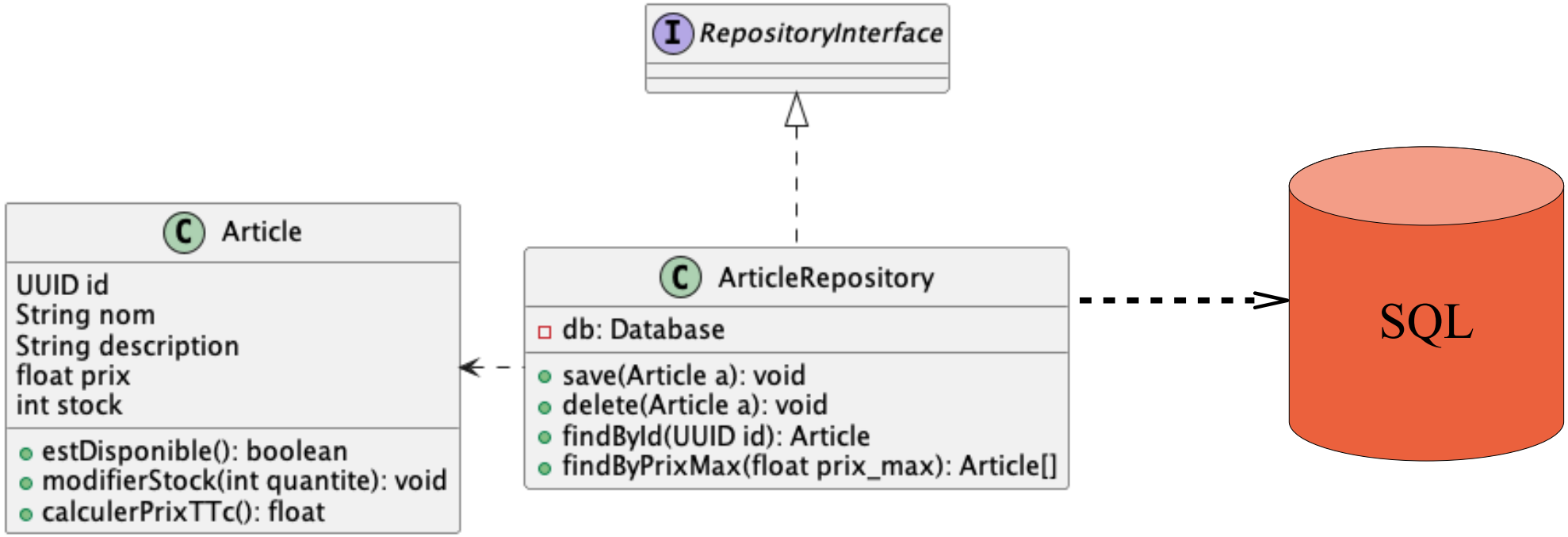
Les ORM basés sur "Repository" ou "DataMapper"

- Modèle d'architecture pour l'accès aux données différent d'active record
- utilisé dans plusieurs "gros" orm : Doctrine, Hibernate

- **Principe** : intercaler un ou plusieurs objets entre les entités et la couche persistance
- **intérêt** : améliorer le découplage entre entités du Domaine et données persistantes

Le modèle Repository (aka DataMapper)



entités,
fonctions métier

repository,
lien avec la BD

principes

- le Repository (ou data mapper) est le seul objet en relation avec la base
- les entités ne connaissent pas le repository et sont des objets PHP classiques : des Pure Old PHP Object (POPO:)

- **intérêt** : permet d'inverser la dépendance
  - Les entités du domaine sont indépendantes de la couche persistance
  - Les repository peuvent implanter une interface métier : la persistance devient dépendante du métier

