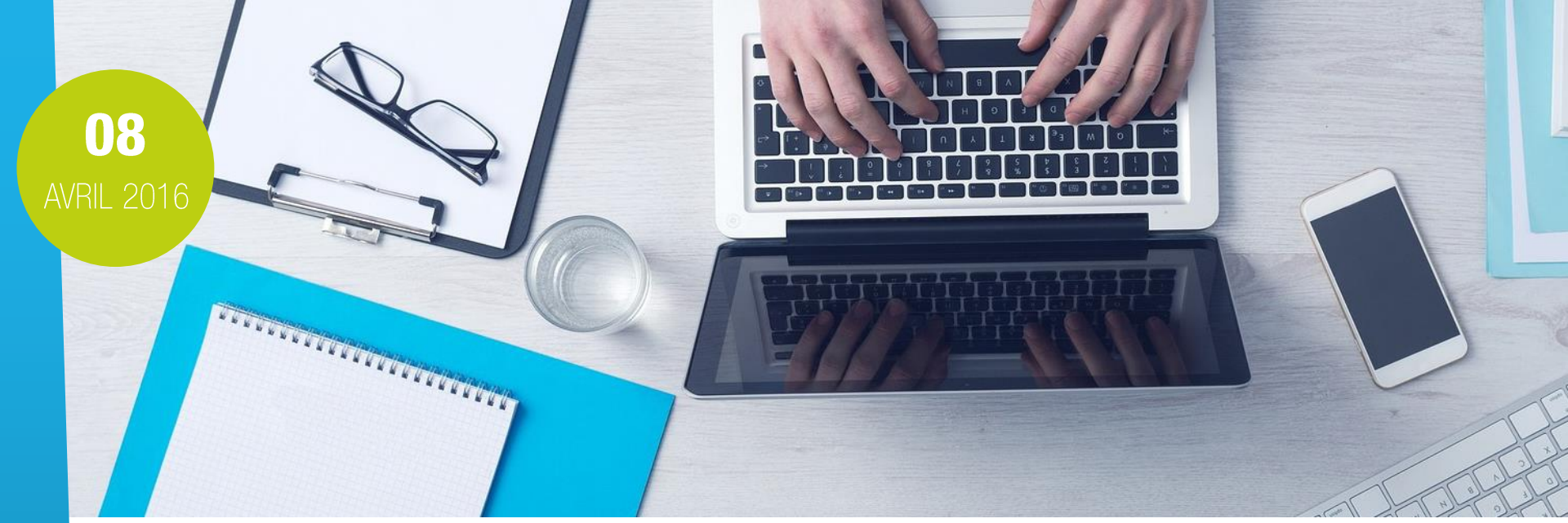

08
AVRIL 2016

ALLER PLUS LOIN AVEC **DOCTRINE 2**

Présentation par André Tapia & Amine Mokeddem

webnet
intégrateur de solutions internet

SymfonyLive
PARIS 2016
7-8 AVRIL

ORDRE DU JOUR

1 /

Etendre le vocabulaire DQL

2 /

Comment tirer profit des events ?

3 /

Optimiser les performances de vos requêtes

4 /

Développer plus rapidement grâce à Doctrine

QUI SOMMES-NOUS ?

QUI SOMMES-NOUS ?

André Tapia

Amine Mokeddem

Architectes techniques chez webnet depuis 2011

webnet
intégrateur de solutions internet

10 ans d'expérience cumulée sur une vingtaine de projets Symfony2 de tous types (système d'information, site web grand public …)

1/4

/// Première partie

ETENDRE LE VOCABULAIRE DQL

Qu'est ce qui peut nous amener à étendre le DQL ?

