

Programmation avancée & Symfony

BUT Informatique - 3ème année

TP 3

ORM Doctrine, BDD, Entity

Jeux vidéo, genres et éditeurs

© Claire BOMBEAUX

Version : 11/10/2025



Objectifs du TP 3

PROJET SYMFONY

- Installer et configurer l'ORM *Doctrine*
- Créer et configurer le schéma de base de données *MySQL*
- Créer les premières entités et leurs repositories (classes PHP) via le *Maker*, et gérer leurs relations
- A partir de ces éléments, créer la structure (les tables) de votre BDD automatiquement via le processus de *migration*
- Créer, insérer des données en BDD via les *Fixtures*

Un jeu vidéo est constitué de différentes propriétés (titre, développeur, éditeur, date de sortie, prix, description, genre, ...).

Un genre est constitué d'un nom et d'une description.

Un éditeur est représenté par un nom et d'une description, ainsi qu'un pays et son site web.

Un jeu vidéo ne peut avoir qu'un seul genre, et qu'un seul éditeur. Un genre peut être rattaché à 0 à n jeux vidéo. Un éditeur peut être rattaché à 0 à n jeux vidéo.



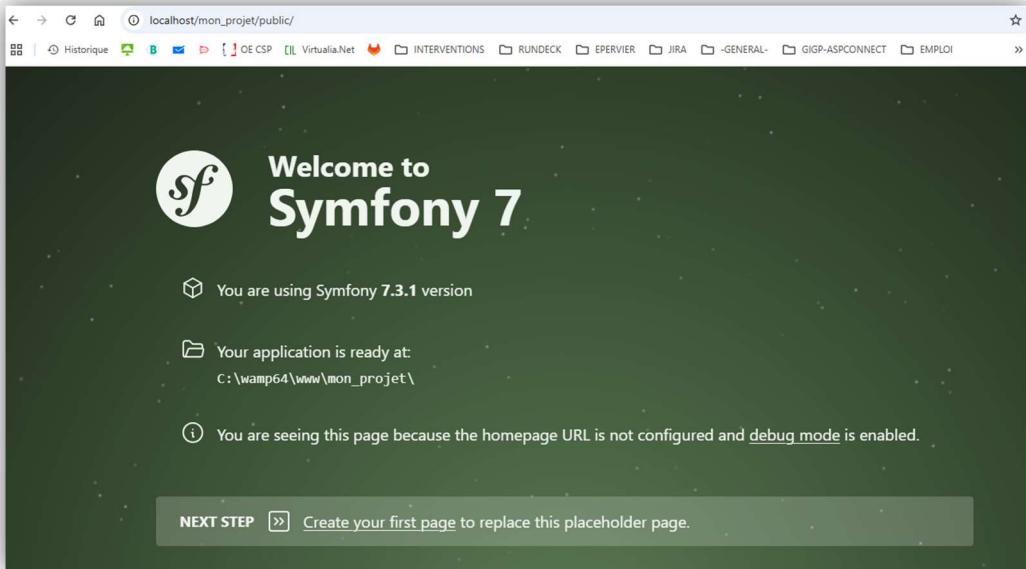
Sommaire

■ Objectifs du TP 3	0
■ Sommaire	1
1. Install. Base de données et ORM Doctrine	3
1.1 Chargement de l'ORM Doctrine.....	3
1.2 Configuration PHP & démarrage MySQL (via WampServer)	3
1.3 Configurer Doctrine	4
1.4 Création de la BDD MySQL.....	5
Méthode 1 : en ligne de commande (recommandé).....	5
Méthode 2 : via IHM (<i>phpMyAdmin</i>)	5
2. Création des entités	6
2.1 Prérequis et installation du MakerBundle de Symfony.....	6
Installation du <i>MakerBundle</i>	6
Autres prérequis :	7
2.2 Création des Entités avec le MakerBundle	7
Créer l'entité Genre et son Repository	7
<i>Doc./Info.</i> : Types possibles des champs d'une entité	8
Créer l'entité Editeur (et son Repository).....	9
<i>Doc./Info.</i> : Le ? devant le type d'une propriété.....	9
Créer l'entité JeuVideo (et son Repository).....	10
<i>Une petite parenthèse</i> : Sauvegarder régulièrement votre code !	13
3. Création des tables et données de la BDD.....	14
3.1 Génération et exécution des migrations	14
Vérifier l'ensemble des entités	14
Générer la migration.....	14
Exécuter la migration	15
Schéma de la BDD	15
3.2 Création des Fixtures (données de test)	16
Installer DoctrineFixturesBundle	16
Créer une classe de fixtures	16
Charger les fixtures	17
4. Récap.....	18

Structure finale attendue Symfony.....	18
Diagramme BDD.....	18
 <i>Pour aller plus loin</i>	19
[TP 3] Annexe1 : Commandes utiles	20
Création d'une BDD (<i>Doctrine</i>).....	20
Suppression d'une BDD (<i>Doctrine</i>).....	20
Vérifier le schéma (<i>Doctrine</i>)	20
Voir et vérifier le mapping des entités (<i>Doctrine</i>)	20
Création / Modification d'une entité (<i>Maker</i>)	20
Migrations (<i>Maker et Doctrine</i>)	20
Debug <i>Doctrine</i>	21

1. Install. Base de données et ORM Doctrine

Prérequis : Fin du TP 1



http://127.0.0.1/mon_projet/public/

1.1 Chargement de l'ORM Doctrine

A l'aide de *Composer*, charger le package de l'ORM Doctrine, via la commande d'install. :

```
cd mon_projet  
composer require symfony/orm-pack
```

```
PS C:\wamp64\www\mon-projet> composer require symfony/orm-pack  
.composer.json has been updated  
Running composer update symfony/orm-pack  
Loading composer repositories with package information  
Restricting packages listed in "symfony/symfony" to "5.4.*"  
Updating dependencies  
Lock file operations: 21 installs, 0 updates, 0 removals  
- Locking doctrine/annotations (2.0.2)  
- Locking doctrine/cache (2.2.0)
```