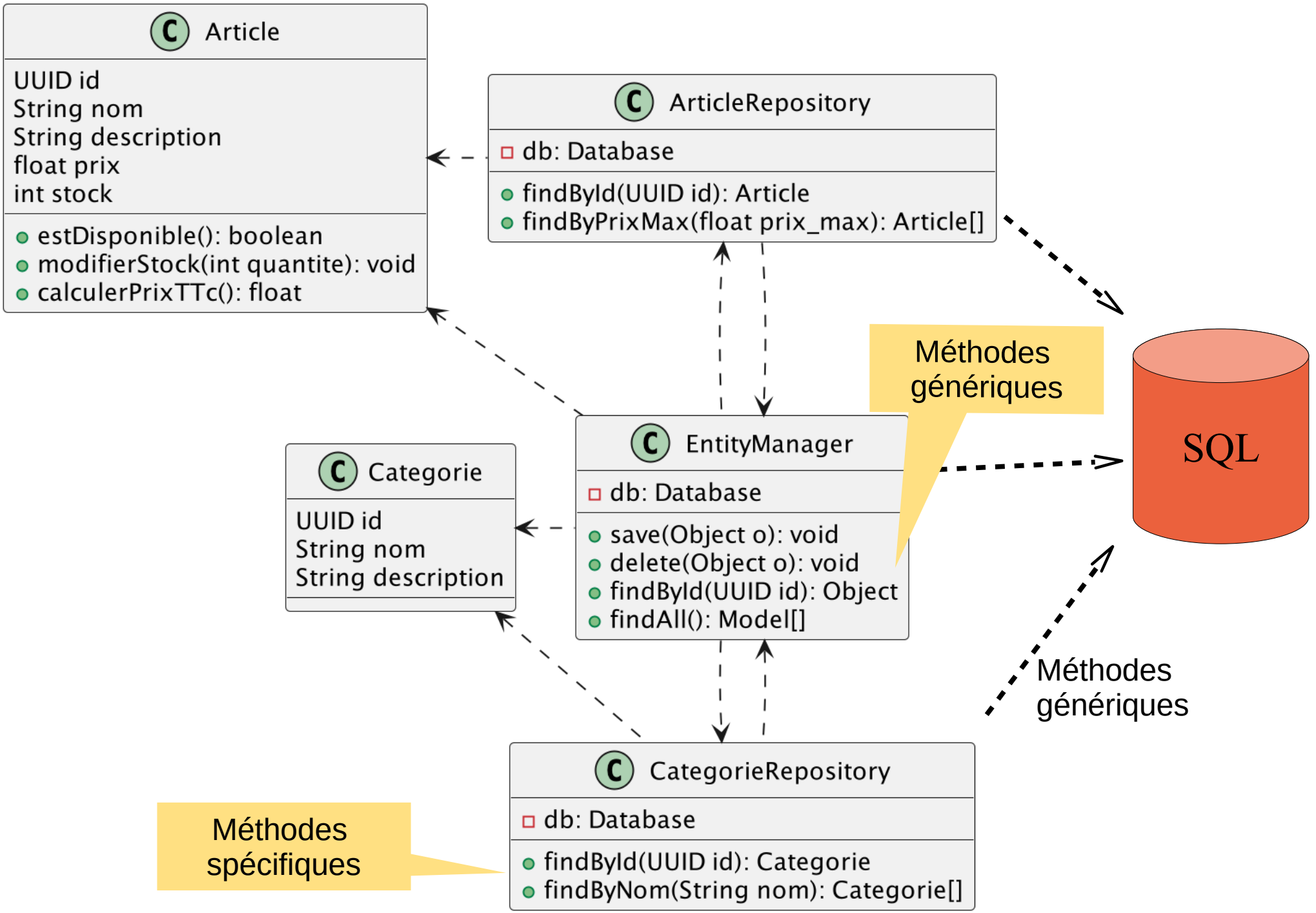
Utilisation

```php
<?php

// Création d'un objet métier et sauvegarde
$a = new Article() ;
$a->nom= 'hello' ;
$a->descr = 'bo velo bleu' ;

$mapper = new ArticleRepository() ;
$mapper->insert($a) ;

// mise à jour

$velo->tarif = 120 ;
$velo->calculerPrixTTC() ;
$mapper->update($velo);

// supprimer un objet
$mapper->delete($a);
```

EntityManager

- ***EntityManager***: extension de Repository dans laquelle on ajoute un objet capable
  - De donner accès aux différents repository
  - De sauvegarder/mettre à jour/supprimer de manière générique toutes les entités

