

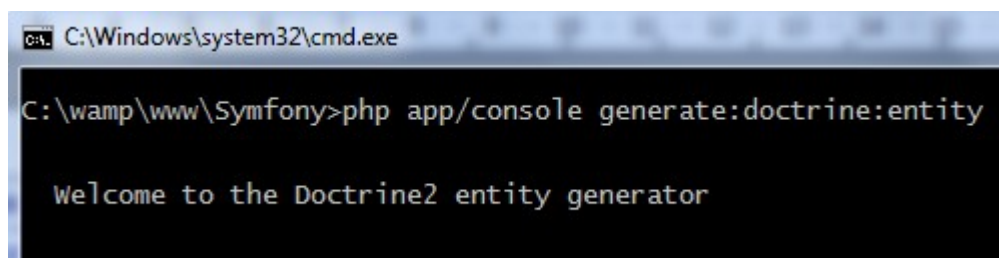
| | | | |
|--|--------|--|-------|
|  SIO-2 | SLAM 5 | SYMPONY 2 : LA COUCHE METIER, L'ORM DOCTRINE | COURS |
| | C07 | | SYMF |

1. QU'EST CE QUE L'ORM DOCTRINE ?

Voir cours n°1 sur symfony.

2. CREATION D'UNE ENTITE

2.1. Création from scratch



```

C:\Windows\system32\cmd.exe

C:\wamp\www\Symfony>php app/console generate:doctrine:entity

Welcome to the Doctrine2 entity generator

```

Il faut préciser :

- le nom de l'entité NORMALISE : **webStageEtudiantBundle:Section**
- le format de configuration : yml, wml, php ou annotation : **annotation**
- le nom de l'attribut : **code**
- le type de chaque attribut:
Available types: array, object, boolean, integer, smallint, bigint, string, text, datetime, datetimetz, date, time, decimal, float, blob : **String** puis longueur de **5**
- creation du repository : **YES**

Résultat :

- 2 fichiers créés dans C:\wamp\www\Symfony\src\webStudent\EtudiantBundle\Entity.
- Le repository contiendra toutes les méthodes interagissant avec la base de données (requêtes DQL ou autres)
- L'ORM a généré l'entité avec tous les getters et setters.
- Il a généré sans qu'on lui demande un attribut id en autoincrement
- Chaque attribut est précédé **d'annotations**. Attention les annotations doivent être dans des commentaires de type « /** », avec précisément deux étoiles.

2.2. L'annotation Column

Les types de colonnes que vous pouvez définir en annotation sont des types Doctrine, *et uniquement Doctrine*. Ne les confondez pas avec leurs homologues SQL ou PHP, ce sont des types à Doctrine seul. Ils font la transition des types SQL aux types PHP (Voir annexe).

Il existe 7 paramètres, tous facultatifs, que l'on peut passer à l'annotation Column afin de personnaliser le comportement. Voici la liste exhaustive dans le tableau suivant.

| Paramètre | Valeur par défaut | Utilisation |
|-----------|-------------------|--|
| type | string | Définit le type de colonne comme nous venons de le voir. |
| name | Nom de l'attribut | Définit le nom de la colonne dans la table. Par défaut, le nom de la colonne est le nom de l'attribut de l'objet, ce qui convient parfaitement. Mais vous pouvez changer le nom de la colonne, par exemple si vous préférez « isExpired » en attribut, mais « is_expired » dans la table. |

| | | | |
|---|---------------|---|--------------|
|  | SLAM 5 | SYMFONY 2 : LA COUCHE METIER, L'ORM DOCTRINE | COURS |
| | C07 | | SYMF |

| Paramètre | Valeur par défaut | Utilisation |
|-----------|-------------------|--|
| length | 255 | Définit la longueur de la colonne. Applicable uniquement sur un type de colonne <code>string</code> . |
| unique | false | Définit la colonne comme unique. Par exemple sur une colonne e-mail pour vos utilisateurs. |
| nullable | false | Permet à la colonne de contenir des <code>NULL</code> . |
| precision | 0 | Définit la précision d'un nombre à virgule, c'est-à-dire le nombre de chiffres en tout. Applicable uniquement sur un type de colonne <code>decimal</code> . |
| scale | 0 | Définit le <i>scale</i> d'un nombre à virgule, c'est-à-dire le nombre de chiffres après la virgule. Applicable uniquement sur un type de colonne <code>decimal</code> . |

