2. Working with the database

2.1 Fetch categories and products from the database

2.1.1 Data schema

The database structure can have a big impact on your application's performance, scalability, security, availability, and many other aspects you can observe. We will not have a database architecture course here, there are plenty of books and online resources about the subject. We will highlight some common design patterns whenever we encounter one.

We have already implemented some logic about categories and products. We will continue from there. Instead of having hard-coded categories and products, now we want to persist them into a database server. We are going to use MySQL server.

We are still adopting the TBD method. That means our first database design is not final and we will eventually update it while implementing new features.

Let's start with the following database structure

```sql
CREATE SCHEMA IF NOT EXISTS `symfony`
DEFAULT CHARACTER SET 'utf8' COLLATE 'utf8_unicode_ci';

USE `symfony`;

/* Category */
CREATE TABLE IF NOT EXISTS `category` (
  `id` SMALLINT(5) UNSIGNED NOT NULL AUTO_INCREMENT,
  `parent_category_id` SMALLINT(5) UNSIGNED NULL DEFAULT NULL,
  `label` VARCHAR(45) NULL DEFAULT NULL,
  PRIMARY KEY (`id`) ,
  INDEX `idx_parent_category_id` (`parent_category_id` ASC) ,
  CONSTRAINT `fk_category_parent_category_id`
    FOREIGN KEY (`parent_category_id`)
    REFERENCES `category` (`id`))
ENGINE = InnoDB;

/* Product */
CREATE TABLE IF NOT EXISTS `product` (
  `id` INT(11) UNSIGNED NOT NULL AUTO_INCREMENT,
  `code` VARCHAR(45) NOT NULL,
  `title` VARCHAR(200) NULL DEFAULT NULL,
  `description` LONGTEXT NULL DEFAULT NULL,
  PRIMARY KEY (`id`) ,
  INDEX `idx_code` (`code` ASC))
ENGINE = InnoDB;

/* Category has Product */
CREATE TABLE IF NOT EXISTS `category_has_product` (
  `category_id` SMALLINT(5) UNSIGNED NOT NULL,
  `product_id` INT(11) UNSIGNED NOT NULL,
  PRIMARY KEY (`category_id`, `product_id`) ,
  INDEX `idx_categoy_id` (`category_id` ASC) ,
  INDEX `idx_product_id` (`product_id` ASC) ,
  CONSTRAINT `fk_category_has_product_category_id`
    FOREIGN KEY (`category_id`)
    REFERENCES `category` (`id`),
  CONSTRAINT `fk_category_has_product_product_id`
    FOREIGN KEY (`product_id`)
    REFERENCES `product` (`id`))
ENGINE = InnoDB;
```

- Most of the time you may better define your primary keys as `UNSIGNED` because you don't usually need negative values.
- Carefully choose every column's data type.

MySQL integer types

Type		Storage (bytes)	Minumum value	Maximum value
TINYINT	SIGNED	1	-128	127
	UNSIGNED		0	255
SMALLINT	SIGNED	2	-32768	32767
	UNSIGNED		0	65535
MEDIUMINT	SIGNED	3	-8388608	8388607
	UNSIGNED		0	16777215
INT	SIGNED	4	-2147483648	2147483647
	UNSIGNED		0	4294967295
MEDIUMINT	SIGNED	8	-9223372036854775808	9223372036854775807
	UNSIGNED		0	18446744073709551615

> Recommanded reading: **High Performance MySQL** [ISBN: 978-1449314286] by *Baron Schwartz* (O'Reilly)

2.1.2 Doctrine ORM

Now we need to update our code to use the created tables in order to store categories and products. We are not going to write plain SQL queries. Is a tedious work, and prone to errors. We will use an Object-Relational Mapping (ORM) tool that will abstract our data structures as PHP objects that are called **entities**. For this project, we are going to use Doctrine ORM. For more detailed informations, please check the official documentation

Before proceeding make sure you created the database structure. Also make sure that your database details in app/config/parameters.yml are correct.

Let's start by creating the category entity.

