

Prise en main de Symfony 4.4

Doctrine

Structure

**Objectifs :**

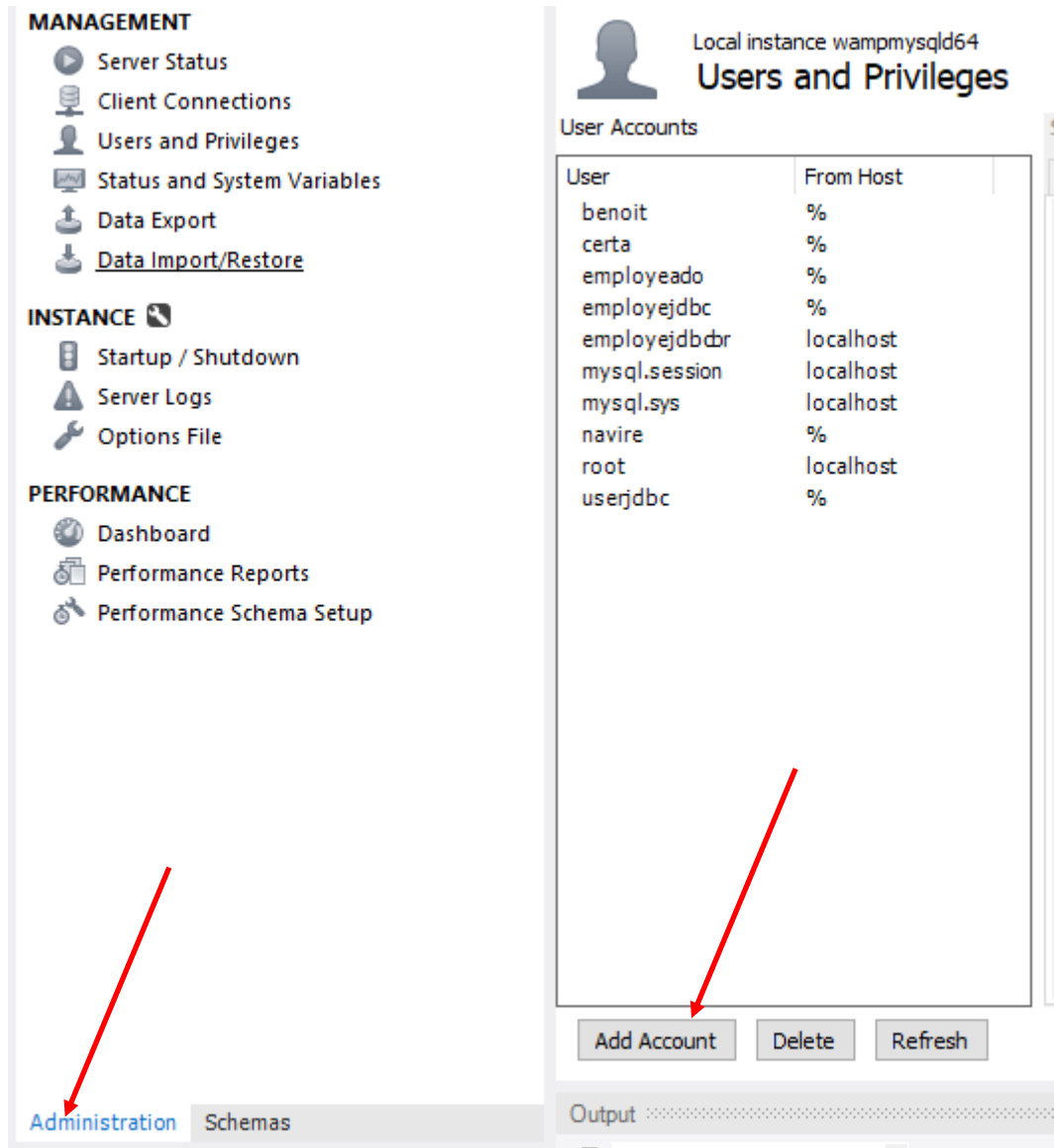
Développer une petite application qui va gérer un les navires.
Versionner le projet
Créer des contrôleurs
Créer et valider des formulaires

Environnement :

- ✓ Serveur apache : celui de wamp
- ✓ Base de données : mysql en local (wamp)
- ✓ IDE : netbeans
- ✓ virtualHost : navire.sio
- ✓ PHP 7.4.x
- ✓ Symfony 4.4
- ✓ Mysql 5.7.xx

**Partie 1 : PREPARATION DU TD**

Création de l'utilisateur Symfony/symfony dans mysql avec le client MySQL workbench :
Connectez-vous en root :



MANAGEMENT

- Server Status
- Client Connections
- Users and Privileges
- Status and System Variables
- Data Export
- Data Import/Restore

INSTANCE

- Startup / Shutdown
- Server Logs
- Options File

PERFORMANCE

- Dashboard
- Performance Reports
- Performance Schema Setup

Local instance wampmysqld64
Users and Privileges

User Accounts

User	From Host
benoit	%
certa	%
employeado	%
employejdbc	%
employejdbdr	localhost
mysql.session	localhost
mysql.sys	localhost
navire	%
root	localhost
userjdbc	%

Add Account Delete Refresh

Output

Administration Schemas

Login Account Limits Administrative Roles Schema Privileges

Login Name: You may create multiple accounts with the same name to connect from different hosts.

Authentication Type: For the standard password and/or host based authentication, select 'Standard'.

Limit to Hosts Matching: % and _ wildcards may be used

Password: Type a password to reset it.

Weak password.

Confirm Password: Enter password again to confirm.

Expire Password

Puis passez à l'onglet Administrative Roles :

Login Account Limits **Administrative Roles** Schema Privileges

Cochez DBA :

Login	Account Limits	Administrative Roles	Schema Privileges
Role	Description		
<input checked="" type="checkbox"/> DBA	grants the rights to perform all tasks		
<input checked="" type="checkbox"/> MaintenanceAdmin	grants rights needed to maintain server		
<input checked="" type="checkbox"/> ProcessAdmin	rights needed to assess, monitor, and kill any user proce...		
<input checked="" type="checkbox"/> UserAdmin	grants rights to create users logins and reset passwords		
<input checked="" type="checkbox"/> SecurityAdmin	rights to manage logins and grant and revoke server an...		
<input checked="" type="checkbox"/> MonitorAdmin	minimum set of rights needed to monitor server		
<input checked="" type="checkbox"/> DBManager	grants full rights on all databases		
<input checked="" type="checkbox"/> DBDesigner	rights to create and reverse engineer any database sche...		
<input checked="" type="checkbox"/> ReplicationAdmin	rights needed to setup and manage replication		
<input checked="" type="checkbox"/> BackupAdmin	minimal rights needed to backup any database		

Il ne vous reste plus qu'à valider :

Revert Apply



On réduira les privilèges de l'utilisateur symfony une fois la BD créée

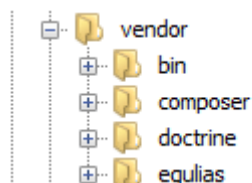
Vérifiez que le recipe symfony/orm-pack est installé. Vous pouvez le voir dans le fichier composer.json

```

"doctrine/doctrine-bundle": "^2.2",
"doctrine/doctrine-migrations-bundle": "^3.0",
"doctrine/orm": "^2.7",
"sensio/framework-extra-bundle": "^5.6",

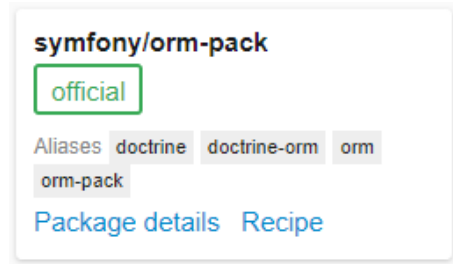
```

Vous pouvez aussi le repérer dans le dossier config\prod ou dans le dossier vendor



Si vous ne trouvez pas le recipe, alors vous devrez l'installer :

Rendez-vous sur le site <https://flex.symfony.com/>, repérez le bundle à installer :



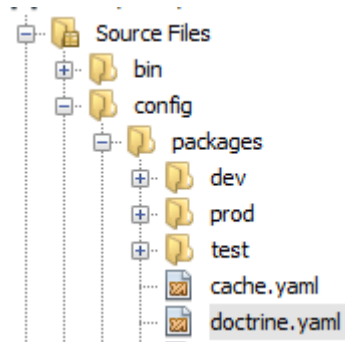
Installez-le avec composer à la racine du projet :

```
> composer require symfony/orm-pack
```

Une fois Doctrine installé, vous allez configurer la base de données.

