Doctrine : requêtes avancées, associations *--*

- Requêtes utilisant des critères composés
- Repository et inversion de dépendance
- Doctrine Query Language (DQL)
- associations *--*

critères composés

- Objets de type Criteria représentant des critères composés pouvant utiliser tous les types de comparaison
- peuvent être appliqués à un repository pour exprimer des conditions de sélection
- peuvent être appliqués à une Collection pour appliquer une condition de sélection sur une association

```php
// spectacles dont le titre contient "Stone"
$pestacles = $spectacleRepository->matching(
 Criteria::create()
        ->where(Criteria::expr()->contains("titre", "Stone"))
        ->orderBy(["titre" => "ASC"])
        ->setMaxResults(5)
);
// spectacles dont le titre contient "Stone" et la durée est >= 120
$pestacles = $spectacleRepository->matching(
    Criteria::create()
        ->where(Criteria::expr()->contains("titre", "Stone"))
        ->andWhere(Criteria::expr()->gte("duree", 120))
        ->orderBy(["titre" => "ASC"])
);

// spectacles dont le titre contient "Stone" et la durée est >= 120
// ou dont le titre contient "cygnes" et la durée est < 30
$e1 = Criteria::expr();
$e2 = Criteria::expr();
$pestacles = $spectacleRepository->matching(
    Criteria::create()
        ->where($e1->andX($e1->contains("titre", "Stone"),
                          $e1->gte("duree", 120)))
        ->orWhere($e2->andX($e2->contains("titre", "cygnes"),
                            $e2->lt("duree", 30)))
        ->orderBy(["titre" => "ASC"])
);
```

- La méthode matching() est également disponible pour les collections, ce qui permet d'exprimer des critères de filtrage sur les associations

```php
$seances = $spectacleRepository
    ->findOneBy(["titre" => "Le lac des cygnes"])
    ->seances
    ->matching(Criteria::create()
            ->where(Criteria::expr()->eq("date", "2023-12-06"))
            ->orderBy(["heure" => "ASC"])
);
```

Repository et inversion de dépendance

- Doctrine fournit des repository par défaut associés à chaque Entité
- Ces repository héritent de EntityRepository qui implante des méthodes génériques
- il est possible de créer des classes Repository spécialisées et adaptées
  - pour chaque entité ou
  - partagées entre plusieurs entités

- Ces repository :
  - Peuvent implanter une interface métier
  - Héritent de EntityRepository pour bénéficier des méthodes génériques de l'ORM
- On y déclare toutes les méthodes utilisées par les services métier pour dialoguer avec la persistance
- Ces classes doivent être indiquées dans les attributs de mapping de chaque entité
- Bonne pratique : à utiliser de manière systématique

On associe l'entité `Spectacle` à un Repository spécifique

```php
#[Entity(repositoryClass: SpectacleRepository::class)]
#[Table(name: "spectacle")]
class Spectacle
{
    #[Id]
    #[Column(type: Types::INTEGER)]
    #[GeneratedValue(strategy: "AUTO")]
    public int $id;

  /* … */
}
```

```php
$spectacleRepository = $em->getRepository(Spectacle::class);
```

C’est ce repository qui est retourné

```php
interface SpectacleRepositoryInterface
{
    public function saveSpectacle(Spectacle $spectacle): Spectacle;
    public function getSpectaclesByKeyword(string $keyword): Array;
}
```

SpectacleRepository hérite de EntityRepository et
Implante l'interface métier SpectacleRepositoryInterface

```php
class SpectacleRepository extends EntityRepository
                    implements SpectacleRepositoryInterface
{
    public function saveSpectacle(Spectacle $spectacle) : Spectacle
    {
        $this->getEntityManager()->persist($spectacle);
        $this->getEntityManager()->flush();
        return $spectacle;
    }
    public function getSpectaclesByKeyword(string $keyword): Array
    {
        return $this->matching(Criteria::create()
            ->where(Criteria::expr()->contains('description', $keyword))
            ->orderBy(['titre' => 'ASC'])
        )->toArray();
    }
```

Doctrine Query Language - DQL

- Les méthodes findBy() et matching() des repository permettent de construire des requêtes simples portant sur une seule table
- Pour construire des requêtes plus élaborées, notamment des jointures, Doctrine se base sur un langage de requêtes de haut niveau : DQL
- DQL est inspiré de SQL mais les requêtes concernent et s'expriment sur des entités et non des tables
  - nom des classes entités (et non des tables)
  - nom des propriétés des objets (et non des colonnes de tables)

DQL - SELECT

- La claude SELECT permet d'exprimer une sélection et d'obtenir un résultat sous forme d'entités

```
SELECT s FROM ticketnet\entities\Spectable s
WHERE s.titre LIKE ' %cygnes %'
```

  - retourne un tableau d'entités Spectacle dont le titre contient "cygnes"

- Le type des résultats
  - tableau d'objets : `SELECT s FROM ticketnet\entities\Spectable s`
  - tableau de tableaux :

```
SELECT s.titre, s.description
FROM ticketnet\entities\Spectable s
```

  - tableau de tableaux contenant des objets et des valeurs

```
SELECT s, s.titre, s.description
FROM ticketnet\entities\Spectable s
```

Jointures DQL

- Une requête SELECT peut contenir des jointures
  - les conditions de jointures qui correspondent aux associations n'ont pas besoin d'être spécifiées
  - jointure normale : pour suivre et contraindre les associations

```
SELECT s FROM ticketnet\entities\Spectable s JOIN s.lieu l
WHERE l.nom LIKE ' %Zenith%'
```