Base de données : MySQL
Table « products »

id	title	pretax_price	in_stock	on_order
1	Produit 1	120,53	16	15
2	Produit 2	53,69	23	NULL
3	Produit 3	27,17	39	20

```
SELECT title, pretax_price * (in_stock – on_order) AS total
FROM products;
```

title	total
Produit 1	120,53
Produit 2	NULL
Produit 3	297,73

```
SELECT title, pretax_price * (in_stock – IFNULL(on_order, 0)) AS total
FROM products;
```

title	total
Produit 1	120,53
Produit 2	1 234,87
Produit 3	297,73

COMMENT FAIRE DE MÊME EN DQL ?

app/config/config.yml

```yaml
# Doctrine Configuration
doctrine:
    orm:
        entity_managers:
            default:
                ...
                dql:
                    string_functions:
                        IFNULL: MyProject\MyBundle\Doctrine\IfNull
```

COMMENT FAIRE DE MÊME EN DQL ?

```php
use Doctrine\ORM\Query\AST\Functions\FunctionNode,
    Doctrine\ORM\Query\Lexer;

class IfNull extends FunctionNode
{
    private $expr1;
    private $expr2;

    public function parse(\Doctrine\ORM\Query\Parser $parser)
    {
        $parser->match(Lexer::T_IDENTIFIER);
        $parser->match(Lexer::T_OPEN_PARENTHESIS);
        $this->expr1 = $parser->ArithmeticExpression();
        $parser->match(Lexer::T_COMMA);
        $this->expr2 = $parser->ArithmeticExpression();
        $parser->match(Lexer::T_CLOSE_PARENTHESIS);
    }

    public function getSql(\Doctrine\ORM\Query\SqlWalker $sqlWalker)
    {
        return 'IFNULL('
        . $sqlWalker->walkArithmeticPrimary($this->expr1) . ', '
        . $sqlWalker->walkArithmeticPrimary($this->expr2) . ')';
    }
}
```

COMMENT FAIRE DE MÊME EN DQL ?

```php
use Doctrine\ORM\Query\AST\Functions\FunctionNode,
    Doctrine\ORM\Query\Lexer;

class IfNull extends FunctionNode
{
    private $expr1;
    private $expr2;

    public function parse(\Doctrine\ORM\Query\Parser $parser)
    {
        $parser->match(Lexer::T_IDENTIFIER);
        $parser->match(Lexer::T_OPEN_PARENTHESIS);
        $this->expr1 = $parser->ArithmeticExpression();
        $parser->match(Lexer::T_COMMA);
        $this->expr2 = $parser->ArithmeticExpression();
        $parser->match(Lexer::T_CLOSE_PARENTHESIS);
    }

    public function getSql(\Doctrine\ORM\Query\SqlWalker $sqlWalker)
    {
        return 'IFNULL('
        . $sqlWalker->walkArithmeticPrimary($this->expr1) . ', '
        . $sqlWalker->walkArithmeticPrimary($this->expr2) . ')';
    }
}
```

COMMENT FAIRE DE MÊME EN DQL ?

```php
use Doctrine\ORM\Query\AST\Functions\FunctionNode,
    Doctrine\ORM\Query\Lexer;

class IfNull extends FunctionNode
{
    private $expr1;
    private $expr2;

    public function parse(\Doctrine\ORM\Query\Parser $parser)
    {
        $parser->match(Lexer::T_IDENTIFIER);
        $parser->match(Lexer::T_OPEN_PARENTHESIS);
        $this->expr1 = $parser->ArithmeticExpression();
        $parser->match(Lexer::T_COMMA);
        $this->expr2 = $parser->ArithmeticExpression();
        $parser->match(Lexer::T_CLOSE_PARENTHESIS);
    }

    public function getSql(\Doctrine\ORM\Query\SqlWalker $sqlWalker)
    {
        return 'IFNULL('
        . $sqlWalker->walkArithmeticPrimary($this->expr1) . ', '
        . $sqlWalker->walkArithmeticPrimary($this->expr2) . ')';
    }
}
```