Documentation : <https://symfony.com/packages/ORM%20Pack>

1.2 Configuration PHP & démarrage MySQL (via WampServer)

Vérifier / activez les extensions PHP suivantes

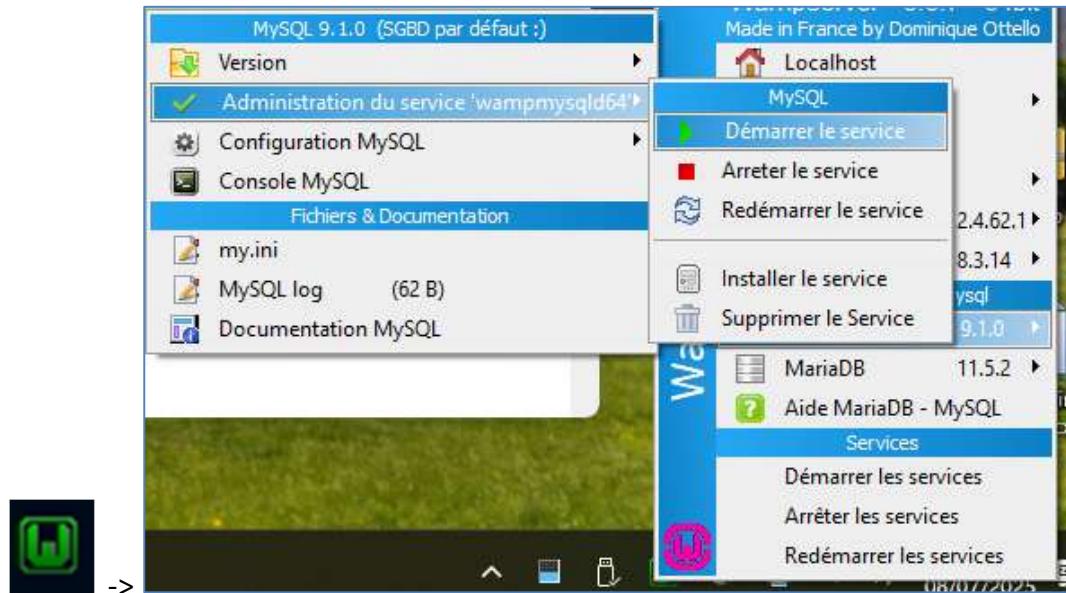
(Via WAMP -> PHP → Extensions PHP ou fichier *php.ini*)

```
extension=pdo_mysql // php_pdo_mysql  
extension=mysqli // php_mysqli
```

ini

+ Redémarrer WAMP

Vérifier que MySQL est démarré, sinon le faire :



1.3 Configurer Doctrine

Rajouter les éléments manquants dans le fichier [mon-projet /config/packages/doctrine.yaml](#)

Attention, le YAML utilise une structure basée sur le respect des indentations (*espaces*).

[Exemple yaml](#)

```
doctrine:  
    dbal:  
        driver: 'pdo_mysql'  
        server_version: '8.0'  
        charset: utf8mb4  
  
        orm:  
            auto_generate_proxy_classes: true  
            naming_strategy: doctrine.orm.naming_strategy.underscore_number_aware  
            auto_mapping: true
```

Dans le fichier `.env` (situé à la racine) de votre projet Symfony, configurez la connexion à la BDD MySQL :

[Exemple .env](#)

```
# mon-projet/.env  
# Exemples :  
  
DATABASE_URL="mysql://root:@127.0.0.1:3306/bdd_mon_projet"
```

```

DATABASE_URL="mysql://root:@127.0.0.1:3306/bdd_mon_projet?serverVersion=8&charset=utf8mb4"

DATABASE_URL="postgresql://app:!ChangeMe!@127.0.0.1:5432/app?serverVersion=16& charset=utf8"

DATABASE_URL="sqlite:///%kernel.project_dir%/var/data_%kernel.environment%.db"

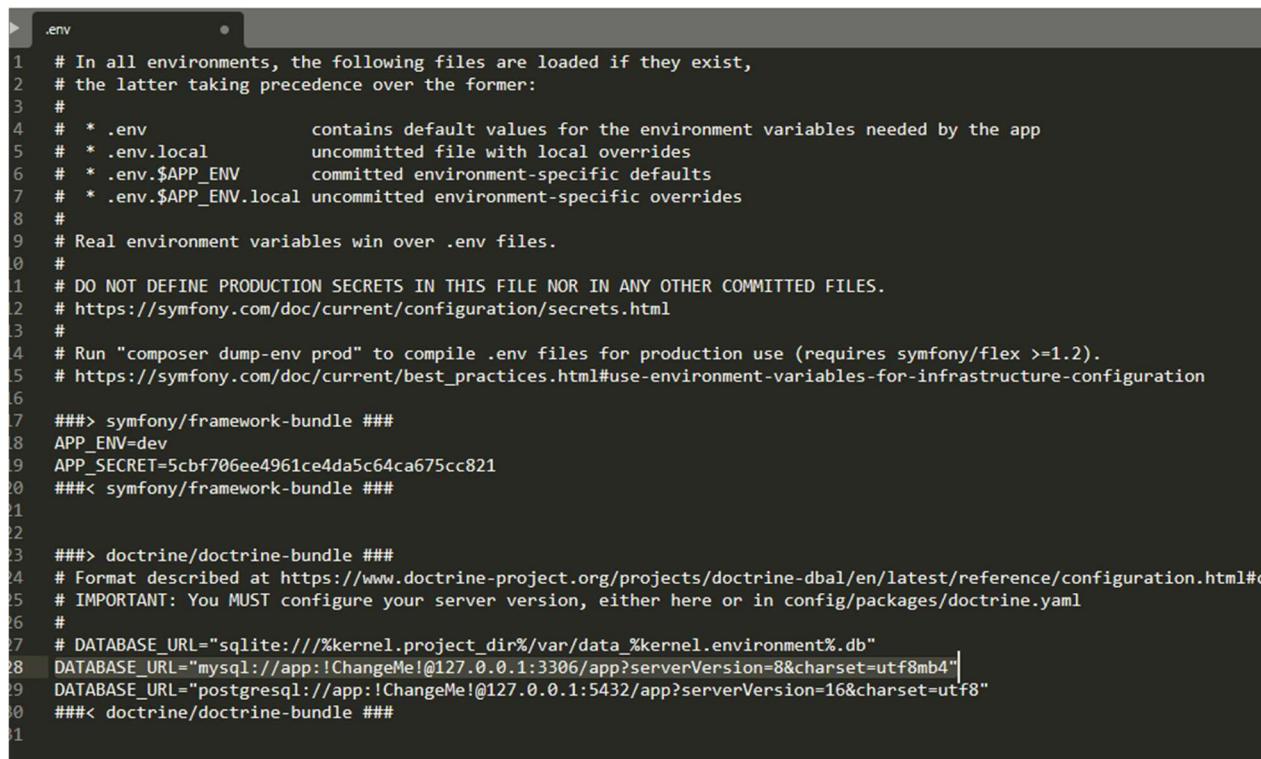
```

