**Symfony lab6**

**Gheorghita Pavel IA1902**

**php bin/console make:entity Category**

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

> name

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

> string

Field length [255]:

> 255

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

> no

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

>

(press enter again to finish)

Это создаст новый класс сущностей:

<?php

// src/Entity/Category.php

namespace App\Entity;

use App\Repository\CategoryRepository;

use Doctrine\ORM\Mapping as ORM;

/\*\*

\* @ORM\Entity(repositoryClass=CategoryRepository::class)

\*/

class Category

{

/\*\*

\* @ORM\Id

\* @ORM\GeneratedValue

\* @ORM\Column(type="integer")

\*/

private $id;

/\*\*

\* @ORM\Column(type="string", length=255)

\*/

private $name;

public function getId(): ?int

{

return $this->id;

}

public function getName(): ?string

{

return $this->name;

}

public function setName(string $name): self

{

$this->name = $name;

return $this;

}

}

**Отображение отношений ManyToOne:**

В этом примере каждая категория может быть связана со многими продуктами. Но каждый продукт может быть отнесен только к одной категории. Это отношение можно резюмировать следующим образом: многие продукты относятся к одной категории (или, что эквивалентно, от одной категории ко многим продуктам).

С точки зрения Productсущности, это отношение «многие к одному». С точки зрения Categoryсущности, это отношение «один ко многим».

Чтобы отобразить это, сначала создайте categoryсвойство в Productклассе с ManyToOneаннотацией. Вы можете сделать это вручную или с помощью make:entity команды, которая задаст вам несколько вопросов о ваших отношениях. Если вы не уверены в ответе, не волнуйтесь! Вы всегда можете изменить настройки позже:

**php bin/console make:entity**

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

> Product

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

> category

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

> relation

What class should this entity be related to?:

> Category

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

> ManyToOne

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

> no

Do you want to add a new property to Category so that you can access/update

Product objects from it - e.g. $category->getProducts()? (yes/no) [yes]:

> yes

New field name inside Category [products]:

> products

Do you want to automatically delete orphaned App\Entity\Product objects

(orphanRemoval)? (yes/no) [no]:

> no

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

>

(press enter again to finish)

Это внесло изменения в две сущности. Во-первых, он добавил новое свойство категории в сущность продукта (и методы получения и установки):

<?php

// src/Entity/Product.php

namespace App\Entity;

use App\Repository\ProductRepository;

use Doctrine\ORM\Mapping as ORM;

/\*\*

\* @ORM\Entity(repositoryClass=ProductRepository::class)

\*/

class Product

{

/\*\*

\* @ORM\Id

\* @ORM\GeneratedValue

\* @ORM\Column(type="integer")

\*/

private $id;

/\*\*

\* @ORM\Column(type="string", length=255)

\*/

private $name;

/\*\*

\* @ORM\Column(type="integer")

\*/

private $price;

/\*\*

\* @ORM\ManyToOne(targetEntity=Category::class, inversedBy="products")

\* @ORM\JoinColumn(nullable=false)

\*/

private $category;

public function getId(): ?int

{

return $this->id;

}

public function getName(): ?string

{

return $this->name;

}

public function setName(string $name): self

{

$this->name = $name;

return $this;

}

public function getPrice(): ?int

{

return $this->price;

}

public function setPrice(int $price): self

{

$this->price = $price;

return $this;

}

public function getCategory(): ?Category

{

return $this->category;

}

public function setCategory(?Category $category): self

{

$this->category = $category;

return $this;

}

}

Это ManyToOneсопоставление обязательно. Он сообщает Doctrine использовать category\_id столбец в productтаблице, чтобы связать каждую запись в этой таблице с записью в categoryтаблице.