COMMENT FAIRE DE MÊME EN DQL ?

```php
use Doctrine\ORM\Query\AST\Functions\FunctionNode,
    Doctrine\ORM\Query\Lexer;

class IfNull extends FunctionNode
{
    private $expr1;
    private $expr2;

    public function parse(\Doctrine\ORM\Query\Parser $parser)
    {
        $parser->match(Lexer::T_IDENTIFIER);
        $parser->match(Lexer::T_OPEN_PARENTHESIS);
        $this->expr1 = $parser->ArithmeticExpression();
        $parser->match(Lexer::T_COMMA);
        $this->expr2 = $parser->ArithmeticExpression();
        $parser->match(Lexer::T_CLOSE_PARENTHESIS);
    }

    public function getSql(\Doctrine\ORM\Query\SqlWalker $sqlWalker)
    {
        return 'IFNULL('
        . $sqlWalker->walkArithmeticPrimary($this->expr1) . ', '
        . $sqlWalker->walkArithmeticPrimary($this->expr2) . ')';
    }
}
```

COMMENT FAIRE DE MÊME EN DQL ?

```php
use Doctrine\ORM\Query\AST\Functions\FunctionNode,
    Doctrine\ORM\Query\Lexer;

class IfNull extends FunctionNode
{
    private $expr1;
    private $expr2;

    public function parse(\Doctrine\ORM\Query\Parser $parser)
    {
        $parser->match(Lexer::T_IDENTIFIER);
        $parser->match(Lexer::T_OPEN_PARENTHESIS);
        $this->expr1 = $parser->ArithmeticExpression();
        $parser->match(Lexer::T_COMMA);
        $this->expr2 = $parser->ArithmeticExpression();
        $parser->match(Lexer::T_CLOSE_PARENTHESIS);
    }

    public function getSql(\Doctrine\ORM\Query\SqlWalker $sqlWalker)
    {
        return 'IFNULL('
        . $sqlWalker->walkArithmeticPrimary($this->expr1) . ', '
        . $sqlWalker->walkArithmeticPrimary($this->expr2) . ')';
    }
}
```

COMMENT FAIRE DE MÊME EN DQL ?

```php
use Doctrine\ORM\Query\AST\Functions\FunctionNode,
    Doctrine\ORM\Query\Lexer;

class IfNull extends FunctionNode
{
    private $expr1;
    private $expr2;

    public function parse(\Doctrine\ORM\Query\Parser $parser)
    {
        $parser->match(Lexer::T_IDENTIFIER);
        $parser->match(Lexer::T_OPEN_PARENTHESIS);
        $this->expr1 = $parser->ArithmeticExpression();
        $parser->match(Lexer::T_COMMA);
        $this->expr2 = $parser->ArithmeticExpression();
        $parser->match(Lexer::T_CLOSE_PARENTHESIS);
    }

    public function getSql(\Doctrine\ORM\Query\SqlWalker $sqlWalker)
    {
        return 'IFNULL('
        . $sqlWalker->walkArithmeticPrimary($this->expr1) . ', '
        . $sqlWalker->walkArithmeticPrimary($this->expr2) . ')';
    }
}
```

UTILISATION

```
$this->createQueryBuilder('p')
     ->select('p.title')
     ->addSelect('p.pretaxPrice * (p.inStock - IFNULL(p.onOrder, 0)) AS total')
     ->getQuery()
     ->getArrayResult();
```

RESULTAT

```
array:3 [▼
  0 => array:2 [▼
    "title" => "produit 1"
    "total" => 3736.43
  ]
  1 => array:2 [▼
    "title" => "produit 2"
    "total" => 1234.87
  ]
  2 => array:2 [▼
    "title" => "produit 3"
    "total" => 1603.03
  ]
]
```

LES BUNDLES À LA RESCOUSSE

https://github.com/beberlei/DoctrineExtensions

DB	Functions
MySQL	ACOS, ASCII, ASIN, ATAN, ATAN2, BINARY, CEIL, CHAR_LENGTH, CONCAT_WS, COS, COT, COUNTIF, CRC32, DATE, DATE_FORMAT, DATEADD, DATEDIFF, DATESUB, DAY, DAYNAME, DEGREES, FIELD, FIND_IN_SET, FLOOR, FROM_UNIXTIME, GROUP_CONCAT, HOUR, IFELSE, IFNULL, LAST_DAY, LEAST, LPAD, MATCH_AGAINST, MD5, MINUTE, MONTH, MONTHNAME, NULLIF, PI, POWER, QUARTER, RADIANS, RAND, REGEXP, REPLACE, ROUND, RPAD, SECOND, SHA1, SHA2, SIN, SOUNDEX, STD, STRTODATE, SUBSTRING_INDEX, TAN, TIME, TIMEDIFF, TIMESTAMPADD, TIMESTAMPDIFF, UNIX_TIMESTAMP, UUID_SHORT, WEEK, WEEKDAY, YEAR, YEARWEEK
Oracle	DAY, MONTH, NVL, TODATE, TRUNC, YEAR
Sqlite	DATE, MINUTE, HOUR, DAY, WEEK, WEEKDAY, MONTH, YEAR, STRFTIME, DATE_FORMAT*, IFNULL, REPLACE, ROUND
PostgreSQL	TO_DATE, TO_CHAR