- Create **src/AppBundle/Entity/Category.php**

```php
<?php

namespace AppBundle\Entity;

use Doctrine\ORM\Mapping as ORM;

/**
 * Category
 *
 * @ORM\Table(
 *  name="category",
 *  indexes={@ORM\Index(name="idx_parent_category_id",columns={"parent_category_id"})}
 * )
 * @ORM\Entity
 */
class Category
{
    const REPOSITORY = 'AppBundle:Category';

    /**
     * @var integer
     *
     * @ORM\Column(name="id", type="smallint", nullable=false)
     * @ORM\Id
     * @ORM\GeneratedValue(strategy="IDENTITY")
     */
    private $id;

    /**
     * @var string
     *
     * @ORM\Column(name="label", type="string", length=45, nullable=true)
     */
    private $label;
```

```
    /**
     * @var \Category
     *
     * @ORM\ManyToOne(targetEntity="Category")
     * @ORM\JoinColumns({
     *   @ORM\JoinColumn(name="parent_category_id", referencedColumnName="id")
     * })
     */
    private $parentCategory;

    public function getId()
    {
        return $this->id;
    }

    public function getLabel()
    {
        return $this->label;
    }

    public function getParent()
    {
        return $this->parentCategory;
    }

}
```

`ManyToOne` relation states that many categories can belong to the same parent, and one category can have at most one parent.

We need to update the CategoryService class to load the categories from the database. At this point, we will rewrite the whole class as following.

- Edit **src/AppBundle/Service/CategoryService.php**

```php
<?php

namespace AppBundle\Service;

use AppBundle\Entity\Category;
use Doctrine\ORM\EntityManagerInterface;

class CategoryService
{

    const ID = 'app.category';

    /**
     *
     * @var EntityManagerInterface
     */
    private $entityManager;

    public function __construct(EntityManagerInterface $manager)
    {
        $this->entityManager = $manager;
    }

    public function getCategories()
    {
        return $this->entityManager->getRepository(Category::REPOSITORY)->findAll();
    }

    public function getCategory($categoryId)
    {
        $category = $this->entityManager
                         ->getRepository(Category::REPOSITORY)->find($categoryId);
        if (empty($category)) {
            throw new \Exception(sprintf('Invalid categoryId %s', $categoryId));
        }
        return $category;
    }

}
```

Now our CategoryService depends on an EntityManagerInterface. This makes the instantiation of this class harder because we need to know how to instantiate an EntityManagerInterface. Here we will take advantage of Symfony's dependency injection capabilities and delegate the creation of our service to Symfony. So we will register CategoryService as a Symfony service.

- Create **src/AppBundle/Resources/config/services.yml**

```yaml
services:
  app.category:
      class: AppBundle\Service\CategoryService
      arguments: ["@doctrine.orm.entity_manager"]
```

Now we need to tell Symfony to load this new configuration file.

- Edit **app/config/services.yml**, discard the sample content, and put

```
imports:
  - { resource: @AppBundle/Resources/config/services.yml }
```

Now we don't need to instantiate the CategoryService manually anymore, we can get an instance of CategoryService from the *service container*.

- Edit **src/AppBundle/Controller/CategoryController.php** and change occurrences of
  `$categoryService = new CategoryService();`
  to
  `$categoryService = $this->container->get(CategoryService::ID);`

> Symfony is a dependency injection framework. So keep the pace with it. In few words, give your services the instances they need instead of letting them create their own instances.

In our example, this is how **you should not** implement `CategoryService::__construct`

```
public function __construct(\Doctrine\Bundle\DoctrineBundle\Registry $doctrine)
{
    $this->entityManager = $doctrine->getManager();
}
```

```
public function __construct(
  \Symfony\Component\DependencyInjection\ContainerInterface $container
)
{
    $this->entityManager = $container->get('doctrine')->getManager();
}
```

We are done with categories. Insert some records into the category table and check if everything is working as before.

We will do the same with products.

- Create **src/AppBundle/Entity/Product.php**

```php
<?php

namespace AppBundle\Entity;

use Doctrine\ORM\Mapping as ORM;

/**
 * Product
 *
 * @ORM\Table(name="product", indexes={@ORM\Index(name="idx_code", columns={"code"})})
 * @ORM\Entity
 */
class Product
{
    const REPOSITORY = 'AppBundle:Product';

    /**
     * @var integer
     *
     * @ORM\Column(name="id", type="integer", nullable=false)
     * @ORM\Id
     * @ORM\GeneratedValue(strategy="IDENTITY")
     */
    private $id;

    /**
     * @var string
     *
     * @ORM\Column(name="code", type="string", length=45, nullable=false)
     */
    private $code;

    /**
     * @var string
     *
     * @ORM\Column(name="title", type="string", length=200, nullable=true)
     */
    private $title;

    /**
     * @var string
     *
     * @ORM\Column(name="description", type="text", nullable=true)
     */
    private $description;
```

```
    /**
     * @var \Doctrine\Common\Collections\Collection
     *
     * @ORM\ManyToMany(targetEntity="Category")
     * @ORM\JoinTable(name="category_has_product",
     *   inverseJoinColumns={
     *     @ORM\JoinColumn(name="category_id", referencedColumnName="id")
     *   },
     *   joinColumns={
     *     @ORM\JoinColumn(name="product_id", referencedColumnName="id")
     *   }
     * )
     */
    private $categories;

    /**
     * Constructor
     */
    public function __construct()
    {
        $this->category = new \Doctrine\Common\Collections\ArrayCollection();
    }

    public function getId()
    {
        return $this->id;
    }

    public function getCode()
    {
        return $this->code;
    }

    public function getTitle()
    {
        return $this->title;
    }

    public function getDescription()
    {
        return $this->description;
    }

    public function getCategories()
    {
        return $this->categories;
    }

}
```