Затем, поскольку один Category объект будет относиться ко многим Product объектам, make:entityкоманда также добавила productsсвойство в Category класс, который будет содержать эти объекты:

<?php

// src/Entity/Category.php

namespace App\Entity;

use App\Repository\CategoryRepository;

use Doctrine\ORM\Mapping as ORM;

use Doctrine\Common\Collections\ArrayCollection;

use Doctrine\Common\Collections\Collection;

/\*\*

\* @ORM\Entity(repositoryClass=CategoryRepository::class)

\*/

class Category

{

/\*\*

\* @ORM\Id

\* @ORM\GeneratedValue

\* @ORM\Column(type="integer")

\*/

private $id;

/\*\*

\* @ORM\Column(type="string", length=255)

\*/

private $name;

public function getId(): ?int

{

return $this->id;

}

public function getName(): ?string

{

return $this->name;

}

public function setName(string $name): self

{

$this->name = $name;

return $this;

}

/\*\*

\* @ORM\OneToMany(targetEntity="App\Entity\Product", mappedBy="category")

\*/

private $products;

public function \_\_construct()

{

$this->products = new ArrayCollection();

}

/\*\*

\* @return Collection|Product[]

\*/

public function getProducts(): Collection

{

return $this->products;

}

public function addProduct(Product $product): self

{

if (!$this->products->contains($product)) {

$this->products[] = $product;

$product->setCategory($this);

}

return $this;

}

public function removeProduct(Product $product): self

{

if ($this->products->removeElement($product)) {

// set the owning side to null (unless already changed)

if ($product->getCategory() === $this) {

$product->setCategory(null);

}

}

return $this;

}

}

php bin/console doctrine:migrations:diff

php bin/console doctrine:migrations:migrate

**Сохранение связанных объектов:**

Теперь вы можете увидеть этот новый код в действии! Представьте, что вы внутри контроллера:

<?php

namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;

use Symfony\Component\HttpFoundation\Response;

use Symfony\Component\Routing\Annotation\Route;

use App\Entity\Category;

use App\Entity\Product;

use Doctrine\ORM\EntityManagerInterface;

class ProductController extends AbstractController

{

/\*\*

\* @Route("/product", name="create\_product")

\*/

public function index(): Response

{

$category = new Category();

$category->setName('Computer Peripherals');

$product = new Product();

$product->setName('Keyboard');

$product->setPrice(19.99);

// relates this product to the category

$product->setCategory($category);

$entityManager = $this->getDoctrine()->getManager();

$entityManager->persist($category);

$entityManager->persist($product);

$entityManager->flush();

return new Response(

'Saved new product with id: '.$product->getId()

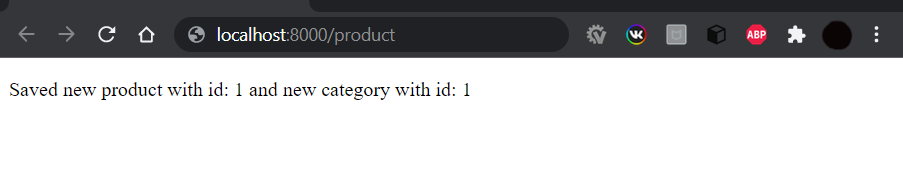
.' and new category with id: '.$category->getId()

);

}

}

Когда вы переходите к /product, к таблицам categoryи добавляется одна строка product. В product.category\_idстолбце для нового продукта устанавливается значение, idсоответствующее новой категории. Doctrine обеспечивает сохранение этих отношений за вас:





**Получение связанных объектов:**

Когда вам нужно получить связанные объекты, ваш рабочий процесс выглядит так же, как и раньше. Сначала выберите $productобъект, а затем получите доступ к связанному с ним Categoryобъекту:

public function show(int $id): Response

{

$product = $this->getDoctrine()

->getRepository(Product::class)

->find($id);

$categoryName = $product->getCategory()->getName();

}

Важен тот факт, что у вас есть доступ к категории, связанной с продуктом, но данные категории фактически не извлекаются, пока вы не запросите категорию (т. Е. Она «загружается лениво»).

