



DQL

Doctrine Query Language

UP Web

AU: 2017/2018

Repository



- Les Repository servent à récupérer les entités
- Ces services utilisent un Entity Manager:
- 2 techniques de récupération :
 - QueryBuilder: permettant de construire les requêtes par étape
 - DQL: semblable au langage SQL

Méthodes find du QueryBuilder



<code>find(\$id)</code>	recherche l'entité par sa clé primaire
<code>findAll()</code>	recherche toutes les entités
<code>findBy(array \$criteres, array \$orderBy = null, \$limite = null, \$offset = null)</code> Exemple <code>findBy(array('Pays'=>'Tunisie') , array('id'=>'asc'), 5, 0);</code>	recherche toutes les entités selon des critères : Cet exemple va récupérer tous les Modèles ayant comme Pays = Tunisie en les classant par id croissant et en en sélectionnant cinq (5) à partir du début (0=1 ^{er} résultat trouvé).
<code>findOneBy(array \$criteres)</code>	Recherche une entité selon des critères

Méthodes find du QueryBuilder



findByX(\$valeur) en remplaçant X par une propriété de l'entité	Similaire à findBy() avec un seul critère, celui du nom de la méthode
findOneByX(\$valeur) en remplaçant X par une propriété de l'entité	Similaire à findOneBy() avec un seul critère, celui du nom de la méthode

Personnaliser le Repository



- L'EntityRepository « ModeleRepository » est le dépôt de toutes les méthodes d'accès aux données de la base de données concernant l'Entité Modele:
 - l'interface entre les données BD et les données objet y est centralisée
 - Ainsi, les contrôleurs et vues ne manipulent que les objets entités du modèle de donnée et non du SQL.

ModeleRepository



- Créer une classe ModeleRepository dans le dossier Entity.

```
<?php

namespace Esprit\ParcBundle\Entity;
use Doctrine\ORM\EntityRepository;

class ModeleRepository extends EntityRepository{
}
```

- Référencer le nouveau Repository dans l'entité Modele

```
namespace Esprit\ParcBundle\Entity;
use Doctrine\ORM\Mapping as ORM;

/**
 * @ORM\Table(name="TestModele")
 * @ORM\Entity(repositoryClass="Esprit\ParcBundle\Entity\ModeleRepository")
 */
class Modele {
    |
```

- Pour l'instant, il est « vide » : il contient les méthodes standards de la classe EntityRepository et n'est pas encore personnalisé.

Exemple avec QueryBuilder



- Ajouter findPaysQB dans ModeleRepository

```
public function findPaysQB() {  
    $qb=$this->createQueryBuilder('s');  
    $qb->where('s.Pays=:pays')  
        ->setParameter('pays', 'Tunisie');  
    return $qb->getQuery()->getResult();  
}
```

- Ajouter l'action correspondante dans ModeleController

```
public function findPays1Action() {  
  
    $em=$this->getDoctrine()->getManager();  
    $modeles=$em->getRepository("EspritParcBundle:Modele")  
        ->findPaysQB();  
    return ($this->render("EspritParcBundle:Modele:list.html.twig",array("modeles"=>$modeles)));  
}
```

Exemple avec QueryBuilder


- Ajouter le routing correspondant

```
esprit_parc_findPays_QB:  
  path:    /QB/findPays  
  defaults: { _controller: EspritParcBundle:Modele:findPays1}
```

- Et la vue mentionnée dans l'action list.html.twig

```
<h2>Liste des modèles</h2>  
  
<table border="1">  
  <tr>  
    <td>Id</td>  
    <td>Libelle</td>  
    <td>Pays</td>  
  </tr>  
  <tr>  
    {% for modele in modeles %}  
      <td>{{modele.id}}</td>  
      <td>{{modele.libelle}}</td>  
      <td>{{modele.pays}}</td>  
    </tr>  
  {%endfor%}  
</table>
```


DQL = Doctrine Query Language



- un langage de requêtes qui ressemble beaucoup à SQL, sans avoir les particularités des différentes implémentations
- adapté à la vision par objets que Doctrine utilise : c'est un langage de requête sur des objets et non sur des tables !
- Il ne peut faire que des SELECT, UPDATE, DELETE : l'insertion se fait par la persistance des entités
- Ref = <http://docs.doctrine-project.org/en/latest/reference/dql-doctrine-query-language.html>