- Create **src/AppBundle/Service/ProductService.php**

```php
<?php

namespace AppBundle\Service;

use AppBundle\Entity\Product;
use Doctrine\ORM\EntityManagerInterface;

class ProductService
{

    const ID = 'app.product';

    /**
     *
     * @var EntityManagerInterface
     */
    private $entityManager;

    public function __construct(EntityManagerInterface $manager)
    {
        $this->entityManager = $manager;
    }

    public function getProducts()
    {
        return $this->entityManager->getRepository(Product::REPOSITORY)->findAll();
    }

    public function getProduct($productId)
    {
        $product = $this->entityManager
                        ->getRepository(Product::REPOSITORY)->find($productId);
        if (empty($product)) {
            throw new \Exception(sprintf('Invalid product %s', $productId));
        }
        return $product;
    }

}
```

- Edit **src/AppBundle/Resources/config/services.yml** and add the following service definition

```
app.product:
  class: AppBundle\Service\ProductService
  arguments: ["@doctrine.orm.entity_manager"]
```

- Edit **src/AppBundle/Controller/ProductController.php** Add `use AppBundle\Service\ProductService;`
  Replace the body of `getProducts` method by

```
$productService = $this->container->get(ProductService::ID);
return $productService->getProducts();
```

Replace the body of `getProduct()` method by

```
$productService = $this->container->get(ProductService::ID);
return $productService->getProduct($productId);
```

We are done. Insert some records into *product* and *category_has_product* tables and check if everything is working well.

2.2 Data fixtures

2.2.1 Category fixtures

Inserting data manually in the database is not efficient, neither fun to do. We will need to automate populating the database with some initial records.

We want to create some mock data that is as close as possible to the production database. The best way is to actually use a copy of the production data (after removing or obfuscating sensitive informations like user emails or passwords). That's however a luxury not everyone can have. Since we are developing a new application, we will create some data fixtures.

To install **DoctrineFixturesBundle** add `"doctrine/doctrine-fixtures-bundle": "2.2.0"` to the *require* section, in **composer.json** then run `composer update`.

To enable the bundle within the application, edit **app**/**AppKernel.php** and add a new item to the $bundles array in registerBundles()
`new Doctrine\Bundle\FixturesBundle\DoctrineFixturesBundle(),`

Before writing any fixtures, we need to update the category entity. We have to add setters for `label` and `parentCategory`

- Edit **src**/**AppBundle**/**Entity**/**Category.php** and add setters for `label` and `parentCategory`

```
public function setLabel($label)
{
    $this->label = $label;
    return $this;
}

public function setParentCategory(Category $parentCategory = null)
{
    $this->parentCategory = $parentCategory;
    return $this;
}
```

Now we can create the category fixture class.