2.3. Régénérer une entité existante

```
C:\wamp\www\Symfony>php app/console generate:doctrine:entities webStudentEtudiantBundle:Etudiant
Generating entity "webStudent\EtudiantBundle\Entity\Etudiant"
> backing up Etudiant.php to Etudiant.php~
> generating webStudent\EtudiantBundle\Entity\Etudiant
```

2.4. Les constructeurs php

Il est possible lorsqu'il n'existe pas d'ajouter dans la classe un constructeur afin par exemple, de valoriser des propriétés dès l'instantiation d'un objet

Exemple :

```
class Article
{
    // La définition des attributs...

    public function __construct()
    {
        $this->date = new \Datetime(); // Par défaut, la date de l'article est la date d'aujourd'hui
    }
    ...
}
```

3. CREER LES TABLES EN BASE DE DONNEES A PARTIR DE SES ENTITES

3.1. CREATION DE LA BASE DE DONNES

Avec phpMyadmin (ou autre méthode), créer la base de données qui persistera vos entités. (ex : webstudent)

3.2. CONFIGURATION DE LA BDD

Changer les paramètres Dans le fichier `app/config/parameters.yml`

| | | | |
|--|--------|---|-------|
|  SIO-2 | SLAM 5 | SYMPHONY 2 : LA COUCHE METIER, L'ORM DOCTRINE | COURS |
| | C07 | | SYMF |

3.3. INSTRUCTIONS DOCTRINE PERMETTANT D'AGIR SUR LA BDD

Pour voir les requêtes que Doctrine s'apprête à exécuter :

```
C:\wamp\www\Symfony>php app/console doctrine:schema:update --dump-sql
```

Vous venez de créer un ou plusieurs entités. Pour créer les tables correspondantes en bdd :

```
C:\wamp\www\Symfony>php app/console doctrine:schema:update --force
```

En résumé :

```
C:\wamp\www\Symfony>php app/console doctrine:schema:update --dump-sql
ALTER TABLE section ADD nbEtudiant INT NOT NULL

C:\wamp\www\Symfony>php app/console doctrine:schema:update --force
Updating database schema...
Database schema updated successfully! "1" queries were executed
```

4. INSERER DES DONNEES A PARTIR DES ENTITES

Dans le controleur ajouter la méthode ci-dessous.

Ne pas oublier d'ajouter le use avant la déclaration de la classe

```
use webStudent\EtudiantBundle\Entity\Section;

public function ajouterSectionAction()
{
    // Etape 0 - creation de l'objet Section
    $section = new Section();
    $section->setCode('CGO');
    $section->setNom('BTS Comptabilité');
    $section->setNbEtudiant(64);

    // Etape 1 On récupère l'EntityManager
    $em = $this->getDoctrine()->getManager();

    // Étape 2 : On « persiste » l'entité
    $em->persist($section);

    // Étape 3 : On « flush » tout ce qui a été persisté avant
    $em->flush();

    return $this->render('webStageEtudiantBundle:Etudiant:index.html.twig');
}
```

EXPLICATIONS

- Etape 0 : on crée l'objet section. Par la suite on utilisera des formulaires
- Etape 1: On récupère l'Entity Manager. Le service Doctrine est celui qui va nous permettre de gérer la persistance de nos objets. Ce service est accessible depuis le contrôleur comme n'importe quel service : `$doctrine = $this->getDoctrine();` Mais le service que l'on va utiliser le plus est l'entityManager qui gère la partie ORM de Doctrine : `$em = $this->getDoctrine()->getManager();`
- L'étape 2 dit à Doctrine de « persister » l'entité. Cela veut dire qu'à partir de maintenant cette entité (qui n'est qu'un simple objet !) est gérée par Doctrine. Cela n'exécute pas encore de requête SQL, ni rien d'autre.
- L'étape 3 dit à Doctrine d'exécuter effectivement les requêtes nécessaires pour sauvegarder les

| | | | |
|---|---------------|---|--------------|
|  | SLAM 5 | SYMFONY 2 : LA COUCHE METIER, L'ORM DOCTRINE | COURS |
| | C07 | | SYMF |

entités qu'on lui a dit de persister ;notre Section étant maintenant enregistré en base de données grâce au flush(), Doctrine2 lui a attribué un id !