- *EntityManager* : synchronisation des entités avec la base de données persistante
  - insertion, mise à jour, suppression
- *Repository* : requêtes d'accès à la base, un repository par entité
- modèle utilisé par Hibernate et Doctrine 2


Article
UUID id
String nom
String description
float prix
int stock
estDisponible(): boolean
modifierStock(int quantite): void
calculerPrixTTc(): float
ArticleRepository
db: Database
findById(UUID id): Article
findByPrixMax(float prix_max): Article[]
Méthodes génériques
SQL
Categorie
UUID id
String nom
String description
EntityManager
db: Database
save(Object o): void
delete(Object o): void
findById(UUID id): Object
findAll(): Model[]
Méthodes génériques
CategorieRepository
db: Database
findById(UUID id): Categorie
findByNom(String nom): Categorie[]
Méthodes spécifiques

exemple avec Doctrine

```php
<?php
/* ...*/

// obtaining the Doctrine entity manager
$entityManager = EntityManager::create($conn, $config);

$product = new Article();
$product->setName( 'velo rouge' );

$entityManager->persist($product);  // l'entité est gérée par l'ORM
$entityManager->flush();            // synchronisation entités - BD

$product->setName( 'velo rouge et bleu' );
$entityManager->flush();            // synchronisation entités - BD

// accès au repository d'articles et recherche d'un article
$productRepository = $entityManager->getRepository('Article');
$products = $productRepository->findByProxMax(20.0);

print $product->id . $product->id . "\n" ;

$entityManager->remove($product);   // l'entité est supprimée
$entityManager->flush();            // synchronisation entités - BD
```

le mapping entités - tables

- L’entity manager à besoin d'information pour gérer les transformations entités <--> lignes SQL
  - nom des tables
  - colonne clé primaire
  - correspondance propriété ↔ colonnes de tables
  - description des associations

- différentes approches selon l'ORM :
  - fichiers de description en plus des classes entités
  - attributs dans les classes entités

intérêts du pattern Repository+EntityManager

- Découplage complet entre entités du domaine vs database : les entités sont des POPO (sic) (*Pure Old Php Object)*
- Inversion de dépendance possible entre le métier et l'infrastructure

- les annotations ou fichiers de configuration décrivant le lien entités – tables peuvent être utilisés pour générer les tables