Ouvrez le fichier de configuration de doctrine (doctrine.yaml) :

Voir documentation : <https://symfony.com/doc/4.4/reference/configuration/doctrine.html>



```
doctrine:
  dbal:
    #url: '%env(resolve:DATABASE_URL)%'

    # IMPORTANT: You MUST configure your server version,
    # either here or in the DATABASE_URL env var (see .env file)
    #server_version: '13'

    #####
    ## Benoît
    #####

    driver: 'pdo_mysql'
    server_version: '5.7'
    charset: utf8mb4
    default_table_options:
      charset: utf8mb4
      collate: utf8mb4_unicode_ci
    dbname: '%env(DATABASE_NAME)%'
    host: 'localhost'
    user: '%env(DATABASE_USER)%'
    password: '%env(DATABASE_PWD)%'

    #####
    ## Fin Benoît
    #####

  orm:
    auto_generate_proxy_classes: true
    naming_strategy: doctrine.orm.naming_strategy.underscore_number_aware
    auto_mapping: true
    mappings:
      App:
        is_bundle: false
        type: annotation
        dir: '%kernel.project_dir%/src/Entity'
        prefix: 'App\Entity'
        alias: App
```

Vous allez maintenant modifier les paramètres et en rajouter dans le fichier .env

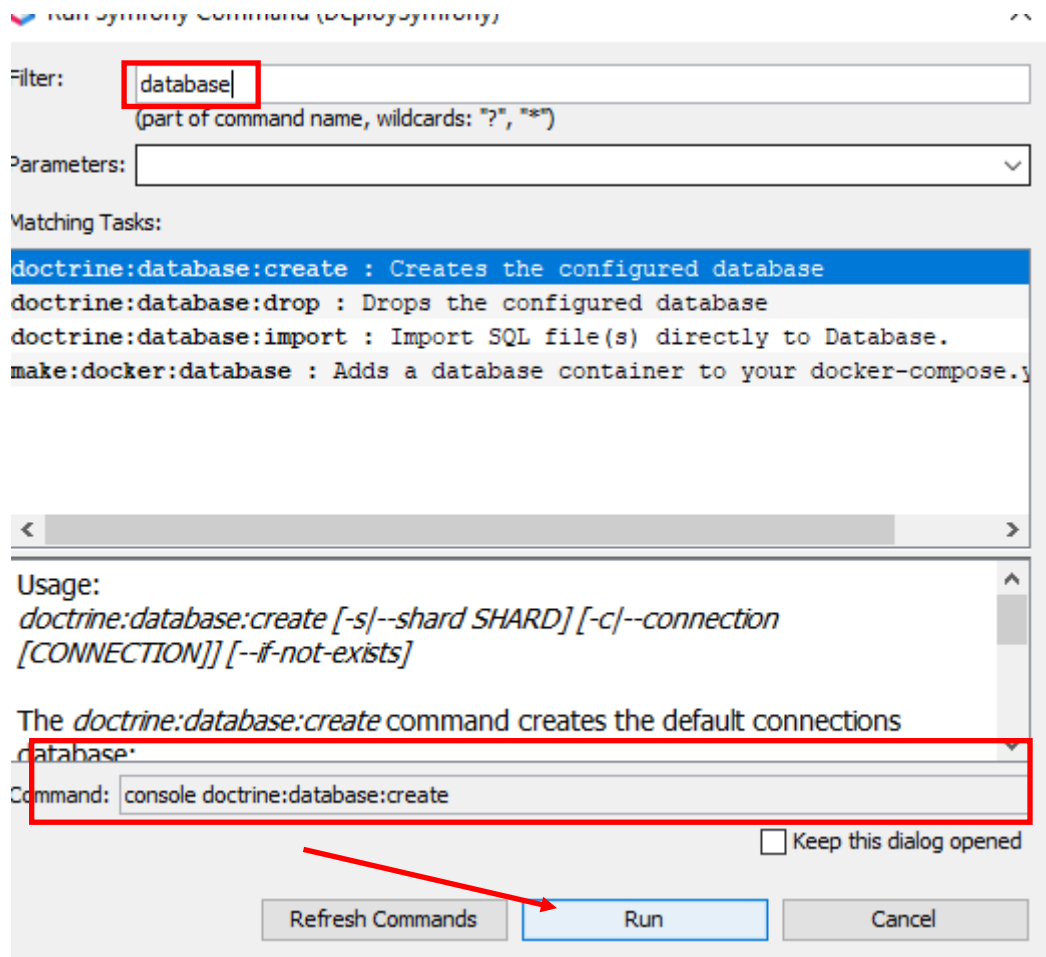
```
#####
DATABASE_URL=mysql://db_user:db_password@127.0.0.1:3306/db_name
DATABASE_USER=symfony
DATABASE_PWD=symfony
DATABASE_NAME=naviresymfo

###< doctrine/doctrine-bundle ###
```

Vous pouvez désormais créer la base de données avec la commande :

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

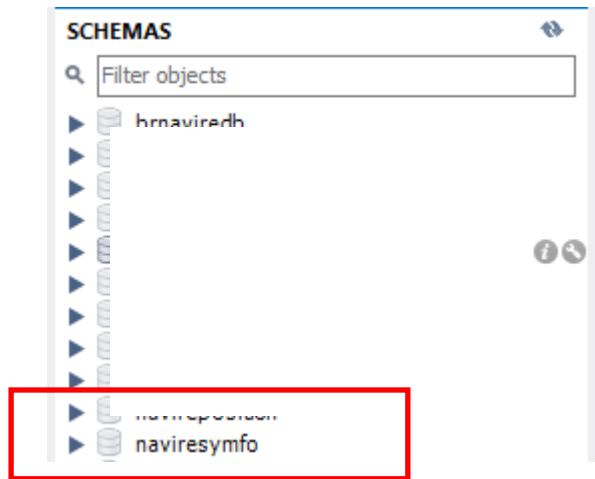
Ou par l'interface graphique des commandes symfony de netbeans