  - seul le spectacle est chargé

  - jointure avec chargement : idem + fetch des objets associés

```
SELECT s, l FROM ticketnet\entities\Spectable s JOIN s.lieu l
WHERE l.nom LIKE ' %Zenith%'
```

  - le spectacle et le lieu sont chargés

exécution, paramètres

```
class SpectacleRepository extends EntityRepository
                          implements SpectacleRepositoryInterface{

    public function getSpectaclesBallet(): array {
      $dql = "SELECT s FROM ticketnet\\entities\Produit p
              WHERE s.titre LIKE '%ballet%'";

      $query= $this->getEntityManager()->createQuery($dql);
      return $query->getResult();
    }
}
```

- les requêtes peuvent être paramétrées, avec des paramètres nommés ou positionnels

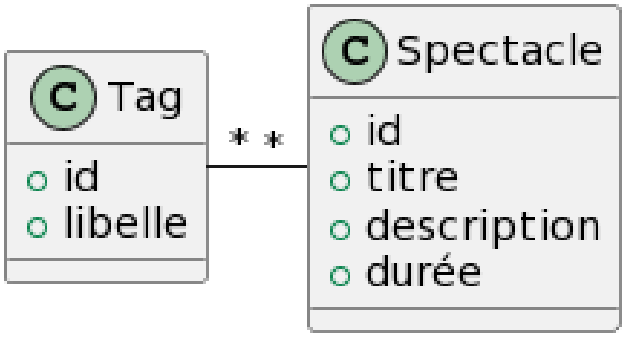
```
public function getSpectaclesByLieu(string $nom_lieu) : array {
    $dql = "SELECT s, l FROM ticketnet\\entities\Spectacle s
              JOIN s.lieu l WHERE l.nom = :nl
              ORDER BY s.titre ASC";

    $query= $this->getEntityManager()->createQuery($dql);
    $query->setParameter('nl', $nom_lieu);
    return $query->getResult();
}
```

exemples

```
SELECT s FROM \ticketnet\entities\Spectacle s

SELECT s.id, s.titre FROM \ticketnet\entities\Spectacle s

SELECT s FROM \ticketnet\entities\Spectacle s
         JOIN s.lieu l
         ORDER BY l.capacite DESC

SELECT s, l FROM \ticketnet\entities\Spectacle s
         JOIN s.lieu l
         ORDER BY l.nom ASC, s.titre DESC

SELECT s, l, se FROM \ticketnet\entities\Spectacle s
         JOIN s.lieu l WITH l.nom = ?1
         JOIN s.seances se
         WHERE se.date > : ?2
         ORDER BY se.date ASC

SELECT s, l.nom, se.date, se.heure FROM \ticketnet\entities\Spectacle s
         JOIN s.lieu l
         JOIN s.seances se
         WHERE l.nom = ?1
         ORDER BY se.date ASC
```

compléments sur les associations

- associations *--* basées sur une table pivot

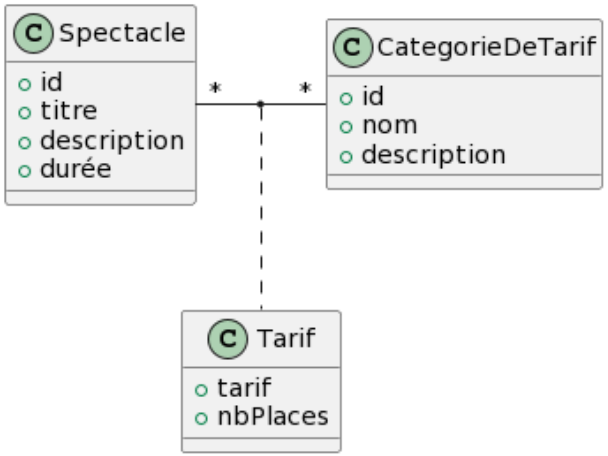



```
tag( #id, libelle)
spectacle( #id, titre, descr, duree )
spectacle_tag( #spectacle_id, #tag_id )
```

```php
#[Entity(repositoryClass: SpectacleRepository::class)]
#[Table(name: "spectacle")]
class Spectacle
{
    #[Id]
    #[Column(type: Types::INTEGER)]
    #[GeneratedValue(strategy: "AUTO")]
    public int $id;

    #[ManyToMany(targetEntity: Tag::class, inversedBy: "spectacles")]
    #[JoinTable(name: "spectacle_tag")]
    #[JoinColumn(name: "spectacle_id", referencedColumnName: "id")]
    #[InverseJoinColumn(name: "tag_id", referencedColumnName: "id")]
    public Collection $tags;
```

```php
#[Entity]
#[Table(name: 'tag')]
class Tag {
    #[Id]
    #[Column(type: Types::INTEGER)]
    #[GeneratedValue(strategy: 'AUTO')]
    public int $id;

    #[ManyToMany(targetEntity: Spectacle::class, mappedBy: 'tags')]
    public Collection $spectacles;
```

- associations *--* portant des attributs




```
spectacle( #id, titre, descr, duree )
categorieTarif( #id, nom, descr )
tarif( #spec_id, #cat_id, tarif, nb_place )
```

- Doctrine ne permet pas de gérer et manipuler des attributs portés par une association *--*
- La classe association doit être transformée en une entité classique