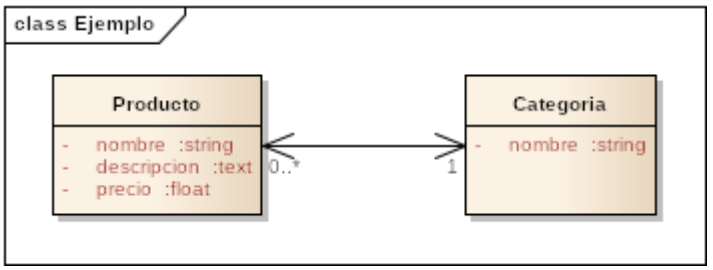
# Unidad 11 - Relaciones entre entidades

Existen dos principales tipos de relaciones entre entidades:

- ManyToOne / OneToMany: Es la relación más común, mapeada en la base de datos con una columna de clave externa (por ejemplo, una columna category_id en la tabla de product). Este es en realidad solo un tipo de asociación, pero visto desde los dos lados diferentes de la relación.
- ManyToMany: Utiliza una tabla de unión y es necesaria cuando ambos lados de la relación pueden tener muchos del otro lado (por ejemplo, "estudiantes" y "clases": cada estudiante está en muchas clases y cada clase tiene muchos estudiantes).

## Asociación ManyToOne / OneToMany

Supongamos que tenemos las siguiente relación:




Cada producto tiene una categoría y una categoría puede tener 0 o muchos productos.

Comenzamos creando la entidad categoria

```
$ 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)
```

Esto genera el siguiente código:

```
// src/Entity/Category.php
namespace App\Entity;

// ...

#[ORM\Entity(repositoryClass: CategoryRepository::class)]
class Category
{
      #[ORM\Id]
      #[ORM\GeneratedValue]
      #[ORM\Column]
      private $id;

      #[ORM\Column]
      private string $name;

      // ... getters and setters
}
```

En este ejemplo, cada categoría se puede asociar con muchos productos. Pero, cada producto se puede asociar con una sola categoría. Esta relación se puede resumir como: muchos productos a una categoría (o de manera equivalente, una categoría a muchos productos).

Desde la perspectiva de la entidad Producto, esta es una relación de muchos a uno. Desde la perspectiva de la entidad Categoría, esta es una relación de uno a muchos.

Para mapear esto, primero debemos crear la propiedad categoría en la clase Producto con la anotación ManyToOne. Puede hacerlo a mano o usando el comando make:entity, que nos hará varias preguntas sobre su relación.

```
$ 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)
```

Esta asignación ManyToOne es obligatoria. Le dice a Doctrine que use la columna category_id en la tabla de productos para relacionar cada registro en esa tabla con un registro en la tabla de categorías.

A continuación, dado que un objeto Categoría se relacionará con muchos objetos Producto, el comando make:entity también agregó una propiedad de productos a la clase Categoría que contendrá estos objetos:

```
// src/Entity/Category.php
namespace App\Entity;

// ...
use Doctrine\Common\Collections\ArrayCollection;
use Doctrine\Common\Collections\Collection;

class Category
{
      // ...

      #[ORM\OneToMany(targetEntity: Product::class, mappedBy: 'category')]
      private Collection $products;

      public function __construct()
      {
      $this->products = new ArrayCollection();
      }

      /**
      * @return Collection<int, Product>
      */
      public function getProducts(): Collection
      {
      return $this->products;
      }

      // addProduct() and removeProduct() were also added
}
```

