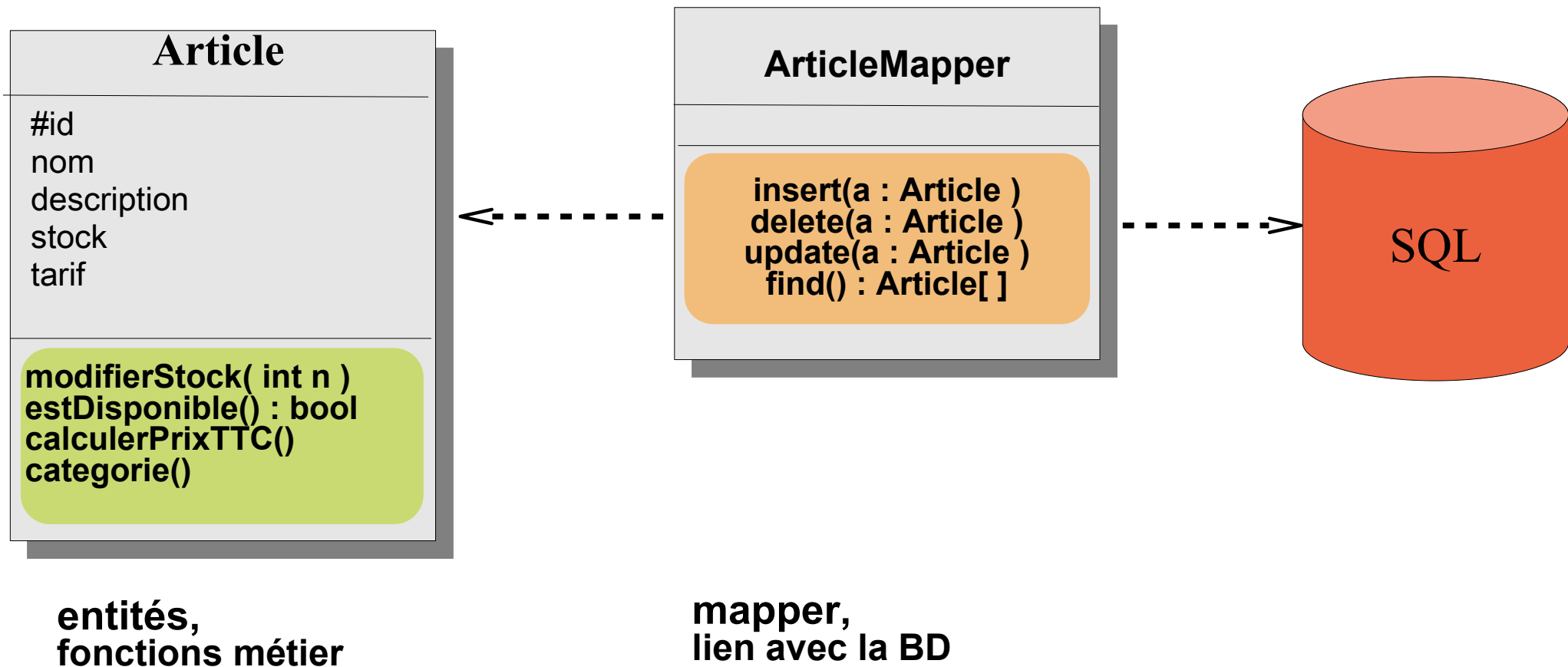


Les Modèles « DataMapper » et «Repository»

- Modèles d'architecture pour l'accès aux données différents 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 couche Domaine et couche données persistantes

Le modèle DataMapper (aka DAO, aka Repository)



principes

- le DataMapper (ou DAO, ou Repository) est le seul objet en relation avec la base
- les entités ne connaissent pas le DAO et sont des objets PHP classiques : des Pure Old PHP Object (POPO:)
- **intérêt** : découplage entre entités et BD
 - La couche Domaine est indépendante de la couche persistance
 - Les entités de la couche Domaine sont indépendantes du modèles de stockage et de l'ORM

Utilisation

```
<?php
```

```
// Création d'un objet métier et sauvegarde
```

```
$a = new Article() ;
```

```
$a->nom= 'hello' ;
```

```
$a->descr = 'bo velo bleu' ;
```

```
$mapper = new ArticleMapper() ;
```

```
$mapper->insert($a) ;
```

```
// mise à jour
```

```
$velo->tarif = 120 ;
```

```
$velo->calculerPrixTTC() ;
```