SGBD : la syntaxe de la chaîne de connexion dépend du SGBD utilisé

Nom de la BDD

Login

Mot de passe



```

.env
1 # In all environments, the following files are loaded if they exist,
2 # the latter taking precedence over the former:
3 #
4 # * .env           contains default values for the environment variables needed by the app
5 # * .env.local     uncommitted file with local overrides
6 # * .env.$APP_ENV   committed environment-specific defaults
7 # * .env.$APP_ENV.local uncommitted environment-specific overrides
8 #
9 # Real environment variables win over .env files.
10 #
11 # DO NOT DEFINE PRODUCTION SECRETS IN THIS FILE NOR IN ANY OTHER COMMITTED FILES.
12 # https://symfony.com/doc/current/configuration/secrets.html
13 #
14 # Run "composer dump-env prod" to compile .env files for production use (requires symfony/flex >=1.2).
15 # https://symfony.com/doc/current/best_practices.html#use-environment-variables-for-infrastructure-configuration
16 #
17 ##> symfony/framework-bundle ##
18 APP_ENV=dev
19 APP_SECRET=5cbf706ee4961ce4da5c64ca675cc821
20 ##< symfony/framework-bundle ##
21 #
22 #
23 ##> doctrine/doctrine-bundle ##
24 # Format described at https://www.doctrine-project.org/projects/doctrine-dbal/en/latest/reference/configuration.html#connection-settings
25 # IMPORTANT: You MUST configure your server version, either here or in config/packages/doctrine.yaml
26 #
27 # DATABASE_URL="sqlite:///%kernel.project_dir%/var/data_%kernel.environment%.db"
28 DATABASE_URL="mysql://app:!ChangeMe!@127.0.0.1:3306/app?serverVersion=8&charset=utf8mb4"
29 DATABASE_URL="postgresql://app:!ChangeMe!@127.0.0.1:5432/app?serverVersion=16&charset=utf8"
30 ##< doctrine/doctrine-bundle ##
31

```

1.4 Crédation de la BDD MySQL

Méthode 1 : en ligne de commande (recommandé)

```

cd mon_projet
php bin/console doctrine :database :create

```

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

Méthode 2 : via IHM (*phpMyAdmin*)

Via l'IHM de *PhpMyAdmin*.

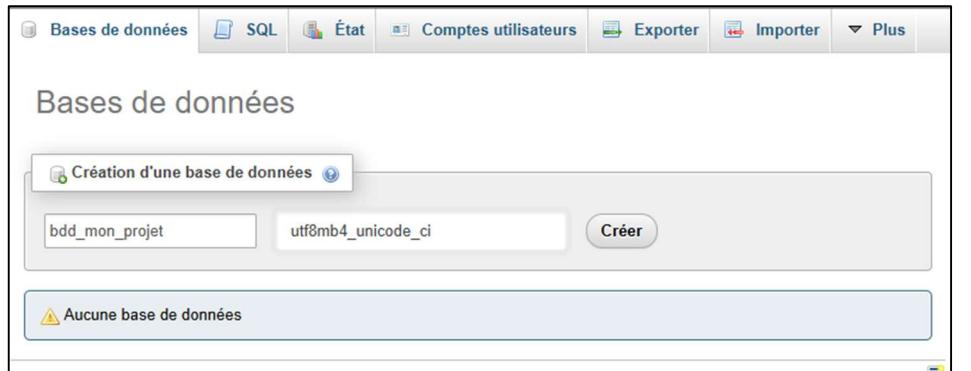


1ere connexion : login par défaut : *root* + mdp vide

Encodage BDD conseillé :

`utf8mb4_unicode_ci`

Pourquoi cet encodage ? => _____



2. Création des entités

2.1 Prérequis et installation du MakerBundle de Symfony

Documentation : <https://symfony.com/bundles/SymfonyMakerBundle/current/index.html>

Installation du *MakerBundle*

Exécuter cette commande pour installer et activer ce bundle dans votre application :

```
cd mon_projet
composer require --dev symfony/maker-bundle
```

```
C:\wamp64\www\mon_projet>composer require --dev symfony/maker-bundle
./composer.json has been updated
Running composer update symfony/maker-bundle
Loading composer repositories with package information
Restricting packages listed in "symfony/symfony" to "7.3.*"
Updating dependencies
Lock file operations: 3 installs, 0 updates, 0 removals
- Locking nikic/php-parser (v5.5.0)
- Locking symfony/maker-bundle (v1.64.0)
- Locking symfony/process (v7.3.0)
Writing lock file
Installing dependencies from lock file (including require-dev)
Package operations: 3 installs, 0 updates, 0 removals
- Downloading symfony/process (v7.3.0)
- Downloading nikic/php-parser (v5.5.0)
- Downloading symfony/maker-bundle (v1.64.0)
- Installing symfony/process (v7.3.0): Extracting archive
- Installing nikic/php-parser (v5.5.0): Extracting archive
- Installing symfony/maker-bundle (v1.64.0): Extracting archive
Generating autoload files
43 packages you are using are looking for funding.
Use the `composer fund` command to find out more!

Symfony operations: 1 recipe (00383a0117e07c9e5d3e04331cffec3b)
- Configuring symfony/maker-bundle (>=1.0): From github.com/symfony/recipes:main
Executing script cache:clear [OK]
Executing script assets:install public [OK]

What's next?
```

Some files have been created and/or updated to configure your new packages.
Please review, edit and commit them: these files are yours.

No security vulnerability advisories found.
Using version ^1.64 for symfony/maker-bundle

Autres prérequis :

- La configuration de la base de données doit avoir été réalisé, dans le fichier .env
- La base de données doit avoir été créée

2.2 Création des Entités avec le MakerBundle

Créer l'entité Genre et son Repository

```
php bin/console make:entity Genre
```

💡 Rappels : Un Repository dans Symfony est une classe qui sert d'intermédiaire entre votre code métier et la base de données. C'est un concept issu du pattern "Repository" de Domain-Driven Design (DDD).

Les Repositories permettent de **centraliser les requêtes de BDD** (regroupent requêtes SQL ou DQL) et donc d'isoler toute la logique d'accès aux données des contrôleurs.