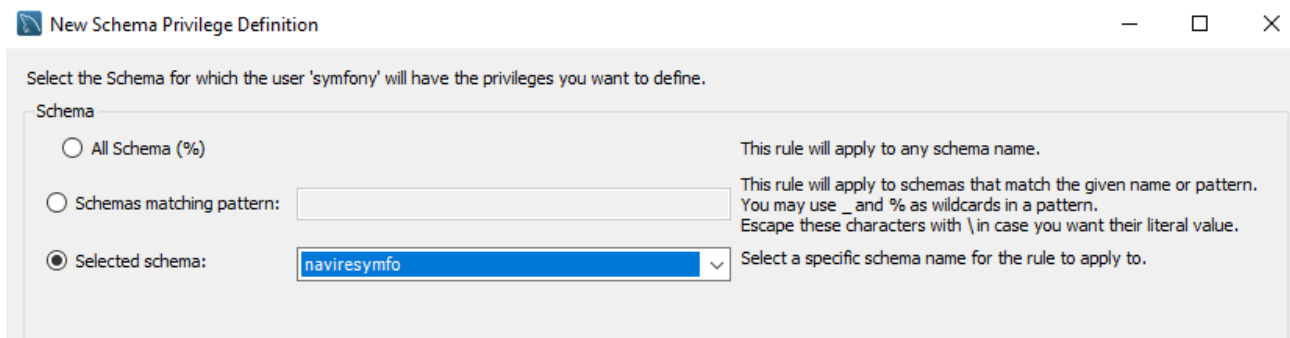
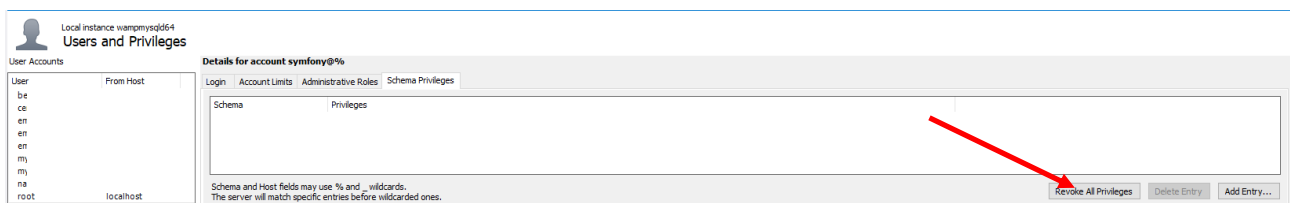
La base de données a été créée :

```
PS T:\Wampsites\CoursSymfony\DeploySymfony> php bin/console doctrine:database:create
Created database 'naviresymfo' for connection named default
PS T:\Wampsites\CoursSymfony\DeploySymfony> |
```

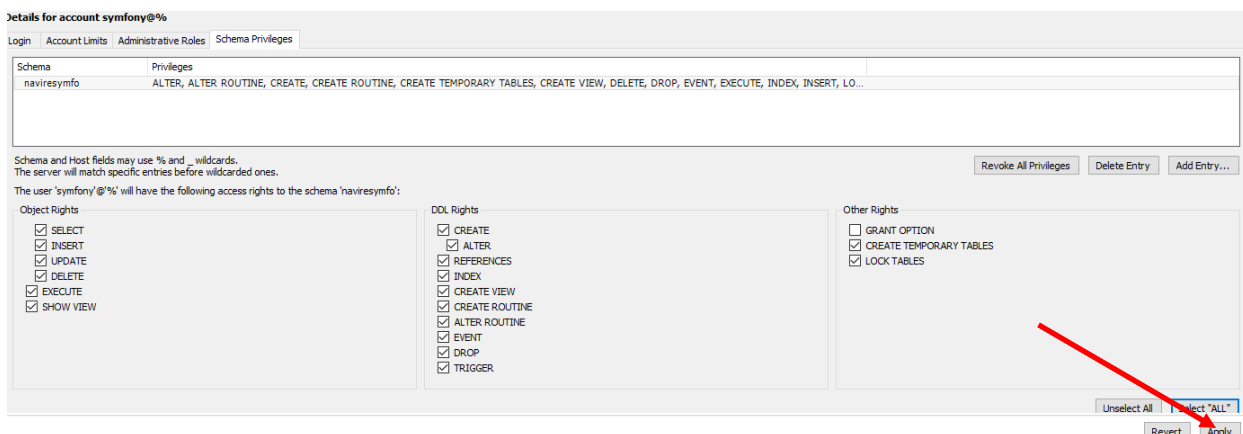
Et dans mysql Workbench :



Vous allez maintenant donner à l'utilisateur symfony tous les privilèges sur cette base :
Dans Mysqlworkbench, connecté en root :



Et cliquez sur OK



Validez.

Vous allez maintenant retirer le privilège DBA à l'utilisateur symfony en décochant la case DBA, cela devrait décocher les autres.

Role	Description
<input type="checkbox"/> DBA	grants the rights to perform all tasks
<input type="checkbox"/> MaintenanceAdmin	grants rights needed to maintain server
<input type="checkbox"/> ProcessAdmin	rights needed to assess, monitor, and kill any user proce...
<input type="checkbox"/> UserAdmin	grants rights to create users logins and reset passwords
<input type="checkbox"/> SecurityAdmin	rights to manage logins and grant and revoke server an...
<input type="checkbox"/> MonitorAdmin	minimum set of rights needed to monitor server
<input type="checkbox"/> DBManager	grants full rights on all databases
<input type="checkbox"/> DBDesigner	rights to create and reverse engineer any database sche...
<input type="checkbox"/> ReplicationAdmin	rights needed to setup and manage replication
<input type="checkbox"/> BackupAdmin	minimal rights needed to backup any database

Partie 2 : MIGRATION DES CLASSES ENTITY CREEES LORS DU TD PRECEDENT

Vous allez maintenant effectuer votre première mise à jour du schéma de la BD. La mise à jour se fera par comparaison du schéma actuel et des classes Entity de votre application Symfony.

Vous allez maintenant mettre à jour la BD

- ✓ générer les ordres SQL de synchronisation des tables par rapport aux entités avec la commande

doctrine:migrations:diff
ou
make:migration



Ces deux commandes sont équivalentes, vous les trouverez indifféremment dans les documentations ou les cours symfony.

- ✓ Exécuter ces ordres

doctrine:migrations:migrate

Commande make:migration :

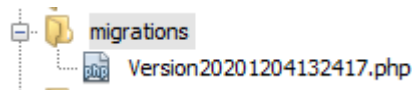
Filter:	make:mi
	(part of command name, wildcards: "?", "*")
Parameters:	
Matching Tasks:	
	make:migration : Creates a new migration based on database changes

La génération du script de mise à jour s'est bien passée :

Success!

Next: Review the new migration "`migrations/Version20201204132417.php`"
Then: Run the migration with `php bin/console doctrine:migrations:migrate`
See <https://symfony.com/doc/current/bundles/DoctrineMigrationsBundle/index.html>
Done.

Et on peut voir le fichier généré des modifications dans le dossier `src/migrations` :



Ce fichier contient les ordres de mise à jour de la base. Vous pouvez aller voir le contenu de ce fichier.



Si vous lisez bien le message on peut voir que l'on peut faire 2 actions sur une migration :

- ✓ L'exécuter pour reporter les modifications sur la BD (`doctrine:migrations:execute --up` => exécutes a single migration version)

```
public function up(Schema $schema) : void
{
    // this up() migration is auto-generated, please modify it to your needs
    $this->addSql('CREATE TABLE ais_ship_type (id INT AUTO_INCREMENT NOT NULL, ais_ship_type INT NOT NULL, libelle VAR
    $this->addSql('CREATE TABLE message (id INT AUTO_INCREMENT NOT NULL, nom VARCHAR(60) NOT NULL, prenom VARCHAR(60) :
    $this->addSql('CREATE TABLE navire (id INT AUTO_INCREMENT NOT NULL, imo VARCHAR(7) NOT NULL, nom VARCHAR(100) NOT :
    $this->addSql('ALTER TABLE navire ADD CONSTRAINT FK_EED1038E62DB837 FOREIGN KEY (idAisShipType) REFERENCES ais_shi
```

- ✓ Annuler une migration (`doctrine:migrations:execute --down`)