- Create **src/AppBundle/DataFixtures/ORM/CategoryFixtures.php**

```php
<?php

namespace AppBundle\DataFixtures\ORM;

use AppBundle\Entity\Category;
use Doctrine\Common\DataFixtures\AbstractFixture;

class CategoryFixtures extends AbstractFixture
{

    /**
     *
     * @var \Doctrine\Common\Persistence\ObjectManager
     */
    protected $manager;
    private $categoriesCount = 0;
    private $categories = array(
        'computers' => array('servers', 'desktop', 'laptops','components','peripherals'),
        'phones and tablets' => array('phones', 'tablets', 'accessories'),
        'appliances' => array('coffee machines', 'washing machines', 'blenders',
        'juicers', 'fridges', 'air conditioner'),
        'video games' => array('consoles', 'games', 'accessories'),
        'televisions' => array('smart TV', 'curved TV', '3D TV', 'projectors'),
        'video' => array('video cameras', 'cam corder', 'mini cameras'),
        'books' => array('textbooks', 'magazines', 'e-books', 'audible books'),
        'sports' => array('tennis', 'football', 'fitness', 'athletic clothing', 'golf'),
        'alcoholic drinks' => array('beer', 'wine', 'strong alcohols'),
        'water' => array('sparkling', 'non sparkling'),
        'sweets' => array('chocolate', 'biscuits', 'candies'),
        'clothing' => array('for men', 'for women', 'for kids', 'casual', 'sport',
        'fancy', 'for winter', 'for summer'),
        'natural products' => array('cosmetics', 'hygiene', 'medicinal plants'),
    );

    public function load(\Doctrine\Common\Persistence\ObjectManager $manager)
    {
        $this->manager = $manager;
        $this->createAndPersistData();
        $this->manager->flush();
    }

    protected function createAndPersistData()
    {
        foreach ($this->categories as $parent => $children) {
            $parentCategory = $this->createAndPersistCategory($parent);
            foreach ($children as $label) {
                $this->createAndPersistCategory($label, $parentCategory);
            }
        }
    }
```

```
    private function createAndPersistCategory($label, $parentCategory = null)
    {
        $this->categoriesCount ++;
        $category = new Category();
        $category->setLabel($label)->setParentCategory($parentCategory);
        $this->manager->persist($category);

        return $category;
    }

}
```

Done, to load the fixtures into the database, run the command

```
$ app/console doctrine:fixtures:load
Careful, database will be purged. Do you want to continue Y/N ?y
  > purging database
  > loading AppBundle\DataFixtures\ORM\CategoryFixtures
```

2.2.2 Product fixtures

Creating product fixtures can be a bit tricky.

- We need to have access to the categories entities to assign them to the created product entities.
- We need to make sure that the category fixtures are executed before the product fixtures.

- Edit **src/AppBundle/Entity/Product.php** and add the required setters

```
public function setCode($code)
{
    $this->code = $code;
    return $this;
}

public function setTitle($title)
{
    $this->title = $title;
    return $this;
}

public function setDescription($description)
{
    $this->description = $description;
    return $this;
}

public function setCategories(\Doctrine\Common\Collections\Collection $categories)
{
    $this->categories = $categories;
    return $this;
}
```

- Edit **src/AppBundle/DataFixtures/ORM/CategoryFixtures.php**
  - Make it implement `Doctrine\Common\DataFixtures\OrderedFixtureInterface`
  - Add `$this->setReference(sprintf('category_%s', $this->categoriesCount), $category);` before `return $category;` in `load` method
  - Add `getOrder` method, it will define the order in which fixtures are executed

```
public function getOrder()
{
  return 1;
}
```

- Create **src/AppBundle/DataFixtures/ORM/ProductFixtures.php**

```php
<?php
namespace AppBundle\DataFixtures\ORM;
use AppBundle\Entity\Category;
use AppBundle\Entity\Product;
use Doctrine\Common\Collections\ArrayCollection;
use Doctrine\Common\DataFixtures\AbstractFixture;
use Doctrine\Common\DataFixtures\OrderedFixtureInterface;
use Doctrine\Common\Persistence\ObjectManager;

class ProductFixtures extends AbstractFixture implements OrderedFixtureInterface
{
    private $productsCount = 0;
    private $productsPerCategory = 500;

    public function load(ObjectManager $manager)
    {
        $this->manager = $manager;
        $this->createAndPersistData();
        $this->manager->flush();
    }

    private function createAndPersistProducts(Category $category)
    {
        for ($i = 1; $i <= $this->productsPerCategory; $i++) {
            $this->productsCount++;
            $product = new Product();
            $product->setCode(sprintf('code-%s-%s', $category->getId(), $i))
                    ->setTitle(sprintf('title %s %s %s', $category->getId(),$i,uniqid()))
                    ->setDescription(sprintf('product description %s', $i))
                    ->setCategories(new ArrayCollection(array($category)));
            $this->manager->persist($product);
            $this->setReference(sprintf('product_%s', $this->productsCount), $product);
        }
    }

    protected function createAndPersistData()
    {
        foreach ($this->getReferences('category') as $category) {
            $this->createAndPersistProducts($category);
            $this->manager->flush();
        }
    }

    protected function getReferences($prefix)
    {
        $entities = array();
        for ($i = 1; true; $i++) {
            try {
                $entities[] = $this->getReference(sprintf('%s_%s', $prefix, $i));
            } catch (\OutOfBoundsException $exception) {
                break;
            }
        }
        return $entities;
    }
```

```
    public function getOrder()
    {
        return 2;
    }

}
```