Propriétés à ajouter :

- nom : string, 100, not null

```
PS C:\wamp64\www\mon_projet> php bin/console make:entity Genre
  created: src/Entity/Genre.php
  created: src/Repository/GenreRepository.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):
  > nom

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

  Field length [255]:
  > 100

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

- description : text, nullable

- actif : boolean, default true

```
private ?bool $actif = true;
```

Car + tard, il devra être possible de désactiver un genre

- createdAt : datetime_immutable, not null

- updatedAt : datetime_immutable, nullable

⚠ Une bonne pratique consiste à ajouter les champs createdAt et updatedAt dans chaque Entity, afin de pouvoir enregistrer les dates de création et de modification de chaque éléments (enregistrements en BDD). Ils sont donc attendus dans toutes les tables de votre BDD.

L'entité *Genre* et son Repository *GenreRepository* doivent avoir été créées :

The screenshot shows two code editors side-by-side. The left editor contains `Genre.php` with the following content:

```
1 <?php
2
3 namespace App\Entity;
4
5 use App\Repository\GenreRepository;
6 use Doctrine\DBAL\Types\Types;
7 use Doctrine\ORM\Mapping as ORM;
8
9 #[ORM\Entity(repositoryClass: GenreRepository::class)]
10 class Genre
11 {
12     #[ORM\Id]
13     #[ORM\GeneratedValue]
14     #[ORM\Column]
15     private ?int $id = null;
16
17     #[ORM\Column(length: 100)]
18     private ?string $nom = null;
19
20     #[ORM\Column(type: Types::TEXT, nullable: true)]
21     private ?string $description = null;
22
23     #[ORM\Column]
24     private ?bool $actif = null;
25
26     public function getId(): ?int
27     {
28         return $this->id;
29     }
30
31     public function getNom(): ?string
32     {
```

The right editor contains `GenreRepository.php` with the following content:

```
1 <?php
2
3 namespace App\Repository;
4
5 use App\Entity\Genre;
6 use Doctrine\Bundle\DoctrineBundle\Repository\ServiceEntityRepository;
7 use Doctrine\Persistence\ManagerRegistry;
8
9 /**
10  * @extends ServiceEntityRepository<Genre>
11 */
12 class GenreRepository extends ServiceEntityRepository
13 {
14     public function __construct(ManagerRegistry $registry)
15     {
16         parent::__construct($registry, Genre::class);
17     }
18
19     /**
20      * @return Genre[] Returns an array of Genre objects
21      */
22     public function findByExampleField(?string $value): array
23     {
24         return $this->createQueryBuilder('g')
25             ->andWhere('g.exampleField = :val')
26             ->setParameter('val', $value)
27             ->orderBy('g.id', 'ASC')
28             ->setMaxResults(10)
29             ->getQuery()
30             ->getResult()
31     }
```

Doc./Info. : Types possibles des champs d'une entité

```
Main Types
* string or ascii_string
* text
* boolean
* integer or smallint or bigint
* float

Relationships/Associations
* relation a wizard will help you build the relation
* ManyToOne
* OneToMany
* ManyToMany
* OneToOne

Array/Object Types
* array or simple_array
* json
* object
* binary
* blob

Date/Time Types
* datetime or datetime_immutable
* datetimetz or datetimetz_immutable
* date or date_immutable
* time or time_immutable
* dateinterval

Other Types
* enum
* decimal
* guid
```

Créer l'entité Editeur (et son Repository)

Vous pouvez, comme précédemment, utiliser le *MakerBundle* pour créer cette Entity et son Repository, via la commande :

```
php bin/console make:entity Editeur
```

Ou bien vous pouvez également créer et coder les fichiers `Editeur.php` et `EditeurRepository.php` à la main.

Propriétés à ajouter :

- nom : string, 150, not null
- pays : string, 100, nullable
- siteWeb : string, 255, nullable
- description : text, nullable
- (Et `createdAt` + `updatedAt`)

Doc./Info. : Le ? devant le type d'une propriété

En PHP (et donc dans *Symfony*), le `?` devant un type comme `int` indique que la propriété peut être *nullable*, c'est-à-dire qu'elle peut contenir soit une valeur de type X (`int` dans l'exemple), soit `null`.

php

```
private ?int $id = null;
```

Cela signifie que :

- `$id` peut contenir un entier (`int`)
- `$id` peut aussi valoir `null`
- Par défaut, initialisé à `null`

Sans le `?`, si vous essayiez d'assigner `null` à une propriété typée `int`, PHP lèverait une erreur de type.

Exemples :

Php

```
private ?int $id = null;
        $this->id = 42;           // OK
        $this->id = null;         // OK

private int $id;
        $this->id = 42;           // OK
        $this->id = null;         // KO : Erreur TypeError
```

C'est particulièrement utile pour les entités Doctrine où l'ID est souvent auto-généré et donc null avant la première sauvegarde en BDD.

Créer l'entité JeuVideo (et son Repository)

```
php bin/console make:entity JeuVideo
```

Notez que cette commande permet la création d'entités, mais également leurs modifications si celles existent déjà.

Propriétés à ajouter :

- titre : Nom du jeu. String, 255 caractères max. Not null
- developpeur : string, 255, not null
- dateSortie : date, nullable
- prix : decimal, precision 6, scale 2, nullable (*le prix sera en euros*)

Precision = nombre de chiffre. *Scale* = nombre de chiffres après la virgule

- description : text, nullable
- imageUrl : string , 255 nullable

Relations à ajouter :

- **éditeur** : relation ManyToOne vers Editeur, not null

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

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

What class should this entity be related to?:
> Editeur
```

Nous allons créer une relation `ManyToOne` puisque chaque jeu vidéo a un seul éditeur et un éditeur peut éditer plusieurs jeux vidéo.

Entrez donc le bon type de relation (`ManyToOne`) dans votre terminal. Le Maker vous demande si cette propriété peut être nulle, répondez non. Il vous demande ensuite si vous souhaitez ajouter une propriété dans Editeur pour accéder aux objets JeuVideo qui lui sont liés.