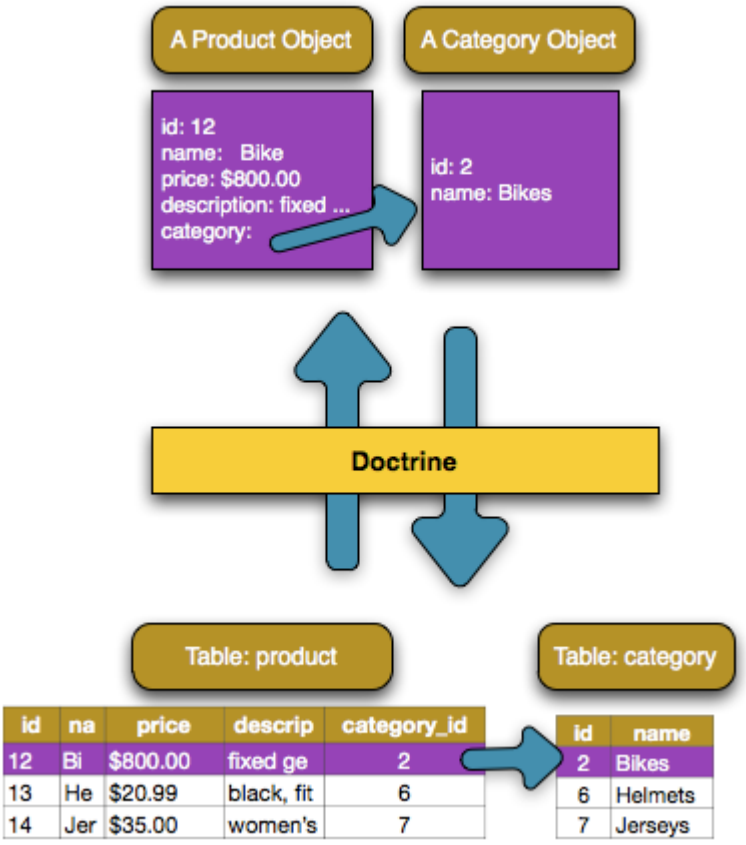
El mapeo ManyToOne que se muestra es obligatorio, pero el mapeo OneToMany es opcional: solo debemos agregarlo si se desea poder acceder a los productos que están relacionados con una categoría (esta es una de las preguntas que le hace make:entity). En este ejemplo, sería útil poder llamar a $category->getProducts().

### Guardar entidades relacionadas

Ahora vemos este nuevo código en acción. Imaginemos que estamos dentro de un controlador:

```
// src/Controller/ProductController.php
namespace App\Controller;

// ...
use App\Entity\Category;
use App\Entity\Product;
use Doctrine\ORM\EntityManagerInterface;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;

class ProductController extends AbstractController
{
      #[Route('/product', name: 'product')]
      public function index(EntityManagerInterface $entityManager): Response
      {
      $category = new Category();
      $category->setName('Computer Peripherals');

      $product = new Product();
      $product->setName('Keyboard');
      $product->setPrice(19.99);
      $product->setDescription('Ergonomic and stylish!');

      // relates this product to the category
      $product->setCategory($category);

      $entityManager->persist($category);
      $entityManager->persist($product);
      $entityManager->flush();

      return new Response(
            'Saved new product with id: '.$product->getId()
            .' and new category with id: '.$category->getId()
      );
      }
}
```

Cuando vamos a /producto, se agrega una sola fila a las tablas de categorías y productos. La columna product.category_id para el nuevo producto se establece en el id de la nueva categoría. Doctrine gestiona la persistencia de esta relación automáticamente.



