La couche métier

Plan :

1. Introduction à Doctrine
2. Configurer Doctrine
3. Les Entités
4. Requêtes avec Repository
5. Relations entre Entités
6. Migrations et Synchronisation
7. Requêtes Personnalisées
8. Conclusion

Introduction à Doctrine

•**Définition ?**

Doctrine est un ORM (Object Relational Mapper) qui facilite l'interaction entre des objets PHP et une base de données relationnelle.

•**Rôle dans Symfony** :

- ✓ Abstraction du SQL grâce à un modèle orienté objet.
- ✓ Simplification de la gestion des données.
- ✓ Automatisation du mapping entre les entités (classes PHP) et les tables de la base de données.

•**Principales fonctionnalités** :

- ✓ Création et gestion des entités.
- ✓ Requêtes simplifiées grâce à DQL (Doctrine Query Language).
- ✓ Gestion des relations complexes entre les entités.

Configuration de Doctrine

- Installation de Doctrine

composer require symfony/orm-pack

- Configuration de la base de données :

Modifiez le fichier .env pour définir les paramètres de connexion à votre base de données :

DATABASE_URL="mysql://db_user:db_password@127.0.0.1:3306/db_name"

- créer la BDD

php bin/console doctrine:database:create

Les entités avec Doctrine

- Définition
  - Une entité est une classe PHP qui représente une table dans une base de données.
  - Doctrine mappe automatiquement les attributs de la classe avec les colonnes de la table.
- Définir une Entité

Une classe d'entité doit :

1. Être annotée avec *#[ORM\Entity]*.
2. Contenir un identifiant unique (clé primaire) avec *#[ORM\Id]*.
3. Définir les colonnes avec *#[ORM\Column]*.

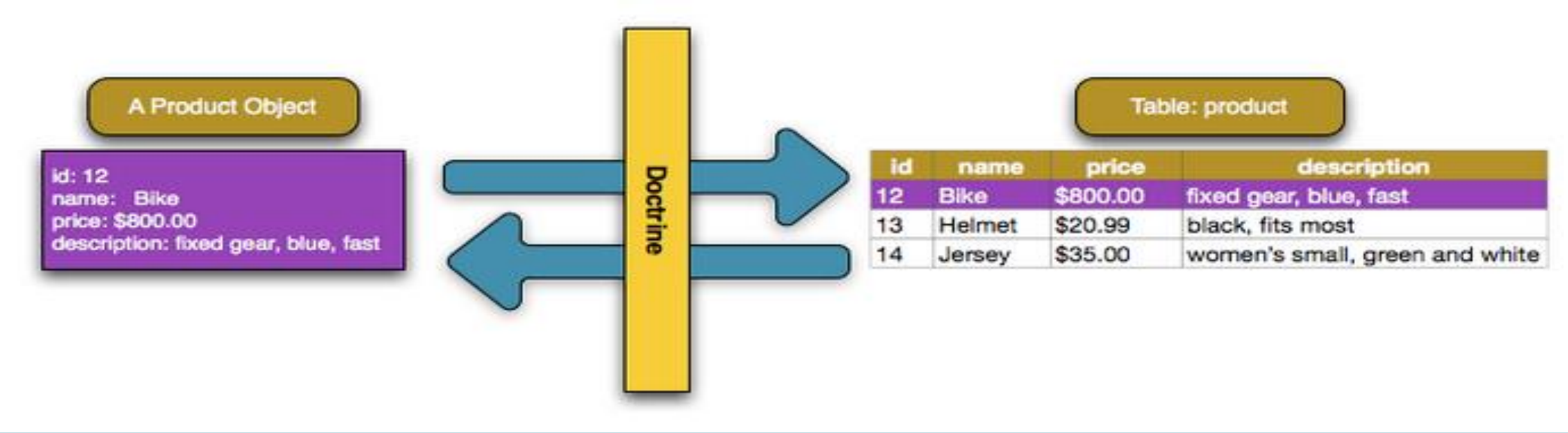
Les entités avec Doctrine

Exemple

```
#[ORM\Entity()]
class User {
    #[ORM\Id, ORM\GeneratedValue, ORM\Column(type: "integer")]
    private $id;

    #[ORM\Column(type: "string", length: 100)]
    private $name;

    #[ORM\Column(type: "string", unique: true)]
    private $email;

#[ORM\Column(type: "string", unique: true)]
    private $pswd;

private $confirmpswd;


}
```

