

ENTITÉ & DOCTRINE: CHAPITRE 4

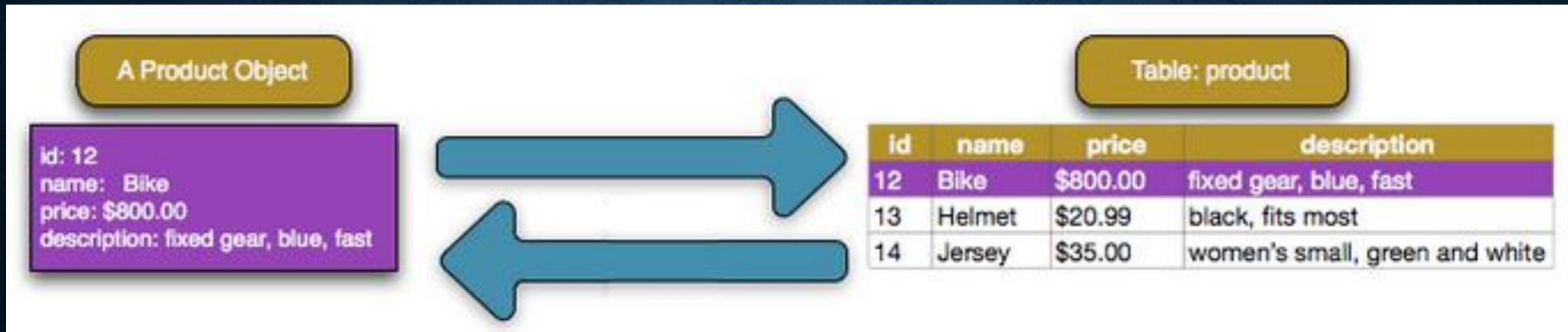
Cours Symfony 4

Mohamed CHEMINGUI

Institut Supérieur des Arts Multimédia de la Manouba

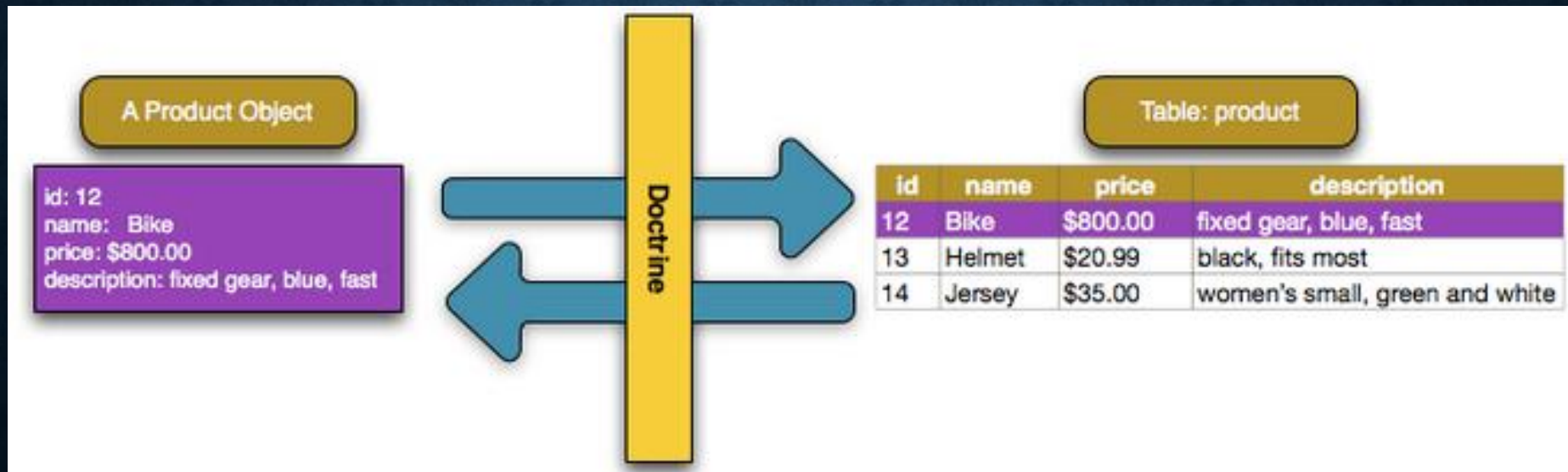
5.1 INTRODUCTION

- Une entité est une classe PHP, se trouve dans le dossier src/Entity
- Une entité: un objet qui représente/contenir les données,,,
- Chaque instance d'entité contient exactement une ligne de table de base de données ciblée



5.1 INTRODUCTION

- Les entités représentent le modèle entier (couche model) de l'application
- Dans Symfony, le modèle entier (le niveau de données) est conservé (enregistré, mis à jour) et géré via Doctrine.
- Doctrine est un ORM (Object Relational Mapping) qui permet d'écrire et lire dans une base de données en utilisant que du PHP.



5.1 INTRODUCTION

L'ORM Doctrine implémente 2 patterns objets pour mapper un objet PHP à des éléments d'un système de persistance :

le pattern "Data Mapper" :

→ synchronise la donnée stockée en base avec les objets PHP (Entités)

le pattern "Unit of Work" :

→ synchronise en base de donnée les instances des Entités (Objets)

Le lien entre les "Entités« (simples objets PHP) et la base de données

Se fait par le Data Mapper « l'Entity Manager »

5.1 INTRODUCTION

Le lien entre les "Entités« (simples objets PHP) et la base de données

Se fait par le Data Mapper « l'Entity Manager »:

- La fonction `connect()`, pour se connecter à la base,
- La fonction **find**, il retrouve un objet à partir d'informations en base.
- La fonction **persist**, il ajoute l'objet manipulé dans l'Unit of Work ;
- La fonction **flush**, tous les objets "marqués" pour être ajoutés, mis à jour et supprimés (fonction **remove**) conduiront à l'exécution d'une transaction avec la base de données.