4.1. AUTRES METHODES UTILES DE L'ENTITY MANAGER

Il existe un certain nombre de méthodes permettant de gérer ses entités grâce à l'entity manager.

- clear(\$nomEntite)** annule tous les persist() effectués. Si le nom d'une entité est précisé (son namespace complet ou son raccourci), seuls les persist() sur des entités de ce type seront annulés.

```
<?php
$em->persist($etudiant);
$em->persist($section);
$em->clear();
$em->flush(); // N'exécutera rien, car les deux persists sont annulés par le clear
```

- detach(\$entite)** annule le persist() effectué sur l'entité en argument. Au prochain flush(), aucun changement ne sera donc appliqué à l'entité.

```
<?php
$em->persist($etudiant);
$em->persist($section);
$em->detach($etudiant);
$em->flush(); // Enregistre $section mais pas $etudiant
```

- contains(\$entite)** retourne true si l'entité donnée en argument est gérée par l'EntityManager (s'il y a eu un persist() sur l'entité donc).

```
<?php
$em->persist($etudiant);
var_dump($em->contains($etudiant)); // Affiche true
var_dump($em->contains($section)); // Affiche false
```

- refresh(\$entite)** met à jour l'entité donnée en argument dans l'état où elle est en base de données. Cela écrase et donc annule tous les changements qu'il a pu y avoir sur l'entité concernée.

```
<?php
$article->setNom('nouveau nom');
$em->refresh($etudiant);
var_dump($etudiant->getNom()); // Affiche « Un ancien nom »
```

- remove(\$entite)** supprime l'entité donnée en argument de la base de données. Effectif au prochain flush().

```
<?php
$em->remove($etudiant);
$em->flush(); // Exécute un DELETE sur $etudiant
```

5. RECUPERER UNE ENTITE

```
public function consulterSectionAction($id)
{
    $repository = $this->getDoctrine()
        ->getManager()
        ->getRepository('webStageEtudiantBundle:Section');

    // On récupère l'entité correspondant à l'id $id
    $section = $repository->find($id);

    // Ou null si aucune section n'a été trouvée avec l'id $id
    if($section === null)
    {
        throw $this->createNotFoundException('Section[id='.$id.'] inexistant.');
```

| | | | |
|---|--------|--|-------|
|  | SLAM 5 | SYMPONY 2 : LA COUCHE METIER, L'ORM DOCTRINE | COURS |
| | C07 | | SYMF |

```

return $this->render('webStageEtudiantBundle:Etudiant:index.html.twig', array(
    'id' => $section->getNom()
));
}

```

➤ TRAVAIL A FAIRE

Créer toutes vos entités indépendamment les unes des autres. Ne les générez pas en base de données.

6. LES RELATIONS ENTRE ENTITES

6.1. L'HERITAGE

Un etudiant et un enseignant héritent des propriétés d'Utilisateur. Un utilisateur est lié à une section. Doctrine gère l'héritage de 2 façons différentes :

Cas 1 : 1 seule table créée en bdd

```

* @ORM\InheritanceType("SINGLE_TABLE")
* @ORM\DiscriminatorColumn(name="type", type="string")
* @ORM\DiscriminatorMap({"etudiant" = "Etudiant", "enseignant" = "Enseignant"})
*/

```

Cas 2 : 3 tables créées en bdd.

```

/**
 * @Entity
 * @InheritanceType("JOINED")
 * @DiscriminatorColumn(name="discr", type="string")
 * @DiscriminatorMap({"person" = "Person", "employee" = "Employee"})
 */

```

Etape 1) Ajouter les annotations du cas 2 dans Utilisateur.

Etape 2) Régénérer les entités

Etape 3) Régénérer la bdd

6.2. RELATION MANY-TO-ONE

Exemple : Un étudiant (utilisateur) est rattaché à une seule section mais une section inclus plusieurs utilisateurs.

Etape 0) Les entités utilisateurs et section sont créées.

