|  |
| --- |
| Symfony en Databases |
| De 1-n relatie. |
|  |
| Voor de examens K2 en K4 |
|  |

Symfony en Databases

De 1-n relatie. Ga naar de Symfony website, onderdeel Databases



1. Maak een leeg website project aan.



1. Configureer de database, pas het **.env** bestand aan voor

* gebruiker “root”
* wachtwoord “”
* database “database\_example”

1. Maak de database aan.



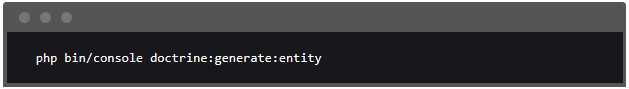
1. De volgende ERD gaan we realiseren.



Each **User** must works in one and only one **Afdeling**

Each **Afdeling** may be responsible for one or more **User**s

De User Entity is nog niet aangemaakt, deze maken we aan met



Geef antwoord op de gestelde vragen en de User Entity wordt aangemaakt!

(let op: “roles”=>json en “salaris”=>decimal(10,2) alle andere type string)

Class name of the entity to create or update (e.g. GentleKangaroo):

> User

created: src/Entity/User.php

created: src/Repository/UserRepository.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):

> username

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

>

Field length [255]:

>

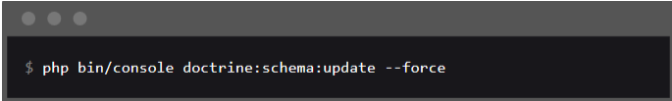
Can this field be null in the database (nullable) (yes/no) [no]:

>

updated: src/Entity/User.php

Let op, alleen de velden zijn toegevoegd. Maak zelf de getters en setters.

Vervolgens even de database updaten



**src\Entity\User.php**

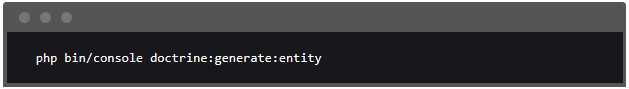
**<?php  
  
namespace** App\Entity;  
  
**use** Doctrine\ORM\Mapping **as** ORM;  
  
*/\*\*  
 \** ***@ORM\Entity****(repositoryClass="App\Repository\UserRepository")  
 \*/***class** User  
{  
 */\*\*  
 \** ***@ORM\Id****()  
 \** ***@ORM\GeneratedValue****()  
 \** ***@ORM\Column****(type="integer")  
 \*/* **private $id**;  
  
 */\*\*  
 \** ***@ORM\Column****(type="string", length=255)  
 \*/* **private $username**;  
  
 */\*\*  
 \** ***@ORM\Column****(type="string", length=255)  
 \*/* **private $password**;  
  
 */\*\*  
 \** ***@ORM\Column****(type="json")  
 \*/* **private $roles** = [];  
  
 */\*\*  
 \** ***@ORM\Column****(type="string", length=255)  
 \*/* **private $naam**;  
  
 */\*\*  
 \** ***@ORM\Column****(type="decimal", precision=10, scale=2)  
 \*/* **private $salaris**;  
  
 **public function** getId(): ?int  
 {  
 **return** $this->**id**;  
 }  
  
 **public function** getUsername(): ?string  
 {  
 **return** $this->**username**;  
 }

…

Doctrine gaat kijken wat het **type** van het veld is om vervolgens dit om te zetten naar een mysql-type. zie onderstaande tabel.



De Afdeling Entity is nog niet aangemaakt, deze maken we aan met



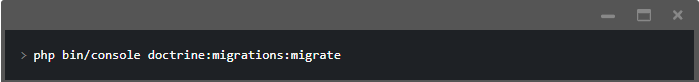
Geef antwoord op de gestelde vragen en de Afdeling Entity wordt aangemaakt!

1. Voordat de tabellen aangemaakt worden moet eerst aangegeven worden wat de versie van de databaseserver is. Op basis hiervan maakt doctrine sql-code aan. Aangezien onze db-versie geen json ondersteunt moeten we dat aangeven door serverVersion (in .env) op 5.6 te zetten. Voor json wordt nu LONGTEXT gebruikt. Dit begrijpt de database wel



Maak de tabellen aan in de database





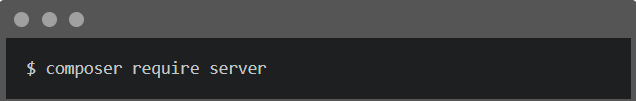
Controleer in de database of de tabellen aangemaakt zijn! Naast de entity maakt dit commando ok de repository klasse aan.