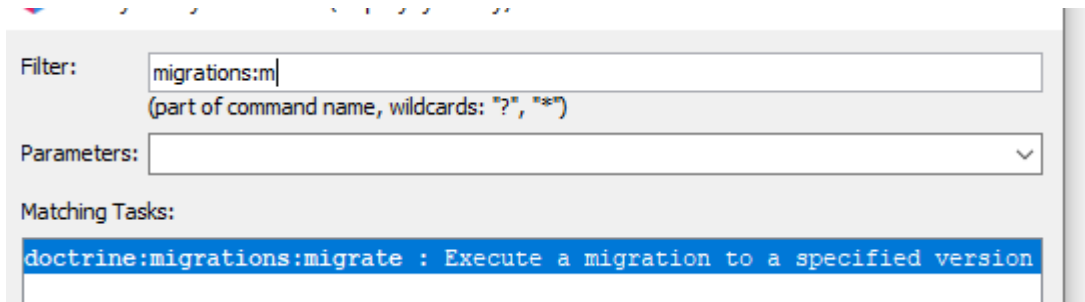
```
public function down(Schema $schema) : void
{
    // this down() migration is auto-generated, please modify it to your needs
    $this->addSql('ALTER TABLE navire DROP FOREIGN KEY FK_EED1038E62DB837');
    $this->addSql('DROP TABLE ais_ship_type');
    $this->addSql('DROP TABLE message');
    $this->addSql('DROP TABLE navire');
}
```

Cette dernière méthode permettra de remonter en arrière sur les migrations effectuées. C'est du versionning !

Exécutez maintenant la commande

`doctrine:migrations:migrate`

Cette commande exécutera toutes les migrations qui ont été générées et qui n'ont pas encore été reportées en BD.

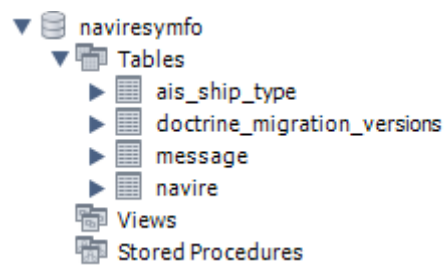


N'oubliez pas de répondre par "y" à la demande de confirmation d'exécution du script :

```
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? (yes/no) [yes]:
> yes
[notice] Migrating up to DoctrineMigrations\Version20201204132417
[notice] finished in 268.7ms, used 16M memory, 1 migrations executed, 4 sql queries
Done.
```

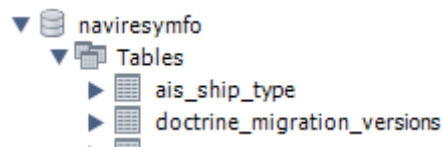
Vous voyez que le fichier de migration a été exécuté avec succès. 4 ordres SQL ont été exécutés.

Ouvrez votre client mysql : MysqlWorkbench ou phpMyAdmin et constatez que la table a été créée dans la bonne BD



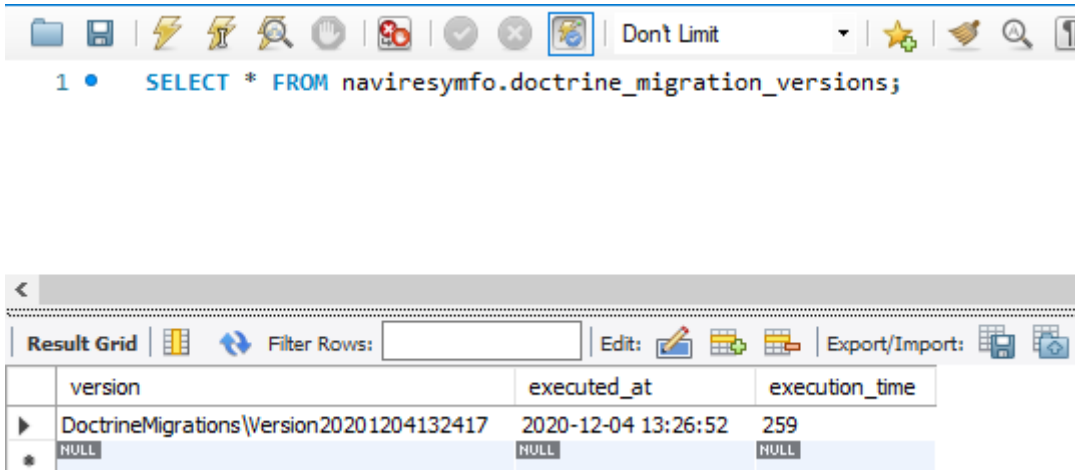
... et elles sont vides, vous pouvez le vérifier : `SELECT * FROM naviresymfo.navire;`

Notez aussi la création de la table des migrations :



Qui elle contient une ligne correspondant à la version de migration effectuée :

Faites un select dessus :



1 • `SELECT * FROM naviresymfo.doctrine_migration_versions;`

version	executed_at	execution_time
DoctrineMigrations\Version20201204132417	2020-12-04 13:26:52	259
NULL	NULL	NULL

La première ligne correspond au nom du fichier des migrations généré par la commande `doctrine:migrations:diff`

On y reviendra.



Remarquez que le nom de la table des types de navire : `ais_ship_type`
.. pas très beau,
on va la renommer grâce aux annotations dans l'entity `AisShipType`

Vous allez apporter la modification suivante à l'annotation de la classe Entity `AisShipType` :

```
/**
 * @ORM\Entity(repositoryClass=AisShipTypeRepository::class)
 * @ORM\Table(name="aisshiptype")
 */
class AisShipType
```

Et vous allez générer un nouveau fichier de migration : `php bin/console make:migration`

Examinez le fichier généré

```
public function up(Schema $schema) : void
{
    // this up() migration is auto-generated, please modify it to your needs
    $this->addSql('ALTER TABLE navire DROP FOREIGN KEY FK_EED1038E62DB837');
    $this->addSql('CREATE TABLE aisshiptype (id INT AUTO_INCREMENT NOT NULL, ais_ship_type INT NOT NULL, libelle V
    $this->addSql('DROP TABLE ais_ship_type');
    $this->addSql('ALTER TABLE navire DROP FOREIGN KEY FK_EED1038E62DB837');
    $this->addSql('ALTER TABLE navire ADD CONSTRAINT FK_EED1038E62DB837 FOREIGN KEY (idAisShipType) REFERENCES ais
```



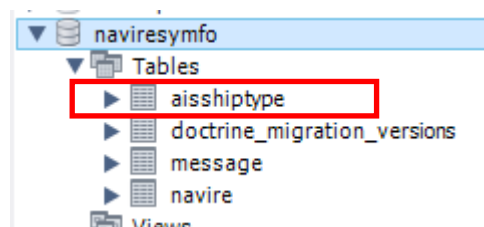
Il y a une erreur !!!
 L'ordre de suppression de la clé étrangère est générée 2 fois
 Commentez ou supprimez le deuxième ordre dans les 2 méthodes de la migration.
 N'oubliez pas d'enregistrer le fichier.
Temps perdu pour moi ... 30mn

Vous pouvez maintenant lancer la migration sur la BD : doctrine:migration:migrate

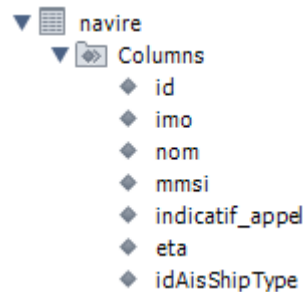
Tout s'est bien passé :

```
[notice] Migrating up to DoctrineMigrations\Version20201204134756
[notice] finished in 215.4ms, used 16M memory, 1 migrations executed, 4 sql queries
```

Et nous avons désormais un nom de table plus présentable !



Voyez plus en détails la table navire :



Doctrine a crée la colonne idAisShipType en tant que clé étrangère dans la table Navire !