2/ 4

/// Seconde partie

COMMENT TIRER PROFIT DES EVENTS ?

Les évenements de type LifeCycleCallbacks

- prePersist / postPersist
- preUpdate / postUpdate
- preRemove / postRemove
- postLoad

Les évenements qui ne font pas partie des LifeCycleCallbacks

- loadClassMetadata
- preFlush / onFlush / postFlush

CALLBACKS

```php
/**
 * Article
 *
 * @ORM\Table(name="article")
 * @ORM\Entity()
 * @ORM\HasLifecycleCallbacks()
 */
class Article
{
    /**
     * Date de création
     * @var \DateTime $createdAt
     *
     * @ORM\Column(name="created_at", type="datetime", nullable=true)
     */
    private $createdAt;
```

CALLBACKS

```php
/**
* Article
*
* @ORM\Table(name="article")
* @ORM\Entity()
* @ORM\HasLifecycleCallbacks()
*/
class Article
{
    (...)

    /**
     * Callback sur la création d'un article
     * @ORM\PrePersist
     */
    public function onCreate()
    {
        $this->createdAt = new \DateTime();
    }
```

Pratique ! Mais :

- Restreint à l'instance de l'entité elle-même
- Duplication de code pour les comportements récurrents
- Les modifications sur les collections ne sont pas gérées
- L'ajout d'une nouvelle entité n'est pas géré
- Impossible d'accéder aux services de l'application
- Difficile à utiliser sur les entités éventuelles de bundles tiers

SUBSCRIBERS

```yaml
services:
    app.subscriber.entity_timestamper.:
        class: AppBundle\EventListener\EntityTimestamperSubscriber
        tags:
            - { name: doctrine.event_subscriber, connection: default }
```

SUBSCRIBERS

```php
/**
 * Décrit une entité dont la date de création peut être mise à jour automatiquement
 *
 * @package AppBundle\Entity
 */
interface CreationTimestampableEntityInterface {

    /**
     * Permet de déterminer la date de création de l'entité
     *
     * @param \DateTime $createdAt
     * @return mixed
     */
    public function setCreatedAt(\DateTime $createdAt);

}
```

SUBSCRIBERS

```php
/**
 * Décrit une entité dont la date de création peut être mise à jour automatiquement
 *
 * @package AppBundle\Entity
 */
interface CreationTimestampableEntityInterface {

    /**
     * Permet de déterminer la date de création de l'entité
     *
     * @param \DateTime $createdAt
     * @return mixed
     */
    public function setCreatedAt(\DateTime $createdAt);

}
```

```php
class Article implements CreationTimestampableEntityInterface
```

SUBSCRIBERS / PERSIST

```php
class EntityTimestamperSubscriber implements EventSubscriber {

    (...)

    public function prePersist(LifecycleEventArgs $args)
    {
        $entity = $args->getEntity();

        if (!$entity instanceof CreationTimestampableEntityInterface) {
            return;
        }

        $entity->setCreatedAt(new \DateTime());
    }
```

SUBSCRIBERS / UPDATE

```php
class EntityTimestamperSubscriber implements EventSubscriber {

    (...)

    public function preUpdate(PreUpdateEventArgs $args)
    {
        $entity = $args->getEntity();

        if (!$entity instanceof UpdateTimestampableEntityInterface) {
            return;
        }

        $entity->setUpdatedAt(new \DateTime());

        $args->getEntityManager()->getUnitOfWork()->recomputeSingleEntityChangeSet(
            $args->getEntityManager()->getClassMetadata(get_class($entity)),
            $entity
        );
    }
```

