayResult()
`$liste = $queryBuilder->getQuery()`
`->getArrayResult() ;`
	Exécute la requête et retourne le résultat sous forme de tableau.
	Depuis une vue Twig,
	{{ entite.attribut }} exécute :
	- $entite->getAttribut() si $entite est un objet
	- $entite['attribut'] si $entite est un tableau

GetScalarResult()
`$liste = $queryBuilder->getQuery()`
`->getScalarResult() ;`
	Exécute la requête et retourne un tableau contenant les résultats sous forme de valeurs. Utilisée dans le cas où une seule valeur est utilisée dans le SELECT

GetOneOrNullResult()
`$unobjet = $queryBuilder->getQuery()`
`->getOneOrNullResult() ;`
	Retourne un seul résultat (donc une seule instance d'entité) ou null si pas de résultat. Une exception est déclenchée si la requête retourne plus d'une ligne.

GetSingleResult()
`try {`
`$unObjet = $queryBuilder->getQuery()`
`->getSingleResult();`
`}`
`catch(\Doctrine\Orm\NoResultException $e)`
`{`
`$unObjet = null;`
`}`
	Retourne un seul résultat. Déclenche une exception si aucun résultat n'est retourné.

GetSingleScalarResult()
`$nb = $queryBuilder->getQuery()`
`->getSingleScalarResult();`
	Retourne une seule valeur. Exemple de requêtes avec des fonctions d'agrégation.

Execute() :
`$this->getDoctrine()->getManager()`
`->prepare($sql)->execute() ;`
	Exécute les requêtes qui ne retournent pas de résultats (UPDATE, INSERT...)

2. Utilisation de DQL (Doctrine Query Language)

Le DQL est une sorte de SQL adapté à Doctrine. Pour cela, on utilisera seulement les objets Query et les résultats. On l'utilisera depuis un repository.

La création de la requête DQL s'effectue grâce à la méthode createQuery() de l'EM.

$query = $this->**getEntityManager()->createQuery**('requêteDQL') ->getResult() ;

Dans les FROM et les JOIN de la requête DQL, il faut utiliser le **nom des entités** (alias ou namespace complet) et non pas les tables de la base de données.

Exemples de syntaxe de requêtes DQL :

SELECT alias FROM NomEntite alias

```
    /**
     * @return Coach[] Returns an array of Coach objects with DQL
     */
    public function getCoachsByDql(): ?array
    {   $dql = 'select c from App\Entity\Coach c';
        return $this->getEntityManager()->createQuery($dql)->getResult();
    }
```

SELECT alias FROM Entite alias WHERE alias.attribut = 'valeur' ;

```
    /**
     * @return Coach [] Returns an array of Coach objects with DQL
     */
    public function getCoachsByDql(): ?array
    {
        $dql = "select c from App\Entity\Coach c where c.discipline='Basket'";
        return $this->getEntityManager()->createQuery($dql)->getResult();

    }
```

SELECT alias FROM Entite alias WHERE alias.attribut IN(valeur1, valeur2, .., valeurN)

```
public function getCoachsByDql(): ?array
    {
        $dql = "select c from App\Entity\Coach c
                where c.discipline IN ('Basket', 'Rugby')";
        return $this->getEntityManager()->createQuery($dql)->getResult();

    }
```

Si on utilise des paramètres comme valeurs de sélection :

$query = $this->getEntityManager()

->createQuery('SELECT alias FROM Entite alias WHERE alias.attribut =:parametre')

->setParameter(‘parametre’, $parametre) ;

Exemple :

```
public function getCoachsByDql($discipline): ?array
    {
        return $this->getEntityManager()
                    ->createQuery(' select c from App\Entity\Coach c
                                    where v.discipline = :discipline')
                    ->setParameter('discipline', $discipline)
                    ->getResult();
    }
```

Remarques :

- ⇨ Les résultats seront sous forme de tableaux ou d'objets selon l’expression utilisée dans le SELECT. Quand on sélectionne des attributs d’un objet, le résultat est un tableau contenant les champs sélectionnés.
- ⇨ Il est possible de tester ses requêtes DQL grâce à la commande doctrine:query:dql

```
php bin/console doctrine:query:dql "select c from App\Entity\Coach c"
```

3. Les jointures

Les jointures traduisent les relations qui existent entre les entités. On ne peut faire une jointure que si l'entité du FROM possède un attribut (entité propriétaire) vers l'entité à joindre, ou si la relation est bidirectionnelle.

On crée des jointures avec le **QueryBuilder** via les méthodes **leftJoin()** ou **join()** (équivalent du INNER JOIN) permettent de créer une jointure.

Code :

SELECT aliasE1, aliasE2 FROM App\Entity\Entite1 aliasE1 **JOIN** aliasE1.PK aliasE2

$qb = $this->createQueryBuilder('aliasE1')->**Join**('E1.attribut', 'aliasE2', ['WITH'], 'Condition')->addSelect('aliasE2') ;

$resultat = $qb->getQuery()->getResult() ;

Équivalent SQL : **SELECT * FROM E1 aliasE1 JOIN E2 aliasE2**
ON ConditionJointure AND Condition ;

aliasE1 : alias de l'entité principale, elle représente l'entité du FROM.

E1.attribut : attribut de l'entité principale, c'est l'attribut de jointure.

AliasE2 : alias de l'entité jointe (entité inverse dans le cas des relations bidirectionnelles)

addSelect() : permet de rajouter à la première sélection le résultat de la deuxième sélection pour réaliser la jointure.

['WITH'] : facultatif (sans les crochets). Ajoute une condition pour la jointure.

Exemple :

SQL : select e.libelle, c.nom_coach FROM equipe e inner join coach c ON e.id_coach = c.id ;

```
// EquipeRepository
public function findByEquipeCoach(): ?Array
{
    $qb = $this->createQueryBuilder('e')
               ->join ('e.coach', 'c')
               ->addSelect('e.libelle', 'c.nomCoach') ;
    return $qb->getQuery()-> getResult();
}
```