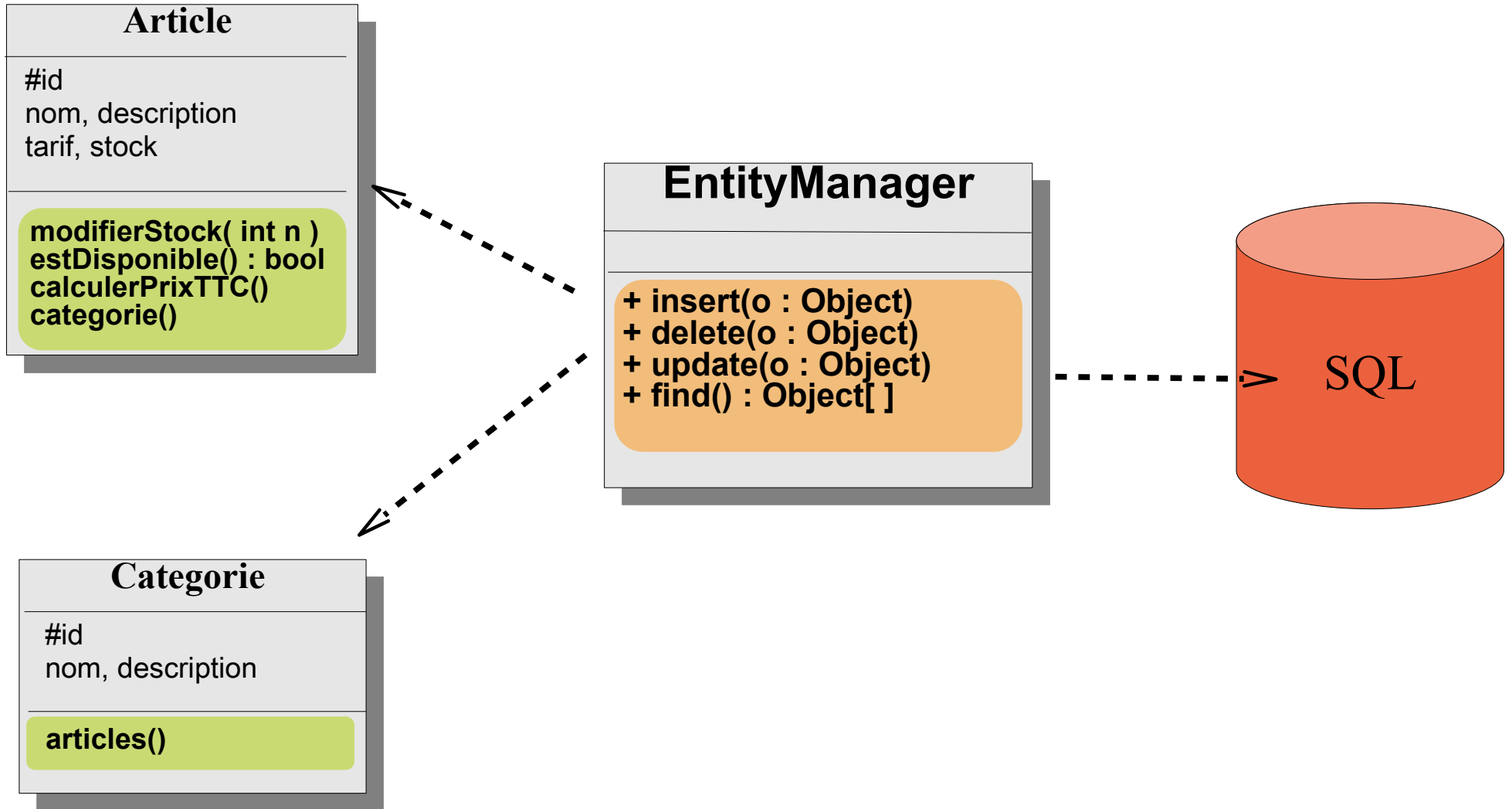
```
$mapper->update($velo);
```

```
// supprimer un objet
```

```
$mapper->delete($a) ;
```

EntityManager

- ***EntityManager***: Variante de DAO dans lequel l'objet faisant l'interface entre objets métiers et persistance est un **objet générique** et non plus un objet spécifique à chaque classe
- modèle utilisé par Hibernate et Doctrine 2