Plusieurs formats pour our mapper des entités à des tables:

- des annotations dans les commentaires (dits "DocBlocks") des objets PHP
- des fichiers XML .
- des fichiers YAML.
- des fichiers PHP de configuration.

5.1 INTRODUCTION

Installation et configuration

Lancez les commandes en ci-dessous:

1-« composer require symfony/orm-pack »

Si vous rencontrez l'erreur en ci-dessous, modifiez la valeur du «memory_limit » qui se trouve dans le fichier php.ini 128 M par -1

puis lancez de nouveau « composer require symfony/orm-pack »

```
C:\Users\Mohamed>composer require symfony/orm-pack
Using version ^2.0 for symfony/orm-pack
./composer.json has been updated
Loading composer repositories with package information
Updating dependencies (including require-dev)

Fatal error: Allowed memory size of 1610612736 bytes exhausted (tried to allocate 294912 bytes) in phar:///C:/ProgramData/ComposerSetup/bin/composer.phar/src/Composer/DependencyResolver/RuleWatchGraph.php on line 49

Check https://getcomposer.org/doc/articles/troubleshooting.md#memory-limit-errors for more info on how to handle out of memory errors.
C:\Users\Mohamed>php -r "echo ini_get('memory_limit').PHP_EOL;"
128M
```

2-« composer require --dev symfony/maker-bundle »

5.2 PREMIÈRE ENTITÉ « VOITURE »

Mini-Projet: Application « Gestion Location des voitures »

-L'objectif de cette application est de gérer une agence de location de voiture.

- Gestion de la parc par un administrateur:
 1. Ajouter, modifier et supprimer des voitures
 2. Louer (retirer une voiture du stock)
 3. Rendre une voiture dans le stock
 4.

5.2 PREMIÈRE ENTITÉ « VOITURE »

Créez notre entité

Pour ce faire, exécutez la commande suivant dans la racine de votre projet ;

- `php bin/console make:entity`

Il va donc vous être demander de rentrer le nom de votre classe, entrer Voiture

```
C:\project\test>php bin/console make:entity

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

created: src/Entity/Voiture.php
created: src/Repository/VoitureRepository.php

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

Entité créée + *Repository* ?

Repository: patron de conception qui utilise le Symfony avec Doctrine,

Repository d'encapsuler les requêtes lancées à la base de données

5.2 PREMIÈRE ENTITÉ « VOITURE »

Chaque dispose d'un repository par défaut accessible dans tous les contrôleurs:

« `getDoctrine()->getRepository(Voiture::class)` »

Ce repository dispose de plusieurs fonctions pour retrouver vos objets parmi la collection disponible (ou concrètement, parmi ceux disponibles en base de données) :

```
/**
```

```
* @method Voiture|null find($id, $lockMode = null, $lockVersion = null)
```

```
* @method Voiture|null findOneBy(array $criteria, array $orderBy = null)
```

```
* @method Voiture[] findAll()
```

```
* @method Voiture[] findBy(array $criteria, array $orderBy = null, $limit = null, $offset = null)
```

```
*/
```

5.2 PREMIÈRE ENTITÉ « VOITURE »

Les champs à ajouter:

- Matricule (matricule: string (30), nullable = no)
- Marque (marque: string (30), nullable = no)
- Couleur (couleur: string (20), nullable = no)
- Type de carburant (carburant : string (20), nullable = yes)
- Description (description : text, nullable = yes)
- Une date de mise en circulation (datemiseencirculation: datetime, nullable: no)
- Et un booléen (disponibilite : boolean, nullable = no)

5.2 PREMIÈRE ENTITÉ « VOITURE »

Maintenant il faut renseigner les champs de l'entité :

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

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

Field length [255]:
> 30

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

updated: src/Entity/Voiture.php

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

Si tout est bon, le message « Success » s'affiche,

```
Success!

Next: When you're ready, create a migration with php bin/console make:migration
```

5.2 PREMIÈRE ENTITÉ « VOITURE »

Ouvrez le fichier src/Entity/Voiture.php

L'annotation `@ORM\Column` permet de mapper une propriété PHP à une colonne de la base de données.

Tous les attributs ont été créés en plus de l'attribut `$id`

Tous les getters et setters ont été créés,

Nom de la table en base de données aura le même nom que la classe
et les champs aussi auront le même nom que les attributs respectifs,

5.2 PREMIÈRE ENTITÉ « VOITURE »

```
<?php

namespace App\Entity;

use App\Repository\VoitureRepository;
use Doctrine\ORM\Mapping as ORM;

/**
 * @ORM\Entity(repositoryClass=VoitureRepository::class)
 */

class Voiture
{
    /**
     * @ORM\Id
     * @ORM\GeneratedValue
     * @ORM\Column(type="integer")
     */
    private $id;

    /**
     * @ORM\Column(type="string", length=30, unique=true)
     */
    private $matricule;

    /**
     * @ORM\Column(type="string", length=30)
     */
    private $marque;

    /**
     * @ORM\Column(type="string", length=20)
     */
    private $couleur;

    /**
     * @ORM\Column(type="string", length=20, nullable=true)
     */
    private $carburant;

    /**
     * @ORM\Column(type="text", nullable=true)
     */
    private $description;
```