Etape 1) Dans l'entité Utilisateur, ajouter la référence de section

```

/**
 * @ORM\ManyToOne(targetEntity="webStudent\EtudiantBundle\Entity\Section")
 * @ORM\JoinColumn(nullable=false)
 */
private $section;

```

Etape 2) générer les accesseurs

```

C:\wamp\www\Symfony>php app/console generate:doctrine:entities webStudentEtudiantBundle:Etudiant
Generating entity "webStudent\EtudiantBundle\Entity\Etudiant"
> backing up Etudiant.php to Etudiant.php~
> generating webStudent\EtudiantBundle\Entity\Etudiant

```

| | | | |
|--|--------|--|-------|
|  SIO-2 | SLAM 5 | SYMPONY 2 : LA COUCHE METIER, L'ORM DOCTRINE | COURS |
| | C07 | | SYMF |

Etape 3) mettre à jour la base de données

```
C:\wamp\www\Symfony>php app/console doctrine:schema:update --force
Updating database schema...
Database schema updated successfully! "3" queries were executed
```

Doctrine crée le champ supplémentaire AVEC la contrainte d'intégrité référentielle!

6.3. INVERSER LA RELATION MANY-TO-ONE

Si l'on souhaite partir de section afin de récupérer la listes des utilisateurs rattachés à une section

etape 1) déclaration de la liste d'utilisateurs dans section

```
/**
 * @ORM\OneToMany(targetEntity="webStudent\EtudiantBundle\Entity\Utilisateur",
mappedBy="utilisateur")
 * @ORM\JoinColumn(nullable=false)
 */
private $utilisateur;
```

etape 2) Il faut également adapter l'entité propriétaire, pour lui dire que maintenant la relation est de type bidirectionnelle et non plus unidirectionnelle. Pour cela, il faut rajouter le paramètre `inversedBy` dans l'annotation *Many-To-One* :

```
/**
 *
 * @ORM\ManyToOne(targetEntity="webStudent\EtudiantBundle\Entity\Section", inversedBy="utilisateurs")
 * @ORM\JoinColumn(nullable=false)
 */
private $section;
```

Etape 3) régénérer les entités.

6.4. RELATION ONE-TO-ONE

Exemple : un étudiant est rattaché à une seule adresse et à une adresse correspond un seul étudiant.

```
/**
 * @ORM\OneToOne(targetEntity="webStage\EtudiantBundle\Entity\Adresse", cascade={"persist"})
 */
private $adresse;
```

Note : il est possible de rendre une relation obligatoire (par défaut, elle est facultative)

Ajouter

```
* @ORM\JoinColumn(nullable=false)
```

6.5. RELATION MANY-TO-MANY AVEC OU SANS ATTRIBUTS

Dans ce cas, Doctrine créera une entité intermédiaire avec ou sans attribut.

Voir la doc.

➤ TRAVAIL A FAIRE

Créer les relations entre vos entités (et donc vos tables en bdd)

7. RECUPERER SES ENTITES AVEC LES DONNEES DE LA BDD

7.1. LE REPOSITORY

Le repository est le système qui permet de récupérer les objets. Toute requête SQL ne doit pas avoir lieu ailleurs. On va donc y construire des méthodes pour récupérer une entité par son id, pour récupérer une liste d'entités suivant un critère spécifique, etc. Bref, à chaque fois que vous devez récupérer des entités dans votre base de données, vous utiliserez le repository de l'entité correspondante. Il existe un repository par entité. Cela permet de bien organiser son code. Bien sûr, cela n'empêche pas qu'un repository utilise plusieurs types d'entité, dans le cas d'une jointure par exemple.

7.2. LES METHODES DE RECUPERATION DE BASE

| | |
|--|---|
| find(\$id) récupère l'entité correspondant à l'id | <pre> \$repository = \$this->getDoctrine() ->getManager() ->getRepository('webStageEtudiantBundle:Section'); // On récupère l'entité correspondant à l'id \$id \$section = \$repository->find(\$id); </pre> |
| findAll() retourne toutes les entités et retourne un tableau | <pre> \$repository = \$this->getDoctrine() ->getManager() ->getRepository('webStageEtudiantBundle:Stage'); \$listeStages = \$repository->findAll(); foreach(\$ listeStages as \$stage) { // \$stage est une instance de Stage echo \$stage->getLibelle(); } ou dans une vue Twig {% for stage in listeStages %} {{ stage.libelle }} {% endfor %} </pre> |
| findBy() findBy(array \$criteres, array \$orderBy = null, \$limite = null, \$offset = null) | <pre> \$repository = \$this->getDoctrine() ->getManager() ->getRepository('webStageEtudiantBundle:Stage'); \$listeStages = \$repository->findBy(array('section_id' => 2), array('dateDebut' => '2013-01-20'), 5, 0); foreach(\$listeStages as \$stage) { // \$stage est une instance de Stage echo \$stage->getLibelle(); } </pre> |
| findOneBy(array \$criteres) fonctionne sur le même principe que la méthode findBy() mais ne retourne qu'une seule entité | |