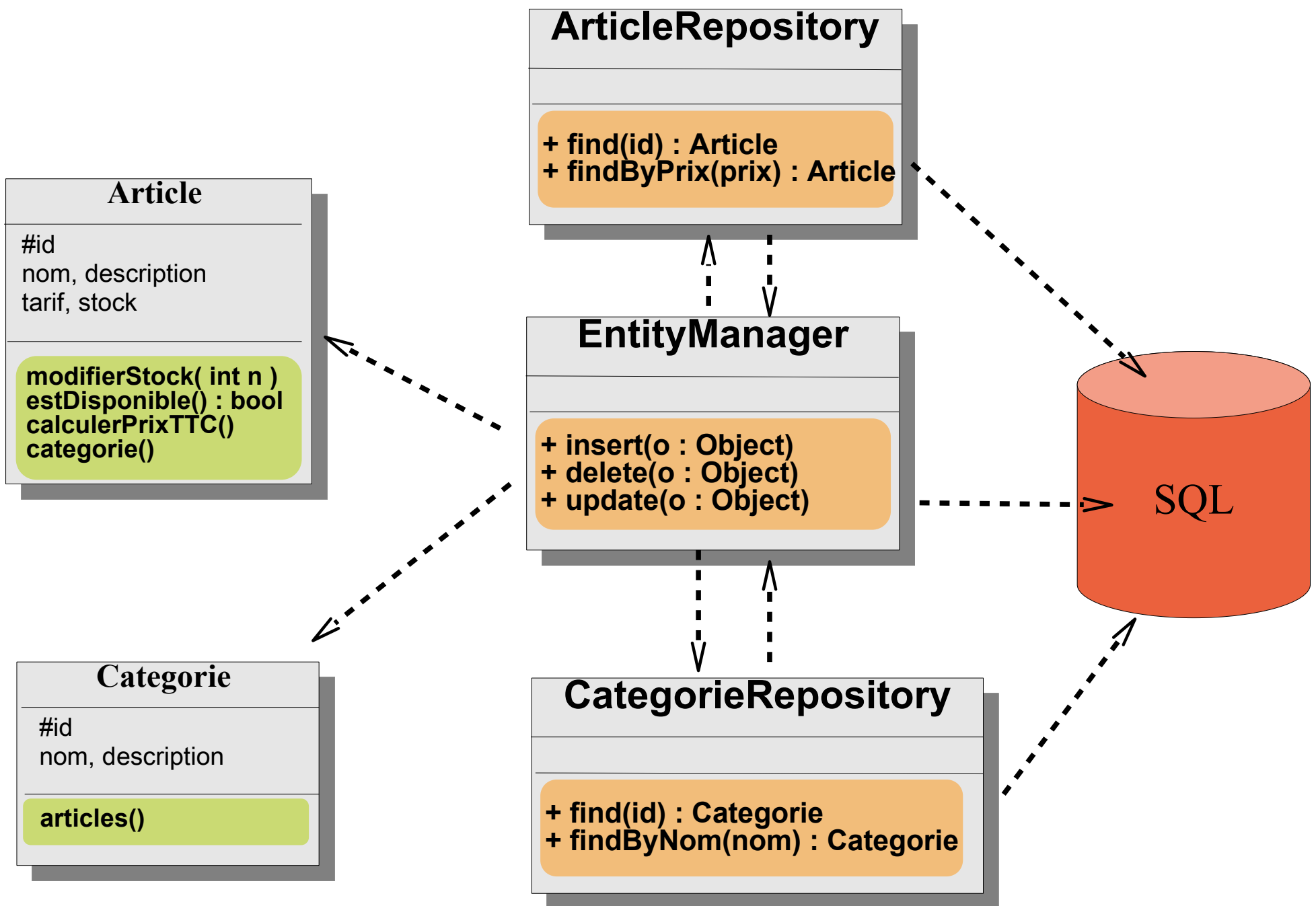


EntityManager

- Intérêts :
 - découplage Domaine - BD/ORM
 - localisation : un seul objet pour l'interface Entités - ORM

variant : EntityManager + Repository

- *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
- Le point d'entrée unique est l'EntityManager