Répondez oui. Il vous demande ensuite le nom que doit prendre cette propriété, en vous proposant `jeuVideos` par défaut, ce qui nous convient.

```
What type of relationship is this?
```

Type	Description
ManyToOne	Each JeuVideo relates to (has) one Editeur. Each Editeur can relate to (can have) many JeuVideo objects.
OneToMany	Each JeuVideo can relate to (can have) many Editeur objects. Each Editeur relates to (has) one JeuVideo.
ManyToMany	Each JeuVideo can relate to (can have) many Editeur objects. Each Editeur can also relate to (can also have) many JeuVideo objects.
OneToOne	Each JeuVideo relates to (has) exactly one Editeur. Each Editeur also relates to (has) exactly one JeuVideo.

```
Relation type? [ManyToOne, OneToMany, ManyToMany, OneToOne]:
```

```
> ManyToOne
```

```
Is the JeuVideo.editeur property allowed to be null (nullable)? (yes/no) [yes]:
```

```
> no
```

```
Do you want to add a new property to Editeur so that you can access/update JeuVideo objects from it - e.g. $editeur->getJeuVideos()? (yes/no) [yes]:
```

```
> yes
```

```
A new property will also be added to the Editeur class so that you can access the related JeuVideo objects from it.
```

```
New field name inside Editeur [jeuVideos]:
```

```
> |
```

On vous demande ensuite si vous souhaitez activer `orphanRemoval` sur cette relation, en précisant qu'un objet JeuVideo sera orphelin quand il est retiré de son Editeur, en d'autres termes, quand la relation JeuVideo <-> Editeur est détruite.

La distinction est importante. Lorsque vous appellerez `$editor->removeJeuVideo($jeuVideo)` pour retirer un jeuVideo de son éditeur, vous ne ferez que supprimer *la relation entre ces deux objets*. Le jeuVideo ne sera pas retiré de la base de données pour autant !

Et c'est là l'enjeu de cette question : voulez-vous automatiquement retirer de la base de données un jeu video que vous avez dissocié de son éditeur, c'est-à-dire "voulez-vous autoriser un jeu video à ne pas avoir d'éditeur ?". Si la réponse est non, activez l'`orphanRemoval`. Ici, nous ne l'activerons pas.

- `genre` : relation ManyToOne vers `Genre`

Procédez de la même façon que nous l'avons fait pour `Editeur`.

```

PS C:\wamp64\www\mon_projet> php bin/console make:entity JeuVideo
Your entity already exists! So let's add some new fields!

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

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

What class should this entity be related to?:
> Genre

Relation type? [ManyToOne, OneToMany, ManyToMany, OneToOne]:
> ManyToOne

Is the JeuVideo.genre property allowed to be null (nullable)? (yes/no) [yes]:
> no

Do you want to add a new property to Genre so that you can access/update JeuVideo objects from it - e.g. $genre->getJeuVideos()? (yes/no) [yes]:
>

A new property will also be added to the Genre class so that you can access the related JeuVideo objects from it.

New field name inside Genre [jeuVideos]:
>

Do you want to activate orphanRemoval on your relationship?
A JeuVideo is "orphaned" when it is removed from its related Genre.
e.g. $genre->removeJeuVideo($jeuVideo)

NOTE: If a JeuVideo may *change* from one Genre to another, answer "no".

```

Nous avons terminé de définir les deux relations, ce doit donner un résultat similaire à :

```

class JeuVideo
{
    # [ORM\Id]
    # [ORM\GeneratedValue]
    # [ORM\Column]
    private ?int $id = null;

    ...
    # [ORM\ManyToOne(inversedBy: 'jeuVideos')]
    # [ORM\JoinColumn(nullable: false)]
    private ?Editeur $editeur = null;

    # [ORM\ManyToOne(inversedBy: 'jeuVideos')]
    # [ORM\JoinColumn(nullable: false)]
    private ?Genre $genre = null;

    ...
}

```

```

class Editeur
{
    ...
}

```

```

    /**
     * @var Collection<int, JeuVideo>
     */
    #[ORM\OneToMany(targetEntity: JeuVideo::class, mappedBy: 'editeur')]
    private Collection $jeuVideos;
    ...
}

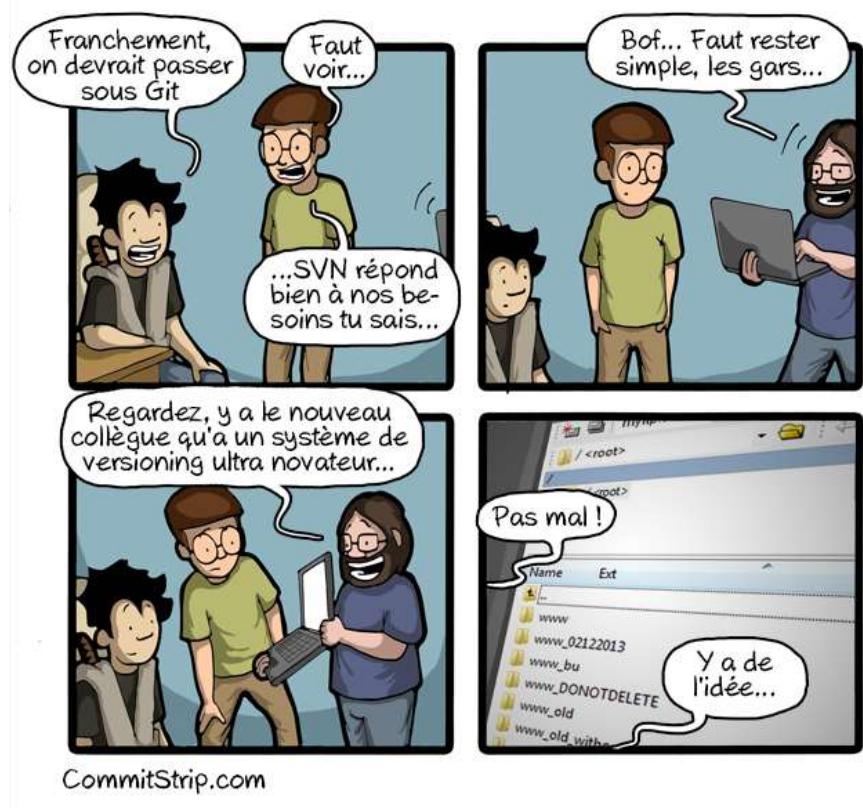
```

```

class Genre
{
    ...
    /**
     * @var Collection<int, JeuVideo>
     */
    #[ORM\OneToMany(targetEntity: JeuVideo::class, mappedBy: 'genre')]
    private Collection $jeuVideos;
    ...
}

```

Une petite parenthèse :
Sauvegarder
régulièrement votre code !