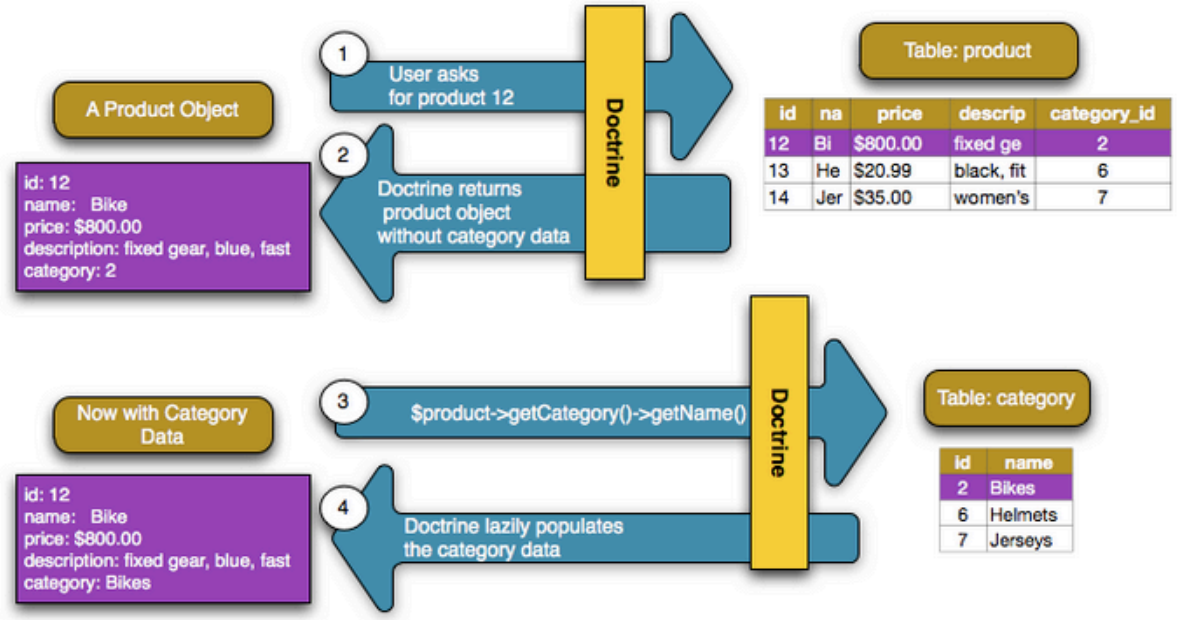

### Recuperando objetos relacionados

Cuando se necesita obtener objetos asociados, el flujo de trabajo se ve como antes. Primero, obtenemos un objeto de $producto y luego se accede a su objeto Categoría relacionado:

```
// src/Controller/ProductController.php
namespace App\Controller;

use App\Entity\Product;
// ...

class ProductController extends AbstractController
{
	public function show(ProductRepository $productRepository, int $id): Response
	{
		$product = $productRepository->find($id);
		// ...

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

		// ...
	}
}
```

En este ejemplo, primero se recupera un objeto Producto en función de la identificación del producto. Esto emite una consulta para obtener solo los datos del producto e hidrata $product. Más tarde,

cuando se llama a $product->getCategory()->getName(), Doctrine silenciosamente realiza una segunda consulta para encontrar la categoría relacionada con este producto. Prepara el objeto $category y lo devuelve.




Lo importante es el hecho de que tiene acceso a la categoría relacionada del producto, pero los datos de la categoría no se recuperan hasta que solicita la categoría (es decir, está "cargado de forma diferida").

Debido a que mapeamos el lado opcional de OneToMany, también se puede consultar en la otra dirección:

```
// src/Controller/ProductController.php

// ...
class ProductController extends AbstractController
{
       public function showProducts(CategoryRepository $categoryRepository, int $id):
Response
       {
              $category = $categoryRepository->find($id);
              $products = $category->getProducts();

       // ...
       }
}
```

En los ejemplos anteriores, se realizaron dos consultas: una para el objeto original (por ejemplo, una categoría) y otra para los objetos relacionados (por ejemplo, los objetos del producto).

Si sabemos de antemano que vamos a necesitar acceder a ambos objetos, se puede evitar la segunda consulta emitiendo una combinación en la consulta original. Para ello debemos realizar la

consulta a través del repositorio. Por lo tanto agregamos el siguiente método a la clase ProductRepository:

```
// src/Repository/ProductRepository.php

// ...
class ProductRepository extends ServiceEntityRepository
{
    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();
    }
}
```

Esto también devolverá un objeto Producto. Pero ahora, cuando llama a $product->getCategory() y usa esos datos, no se realiza una segunda consulta.

Ahora, se puede usar este método en el controlador para consultar un objeto Producto y su Categoría relacionada en una consulta:

```
// src/Controller/ProductController.php

// ...
class ProductController extends AbstractController
{
      public function show(ProductRepository $productRepository, int $id): Response
      {
            $product = $productRepository->findOneByIdJoinedToCategory($id);

            $category = $product->getCategory();

            // ...
      }
}
```