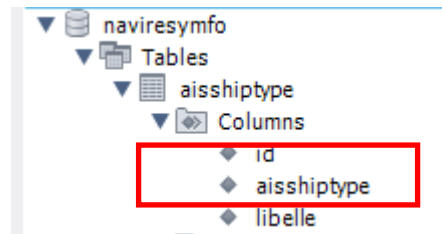


Exercice:

Vous modifierez l'entity AisShipType afin que la colonne générée dans la base ais_ship_type soit renommée en aisshiptype.

```
public function up(Schema $schema) : void
{
    // this up() migration is auto-generated, please modify it to your needs
    $this->addSql('ALTER TABLE aissштиype CHANGE ais_ship_type aissштиype INT NOT NULL');
}
```

Vous devriez obtenir ceci dans le fichier migration :



Prenez l'habitude de renommer tous les attributs et les noms de tables

Bilan :

- ✓ Vous venez de créer votre première relation ManyToOne
- ✓ vous avez vu les répercussions au niveau de la base de données avec la création d'une clé étrangère
- ✓ vous avez appris qu'il faut renommer tous les attributs et les objets

Partie 3 : CREATION DES CLASSES METIER DU PROJET (ENTITIES)

Vous allez dans cette partie créer l'ensemble des classes métier de l'application.

1. Classe Pays



Il va s'agir ici de créer :

- ✓ l'entity Pays
- ✓ la relation unidirectionnelle entre l'entity Navire et l'entity Pays



L'indicatif du pays est constitué de 3 lettres majuscules
 La colonne *indicatif* doit être unique
 On créera un index sur la colonne *indicatif*

Au final, vous devriez avoir ceci :

```
/**
 * @ORM\Entity(repositoryClass=PaysRepository::class)
 * @ORM\Table(
 *     name="pays" ,
 *     uniqueConstraints={@ORM\UniqueConstraint(name="indicatif_unique", columns={"indicatif"})},
 *     indexes={@ORM\Index(name="ind_indicatif", columns={"indicatif"})}
 * )
 */
class Pays
{
    /**
     * @ORM\Id
     * @ORM\GeneratedValue
     * @ORM\Column(type="integer", name="id")
     */
    private $id;

    /**
     * @ORM\Column(type="string", length=60 , name="nom")
     */
    private $nom;

    /**
     * @ORM\Column(type="string", length=3 , name="indicatif")
     * @Assert\Regex(
     *     pattern="/[A-Z]{3}/",
     *     message="L'indicatif Pays a strictement 3 caractères"
     * )
     */
    private $indicatif;
```

Pour rajouter une contrainte à une entité déjà existante vous avez le choix entre :

- ✓ la saisir à la main dans votre IDE préférée
- ✓ la créer via la commande `make entity`

```

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

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

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

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

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

What type of relationship is this?
-----
Type          Description
-----
ManyToOne     Each Navire relates to (has) one Pays.
               Each Pays can relate to (can have) many Navire objects

OneToMany     Each Navire can relate to (can have) many Pays objects.
               Each Pays relates to (has) one Navire

ManyToMany    Each Navire can relate to (can have) many Pays objects.
               Each Pays can also relate to (can also have) many Navire objects

OneToOne      Each Navire relates to (has) exactly one Pays.
               Each Pays also relates to (has) exactly one Navire.
-----

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

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

Do you want to add a new property to Pays so that you can access/update Navire objects from it - e.g. $pays->getNavires()? (yes/no) [yes]:
> no

updated: src/Entity/Navire.php

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

Success!

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

Done.

```

Dans cette partie, on a simplement crée la relation unidirectionnelle de Navire vers Pays.

Dans la question suivante, on va nous demander si on veut une relation bidirectionnelle :

```

Do you want to add a new property to Pays so that you can access/update Navire objects from it - e.g. $pays->getNavires()? (yes/no) [yes]:
> no

```



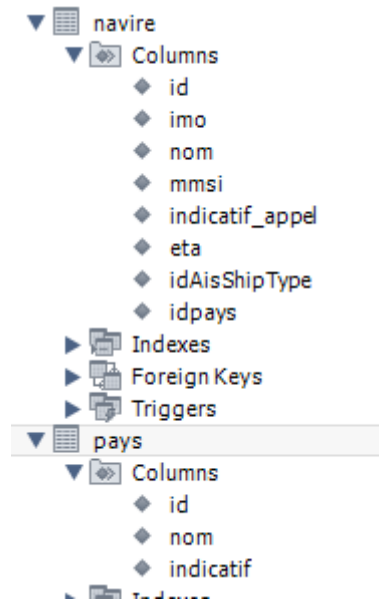
Vous avez répondu no afin de savoir si vous voulez rajouter un attribut à la classe Pays.
On ne veut pas car on veut une relation unidirectionnelle.

Vous modifierez sensiblement le code généré dans l'entity Navire :

```
/**
 * @ORM\ManyToOne(targetEntity=Pays::class)
 * @ORM\JoinColumn(name="idpays", nullable=false)
 */
private $lePavillon;
```

Vous pouvez ici migrer vos modifications dans la base de données.

Voici le résultat dans la base de données :



Observez le code généré dans la migration :

```
'CREATE TABLE pays (id INT AUTO_INCREMENT NOT NULL, nom VARCHAR(60) NOT NULL, indicatif VARCHAR(3) NOT NULL, UNIQUE INDEX indicatif_nique (indicatif),
'ALTER TABLE navire ADD idpays INT NOT NULL);
'ALTER TABLE navire ADD CONSTRAINT FK_EED1038E750CD0E FOREIGN KEY (idpays) REFERENCES pays (id);
'CREATE INDEX IDX_EED1038E750CD0E ON navire (idpays));
```

En plus de la primary Key, un index unique a été créé sur la colonne indicatif de la table pays. Cela assurera donc l'unicité des valeurs des indicatifs pays.



Exercice:

Modifier l'entity navire pour que les numéros IMO et MMSI soient également uniques

On a plusieurs possibilités ici Voici ce que nous allons faire :

- ✓ Créer une contrainte UNIQUE sur la colonne MMSI (On ne peut créer qu'une seule contrainte unique par table)
- ✓ Imposer les valeurs uniques sur la colonne IMO


```
/**
 * @ORM\Entity(repositoryClass=NavireRepository::class)
 * @ORM\Table( name="navire" ,
 *             uniqueConstraints={@ORM\UniqueConstraint(name="mmsi_unique", columns={"mmsi"})}
 * )
 */
class Navire {

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

    /**
     * @ORM\Column(type="string", length=7, unique=true)
     * @Assert\Regex(
     *     pattern="/[1-9]{7}/",
     *     message="Le numéro IMO doit comporter 7 chiffres"
     * )
     */
    private $imo;
```

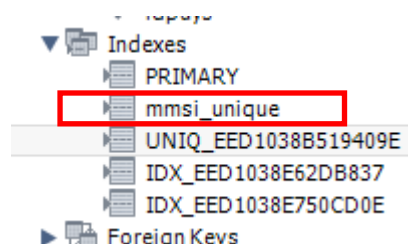
Vous pouvez refaire une migration Cela vous permet de corriger vos erreurs de saisie au fur et à mesure !!!

```
WARNING! You are about to execute a database migration that could result in schema changes
you wish to continue? (yes/no) [yes]:
> y

[notice] Migrating up to DoctrineMigrations\Version20201204160517
[notice] finished in 81.9ms, used 16M memory, 1 migrations executed, 2 sql queries
```



Observez les index de la base de données :



Et plus en détails :

Index: UNIQ_EED1038B519409E

Definition:

Type	BTREE
Unique	Yes
Visible	Yes
Columns	imo

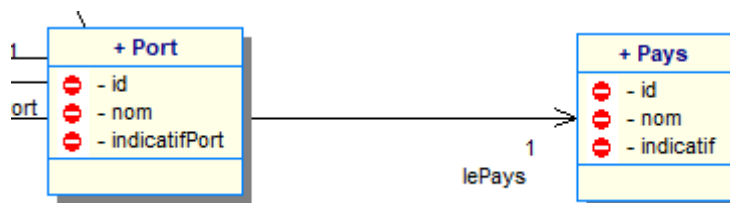
Index: mmsi_unique

Definition:

Type	BTREE
Unique	Yes
Visible	Yes
Columns	mmsi

Les index sont bien uniques

2. Classe Port



Il va s'agir ici de créer :

- ✓ l'entity Port
- ✓ la relation unidirectionnelle entre l'entity Port et l'entity Pays



L'indicatif du port doit être unique
 Il est composé de 5 lettres (les 2 premières représentent le pays
 : <https://www.worldnetlogistics.com/seaport-codes>)

3. classe Escale

```
/**
 * @ORM\Entity(repositoryClass=PortRepository::class)
 * @ORM\Table( name="port" ,
 *             uniqueConstraints={@ORM\UniqueConstraint(name="portindicatif_unique", columns={"indicatif"})}
 * )
 */
class Port
{
    /**
     * @ORM\Id
     * @ORM\GeneratedValue
     * @ORM\Column(type="integer")
     */
    private $id;

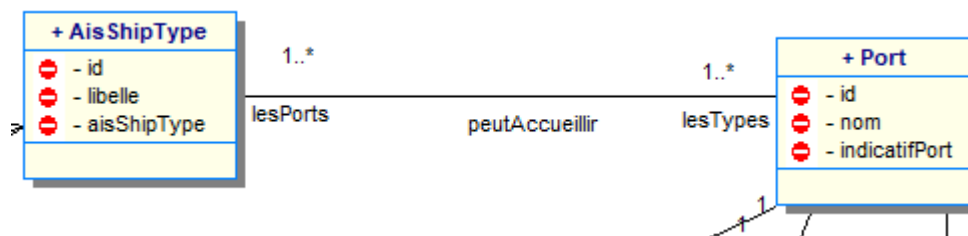
    /**
     * @ORM\Column(type="string", length=60)
     */
    private $nom;

    /**
     * @ORM\Column(type="string", length=5)
     * @Assert\Regex(
     *     pattern="/[A-Z]{5}/",
     *     message="L'indicatif du Port a strictement 5 caractères"
     * )
     */
    private $indicatif;

    /**
     * @ORM\ManyToOne(targetEntity=Pays::class)
     * @ORM\JoinColumn(nullable=false , name="idpays")
     */
    private $lePays;
}
```

4. Association PeutAccueillir

Nous allons nous intéresser maintenant à l'association bidirectionnelle entre la classe AisShipType et la classe Port



Vous allez donc devoir créer
 un attribut lesPorts dans la classe (entity) AisShipType
 un attribut lesTypes dans la classe (entity) port

Ces 2 liens seront reliés entre eux à travers dans les paramètres de l'association afin de permettre d'assurer des mises à jour cohérentes.

Vous allez commencer par modifier l'entity AisShipType par la commande make:entity

```

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

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

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

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

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

What type of relationship is this?
-----
Type      Description
-----
ManyToOne Each AisShipType relates to (has) one Port.
           Each Port can relate to (can have) many AisShipType objects

OneToMany Each AisShipType can relate to (can have) many Port objects.
           Each Port relates to (has) one AisShipType

ManyToMany Each AisShipType can relate to (can have) many Port objects.
           Each Port can also relate to (can also have) many AisShipType objects

OneToOne  Each AisShipType relates to (has) exactly one Port.
           Each Port also relates to (has) exactly one AisShipType.
-----

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

Do you want to add a new property to Port so that you can access/update AisShipType objects from it - e.g. $port->getAisShipTypes()? (yes/no) [yes]:
> yes

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

New field name inside Port [aisShipTypes]:
> lesTypes

updated: src/Entity/AisShipType.php
updated: src/Entity/Port.php

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

Success!

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

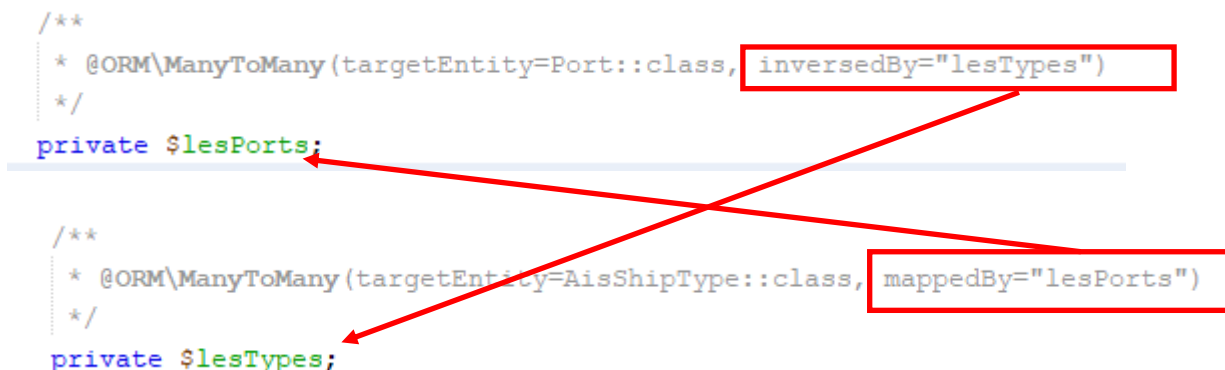
Quand vous avez répondu oui à la question *Do you want to add a new property to Port* alors vous avez demandé à Doctrine de créer une relation bidirectionnelle.

Ce qui a été créé dans l'entity AisShipType et dans l'entity Port :

```

/**
 * @ORM\ManyToMany(targetEntity=Port::class, inversedBy="lesTypes")
 */
private $lesPorts;

/**
 * @ORM\ManyToMany(targetEntity=AisShipType::class, mappedBy="lesPorts")
 */
private $lesTypes;
  
```



Vous voyez le lien entre les 2 entités, doctrine va se charger de maintenir les collections lesPorts et lesTypes à jour.

Je vous propose de générer le fichier migration pour voir ce qu'il va se passer au niveau de la BD :

php bin/console make:migration

Ouvrez le fichier migration généré :

```
public function up(Schema $schema) : void
{
    // this up() migration is auto-generated, please modify it to your needs
    $this->addSql('CREATE TABLE ais_ship_type_port (ais_ship_type_id INT NOT NULL, port_id INT NOT NULL, INDEX IDX_E2C18803479E0B84 (ais_ship_type_id), INDEX IDX_E2C1880376E92A9C (port_id), PRIMARY KEY(ais_ship_type_id, port_id)) DEFAULT CHARACTER SET utf8mb4 COLLATE `utf8mb4_unicode_ci` ENGINE = InnoDB');
    $this->addSql('ALTER TABLE ais_ship_type_port ADD CONSTRAINT FK_E2C18803479E0B84 FOREIGN KEY (ais_ship_type_id) REFERENCES aishiptype (id) ON DELETE CASCADE');
    $this->addSql('ALTER TABLE ais_ship_type_port ADD CONSTRAINT FK_E2C1880376E92A9C FOREIGN KEY (port_id) REFERENCES port (id) ON DELETE CASCADE');
}
```

Si on agrandit :