Exemple 1



- Récupérer tous les modèles correspondant au pays 'France'

ModeleRepository.php

```
public function findPaysDQL() {  
    $query=$this->getEntityManager()  
        ->createQuery("SELECT m from EspritParcBundle:Modele m WHERE m.Pays='Tunisie'");  
    return $query->getResult();  
}
```

ModeleController.php

```
public function findPaysAction() {  
  
    $em=$this->getDoctrine()->getManager();  
    $modeles=$em->getRepository("EspritParcBundle:Modele")  
        ->findPaysDQL();  
    return ($this->render("EspritParcBundle:Modele:list.html.twig",array("modeles"=>$modeles)));  
}
```

Exemple 2



- Récupérer tous les modèles correspondant à un pays donné en paramètre (dans le path)

ModeleRepository.php

```
public function findPaysParametre($pays)
{
    $query=$this->getEntityManager()
        ->createQuery("SELECT m FROM EspritParcBundle:Modele m where m.Pays=:pays")
        ->setParameter('pays',$pays);
    return $query->getResult();
}
```

ModeleController.php

```
public function findPays2Action($pays){

    $em=$this->getDoctrine()->getManager();
    $modeles=$em->getRepository("EspritParcBundle:Modele")
        ->findPaysParametre($pays);
    return ($this->render("EspritParcBundle:Modele:list.html.twig",array("modeles"=>$modeles)));
}
```

routing.yml

```
esprit_parc_findPays2_DQL:
    path:    /DQL/findPays2/{pays}
    defaults:  { _controller: EspritParcBundle:Modele:findPays2}
```

Paramètres dans DQL



DQL supporte les paramètres nommés et les paramètres positionnels.

- *Paramètres nommés (Named Parameter)*: le paramètre est appelé par son nom dans la requête **:param**.
Exemple **:pays, :libelle**
- *Paramètres positionnels (Positional Parameter)*: le paramètre est appelé par sa position dans la requête **?position**
Exemple **?1, ?2**

Exemples



1. sans paramètre

```
$this->getEntityManager()->  
createQuery("SELECT m FROM EspritParcBundle:Modele m  
    where m.Pays='France'");
```

2. utilisation d'un paramètre nommé

```
$this->getEntityManager()->  
createQuery("SELECT m FROM EspritParcBundle:Modele m  
    where m.Pays=:pays");  
$query->setParameter('pays','France');
```

Exemples



3. utilisation de plusieurs paramètres nommés

```
$this->getEntityManager()->createQuery("SELECT m FROM  
EspritParcBundle:Modele m where m.Pays=:pays and  
m.id=:id");
```

```
$query-> setParameters (array ('pays'=>'Allemagne',  
'id'=>2));
```

4. utilisation d'un paramètre positionnel

```
$this->getEntityManager()->createQuery("SELECT m FROM  
EspritParcBundle:Modele m where m.Pays=?1 and  
m.id=?2");
```

```
$query->setParameters(array(1=>'Allemagne', 2=>2));
```

Exemple 3



- Récupérer tous les modèles ayant un id supérieur à une valeur donnée, order by Pays

```
public function findSupId($id)
```

ModeleRepository.php

```
{  
    $query=$this->getEntityManager()  
        ->createQuery("SELECT m FROM EspritParcBundle:Modele m "  
            . "where m.Id>:num ORDER BY m.Pays ASC")  
        ->setParameter('num',$id);  
    return $query->getResult();  
}
```

```
public function findIdAction($id){
```

ModeleController.php

```
$em=$this->getDoctrine()->getManager();  
$modeles=$em->getRepository("EspritParcBundle:Modele")  
    ->findSupId($id);  
return ($this->render("EspritParcBundle:Modele:list.html.twig",array("modeles"=>$modeles)));  
}
```

```
esprit_parc_findId_DQL:
```

routing.yml

```
path: /DQL/findId/{id}  
defaults: { _controller: EspritParcBundle:Modele:findIdAction}
```