3. Création des tables et données de la BDD

3.1 Génération et exécution des migrations

Le processus de **migration** est ce qui vous permet passer de vos entités Symfony à votre BDD, via la génération automatique fichiers contenant les requêtes SQL de création de votre architecture BDD.

Vérifier l'ensemble des entités

```
php bin/console doctrine:mapping:info
```

```
C:\wamp64\www\mon_projet>php bin/console doctrine:mapping:info
Found 8 mapped entities:

[OK] App\Entity\JeuVideo
[OK] App\Entity\Editeur
[OK] App\Entity\Genre
```

Si tout est OK, vous êtes prêts à générer le SQL de votre base de données d'après nos entités créées.

Générer la migration

```
php bin/console make:migration
```

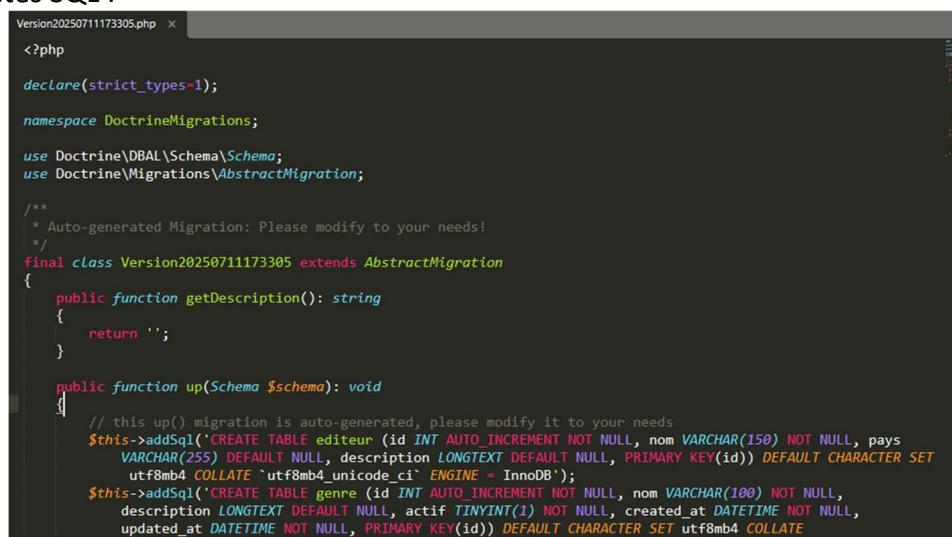
```
PS C:\wamp64\www\mon_projet> php bin/console make:migration
created: migrations/Version20250711173305.php

Success!
```

Remarque : *Maker fait appel à Doctrine.*

Equivalent Doctrine : `symfony console doctrine:migrations:diff`

Ouvrir le fichier créé dans `migrations/` et vérifier que les tables seront correctement créées via les requêtes SQL :



```
<?php

declare(strict_types=1);

namespace DoctrineMigrations;

use Doctrine\DBAL\Schema\Schema;
use Doctrine\Migrations\AbstractMigration;

/**
 * Auto-generated Migration: Please modify to your needs!
 */
final class Version20250711173305 extends AbstractMigration
{
    public function getDescription(): string
    {
        return '';
    }

    public function up(Schema $schema): void
    {
        // this up() migration is auto-generated, please modify it to your needs
        $this->addSql('CREATE TABLE editeur (id INT AUTO_INCREMENT NOT NULL, nom VARCHAR(150) NOT NULL, pays VARCHAR(255) DEFAULT NULL, description LONGTEXT DEFAULT NULL, PRIMARY KEY(id)) DEFAULT CHARACTER SET utf8mb4 COLLATE `utf8mb4_unicode_ci` ENGINE = InnoDB');
        $this->addSql('CREATE TABLE genre (id INT AUTO_INCREMENT NOT NULL, nom VARCHAR(100) NOT NULL, description LONGTEXT DEFAULT NULL, actif TINYINT(1) NOT NULL, created_at DATETIME NOT NULL, updated_at DATETIME NOT NULL, PRIMARY KEY(id)) DEFAULT CHARACTER SET utf8mb4 COLLATE `utf8mb4_unicode_ci` ENGINE = InnoDB');
    }

    public function down(Schema $schema): void
    {
        // this down() migration is auto-generated, please modify it to your needs
        $this->dropTable('editeur');
        $this->dropTable('genre');
    }
}
```

Exécuter la migration

Si tout vous semble correct, vous pouvez alors exécuter la migration, ce qui appliquera les requêtes sur notre BDD.

```
php bin/console doctrine:migrations:migrate
```

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

WARNING! You are about to execute a migration in database "bdd_mon_projet" that could result
in schema changes and data loss. Are you sure you wish to continue? (yes/no) [yes]:
> yes

[notice] Migrating up to DoctrineMigrations\Version20250711173305
[notice] finished in 150.3ms, used 16M memory, 1 migrations executed, 7 sql queries

[OK] Successfully migrated to version: DoctrineMigrations\Version20250711173305
```

Utiliser *phpMyAdmin* pour visualiser vos tables de BDD. Vérifier l'ensemble (structure : clés primaires et secondaires, auto-increment sur les id, ...)

Si vous constatez des erreurs majeures, il convient de rectifier côté Entities et côté BDD (manuellement ou via le *Maker + Migration*).

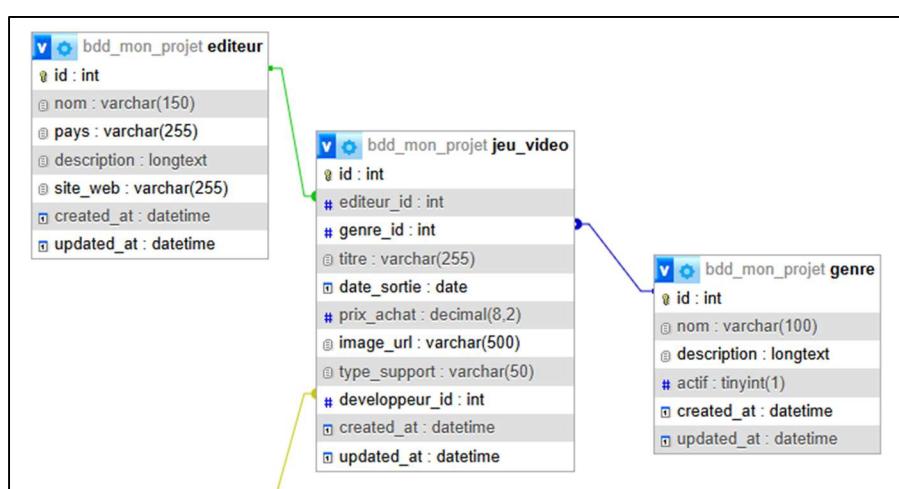
Noter que le processus de migration utilise une **table en BDD. Documentez-vous son rôle.**

Nom de la table => _____

Rôles/Fonctions et fonctionnement => _____

Schéma de la BDD

Utiliser le *Concepteur de phpMyAdmin* pour obtenir le diagramme de la BDD :



Profitez-en pour vous familiariser avec ce *concepteur* si nécessaire (Notez qu'il convient d'enregistrer ses modifications : 

☛ Pensez à exporter le diagramme, pour votre futur dossier de TP

 **Attention :** après chaque modification d'entités, il faudra générer le fichier de migration, et appliquer les modifications sur la base de données.

3.2 Création des Fixtures (données de test)

DoctrineFixturesBundle est un bundle Symfony qui permet de charger des données de test ou d'initialisation dans votre base de données de manière programmatique et reproductible. Il est utile pour initialiser la base de données, charger des données de test et faciliter la création d'un environnement de développement.

Installer DoctrineFixturesBundle