```
$this->addSql('CREATE TABLE ais_ship_type_port
(ais_ship_type_id INT NOT NULL,
port_id INT NOT NULL,
INDEX IDX_E2C18803479E0B84 (ais_ship_type_id),
INDEX IDX_E2C1880376E92A9C (port_id),
PRIMARY KEY(ais_ship_type_id, port_id))
DEFAULT CHARACTER SET utf8mb4
COLLATE `utf8mb4_unicode_ci`
ENGINE = InnoDB');
```



Vous voyez que doctrine :

- ✓ Va créer une table de jointure appelée ais_ship_type_port
- ✓ composée des colonnes ais_ship_type_id et port_id
- ✓ qui forment à elles la clé primaire
- ✓ et que chacune de ces colonnes est clé étrangère ve la clé primaire respectivement des tables aishiptype et port

Bien entendu, nous n'aurons rien à faire sur cette table, doctrine va se charger de la gérer.

Vous le voyez, les noms de table et de colonne ne sont pas très heureux, aussi je vous propose de rajouter les annotations nécessaires pour changer ces noms.

Vous aurez à modifier l'entity AisShipType :

```
/**
 * @ORM\ManyToMany(targetEntity=Port::class, inversedBy="lesTypes")
 * @ORM\JoinTable(
 *     name="porttypecompatible",
 *     joinColumns={@ORM\JoinColumn(name="idaistype", referencedColumnName="id")},
 *     inverseJoinColumns={@ORM\JoinColumn(name="idport", referencedColumnName="id")}
 * )
 */
private $lesPorts;
```

- ✓ La table de jointure va s'appeler porttypecompatible
- ✓ Les colonnes vont s'appeler idaistype et idport

Vous allez générer un nouveau fichier de migrations :

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

Symfony remarque que vous avez une migration qui n'a pas encore été exécutée pour mettre à jour la base de données. Il vous demande si vous voulez quand même générer le nouveau fichier de migration répondez oui

```
PS T:\Wampsites\CoursSymfony\DeploySymfony> php bin/console make:migration

[WARNING] You have 1 available migrations to execute.

Are you sure you wish to continue? (yes/no) [yes]:
> yes

Success!

Next: Review the new migration "migrations/Version20201205084425.php"
```

Examinez le fichier généré :

```
$this->addSql('CREATE TABLE porttypecompatible (
                idaistype INT NOT NULL,
                idport INT NOT NULL,
                INDEX IDX_2C02FFDB53BA86F9 (idaistype), IN
```

C'est beaucoup mieux. Vous pourriez modifier également le nom des index.

Si vous faites maintenant la migration vers la base de données, les deux migrations non encore exécutées le seront maintenant.

La première migration est inutile puisque la deuxième n'est qu'une amélioration de la deuxième.

Vous allez donc modifier la table des migrations de symfony en lui indiquant que la dernière migration à exécutée est celle que vous ne voulez pas exécuter :

C'est la commande

```
php bin/console doctrine:migrations:version 'App\Migrations\Versionxxx' --add
```

remplacez le Versionxxx par le numéro de l'avant dernière migration générée, c'est-à-dire le nom de la classe de cette migration.

Pour moi :

```
<?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 Version20201204153851 extends AbstractMigration
{
    public function getDescription()
    {
        return '';
    }

    public function up(Schema $schema) : void
    {
        // this up() migration is auto-generated, please modify it to your needs
        $this->addSql('CREATE TABLE pays (id INT AUTO_INCREMENT NOT NULL, nom VARCHAR(60) NOT NULL, indicatif VARCHAR(3) NOT NULL,
        $this->addSql('ALTER TABLE navire ADD idpays INT NOT NULL');
        $this->addSql('ALTER TABLE navire ADD CONSTRAINT FK_EED1038E750CD0E FOREIGN KEY (idpays) REFERENCES pays (id)');
        $this->addSql('CREATE INDEX IDX_EED1038E750CD0E ON navire (idpays)');
    }
}
```

L'espace de noms

Le nom de la classe Migration

Ce sera donc la commande php bin/console doctrine:migrations:version
'DoctrineMigrations\Version20201205081327' --add

Run Symfony Command (DeploySymfony)

Filter:

(part of command name, wildcards: "?", "*")

Parameters: 'DoctrineMigrations\Version20201205081327' --add

Matching Tasks:

- doctrine:migrations:version : Manually add and delete migration version
- doctrine:query:dql : Executes arbitrary DQL directly from the command
- doctrine:query:sql : Executes arbitrary SQL directly from the command
- doctrine:schema:create : Executes (or dumps) the SQL needed to generate the database schema
- doctrine:schema:drop : Executes (or dumps) the SQL needed to drop the database schema
- doctrine:schema:update : Executes (or dumps) the SQL needed to update the database schema
- doctrine:schema:validate : Validate the mapping files
- lint:container : Ensures that arguments injected into services match the container configuration

Usage:

```
doctrine:migrations:version [--add] [--delete] [--all] [--range-from [RANGE-FROM]] [--range-to [RANGE-TO]] [--configuration CONFIGURATION] [--db-configuration DB-CONFIGURATION] [--] [<version>]
```

version

Command: console doctrine:migrations:version 'DoctrineMigrations\Version20201205081327' --add

☐ Keep this dialog opened

Refresh Commands Run Cancel

Configuration		
Storage	Type	Doctrine\Migrations\Metadata\Storage\TableMetadataStorageConfiguration
	Table Name	doctrine_migration_versions
	Column Name	version
Database	Driver	Doctrine\DBAL\Driver\PDO\MySQL\Driver
	Name	naviresymfo
Versions	Previous	DoctrineMigrations\Version20201204223536
	Current	DoctrineMigrations\Version20201205081327
	Next	DoctrineMigrations\Version20201205084425
	Latest	DoctrineMigrations\Version20201205084425
Migrations	Executed	8
	Executed Unavailable	0
	Available	9
	New	1
Migration Namespaces	DoctrineMigrations	T:\Wampsites\CoursSymfony\DeploySymfony\migrations

Un aperçu de ce que l'on peut faire avec les migrations :

```
doctrine:migrations:current : Outputs the current version
doctrine:migrations:diff : Generate a migration by comparing your current database to your mapping information.
doctrine:migrations:dump-schema : Dump the schema for your database to a migration.
doctrine:migrations:execute : Execute one or more migration versions up or down manually.
doctrine:migrations:generate : Generate a blank migration class.
doctrine:migrations:latest : Outputs the latest version
doctrine:migrations:list : Display a list of all available migrations and their status.
doctrine:migrations:migrate : Execute a migration to a specified version or the latest available version.
doctrine:migrations:rollback : Rollup migrations by deleting all tracked versions and insert the one version that exists.
doctrine:migrations:status : View the status of a set of migrations.
doctrine:migrations:sync-metadata-storage : Ensures that the metadata storage is at the latest version.
doctrine:migrations:up-to-date : Tells you if your schema is up-to-date.
doctrine:migrations:version : Manually add and delete migration versions from the version table.
```

Vous allez donc maintenant faire la migration pour créer la table de jointure entre AisShipType et Port :

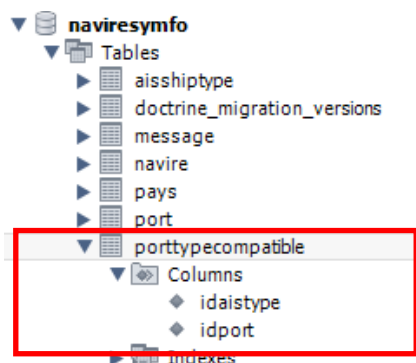
`doctrine:migrations:migrate`

```
"C:\wamp64\bin\php\php7.4.3\php.exe" "T:\Wampsites\CoursSymfony\DeploySymfony\bin\console" "--ansi" "doctrine:migrations:migrate"
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? (yes/no) [yes]:
> yes