```
public function getId(): ?int
{
    return $this->id;
}

public function getMatricule(): ?string
{
    return $this->matricule;
}

public function setMatricule(string $matricule): self
{
    $this->matricule = $matricule;

    return $this;
}

public function getMarque(): ?string
{
    return $this->marque;
}

public function setMarque(string $marque): self
{
    $this->marque = $marque;

    return $this;
}

public function getCouleur(): ?string
{
    return $this->couleur;
}

public function setCouleur(string $couleur): self
{
    $this->couleur = $couleur;

    return $this;
}

public function getCarburant(): ?string
{
    return $this->carburant;
```

```
public function setCarburant(?string $carburant): self
{
    $this->carburant = $carburant;

    return $this;
}

public function getDescription(): ?string
{
    return $this->description;
}

public function setDescription(?string $description): self
{
    $this->description = $description;

    return $this;
}

public function getDateMiseEncirculation(): ?\DateTimeInterface
{
    return $this->dateMiseEncirculation;
}

public function setDateMiseEncirculation(\DateTimeInterface $dateMiseEncirculation): self
{
    $this->dateMiseEncirculation = $dateMiseEncirculation;

    return $this;
}

public function getDisponibilite(): ?bool
{
    return $this->disponibilite;
}

public function setDisponibilite(bool $disponibilite): self
{
    $this->disponibilite = $disponibilite;

    return $this;
}
```

5.2 PREMIÈRE ENTITÉ « VOITURE »

Ouvrez le fichier src/Repository/VoitureRepository.php

Le patron de conception *repository* permet d'encapsuler les requêtes lancées à la base de données:

- Méthodes créées par défaut.
- Développer des méthodes.
- ,,,,

```
VoitureRepository.php x .env.local x ConnectionController.php x VoitureController.php x
1 k?php
2
3 namespace App\Repository;
4
5 use App\Entity\Voiture;
6 use Doctrine\Bundle\DoctrineBundle\Repository\ServiceEntityRepository;
7 use Doctrine\Persistence\ManagerRegistry;
8
9 /**
10  * @method Voiture|null find($id, $lockMode = null, $lockVersion = null)
11  * @method Voiture|null findOneBy(array $criteria, array $orderBy = null)
12  * @method Voiture[]    findAll()
13  * @method Voiture[]    findBy(array $criteria, array $orderBy = null, $limit = null, $offset = null)
14  */
15 class VoitureRepository extends ServiceEntityRepository
16 {
17     public function __construct(ManagerRegistry $registry)
18     {
19         parent::__construct($registry, Voiture::class);
20     }
21
22     // /**
23     //  * @return Voiture[] Returns an array of Voiture objects
24     //  */
25     /*
26     public function findByExampleField($value)
27     {
28         return $this->createQueryBuilder('v')
29             ->andWhere('v.exampleField = :val')
30             ->setParameter('val', $value)
31             ->orderBy('v.id', 'ASC')
32             ->setMaxResults(10)
33             ->getQuery()
34             ->getResult()
35         ;
36     }
37     */
38 }
```


5.2 PREMIÈRE ENTITÉ « VOITURE »

Comment ajouter d'autres contraintes ?

Par exemple : la matricule d'une voiture doit être UNIQUE, Pour cela il faut ajouter les lignes suivantes dans l'entité « Voiture »

- `use Symfony\Component\Validator\Constraints as Assert;`
- `// validateur de Symfony pour obtenir la valeur des champs avec les «getter»`
- Ajouter les annotations suivantes:

```
/**  
 * @ORM\Column(type="string", length=30, unique=true)  
 */  
private $matricule;
```

5.3 CONFIGURATION CONNEXION AVEC LA BASE DE DONNÉE

- Ouvrez le fichier .env qui se trouve à la racine du projet,

A partir de la ligne 27 il y a la les différentes configuration pour la connexion avec les base de données MySQL, PostgreSQL et NOSQL,

MySQL:

« DATABASE_URL=mysql://db_user:db_password@url:port/db_name »

- db_user: le nom d'utilisateur de la base de données
- db_password: mot de passe de la base de données
- url:port: lien du serveur de la base de donnée et le numero du port
- db_name: nom de la base de données

Copiez le fichier '.env' et renommer le nouveau fichier '.env.local'

5.4 TESTER LA CONNEXION AVEC LA BASE DE DONNÉE

Créez un contrôleur nommé « ConnectionController » pour testez la connexion avec la base de donnée en utilisant les instruction suivantes:

```
$em-> $this->getDoctrine()->getManager();
```

méthode Data Mapper « l'Entity Manager »:

```
$em->getConnection()->connect();
```

tentative de la connexion à la base de donnée

```
$connected = $em->getConnection()->isConnected();
```

Affichez le résultat dans une page TWIG

Connexion réussite si \$connected = 1

Sinon Connexion échouée

5.4 TESTER LA CONNEXION AVEC LA BASE DE DONNÉE

- Syntaxe « If » dans TWIG:
- Exemple avec if ... endif

```
{% A > 0 %}
```

 {{ A }} est positif

```
{% endif %}
```

Exemple avec if ... else ... endif

```
{{ A }} est
```

```
{% A > 0 %}
```

 positif

```
{% else %}
```

 négatif

```
{% endif %}
```

Exemple avec if ... else ... endif

```
{{ A }} est
```

 positif

```
{% elseif A < 0 %}
```

 négatif

```
{% else %}
```

 null

```
{% endif %}
```


5.4 TESTER LA CONNEXION AVEC LA BASE DE DONNÉE

```
class ConnectionController extends AbstractController
{
    /**
     * @Route("/connection", name="connection")
     */
    public function index(): Response
    {
        try
        {
            $em = $this->getDoctrine()->getManager();
            $em->getConnection()->connect();
            $connected = $em->getConnection()->isConnected();

            return $this->render('connection/index.html.twig', [
                'connected' => $connected,
            ]);
        }

        catch(\Exception $e)
        {
            return $this->render('connection/index.html.twig'
            );
        }
    }
}
```

5.5 TESTER LA CONNEXION AVEC LA BASE DE DONNÉE

Mettez à jour le fichier .env.local