Nu gaan we een rij toevoegen aan de afdeling tabel.

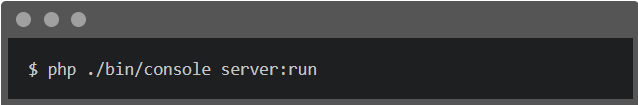
Binnen Doctrine zijn tabellen classes en rijen objecten. Maak een actie aan die een rij toevoegt. Kies zelf een controller.

**<?php  
  
namespace** App\Controller;  
**use** App\Entity\Afdeling;  
**use** Symfony\Bundle\FrameworkBundle\Controller\AbstractController;  
**use** Symfony\Component\HttpFoundation\Response;  
**use** Symfony\Component\Routing\Annotation\Route;  
  
  
**class** BezoekerController **extends** AbstractController  
{  
 */\*\*  
 \** ***@Route****("/create", name="create")  
 \*/* **public function** createAction()  
 {  
 $entityManager=$this->getDoctrine()->getManager();  
  
 $afdeling=**new** Afdeling();  
 $afdeling->setNaam(**"Inkoop"**);  
 $afdeling->setLocatie(**"Amsterdam"**);  
  
 *//dit object wil ik in de tabel afdeling opslaan (nog geen SQL)* $entityManager->persist($afdeling);  
 *//nu wordt er SQL uitgevoerd* $entityManager->flush();  
 **return new** Response(**'Afdeling is opgeslagen met id'**.$afdeling->getId());  
 }  
}

Installeer een interne webserver



Start de server



Voer deze actie uit en controleer de tabel of er een rij toegevoegd is.



Doctrine Associations / Relations

De volgende stap is het maken van de relatie, hiervoor krijgen beide entiteiten een extra veld.

De relatie regels laten we maken door make:entity

C:\xampp\htdocs\database\_example>php bin/console make:entity

Class name of the entity to create or update (e.g. FierceElephant):

> **User**

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

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

> **afdeling**

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

> **relation**

What class should this entity be related to?:

> **afdeling**

What type of relationship is this?

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

Type Description

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

ManyToOne Each User relates to (has) one afdeling.

Each afdeling can relate to (can have) many User objects

OneToMany Each User can relate to (can have) many afdeling objects.

Each afdeling relates to (has) one User

ManyToMany Each User can relate to (can have) many afdeling objects.

Each afdeling can also relate to (can also have) many User objects

OneToOne Each User relates to (has) exactly one afdeling.

Each afdeling also relates to (has) exactly one User.

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

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

> ManyToOne

Is the User.afdeling property allowed to be null (nullable)? (yes/no) [yes]:

>

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

>

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

New field name inside afdeling [users]:

>

updated: src/Entity/User.php

updated: src/Entity/Afdeling.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 make:migration

het volgende is toegevoegd aan de User Entity

*/\*\*  
 \** ***@ORM\ManyToOne****(targetEntity="App\Entity\afdeling", inversedBy="users")  
 \*/***private $afdeling**;

**public function** getAfdeling(): ?afdeling  
{  
 **return** $this->**afdeling**;  
}  
  
**public function** setAfdeling(?afdeling $afdeling): **self**{  
 $this->**afdeling** = $afdeling;  
  
 **return** $this;  
}

het volgende is toegevoegd aan de Afdeling Entity

*/\*\*  
 \** ***@ORM\OneToMany****(targetEntity="App\Entity\User", mappedBy="afdeling")  
 \*/***private $users**;  
  
**public function** \_\_construct()  
{  
 $this->**users** = **new** ArrayCollection();  
}

*/\*\*  
 \** ***@return*** *Collection|User[]  
 \*/***public function** getUsers(): Collection  
{  
 **return** $this->**users**;  
}

De **afdeling** eigenschap in de User Entity bevat een afdeling object. Dit is zijn afdeling.