La couche métier: Les entités

- Une entité est, du point de vue PHP, un simple objet.
- L'entité est une classe qui possède des attributs et se situe dans /Entity
- En symfony, l'entité est une classe PHP qui peut être mappée en une table de base de donnée à travers le service **Doctrine**



La couche métier: Les entités

ORM : Object-Relational Mapping

- Couche d'abstraction d'accès aux données
- Ne plus écrire de requêtes mais des objets
- Lazy loading (chargement paresseux)
- Décrire les relations entre objets
  - One-To-One
  - Many-To-One
  - Many-To-Many
  - Many-To-Many avec attributs
- Gestion personnalisée des accès si nécessaire

Classe PHP ↔ table du SGBD

Propriétés de l'instance ↔ colonnes de la table

Commandes Utiles

Création d'une entité:

php bin/console make:entity

Requêtes avec Repository

Qu’est-ce qu’un Repository ?

•Un repository est une classe qui gère la récupération des données depuis la base.

•Doctrine crée automatiquement un repository pour chaque entité si demandé.

•Permet d’abstraire les requêtes SQL avec des méthodes prêtes à l’emploi.

Utilisation de EntityManager

•L’EntityManager est responsable des opérations de persistance, comme :

- •Enregistrer une entité.
- •Supprimer une entité.
- •Mettre à jour une entité.

Requêtes avec Repository

Exemples :

- $users = $repository->findAll();
- $user = $repository->find($id);
- $users = $repository->findBy(['active' => true], ['name' => 'ASC']);

Relations entre Entités

•Les relations permettent de connecter plusieurs entités entre elles.

•Doctrine gère automatiquement ces relations via des annotations ou des attributs PHP (depuis PHP 8+).

•OneToOne:

```
#[ORM\OneToOne(mappedBy: "user", targetEntity: Profile::class)]
private $profile;
```

•OneToMany et ManyToOne

```
#[ORM\OneToMany(mappedBy: "user", targetEntity: Command::class)]
private $commands;
```

•ManyToMany

```
#[ORM\ManyToMany(targetEntity: Category::class, inversedBy: "products")]
private $categories;
```

Migration et synchronisation de schémas

- Une migration est un mécanisme pour appliquer ou modifier la structure de la base de données de manière contrôlée et visionnée.
- Permet de synchroniser la base de données avec les changements apportés aux entités.

Requêtes Personnalisées avec Doctrine

Pourquoi des Requêtes Personnalisées ?

•Les méthodes de base des repositories (find, findOneBy, etc.) peuvent ne pas suffire dans certains cas complexes.

•Doctrine permet d'écrire des requêtes personnalisées via DQL (Doctrine Query Language).

Exemples:

SELECT u FROM App\Entity\User u WHERE u.active = true

SELECT c FROM App\Entity\Command c JOIN c.user u WHERE u.id = :userId

Requêtes Personnalisées avec Doctrine

```
public function findUsersByRole(string $role) {
    return $this->createQueryBuilder('u')
        ->where('u.role = :role')
        ->setParameter('role', $role)
        ->getQuery()
        ->getResult();
}
```

L'objet entityManger

- **Définition**

L'***EntityManager*** est un composant clé de Doctrine, utilisé pour gérer la persistance des entités et interagir avec la base de données. Voici un aperçu de son fonctionnement avec les méthodes ***persist*** et ***flush***

```
use Doctrine\Persistence\ObjectManager;
…
public function MyFunction(ObjectManager $manager): …
{
…
}
```

L'objet entityManger

- **Rôle :** Il agit comme une interface entre vos entités et la base de données.
- **Responsabilités :**
  - Gérer le cycle de vie des entités (persistantes, détachées, supprimées, etc.).
  - Coordonner les opérations de persistance des entités (insertion, mise à jour, suppression).
  - Effectuer les transactions avec la base de données.

L'objet entityManger

La Méthode persist()

•**Description :** persist signale à l'EntityManager qu'une entité doit être suivie et préparée pour être ajoutée ou mise à jour dans la base de données.

•**Exemple :**

```
$user = new User();
$user->setName('Sofiene');
$user->setEmail('sofiene@example.com');

// Ajouter l'entité dans le cycle de gestion
$entityManager->persist($user);
```

L'objet entityManger

La Méthode Flush()

•**Description :** flush applique les modifications en attente (ajouts, mises à jour, suppressions) dans la base de données.