```
32 DATABASE_URL="mysql://root:root@127.0.0.1:3306/?serverVersion=5.7.31"
```

```
{% extends 'base.html.twig' %}

{% block title %}Hello DataBase!{% endblock %}

{% block body %}
▼ <style>
    .example-wrapper { margin: 1em auto; max-width: 800px; width: 95%; font: 18px/1.5 sans-serif; }
    .example-wrapper code { background: #F5F5F5; padding: 2px 6px; }
</style>

▼ <div class="example-wrapper">

    {% if connected is defined %}
        Connexion réussite
    {% else %}
        Connexion échouée
    {% endif %}

</div>
{% endblock %}
```


5.5 CRÉATION DE LA BASE DE DONNÉE

Ajoutez le nom de la base dans le fichier .env.local

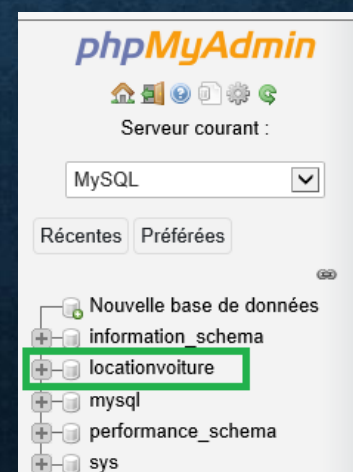
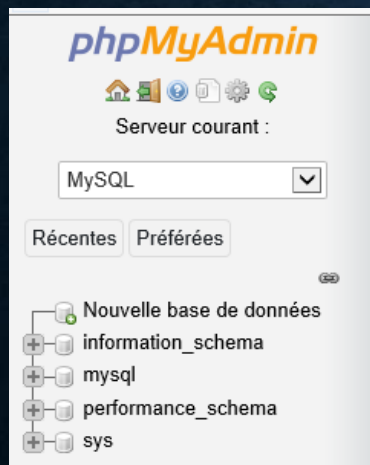
```
32 DATABASE_URL="mysql://root:root@127.0.0.1:3306/locationvoiture?serverVersion=5.7.31"
```

Symfony 4 est fourni avec une intégration de Doctrine ORM qui fournit de nombreuses commandes dans la console.

Lancez la commande :« php bin/console doctrine:database:create »

```
C:\project\test>php bin/console doctrine:database:create
Created database 'locationvoiture' for connection named default

C:\project\test>
```




























5.6 CRÉATION DE LA TABLE VOITURE

Lancez la commande :« php bin/console doctrine:schema:update --force »

```
C:\project\test>php bin/console doctrine:schema:update --force
Updating database schema...
    1 query was executed
[OK] Database schema updated successfully!
```

Vous pouvez vérifier la création de la table dans la base de donnée

#	Nom	Type	Interclassement	Attributs	Null	Valeur par défaut	Commentaires	Extra	Action
<input type="checkbox"/>	1 id 	int(11)			Non	Aucun(e)		AUTO_INCREMENT	 Modifier  Supprimer  Plus
<input type="checkbox"/>	2 matricule	varchar(30)	utf8mb4_unicode_ci		Non	Aucun(e)			 Modifier  Supprimer  Plus
<input type="checkbox"/>	3 marque	varchar(30)	utf8mb4_unicode_ci		Non	Aucun(e)			 Modifier  Supprimer  Plus
<input type="checkbox"/>	4 couleur	varchar(20)	utf8mb4_unicode_ci		Non	Aucun(e)			 Modifier  Supprimer  Plus
<input type="checkbox"/>	5 carburant	varchar(20)	utf8mb4_unicode_ci		Oui	NULL			 Modifier  Supprimer  Plus
<input type="checkbox"/>	6 description	longtext	utf8mb4_unicode_ci		Oui	NULL			 Modifier  Supprimer  Plus
<input type="checkbox"/>	7 datemiseencirculation	datetime			Non	Aucun(e)			 Modifier  Supprimer  Plus
<input type="checkbox"/>	8 disponibilite	tinyint(1)			Non	Aucun(e)			 Modifier  Supprimer  Plus

5.7 MODIFIER L'ENTITÉ « VOITURE »

Ajouter un autre champ: par exemple nrbplace

- Nombre de place (nbrplace : integer , nullable = no)

Pour ce faire, exécutez la commande suivant dans la racine de votre projet ;

- `php bin/console make:entity`

```
C:\project\test>php bin/console make:entity

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

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

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

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

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

updated: src/Entity/Voiture.php

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

Success!
```

5.7 MODIFIER L'ENTITÉ « VOITURE »

Le nouveau champs n'existe pas encore dans la table des voiture.

Pour ce faire, exécutez la commande suivant dans la racine de votre projet ;

Générez une nouvelle migration: `php bin/console make:migration`

```
C:\project\test>php bin/console make:entity
Class name of the entity to create or update (e.g. VictoriousPizza):
> Voiture

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

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

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

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

updated: src/Entity/Voiture.php

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

Success!
```

Table	Action	Lignes	Type	Interclassement	Taille	Perte
<input type="checkbox"/> migration_versions	★ Parcourir Structure Rechercher Insérer Vider Supprimer	0	InnoDB	utf8mb4_unicode_ci	16,0 kio	-
<input type="checkbox"/> voiture	★ Parcourir Structure Rechercher Insérer Vider Supprimer	0	InnoDB	utf8mb4_unicode_ci	16,0 kio	-
2 tables	Somme	0	MyISAM	latin1_swedish_ci	32,0 kio	0 0

5.7 MODIFIER L'ENTITÉ « VOITURE »

Appliquez la migration dans la base : `php bin/console doctrine:migrations:migrate`