Done, however we cannot load the new fixtures exactly like we did previously. Because MySQL will refuse to delete records that are referenced as foreign keys relations. We will update our command to purge the database before loading the new fixtures.

```
$ app/console doctrine:schema:drop --force; app/console doctrine:schema:create; \
app/console doctrine:fixtures:load
Dropping database schema...
Database schema dropped successfully!
ATTENTION: This operation should not be executed in a production environment.

Creating database schema...
Database schema created successfully!
Careful, database will be purged. Do you want to continue Y/N ?y
  > purging database
  > loading [1] AppBundle\DataFixtures\ORM\CategoryFixtures
  > loading [2] AppBundle\DataFixtures\ORM\ProductFixtures
```

2.3 Persist categories to the database

Now we will add the logic to create and edit categories from the user interface. So far we built the edit form manually. We can take advantage of the Symfony Form component to render and handle forms for us.

- Edit **src/AppBundle/Resources/views/category/edit.html.twig** replace the existent form with

```
{% extends 'base.html.twig' %}
{% block body %}
    <h1>Category edit</h1>
    {{ form(edit_form) }}
    <div class="actions">
        <a class="button" href="{{ path('category_list') }}">Back to the list</a>
    </div>
{% endblock %}
```

Now we will create a Category form type that will define the required parameters to handle the form.

- Create **src/AppBundle/Form/Type/CategoryType.php**

```
<?php
namespace AppBundle\Form\Type;
use AppBundle\Entity\Category;
use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\FormBuilderInterface;
use Symfony\Component\OptionsResolver\OptionsResolver;

class CategoryType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder
                ->add('label')
                ->add('parent', 'entity', array('label' => 'parent category',
                    'class' => Category::REPOSITORY,
                    'choice_label' => 'label',
                ))
                ->add('save', 'submit')
        ;
    }
    public function configureOptions(OptionsResolver $resolver)
    {
        $resolver->setDefaults(array(
            'data_class' => 'AppBundle\Entity\Category',
        ));
    }
    public function getName()
    {
        return 'app_category';
    }
}
```

And then we have to update the controller to use the new form.

- Edit **src/AppBundle/Controller/CategoryController.php** Add the uses

```
use AppBundle\Entity\Category;
use AppBundle\Form\Type\CategoryType;
```

Update `editCategoryAction` and `saveCategoryAction` and add `createCategoryEditForm` as following

```
<?php

public function editCategoryAction(Request $request, $categoryId)
{
    try {
        $category = $this->getCategory($categoryId);
    } catch (\Exception $exception) {
        throw $this->createNotFoundException($exception->getMessage(), $exception);
    }
    $form = $this->createCategoryEditForm($category);
    return $this->render('AppBundle:category:edit.html.twig', array(
                'entity' => $category,
                'edit_form' => $form->createView(),
    ));
}
public function saveCategoryAction(Request $request, $categoryId)
{
    try {
        $category = $this->getCategory($categoryId);
    } catch (\Exception $exception) {
        throw $this->createNotFoundException($exception->getMessage(), $exception);
    }
    $form = $this->createCategoryEditForm($category);
    $form->handleRequest($request);
    if ($form->isValid()) {
        $this->getDoctrine()->getManager()->flush();
        return $this->redirect(
                  $this->generateUrl('category_edit', array('categoryId' => $categoryId))
        );
    }
    throw new \Exception($form->getErrors(true)->current()->getMessage());
}
private function createCategoryEditForm(Category $category)
{
    return $this->createForm(new CategoryType(), $category, array(
                'action' => $this->generateUrl(
                        'category_edit', array('categoryId' => $category->getId())
                ),
                'method' => 'POST',
    ));
}
```

Done, try to edit some categories and check if everything is ok.

2.4 Exercises

2.4.1 Probably you have noticed that you cannot save a category without assigning a parent to it. We want to add an item to the top of the parent categories select with the label **None**. Selecting the **None** option will set the parent category to `NULL`

2.4.2 When the user selects the same category as parent, throw an exception **A category cannot be the parent of itself**.

2.4.3 On the category edit page, add a delete button. Add the route and the action required to perform the delete.

2.4.4 Use the **soft delete** technique for categories. Add a boolean column named **deleted** when someone wants to delete a category, set `deleted=true` instead. Update the listing page to not to show *deleted* categories.