exemple avec Doctrine 2

```
<?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 sur ID
$productRepository = $entityManager->getRepository('Article');
$products = $productRepository->find($product->id);

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

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

le mapping

- le mapper générique à besoin d'information pour transformer les entités en 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/annotations 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*)
- Evolutions facilitées du service de stockage
- Possibilité de gérer plusieurs services de stockage
- Un point d'entrée unique : injection facilitée dans les services métiers
- plus simple à utiliser pour le programmeur que DAO : pas besoin de définir les classes DAO
- les annotations ou fichiers de configuration peuvent être utilisés pour générer les tables

Utiliser Doctrine

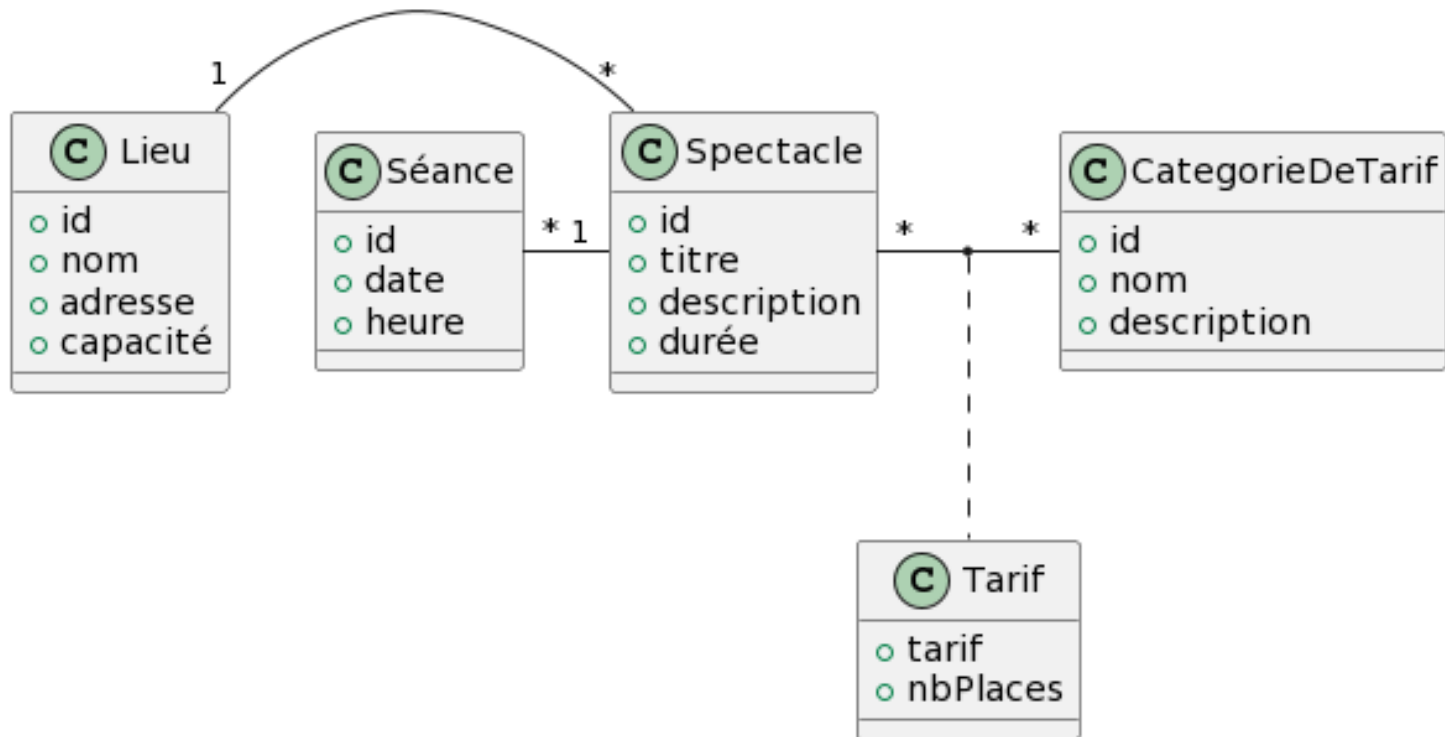
www.doctrine-project.org

- ORM PHP basé sur le pattern repository+entityManager
- Entity Manager : gestion du lien objets-table, synchronisation entités - base de données, accès aux repository
- Repository : requêtes sur la bases de données retournant des entités
- 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

```
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
chaîne de caractère de taille 64

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

Mapping des associations : 1 -- *

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

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

entité cible

clé étrangère, clé primaire
référéncée

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

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

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

association inverse

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

```
}
```

Configuration, création de l'entityManager

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

liste des répertoires
contenant les classes entité

```
$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
associé au type entité

```
$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

Requêtes à conditions simples

```
// 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

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

```
$entityManager->persist($s);
```

enregistre l'entité dans le manager pour la rendre persistante

```
$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);
```

associations

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

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

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

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

installation : avec composer

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