```
C:\project\test> 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 20201112205528 from 0

++ migrating 20201112205528

    -> ALTER TABLE voiture ADD nbrplace INT NOT NULL

++ migrated (took 2637.6ms, used 20M memory)

-----

++ finished in 2714.6ms
++ used 20M memory
++ 1 migrations executed
++ 1 sql queries

C:\project\test>
```

Vous pouvez voir la nouvelle structure de la table Voiture,

5.7 MODIFIER L'ENTITÉ « VOITURE »

Appliquez la migration dans la base : `php bin/console doctrine:migrations:migrate`

```
C:\project\test> 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 20201112205528 from 0

++ migrating 20201112205528

    -> ALTER TABLE voiture ADD nbrplace INT NOT NULL

++ migrated (took 2637.6ms, used 20M memory)

-----

++ finished in 2714.6ms
++ used 20M memory
++ 1 migrations executed
++ 1 sql queries

C:\project\test>
```

Vous pouvez voir la nouvelle structure de la table Voiture,

5.8 MANIPULER LES OBJETS (CRUD)

L'entité « Voiture » est bien définie ✓

Commencez à manipuler l'entité dans nos contrôleurs ,

Manipulation d'objets en base : création, mise à jour et suppression

L'Entity Manager qui est responsable de la gestion des entités, grâce aux

- La fonction `connect()`, pour se connecter à la base,
- La fonction **find**, il retrouve un objet à partir d'informations en base.
- La fonction **persist**, il ajoute l'objet manipulé dans l'Unit of Work ;
- La fonction **flush**, tous les objets "marqués" pour être ajoutés, mis à jour et supprimés (fonction **remove**) conduiront à l'exécution d'une transaction avec la base de données.

5.8 MANIPULER LES OBJETS (CRUD)

Créez un contrôleur nommé « VoitureController » pour manipuler les objet avec la base de donnée: « php bin/console make:controller VoitureController », puis:

- Créez un nouvel objet Voiture.
- Définissez des données dessus.
- Enregistrez dans la base de données

Ouvrez le contrôleur « VoitureControlle » puis ajoutez:

- `use App\Entity\Voiture;`
- `use Doctrine\ORM\EntityManagerInterface;`

Le reste dans code source

5.8 MANIPULER LES OBJETS (CRUD)

Voici le contenu de la Classe « Voiture » pour insérer un objet

```
/**
 * @Route("/createvoiture", name="create_voiture")
 */
public function createVoiture(): Response
{
    $entityManager = $this->getDoctrine()->getManager();

    $voiture = new Voiture();
    $voiture->setMatricule('200TU1500');
    $voiture->setMarque('BMW');
    $voiture->setDescription('Voiture Luxe');
    $voiture->setCouleur('Noir');
    $voiture->setCarburant('Gazoil');
    $date = new \DateTime('2019-06-05 12:15:30');
    $voiture->setDatemiseencirculation($date);
    $voiture->setDisponibilite(1);
    $voiture->setNbrplace(5);

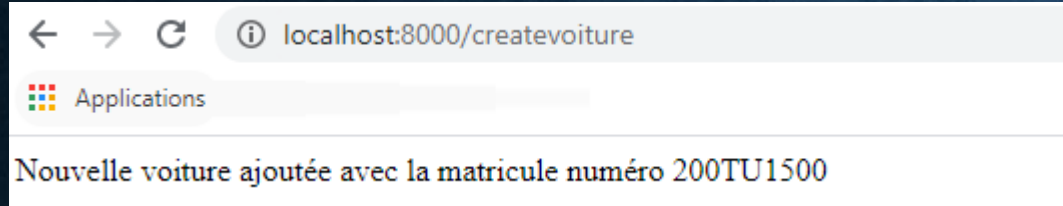
    $entityManager->persist($voiture); // notifier Doctrine qu'il ya un objet à enregistrer

    $entityManager->flush();// exécuter la requête pour enregistrer dans la base

    return new Response('Nouvelle voiture ajoutée avec la matricule numéro |'.$voiture->getMatricule());
}
```

5.8 MANIPULER LES OBJETS (CRUD)

Pour tester: <http://localhost:8000/createvoiture>

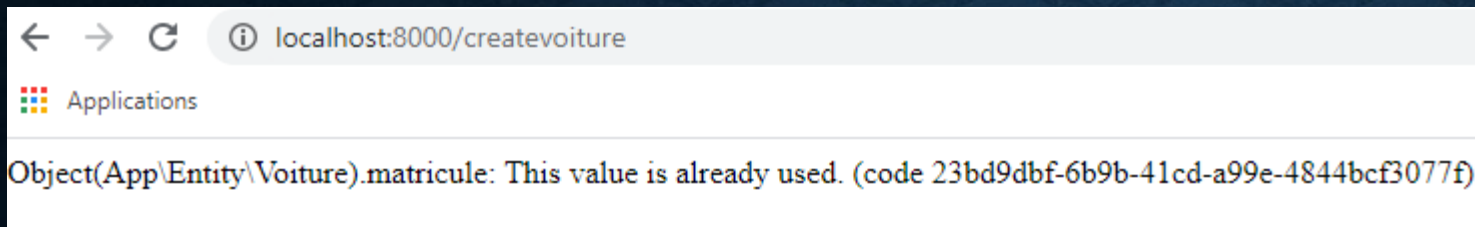


Et dans la base ...

	id	matricule	marque	couleur	carburant	description	datemiseencirculation	disponibilite	nbrplace
<input type="checkbox"/> Éditer Copier Supprimer	1	200TU1500	BMW	Noir	Gazoil	Voiture Luxe	2019-06-05 12:15:30	1	5

Essayez de nouveau le lien avec la même matricule ?

Symfony\Component\Validator ✓



5.8 MANIPULER LES OBJETS (CRUD)

Utilisation du composant « Validator » pour valider les champs avant l'envoi des données vers la base de données