Utiliser Doctrine

www.doctrine-project.org

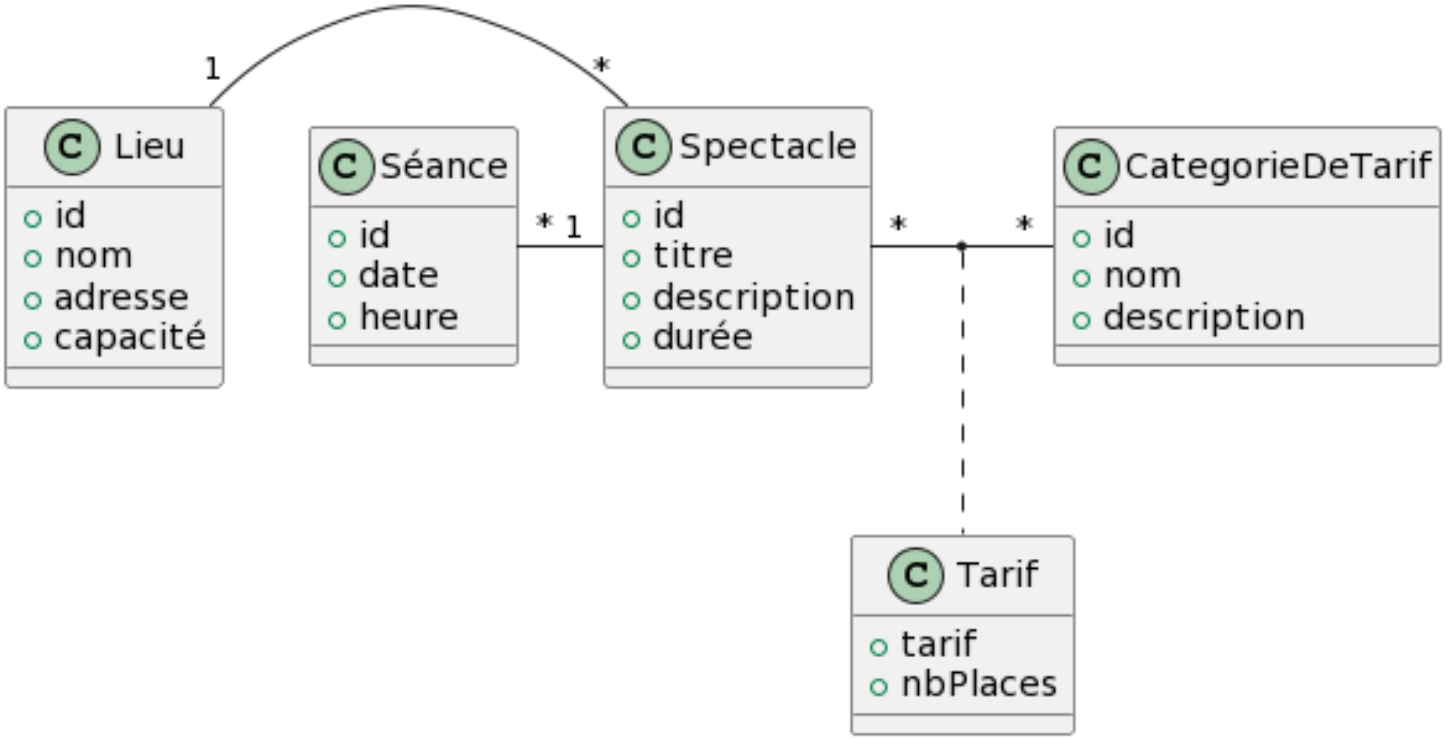
- ORM PHP basé sur le pattern repository+entityManager
- Entity Manager : synchronisation entités - base de données, accès aux repository
- Repository : requêtes sur la bases de données retournant des entités, accès à l'entity manager
  - Un repository par défaut pour chaque entité, basé sur le mapping entité-table

- La synchronisation entités-BD est gérée grâce au pattern *Unit Of Work* : toutes les modifications des entités gérées sont répertoriées puis synchronisées en une fois au moment du **`flush()`**

Le mapping dans Doctrine

- définitions du lien entre entités et tables
- différentes formes utilisables :
  - Attributs (recommandé)
  - XML
  - Code PHP
  - Annotations Docblock (déprécié, retiré de Doctrine 3)
  - YAML (déprécié, retiré de Doctrine 3)

exemple



spectacle(id,nom, descr, durée, *lieu_id*)

lieu(id, nom, adresse, capacité)

seance(id, date, heure, *spect_id*)

categorie_place(id, nom, descr)

tarif(*spect_id, cat_id*, tarif, nbplaces)

Mapping des Entités

```php
use Doctrine\ORM\Mapping\Column;
use Doctrine\ORM\Mapping\Entity;
use Doctrine\ORM\Mapping\GeneratedValue;
use Doctrine\ORM\Mapping\Id;
use Doctrine\ORM\Mapping\Table;

#[Entity]
#[Table(name: "spectacle")]
class Spectacle {

  #[Id]
  #[Column(type: Types::INTEGER)]
  #[GeneratedValue(strategy: "AUTO")]
  private int $id;

  #[Column(name: "title",
           type: Types::STRING,
           length: 64)]
  private string $titre;

  #[Column( type: Types::TEXT)]
  private string $description;

}
```

la classe `Spectacle` est une entité liée à la table `spectacle`

la propriété **`$id`** est une clé primaire de type integer dont la valeur est générée par la base, liée à la colonne `id`

la propriété **`$titre`** est liée à la colonne `title,` dont le type est une chaine de caractère de taille 64

la propriété **`$description`** est liée à la colonne `description,` dont le type est un TEXT