•**Exemple :**

```
// Exécuter les modifications persistées
$entityManager->flush();
```

•**Que fait** flush **?**

- •Envoie une ou plusieurs requêtes SQL à la base de données pour synchroniser les changements.
- •Peut traiter plusieurs entités en même temps, si plusieurs ont été marquées avec persist.

L'objet entityManger

Exemple complet:

```
$entityManager = $this->getDoctrine()->getManager();

// Création d'une nouvelle entité
$user = new User();
$user->setName('Sofiene');
$user->setEmail('sofiene@example.com');

// Marquer l'entité pour persistance
$entityManager->persist($user);

// Appliquer les changements dans la base de données
$entityManager->flush();
```

La couche métier: Les entités

Installation Doctrine

```
> composer require symfony/orm-pack
> composer require --dev symfony/maker-bundle
```

Configuration de la Base de données (.env)

```
# .env (or override DATABASE_URL in .env.local to avoid committing your changes)

# customize this line!
DATABASE_URL="mysql://db_user:db_password@127.0.0.1:3306/db_name"

# to use sqlite:
# DATABASE_URL="sqlite:///%kernel.project_dir%/var/app.db"
```

Création de la Base de données

```
> php bin/console doctrine:database:create
```

La couche métier: Les entités

Création d'une entité

```
C:\wamp64\www\firstapp>php bin/console make:entity

Class name of the entity to create or update (e.g. FierceJellybean):
> Produit

created: src/Entity/Produit.php
created: src/Repository/ProduitRepository.php

Entity generated! Now let's add some fields!
You can always add more fields later manually or by re-running this command.

New property name (press <return> to stop adding fields):
> id


[ERROR] The "id" property already exists.

New property name (press <return> to stop adding fields):
> nom

Field type (enter ? to see all types) [string]:
>

Field length [255]:
> 120

Can this field be null in the database (nullable) (yes/no) [no]:
> yes
```



```
updated: src/Entity/Produit.php
```

La couche métier: Les entités

Migration

- Il s'agit de tirer parti du Doctrine Migrations Bundle qui est déjà installé
- Vérification et préparation de l'environnement de gestion de Base de données selon la configuration préétablie
- La migration nécessite la création de la Base de donnée auparavant.

```
C:\wamp64\www\firstapp>php bin/console doctrine:database:create
Created database `firstapp` for connection named default

C:\wamp64\www\firstapp>php bin/console make:migration


  Success!


Next: Review the new migration "src/Migrations/Version20200519173750.php"
Then: Run the migration with php bin/console doctrine:migrations:migrate
See https://symfony.com/doc/current/bundles/DoctrineMigrationsBundle/index.html
```

La couche métier: Les entités

Exécuter les migrations

- Cette commande exécute tous les fichiers de migration qui n'ont pas encore été exécutés sur votre base de données.
- Vous devez exécuter cette commande sur la production lors de votre déploiement pour maintenir votre base de données de production à jour.

```
C:\wamp64\www\firstapp>php bin/console doctrine:migrations:migrate

                    Application Migrations


WARNING! You are about to execute a database migration that could result in schema changes and data loss. Are you sure you wish to continue? (y/n)y
Migrating up to 20200519173750 from 0

  ++ migrating 20200519173750

     -> CREATE TABLE produit (id INT AUTO_INCREMENT NOT NULL, nom VARCHAR(120) DEFAULT NULL, PRIMARY KEY(id)) DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci ENGINE = InnoDB

  ++ migrated (took 1302.1ms, used 18M memory)

  ------------------------

  ++ finished in 1390.6ms
  ++ used 18M memory
  ++ 1 migrations executed
  ++ 1 sql queries
```

La couche métier: Les entités

Mise à jour des entités

- En cas de mise des entités (attributs et méthodes) il devient indispensable de migrer cette modification ver la Base de données.

```
C:\wamp64\www\firstapp>php bin/console make:entity

Class name of the entity to create or update (e.g. GentleChef):
> Produit

Your entity already exists! So let's add some new fields!

New property name (press <return> to stop adding fields):
> description

Field type (enter ? to see all types) [string]:
>

Field length [255]:
>

Can this field be null in the database (nullable) (yes/no) [no]:
> yes

updated: src/Entity/Produit.php

Add another property? Enter the property name (or press <return> to stop adding fields):
>
```