| | | | |
|--|--------|---|-------|
|  SIO-2 | SLAM 5 | SYMPHONY 2 : LA COUCHE METIER, L'ORM DOCTRINE | COURS |
| | C07 | | SYMF |

| | |
|---|--|
| findByX(\$valeur) Cette méthode fonctionne comme findBy(), sauf que vous ne pouvez mettre qu'un seul critère, celui du nom de la méthode. | <pre>\$repository = \$this->getDoctrine()->getManager()->getRepository('webStudentEtudiantBundle:Etudiant'); \$listeEtudiant = \$repository->findByNom('titi');</pre> <p>// \$listeEtudiant est un array qui contient tous les etudiants correspondant à titi // possibilité de récupérer avec un objet lié = findBySection(\$section)</p> |
| findOneByX(\$valeur) | La même méthode que ci-dessus mais ne ramène qu'un objet. |

7.3. METHODES SPECIFIQUES

Exemple 1 : dans le repository d'etudiant

```
public function rechercherEtudiant($pNom)
{
    // Création de la requête à l'aide du QueryBuilder
    $queryBuilder = $this->_em->createQueryBuilder()
        ->select('e')
        ->where('e.nom like :leNom')
        ->setParameter('leNom', $pNom)
        ->orderBy('e.nom', 'ASC')
        ->from($this->_entityName, 'e');

    // On récupère la Query à partir du QueryBuilder
    $query = $queryBuilder->getQuery();

    // On récupère les résultats à partir de la Query
    $resultats = $query->getResult();

    // On retourne ces résultats dans un tableau
    return $resultats;
}
```

Exemple 2 = equivalent de l'exemple 1 en simplifié

```
public function rechercherEtudiant($pNom)
{
    // Création de la requête
    $queryBuilder = $this->createQueryBuilder('e')
        ->where('e.nom like :leNom')
        ->setParameter('leNom', $pNom)
        ->orderBy('e.nom', 'ASC') ;

    // On récupère la Query à partir du QueryBuilder
    $query = $queryBuilder->getQuery();

    // On récupère les résultats à partir de la Query
    $resultats = $query->getResult();

    // On retourne ces résultats
    return $resultats;
}
```


| | | | |
|--|--------|--|-------|
|  SIO-2 | SLAM 5 | SYMFONY 2 : LA COUCHE METIER, L'ORM DOCTRINE | COURS |
| | C07 | | SYMF |

8. ANNEXE : TABLEAU DE CORRESPONDANCE DES TYPES DOCTRINE

Voici dans le tableau suivant la liste exhaustive des types Doctrine2 disponibles.

| Type Doctrine | Type SQL | Type PHP | Utilisation |
|------------------|----------|------------------------|--|
| string | VARCHAR | string | Toutes les chaînes de caractères jusqu'à 255 caractères. |
| integer | INT | integer | Tous les nombres jusqu'à 2 147 483 647. |
| smallint | SMALLINT | integer | Tous les nombres jusqu'à 32 767. |
| bigint | BIGINT | string | Tous les nombres jusqu'à 9 223 372 036 854 775 807. Attention, PHP reçoit une chaîne de caractères, car il ne supporte pas un si grand nombre (suivant que vous êtes en 32 ou en 64 bits). |
| boolean | BOOLEAN | boolean | Les valeurs booléennes true et false. |
| decimal | DECIMAL | double | Les nombres à virgule. |
| date ou datetime | DATETIME | objet DateTime | Toutes les dates et heures. |
| time | TIME | objet DateTime- | Toutes les heures. |
| text | CLOB | string | Les chaînes de caractères de plus de 255 caractères. |
| object | CLOB | Type de l'objet stocké | Stocke un objet PHP en utilisant serialize/unserialize. |
| array | CLOB | array | Stocke un tableau PHP en utilisant serialize/unserialize. |
| float | FLOAT | double | Tous les nombres à virgule. Attention, fonctionne uniquement sur les serveurs dont la locale utilise un point comme séparateur. |

Les types Doctrine sont sensibles à la casse. Ainsi, le type « String » n'existe pas, il s'agit du type « string ».