Поскольку мы сопоставили необязательную OneToManyсторону, вы также можете запросить в другом направлении:

public function showProducts(int $id): Response

{

$category = $this->getDoctrine()

->getRepository(Category::class)

->find($id);

$products = $category->getProducts();

}

/\*\*

\* @Route("/product/{id}", name="product\_show")

\*/

public function show(int $id): Response

{

$product = $this->getDoctrine()

->getRepository(Product::class)

->find($id);

$categoryName = $product->getCategory()->getName();

if (!$product) {

throw $this->createNotFoundException(

'No product found for id '.$id

);

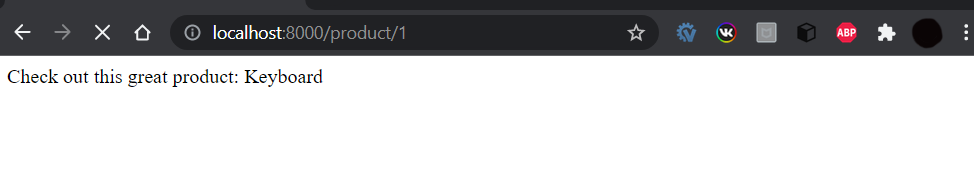
}

return new Response('Check out this great product: '.$product->getName());

}



Название продукта по его id



**Присоединение к связанным записям:**

В приведенных выше примерах было сделано два запроса - один для исходного объекта (например, a Category) и один для связанного объекта (ов) (например, Product объектов).

Если вы заранее знаете, что вам потребуется доступ к обоим объектам, вы можете избежать второго запроса, выполнив соединение в исходном запросе. Добавьте в ProductRepositoryкласс следующий метод :

<?php

// src/Repository/ProductRepository.php

namespace App\Repository;

use App\Entity\Product;

use Doctrine\Bundle\DoctrineBundle\Repository\ServiceEntityRepository;

use Doctrine\Persistence\ManagerRegistry;

/\*\*

\* @method Product|null find($id, $lockMode = null, $lockVersion = null)

\* @method Product|null findOneBy(array $criteria, array $orderBy = null)

\* @method Product[] findAll()

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

\*/

class ProductRepository extends ServiceEntityRepository

{

public function \_\_construct(ManagerRegistry $registry)

{

parent::\_\_construct($registry, Product::class);

}

// /\*\*

// \* @return Product[] Returns an array of Product objects

// \*/

/\*

public function findByExampleField($value)

{

return $this->createQueryBuilder('p')

->andWhere('p.exampleField = :val')

->setParameter('val', $value)

->orderBy('p.id', 'ASC')

->setMaxResults(10)

->getQuery()

->getResult()

;

}

\*/

/\*

public function findOneBySomeField($value): ?Product

{

return $this->createQueryBuilder('p')

->andWhere('p.exampleField = :val')

->setParameter('val', $value)

->getQuery()

->getOneOrNullResult()

;

}

\*/

public function findOneByIdJoinedToCategory(int $productId): ?Product

{

$entityManager = $this->getEntityManager();

$query = $entityManager->createQuery(

'SELECT p, c

FROM App\Entity\Product p

INNER JOIN p.category c

WHERE p.id = :id'

)->setParameter('id', $productId);

return $query->getOneOrNullResult();

}

}

Благодаря этому, если вы позвоните $category->removeProduct($product), в category\_id базе данных Productбудет установлено значение on null.

Но вместо того, чтобы устанавливать значение category\_idnull, что, если вы хотите, Product чтобы объект был удален, если он станет «осиротевшим» (то есть без a Category)? Чтобы выбрать такое поведение, используйте опцию orphanRemoval внутри Category:

/\*\*

\* @ORM\OneToMany(targetEntity="App\Entity\Product", mappedBy="category", orphanRemoval=true)

\*/

private $products;

Благодаря этому, если Productудаляется из Category, он будет удален из базы данных полностью.