```
composer require --dev doctrine/doctrine-fixtures-bundle
```

Créer une classe de fixtures

```
php bin/console make:fixtures AppFixtures
```

Exemples de fixtures :

src/DataFixtures/AppFixtures.php

```
<?php

namespace App\DataFixtures;

use App\Entity\Genre;
use App\Entity\Editeur;
use App\Entity\JeuVideo;
use App\Entity\JeuVideoPs;

use Doctrine\Bundle\FixturesBundle\Fixture;
use Doctrine\Persistence\ObjectManager;

class AppFixtures extends Fixture
{
    public function load(ObjectManager $manager): void
    {
        // Créer les genres

        // Genre ACTION : Jeux d'action, combat
        $genreAction = new Genre();
        $genreAction->setNom('Action');
        $genreAction->setDescription('Jeux d\'action, de combat');
        $manager->persist($genreAction);
        // ...

        // Créer les éditeurs
```

```

$editeurSony = new Editeur();
$editeurSony->setNom('Sony Interactive Entertainment');
$editeurSony->setPays('Japon');
$editeurSony->setSiteWeb('https://www.sie.com');
$manager->persist($editeurSony);
// ...

// Créer un jeu exemple
$jeuVideo = new JeuVideo();
$jeuVideo->setTitre('The Last of Us Part II');
$jeuVideo->setEditeur($editeurSony);
$jeuVideo->setGenre($genreAction);
$jeuVideo->setDeveloppeur('Naughty Dog');
$jeuVideo->setDateSortie(new \DateTime('2020-06-19'));
$jeuVideo->setPrix(59.99);
$manager->persist($jeuVideo);

$manager->flush();
}

}

```

Notez que si vous appelez le constructeur d'une classe (`new Genre()`), il convient de l'inclure (`use App\Entity\Genre;`)

Insérer des **éditeurs** et des **développeurs** de jeux vidéo.

Insérer plusieurs **jeux vidéo** de différents genres.

Les **genres** de Jeux vidéo attendus sont :

Genres de jeux vidéo

- ACTION : Jeux d'action : jeux de plateforme, combat, tir (FPS, TPS, ...)
- AVENTURE : Jeux d'aventure narrative, point and click...
- ACTION_AVENTURE : (Infiltration, survival, ...)
- RPG : Jeux de rôle, MMORPG, ...
- STRATEGIE : Jeux de stratégie (RTS, turn-based)
- SIMULATION : Jeux de simulation, de gestion
- SPORT : Jeux de sport
- COURSE : jeux de course par ex. automobile
- REFLEXION : Jeux de réflexion, puzzles, casse-tête

Ceux-ci sont donc également à créer en BDD (manuellement ou à inclure dans vos Fixtures).

Charger les fixtures

```
php bin/console doctrine:fixtures:load
```

 **Attention** : le chargement de fixtures peut provoquer la purge les données existantes en BDD.

Si tout va bien :

```
C:\wamp64\www\mon_projet>php bin/console doctrine:fixtures:load
Careful, database "bdd_mon_projet" will be purged. Do you want to continue? (yes/no) [no]:
> yes
> purging database
> loading App\DataFixtures\AppFixtures
```

Sinon c'est le moment de corriger d'éventuelles erreurs....

Utiliser *phpMyAdmin* pour visualiser les données de vos tables en BDD.

Si vous constatez des erreurs majeures, il convient de les rectifier (manuellement direct en BDD ou via AppFixtures. Par contre si vous ne le faites qu'en BDD, et que vous réimportez vos fixtures, vous perdrez vos corrections.

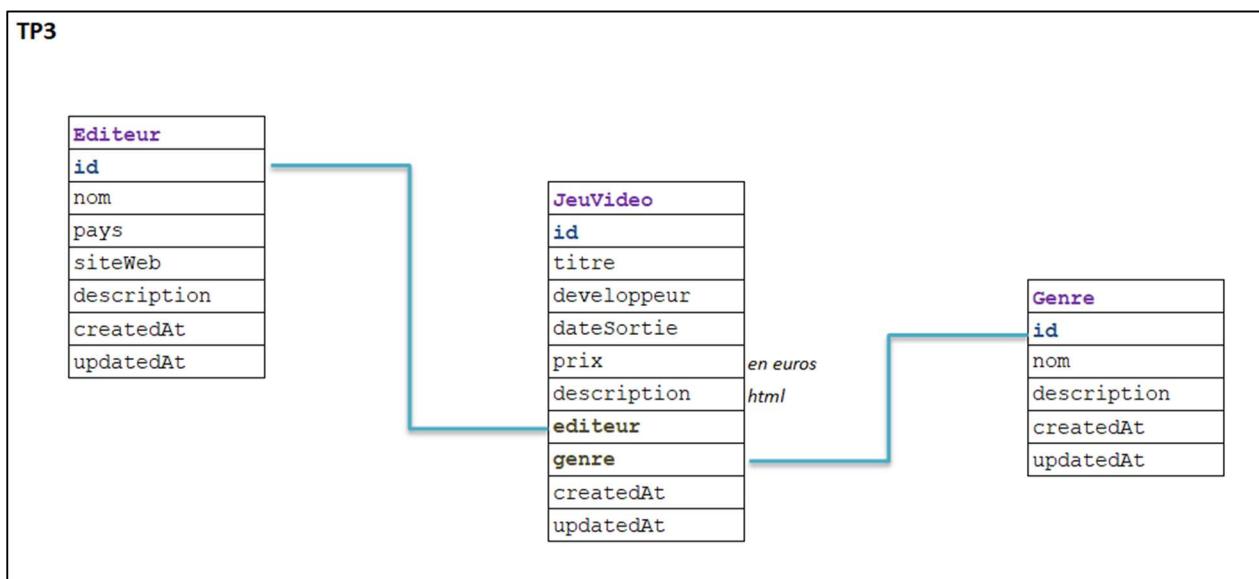
4. Récap.

Structure finale attendue Symfony