Mapping des associations : 1 -- *

```php
/* .. */
use Doctrine\ORM\Mapping\JoinColumn;
use Doctrine\ORM\Mapping\ManyToOne;


#[Entity]
#[Table(name: "spectacle")]
class Spectacle
{
/* .. */

  #[ManyToOne(targetEntity: Lieu::class)]
  #[JoinColumn(name: "lieu_id", referencedColumnName: "id")]
  private ?Lieu $lieu = null;
```

entité cible

clé étrangère, clé primaire référencée

```php
use Doctrine\ORM\Mapping\OneToMany;
use Doctrine\Common\Collections\Collection;
#[Entity]
#[Table(name: "lieu")]
class Lieu {

    #[OneToMany(targetEntity: Spectacle::class, mappedBy: "lieu")]
    private Collection $spectacles;
}
```

classe Doctrine pour gérer des collections, utilisable comme des […]

association inverse

Configuration, création de l'entityManager

liste des répertoires contenant les classes entité

```php
use Doctrine\DBAL\DriverManager;
use Doctrine\ORM\EntityManager;
use Doctrine\ORM\ORMSetup;

$entity_path = [__DIR__  . '/../domain/entities/'];
$isDevMode=true;

$dbParams = parse_ini_file(__DIR__ . '/../config/ticket.ini');

$config = ORMSetup::
      createAttributeMetadataConfiguration($entity_path, $isDevMode);
$connection = DriverManager::getConnection($dbParams, $config);
$entityManager = new EntityManager($connection, $config);
```

```
driver=pgsql
user=ticket
dbname=ticketnet
password=
host=ticket.db
```

```
createAttributeMetadataConfiguration( … )
createXMLMetadataConfiguration( … )
```

Requêtes sur clé primaire, parcours du graphe d'objets

obtenir le repository par défaut associé au type entité

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

$s=$spectacleRepository->find(1);
print $s->titre . "\n";
print $s->description . "\n";

print $s->lieu->nom . "\n";

foreach ($s->seances as $seance) {
    print $seance->date . "\n";
    print $seance->heure . "\n";
    print "---\n";
}
```

accès par l'identifiant (PK)

suivre les associations

- Les repository par défaut héritent de EntityRepository qui fournit des méthodes génériques (find)

Requêtes à conditions simples

```php
// un spectacle dont le titre est "Le lac des cygnes"
$spectacle = $spectacleRepository
             ->findOneBy(["titre" => "Le lac des cygnes"]);

// les spectacles dont la durée est 120
$spectacles = $spectacleRepository->findBy(["duree" => 120]);

// les spectacles dont la durée est 120 et le titre "Le lac des cygnes"
$spectacles = $spectacleRepository
              ->findBy(["duree" => 120,
                        "titre" => "Le lac des cygnes"]);

// les spectacles dont la durée est 110, 120, 130, 140
$spectacles = $spectacleRepository
              ->findBy(["duree" => [110, 120, 130, 140]]);

// les mêmes, ordonnés par titre, avec limit = 5 et offset = 10
$spectacles = $spectacleRepository
              ->findBy(["duree" => [110, 120, 130, 140]],
                       ["titre" => "ASC"],
                       5, 10);
```

insertion/modification:suppression d'objets

```php
$s = new Spectacle();
$s->titre = "Le lac des cygnes";
$s->description = "Ballet en 4 actes, assez célèbre";

$entityManager->persist($s);

$s1 = new Seance(); $s1->date = "2021-12-01"; $s1->heure = "20:00";
$s2 = new Seance(); $s2->date = "2021-12-02"; $s2->heure = "20:00";
$entityManager->persist($s1);
$entityManager->persist($s2);

$s->lieu = $entityManager->getRepository(Lieu::class)->find(1);
$s->seances->add($s2);
$s->seances->add($s1);

$entityManager->flush();

$s2->heure = "20:30";
$entityManager->flush();

$entityManager->remove($s1);
$entityManager->flush();
```

enregistre l'entité dans le manager pour la rendre persistante

associations

synchronise les modifications dans la base de données - des requêtes INSERT - UPDATE - DELETE sont exécutées
Il n'y a jamais de flush automatique

installation : avec composer

```
"require" : {
  "php": ">=8.1",
  "doctrine/orm": "*",
  "symfony/cache": "^6.3"
}
```