```
/**
 * @ORM\Column(type="string", length=30)
 * @Assert\NotBlank
 */
private $marque;
```

```
public function createVoiture(ValidatorInterface $validator): Response
{
    $entityManager = $this->getDoctrine()->getManager();

    $voiture = new Voiture();
    $voiture->setMatricule('200TU1700');
    $voiture->setMarque('');
    $voiture->setDescription('Voiture Luxe');
```

← → ↺ ⓘ localhost:8000/createvoiture

Applications

Object(App\Entity\Voiture).marque: This value should not be blank. (code c1051bb4-d103-4f74-8988-acbcafc7fdc3)

5.8 MANIPULER LES OBJETS (CRUD)

Récupération de tous les données en utilisant la méthode « findAll() » du Repository

```
« getDoctrine()->getRepository(Voiture::class)->findAll(); »
```

- Récupérer le résultat dans un tableau \$voitures
- Afficher le résultat dans une page TWIG

Syntaxes TWIG

```
<h1>Voitures</h1>
```

```
<ul>
```

```
    {% for voiture in voitures %}
```

```
        <li>{{ voiture.marque | e }} {{ voiture.couleur | e }} {{ voiture.Description | e }} {{  
voiture.nbrplace | e }} Places </li>
```

```
    {% endfor %}
```

```
</ul>
```

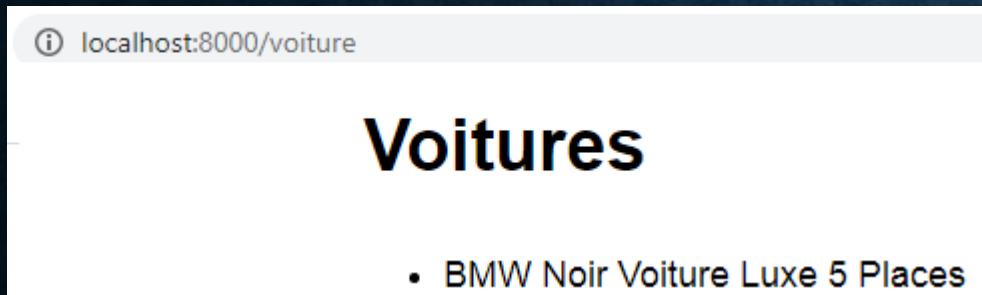

5.8 MANIPULER LES OBJETS (CRUD)

- Modifiez le Contrôleur « Voitures » comme suit

```
/**
 * @Route("/voiture", name="voiture")
 */
public function index(): Response
{
    $voitures = $this->getDoctrine()->getRepository(Voiture::class)->findAll();

    return $this->render('voiture/index.html.twig', [
        'voitures' => $voitures,
    ]);
}
```

- Lancez la page <http://localhost:8000/voiture>



5.8 MANIPULER LES OBJETS (CRUD)

Récupération d'une voiture en utilisant la méthode « `findBy()` » du Repository

“`getDoctrine()->getRepository(Voiture::class)->findBy(array('matricule' => 'matricule'));`”

```
/**
 * @Route("/voiture/{mat}", name="voiturebymat")
 */
public function afficher(String $mat): Response
{
    $voitures=$this->getDoctrine()->getRepository(Voiture::class)->findBy(array('matricule' => $mat));

    return $this->render('voiture/index.html.twig', [
        'voitures' => $voitures,
    ]);
}
```

Lancez la page <http://localhost:8000/voiture/200TU1600>



5.8 MANIPULER LES OBJETS (CRUD)

Modification d'une voiture

```
/**
 * @Route("/modifiervoiture/{mat}", name="editvoiturebymat")
 */
public function modifier(String $mat): Response
{
    $entityManager = $this->getDoctrine()->getManager();
    $voiture=$this->getDoctrine()->getRepository(Voiture::class)->findBy(array('matricule' => $mat));

    if (!$voiture) {
        throw $this->createNotFoundException(
            'Pas de voiture avec la matricule '.$mat
        );
    }

    $voiture[0]->setMarque('POLO');
    $entityManager->flush();

    return $this->redirectToRoute('voiturebymat', ['mat' => $mat ]);
}
```

Lancez: <http://localhost:8000/modifiervoiture/200TU1600>

5.8 MANIPULER LES OBJETS (CRUD)

Suppression d'une voiture

```
/**
 * @Route("/supprimervoiture/{mat}", name="supprimervoiturebymat")
 */
public function supprimer(String $mat): Response
{
    $entityManager = $this->getDoctrine()->getManager();
    $voiture=$this->getDoctrine()->getRepository(Voiture::class)->findBy(array('matricule' => $mat));

    if (!$voiture) {
        throw $this->createNotFoundException(
            'Pas de voiture avec la matricule '.$mat
        );
    }

    $entityManager->remove($voiture[0]);
    $entityManager->flush();

    return $this->redirectToRoute('voiture');
}
```

Lancez: <http://localhost:8000/supprimervoiture/200TU1600>

5.9 RELATION ENTRE LES ENTITÉ

Les objets PHP vont interagir les uns avec les autres à travers les relation entre les entités :

1-1 : un objet A correspond à un objet B (voiture-emplacement)

1-n : un objet A est lié à de nombreuses instances de B (voiture- contrats)

n-n : des objets de type A ont de multiples relations avec des objets de type B.
(contrats-Conducteurs)

5.9.1 RELATION 1-1

La relation ne sera décrite que dans l'un des objets

selon votre propre logique business....

Par exemple une voiture peut posséder un emplacement:

Vous pouvez ajouter l'ID d'une voiture dans l'entité « Emplacement » :

Pour ce faire, il faut créer une entité « Emplacement » et ajouter l'id d'une voiture comme attribut.

5.9.1 RELATION 1-1