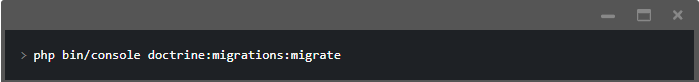
De **users** eigenschap in de Afdeling Entity bevat de users objecten, dit is een array. Dit zijn alle users op die afdeling. (Importeer zelf de ArrayCollection)

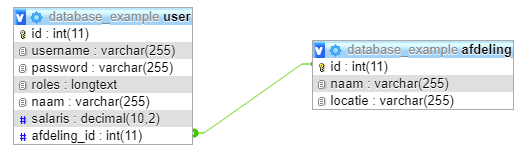
Nu ga je testen of de relatie goed is gemaakt.

Nu nog even de veranderingen in de database opslaan.

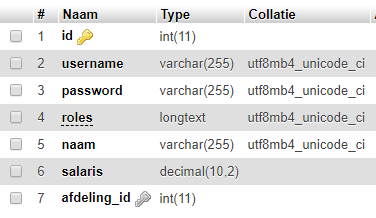


Als er aangegeven wordt dat er niks veranderd is, gooi dan de cache map weg.





De users-tabel zien er nu als volgt uit:



afdeling\_id is de vreemde sleutel.

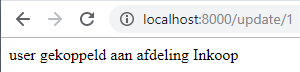
Maak in de user tabel 1 rij aan m.b.v phpmyadmin



Het veld afdeling\_id heeft nog geen waarde. Nu gaan we deze koppelen aan een afdeling. Hiervoor maken we een actie aan.

*/\*\*  
 \** ***@Route****("/update/{userId}", name="update")  
 \*/***public function** updateAction($userId)  
{  
 $entityManager=$this->getDoctrine()->getManager();  
 $user=$entityManager->getRepository(User::***class***)->find($userId);  
  
 **if**(!$userId)  
 {  
 **throw** $this->createNotFoundException(**"Geen user gevonden"**);  
 }  
 $afdeling=$entityManager->getRepository(Afdeling::***class***)->findOneBy([**'naam'**=>**'Inkoop'**]);  
 $user->setAfdeling($afdeling);  
 $entityManager->flush();  
  
 **return new** Response(**"user gekoppeld aan afdeling Inkoop"**);  
}

Voer de actie uit



Bekijk nu de rij in de Users tabel



Repository Classes

Als je zelf queries wil maken doen we dat niet in de Controller, maar in de Repository class van de Entiteit.

Graag wil je weten welke users nog geen afdeling hebben.

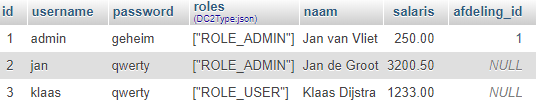
Maak eerst in de user repository de getNoAfdeling() methode aan.

**class** UserRepository **extends** ServiceEntityRepository  
{  
 **public function** \_\_construct(ManagerRegistry $registry)  
 {  
 **parent**::*\_\_construct*($registry, User::***class***);  
 }  
  
 **public function** getNoAfdeling()  
 {  
 $em=$this->getEntityManager();  
  
 $query=$em->createQuery(**"SELECT u FROM App:User u WHERE u.afdeling IS NULL "**);  
  
 *//dump($query->getResult());die();* **return** $query->getResult();  
 }

De query wordt niet gemaakt met SQL maar met DQL. De Doctrine Query Language. Het voordeel is dat je een objecten verzameling terugkrijgt.

<http://docs.doctrine-project.org/projects/doctrine-orm/en/latest/reference/dql-doctrine-query-language.html>

Nu kan in de actie deze methode gebruikt worden. Maak de volgende gebruikers aan



*/\*\*  
 \** ***@Route****("/getnoafdeling", name="no\_afdeling")  
 \*/***public function** getNoAfdeling()  
{  
 $users = $this->getDoctrine()  
 ->getRepository(User::***class***)  
 ->getNoAfdeling();  
  
 **return** $this->render(**'bezoeker/geenAfdeling.html.twig'**,[**'users'**=>$users]);  
}

De twig template ziet er als volgt uit.

{% **extends 'base.html.twig'** %}  
{% **block** body %}  
 <**h1**>Users zonder afdeling</**h1**>  
 <**ul**>  
 {% **for** user **in** users %}  
 <**li**>  
 {{ user.naam }}  
 </**li**>  
 {% **endfor** %}  
 </**ul**>  
{% **endblock** %}