Exemple 3



```
$this->getEntityManager()->createQuery('SELECT m FROM  
M EspritParcBundle:Modele m  
WHERE m.id > :num  
ORDER BY m.id ASC')  
->setParameter('num', '2');
```

- m : alias (variable d'identification) qui réfère à EspritParcBundle:Modele
 - m.id > :num : condition
- => retourner les instances de la classe Modele où l'id est > 2

Exemple 4 Update



- Mettre à jour le pays du modèle correspondant à un id donné.

```
$query=$this->getEntityManager()  
->createQuery('UPDATE  
EspritParcBundle:Modele m SET m.Pays =  
:nvPays WHERE m.id = :idModele');  
$query->setParameter('nvPays ', 'Tunisie');  
$query->setParameter('idModele', '2');  
$query->execute();
```

Exemple 5 Delete



- Supprimer le modèle ayant id = 1

```
$this->getEntityManager()->  
createQuery('DELETE EspritParcBundle:Modele  
m WHERE m.id = :idModele');  
$query->setParameter('idModele', '1');  
$query->execute();
```

Fonctions DQL



- Fonctions utilisées avec les clauses SELECT, WHERE, HAVING:
- ABS(arithmetic_expression)
- CONCAT(str1, str2)
- CURRENT_DATE() – retourner la date courante
- CURRENT_TIME() - retourner le temps courant

Fonctions DQL



- `LENGTH(str)` – Retourne la longueur de la chaîne passée en input
- `LOCATE(needle, haystack [, offset])` – retourne la position de la première occurrence de la sous chaîne dans l'expression.
- `LOWER()`
- `SUBSTRIN(str,start[,length])`
- ...

Fonctions DQL



- AVG : Moyenne
- COUNT : noombre de résultat
- MIN: minimum
- MAX: maximum
- SUM: somme
- Exemple utilisation de COUNT:

```
$query = $em->createQuery('SELECT COUNT(m.id)  
FROM EspritParcBundle:Modele m');  
$result = $query->getSingleScalarResult();
```

Autres expressions



- ALL/ ANY/ SOME
- BETWEEN a AND b
- NOT BETWEEN a AND b
- IN(x1,x2,,,,,)/ NOT IN (x1,x2,,,,,)
- LIKE a/ NOT LIKE a
- IS NULL/ IS NOT NULL
- EXISTS/ NOT EXISTS
- INSTANCE OF
- ORDER BY

Exemples



- Exemple utilisation **ORDER BY**

```
$query = $em->createQuery("SELECT m FROM  
EspritParcBundle:Modele m ORDER BY m.Pays  
ASC") ;
```

```
$result = $query->getResult() ;
```

- Exemple utilisation de **LIKE**

```
$query = $em->createQuery("SELECT m FROM  
EspritParcBundle:Modele m WHERE m.Pays LIKE  
'%F%'") ;
```

```
$result = $query->getResult() ;
```

Jointure



Pour faire une requête avec jointure, il ne faut pas spécifier la clause ON des jointures, puisque la doctrine sait déjà sur eux.

Exemple:

```
$query = $em->createQuery("SELECT u FROM  
User u JOIN u.address a WHERE a.city =  
'Berlin'");  
$users = $query->getResult();
```


Récupération du résultats



- Plusieurs fonctions permettent de récupérer le résultat de la requête
- getResult(): permet de récupérer une collection de résultats.
- getSingleResult(): permet de récupérer un seul résultat. Si plus qu'un résultat est retourné une exception est levée.
- getOneOrNullResult(): permet de récupérer un seul résultat, si pas de résultat NULL est retournée.

Récupération du résultats



- `getArrayResult()`: permet de récupérer un tableau de résultats.
- `getScalarResult()`: permet de récupérer des résultats scalaires
- `getSingleScalarResult()`: permet de récupérer un seul résultat scalaire, sinon exception.

Références



- <http://openclassrooms.com/courses/developpez-votre-site-web-avec-le-framework-symfony2/recuperer-ses-entites-avec-doctrine2>
- <http://php.net/manual/fr/language.oop5.magic.php>
- <http://doctrine-orm.readthedocs.org/en/latest/reference/query-builder.html>