```
C:\project\test>php bin/console make:entity

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

created: src/Entity/Emplacement.php
created: src/Repository/EmplacementRepository.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]:
>

Field length [255]:
> 50

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

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

5.9.1 RELATION 1-1

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

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

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

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

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

Do you want to add a new property to Voiture so that you can access/update Empl
acement objects from it - e.g. $voiture->getEmplacement()? (yes/no) [no]:
> yes

A new property will also be added to the Voiture class so that you can access t
he related Emplacement object from it.

New field name inside Voiture [emplacement]:
>

updated: src/Entity/Emplacement.php
updated: src/Entity/Voiture.php

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


5.9.1 RELATION 1-1

```
/**
 * @ORM\OneToOne(targetEntity=Voiture::class, inversedBy="emplacement", cascade={"persist", "remove"})
 * @ORM\JoinColumn(nullable=false)
 */
private $idvoiture;
```

- OneToOne permet de définir la relation entre les deux entités en permettant de définir une entité liée à l'aide du paramètre targetEntity,
- JoinColumn est une annotation facultative : elle permet de définir la clé étrangère qui fait office de référence dans la table.

5.9.1 RELATION 1-1

php bin/console doctrine:schema:update --dump

```
C:\project\test>php bin/console doctrine:schema:update --dump-sql
```

The following SQL statements will be executed:

```
CREATE TABLE emplacement (id INT AUTO_INCREMENT NOT NULL, idvoiture_id INT
NOT NULL, nom VARCHAR(50) NOT NULL, UNIQUE INDEX UNIQ_C0CF65F6CC28B580 (idvoitur
e_id), PRIMARY KEY(id)) DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci
ENGINE = InnoDB;
ALTER TABLE emplacement ADD CONSTRAINT FK_C0CF65F6CC28B580 FOREIGN KEY (idvo
iture_id) REFERENCES voiture (id);
```

php bin/console doctrine:schema:update --force

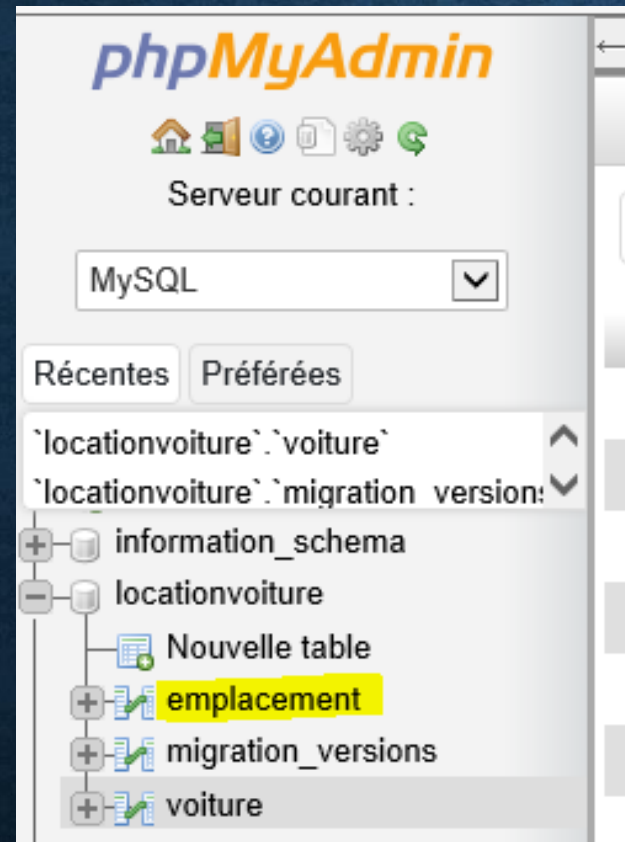
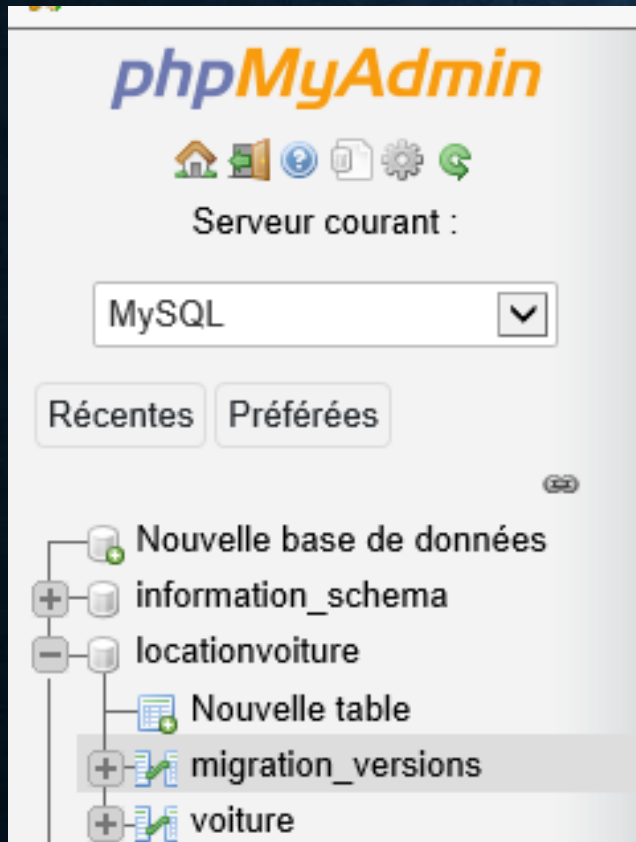
```
C:\project\test>php bin/console doctrine:schema:update --force
```

Updating database schema...