```
src/
└── Entity/
    ├── Genre.php
    ├── Editeur.php
    └── JeuVideo.php
└── Repository/
    ├── GenreRepository.php
    ├── EditeurRepository.php
    └── JeuVideoRepository.php
└── DataFixtures/
    └── AppFixtures.php
```

Diagramme BDD

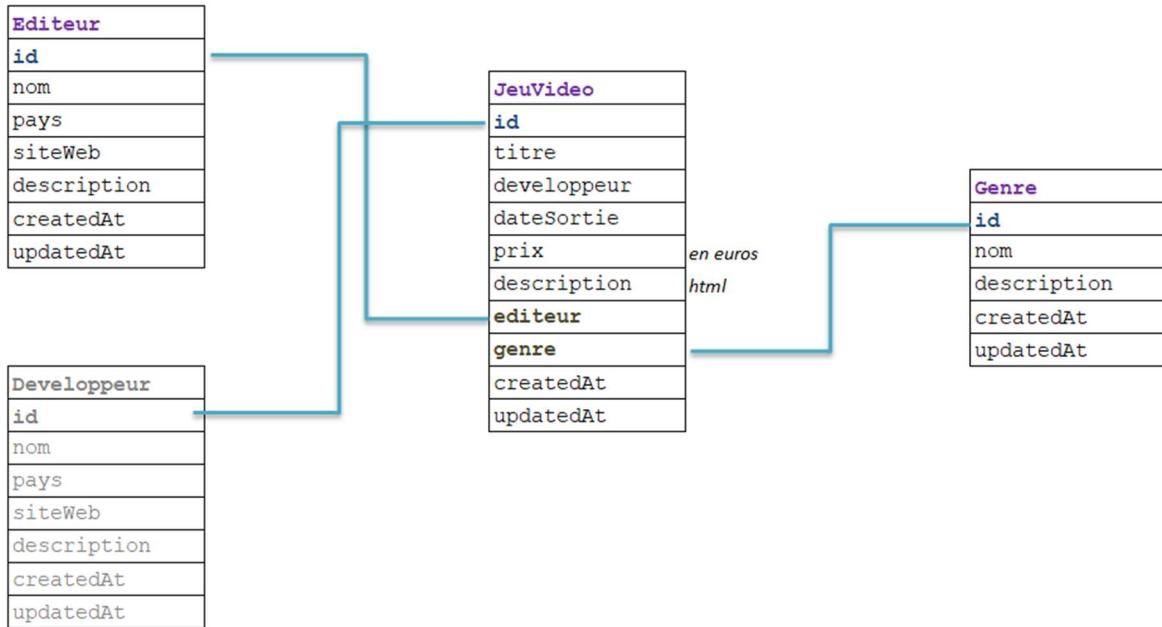
(donné à titre indicatif)



💡 Pour aller plus loin

- Gérer le champ `développeur` de `Jeu_video` comme le champ `éditeur` (c'est dire développer la Classe/Entity `Developpeur`)

TP3



- Ajouter les champs `createdAt` (date de creation) et `updatedAt` (date de modification) dans chaque Entity s'ils sont manquants
- Ajouter des données !
Idéalement, il vous faudra à minima 30 jeux vidéo, rattachés à différents éditeurs, développeurs et genres

[TP 3] Annexe1 : Commandes utiles

Création d'une BDD (*Doctrine*)

```
symfony console doctrine:database:create --if-not-exists
```

```
PS C:\wamp64\www\mon-projet> symfony console doctrine:database:create --if-not-exists
Database `app` for connection named default already exists. Skipped.
PS C:\wamp64\www\mon-projet>
PS C:\wamp64\www\mon-projet> symfony console doctrine:database:create --if-not-exists
Created database `mon_projet` for connection named default
PS C:\wamp64\www\mon_projet>
```

Suppression d'une BDD (*Doctrine*)

⚠️ Attention : irréversible

```
symfony console doctrine:database:drop -force
```

Vérifier le schéma (*Doctrine*)

```
php bin/console doctrine:schema:validate
php bin/console doctrine:schema:validate --verbose
```

Voir et vérifier le mapping des entités (*Doctrine*)

```
php bin/console doctrine:mapping:info
```

```
C:\wamp64\www\mon_projet>php bin/console doctrine:mapping:info
Found 7 mapped entities:

[OK] App\Entity\Editeur
[OK] App\Entity\Genre
[OK] App\Entity\JeuPc
[OK] App\Entity\JeuVideo
[OK] App\Entity\JeuVideoPc

In AttributeReader.php line 115:
  Undefined constant "App\Entity\boolean"

doctrine:mapping:info [--em EM]
```

Création / Modification d'une entité (*Maker*)

```
php bin/console make:entity EntityName
```

Migrations (*Maker et Doctrine*)

Génération (Création) d'une migration (*Maker*)

(Enties → Fichiers de migration contenant des requêtes SQL)

```
php bin/console make:migration
```

Equivalent Doctrine : `symfony console doctrine:migrations:diff`

Génère une migration basée sur les différences détectées (si vous utilisez DoctrineMigrationsBundle).

Vérification du statut des migrations

```
php bin/console doctrine:migrations:status
```

Exécution d'une migration

(Fichiers de migration contenant des requêtes SQL → Schéma de BDD)

```
php bin/console doctrine:migrations:migrate
```

Choix de la migration (ici `Version20250713143754`) :

```
php bin/console doctrine:migrations:migrations:execute -up  
"DoctrineMigrations\Version20250713143754"
```

Différences :

```
symfony console doctrine:migrations:diff
```

Mettre à jour votre base de données (Doctrine)

```
bin/console doctrine:schema:update -f
```

Debug *Doctrine*

```
php bin/console debug:container doctrine
```