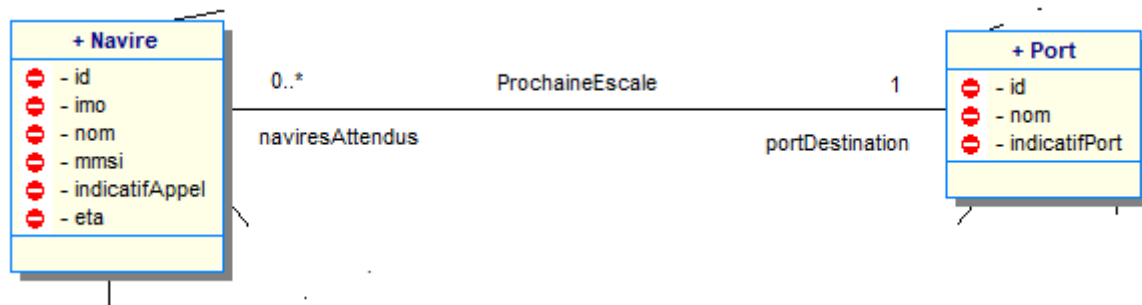
[notice] Migrating up to DoctrineMigrations\Version20201205084425
[notice] finished in 1440.1ms, used 16M memory, 1 migrations executed, 3 sql queries

Done.
```

Tout s'est bien passé, allez vérifier dans la base de données :



5. lePortProchaine escale



Ici il s'agit d'une association **bidirectionnelle ManyToOne** de Navire vers Port.



Il pourra arriver qu'un navire ait pour prochaine destination un Port qui n'est pas encore enregistré dans la base de données.
Dans ce cas, le port de destination du Navire devra être créé en même temps que l'ajout ou la mise à jour d'un navire.

Vous ferez les manipulations au niveau de l'IDE pour arriver au résultat suivant :

```

/**
 * @ORM\ManyToOne(targetEntity=Port::class, inversedBy="naviresAttendus", cascade={"persist"})
 * @ORM\JoinColumn(name="idportdestination", nullable=true)
 */
private $portDestination;

```

Entity Navire

```

/**
 * @ORM\OneToMany(targetEntity=Navire::class, mappedBy="portDestination")
 */
private $naviresAttendus;

```

Entity Port

Après la migration, vous devriez avoir ceci en base de données :

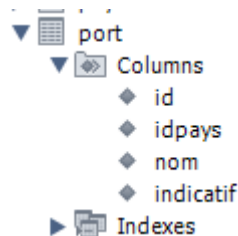
Info	Columns	Indexes	Triggers	Foreign keys	Partitions	Grants	DDL
Column	Type	Default Value	Nullable				
id	int(11)		NO				
imo	varchar(7)		NO				
nom	varchar(100)		NO				
mmsi	varchar(9)		NO				
indicatif_appel	varchar(10)		NO				
eta	datetime		YES				
idAisShipType	int(11)		NO				
idpays	int(11)		NO				
idportdestination	int(11)		YES				

Info	Columns	Indexes	Triggers	Foreign keys	Partitions	Grants	DDL
Name				Schema	Table	Column	
FK_EED1038E62DB837				naviresymfo	navire	idAisShipType	
FK_EED1038E750CD0E				naviresymfo	navire	idpays	
FK_EED1038427CFE1F				naviresymfo	navire	idportdestination	

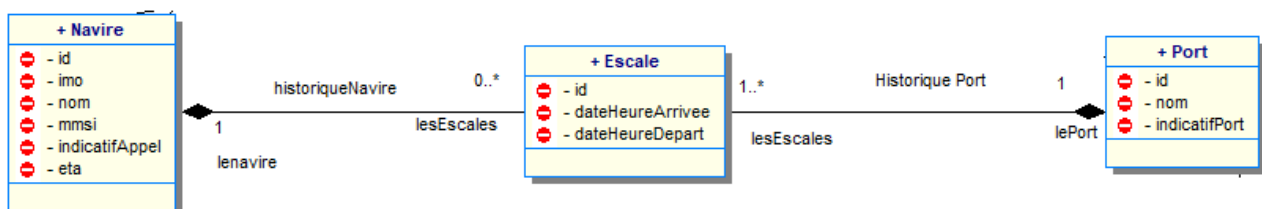


La colonne idportstation a été créée
elle accepte Null.
Et c'est bien une clé étrangère

Il n'y a rien de changé dans la table port.



6. Les escales



Vous remarquerez l'ajout d'une composition sur les Classes Navire et Port. Cela signifie que la classe Escale n'existe que par rapport aux deux classes reliées. Pas d'escale sans Navire et pas d'escale sans Port

Cela aura une conséquence sur les relations que vous allez mettre en place. On va laisser l'application gérer la mise à jour en cascade.

Vous exécuterez la commande `make:entity` pour générer la classe Entity `Escale`.



Vous ferez particulièrement attention quand on vous demandera de répondre à la question suivante à laquelle vous répondrez **yes** !

```
New field name inside Navire [escales]:
> lesEscalaes

Do you want to activate orphanRemoval on your relationship?
A Escale is "orphaned" when it is removed from its related Navire.
e.g. $navire->removeEscale($escale)

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

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

Au final vous devrez obtenir ceci :

```
/**
 * @ORM\Entity(repositoryClass=EscaleRepository::class)
 */
class Escale
{
    /**
     * @ORM\Id
     * @ORM\GeneratedValue
     * @ORM\Column(type="integer")
     */
    private $id;

    /**
     * @ORM\Column(type="datetime" , name="dateheurearrivee")
     */
    private $dateHeureArrivee;

    /**
     * @ORM\Column(type="datetime" , name="dateheuredepart")
     */
    private $dateHeureDepart;

    /**
     * @ORM\ManyToOne(targetEntity=Navire::class, inversedBy="lesEscalaes")
     * @ORM\JoinColumn(nullable=false , name="idnavire")
     */
    private $leNavire;

    /**
     * @ORM\ManyToOne(targetEntity=Port::class, inversedBy="lesEscalaes")
     * @ORM\JoinColumn(nullable=false, name="idPort")
     */
    private $lePort;
```

```
/**
 * @ORM\ManyToOne(targetEntity=Navire::class, inversedBy="lesEscalaes")
 * @ORM\JoinColumn(nullable=false)
 */
private $leNavire;

/**
 * @ORM\ManyToOne(targetEntity=Port::class, inversedBy="lesEscalaes")
 * @ORM\JoinColumn(nullable=false)
 */
private $lePort;
```

Entity Escale : vous retrouverez les rôles du diagramme de classes UML

```
/**
 * @ORM\OneToMany(targetEntity=Escale::class, mappedBy="leNavire", orphanRemoval=true)
 */
private $lesEscalaes;
```

Entity Navire

```
/**
 * @ORM\OneToMany(targetEntity=Escale::class, mappedBy="lePort", orphanRemoval=true)
 */
private $lesEscalaes;
```

Entity Port

Partie 4 : AJOUT DE COLONNES ET MODIFICATION

Finalement, votre product owner vous demande de rajouter les informations suivantes pour gérer les navires :

- ✓ Longueur entier positif
- ✓ Largeur : entier positif
- ✓ tirant d'eau : decimal, 1 chiffre après la virgule.

Par ailleurs, il vous est demandé de préciser le nom de la colonne correspondant à l'attribut indicatifAppel en base de données qui doit s'appeler indicatifappel.

Il vous est demandé de modifier l'entity Navire pour prendre en compte cette évolution du besoin.

Partie 5 : CHARGEMENT DE LA BASE DE DONNEES

Si vous avez bien travaillé, bien respecté les noms et formats de colonnes ainsi que les relations, vous devriez pouvoir exécuter le script fourni par votre formateur pour chargez les données dans la base de données : chargenavire.sql



Symfony étant toujours inachevé, Il ne vous reste plus maintenant qu'à mettre tout ceci en musique !
... Dans le prochain TD



Bravo pour votre travail,