2 queries were executed

[OK] Database schema updated successfully!

5.9.1 RELATION 1-1



Serveur: MySQL:3306 » Base de données: locationvoiture » Table: emplacement

Parcourir Structure SQL Rechercher Insérer Exporter

Structure de table Vue relationnelle

#	Nom	Type	Interclassement	Attributs	Null	Valeur par défaut
<input type="checkbox"/> 1	id	int(11)			Non	Aucun(e)
<input type="checkbox"/> 2	idvoiture_id	int(11)			Non	Aucun(e)
<input type="checkbox"/> 3	nom	varchar(50)	utf8mb4_unicode_ci		Non	Aucun(e)

5.9.2 RELATION 1-N

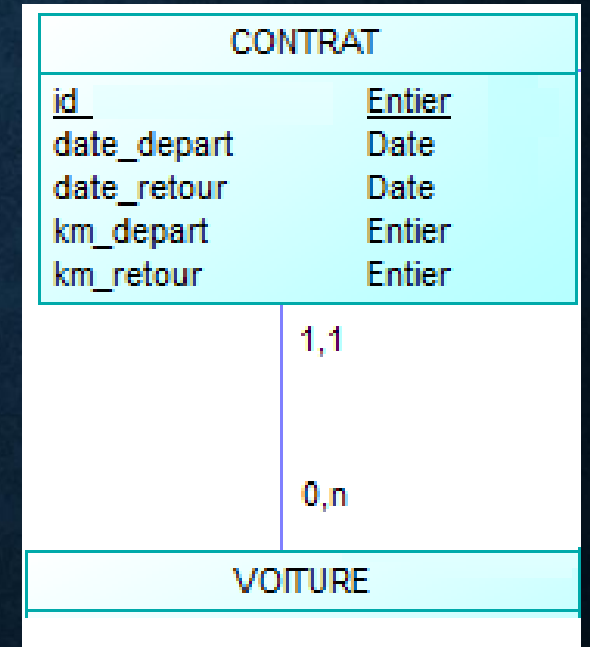
La relation 1-N est représentée de façon "bidirectionnelle" :

-> c'est-à-dire que chaque entité a connaissance de la relation

Dans notre cas une voiture peut avoir 1 ou plusieurs contrat(s):

Pour ce faire, il faut:

- Créer une entité contrat
- Créer l'association « ManyToOne » dans l'entité « Voiture »



5.9.2 RELATION 1-N

```
::\project\test>php bin/console make:entity

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

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

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

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

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

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

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

Do you want to add a new property to Contrat so that you can access/update Voiture objects from it - e.g. $contrat->getVoitures()? (yes/no) [yes]:
> yes

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

New field name inside Contrat [voitures]:
>

Do you want to activate orphanRemoval on your relationship?
A Voiture is "orphaned" when it is removed from its related Contrat.
e.g. $contrat->removeVoiture($voiture)

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

Do you want to automatically delete orphaned App\Entity\Voiture objects (orphanRemoval)? (yes/no) [no]:
> no

updated: src/Entity/Voiture.php
updated: src/Entity/Contrat.php
```

5.9.2 RELATION 1-N

L'entité **propriétaire** doit définir l'attribut "mappedBy" : il correspond à la **propriété de l'objet** "possédé« (Contrat) qui fait le lien entre les deux entités.

```
/**
 * @ORM\OneToMany(targetEntity=Voiture::class, mappedBy="contrat")
 */
private $voitures;
```

L'entité **possédée** doit définir l'attribut "inversedBy" : il correspond à la **propriété de l'objet** "propriétaire« (Voiture) qui fait le lien entre les deux entités.

```
/**
 * @ORM\ManyToOne(targetEntity=Contrat::class, inversedBy="voitures")
 * @ORM\JoinColumn(nullable=false)
 */
private $contrat;
```

Dans ce cas métier, on peut considérer que c'est le client qui est propriétaire de la relation. En effet, une adresse "client" ne peut exister sans client !

5.9.2 RELATION 1-N

php bin/console doctrine:schema:update --dump-sql

```
C:\project\test>php bin/console doctrine:schema:update --dump-sql
```

The following SQL statements will be executed:

```
CREATE TABLE contrat (id INT AUTO_INCREMENT NOT NULL, datedepart DATETIME NOT NULL, dateretour DATETIME NOT NULL, kmdepart INT NOT NULL, kmretour INT NOT NULL, PRIMARY KEY(id)) DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci ENGINE = InnoDB;
ALTER TABLE voiture ADD contrat_id INT NOT NULL;
ALTER TABLE voiture ADD CONSTRAINT FK_E9E2810F1823061F FOREIGN KEY (contrat_id) REFERENCES contrat (id);
CREATE INDEX IDX_E9E2810F1823061F ON voiture (contrat_id);
```

php bin/console doctrine:schema:update --force

```
C:\project\test>php bin/console doctrine:schema:update --force
```

Updating database schema...

In AbstractMySQLDriver.php line 49:

```
An exception occurred while executing 'ALTER TABLE voiture ADD CONSTRAINT FK_E9E2810F1823061F FOREIGN KEY (contrat_id) REFERENCES contrat (id)':
```

Si la table voiture n'est pas vide vous allez rencontrer un erreur d'insertion du L'id du contrat

5.9.3 RELATION N-N

A vous de développer dans vos projet....