SUBSCRIBERS / UPDATE

```php
class EntityTimestamperSubscriber implements EventSubscriber {

    (...)

    public function preUpdate(PreUpdateEventArgs $args)
    {
        $entity = $args->getEntity();

        if (!$entity instanceof UpdateTimestampableEntityInterface) {
            return;
        }

        $entity->setUpdatedAt(new \DateTime());

        $args->getEntityManager()->getUnitOfWork()->recomputeSingleEntityChangeSet(
            $args->getEntityManager()->getClassMetadata(get_class($entity)),
            $entity
        );
    }
```

SUBSCRIBERS / UPDATE

```php
class EntityTimestamperSubscriber implements EventSubscriber {

    (...)

    public function preUpdate(PreUpdateEventArgs $args)
    {
        // Récupérer l'ensemble des modifications sur l'entité
        $changes = $args->getEntityChangeSet();

        // Modifier une valeur précédemment modifiée
        if ($args->hasChangedField('title')) {
           $args->setNewValue('title', $args->getNewValue('title') . ' (edit)');
        }

    }
```

SUBSCRIBERS / UPDATE

```php
class EntityTimestamperSubscriber implements EventSubscriber {

    (...)

    public function preUpdate(PreUpdateEventArgs $args)
    {
        // Récupérer l'ensemble des modifications sur l'entité
        $changes = $args->getEntityChangeSet();

        // Modifier une valeur précédemment modifiée
        if ($args->hasChangedField('title')) {
            $args->setNewValue('title', $args->getNewValue('title') . ' (edit)');
        }

    }
```

SUBSCRIBERS / UPDATE

```php
class EntityTimestamperSubscriber implements EventSubscriber {

    (...)

    public function preUpdate(PreUpdateEventArgs $args)
    {
        // Récupérer l'ensemble des modifications sur l'entité
        $changes = $args->getEntityChangeSet();

        // Modifier une valeur précédemment modifiée
        if ($args->hasChangedField('title')) {
            $args->setNewValue('title', $args->getNewValue('title') . ' (edit)');
        }

    }
```

Encore mieux !

- Mutualisation du code pour les comportements récurrents
- Accès à l'ensemble des opérations de l'application, y compris sur les entités de bundles tiers
- Accès aux services de l'application si besoin

Mais ...

- La modification des collections de l'entité n'est pas prise en compte
- L'ajout/suppression d'une entité ne sont pas gérés
- Le subscriber est appelé pour chaque entité modifiée/ajoutée

L'ÉVÉNEMENT ONFLUSH

```php
public function onFlush(OnFlushEventArgs $eventArgs)
{
    $em = $eventArgs->getEntityManager();
    $uow = $em->getUnitOfWork();

    foreach ($uow->getScheduledEntityInsertions() as $entity) {
        // ....
    }

    foreach ($uow->getScheduledEntityUpdates() as $entity) {
        // ...
    }

    foreach ($uow->getScheduledEntityDeletions() as $entity) {
        // ....
    }

    foreach ($uow->getScheduledCollectionDeletions() as $col) {
        // .....
    }

    foreach ($uow->getScheduledCollectionUpdates() as $col) {
        // .....
    }
}
```

L'ÉVÉNEMENT ONFLUSH

```php
public function onFlush(OnFlushEventArgs $eventArgs)
{
    $em = $eventArgs->getEntityManager();
    $uow = $em->getUnitOfWork();

    foreach ($uow->getScheduledEntityInsertions() as $entity) {
        // ....
    }

    foreach ($uow->getScheduledEntityUpdates() as $entity) {
        // ...
    }

    foreach ($uow->getScheduledEntityDeletions() as $entity) {
        // ....
    }

    foreach ($uow->getScheduledCollectionDeletions() as $col) {
        // .....
    }

    foreach ($uow->getScheduledCollectionUpdates() as $col) {
        // .....
    }
}
```

L'ÉVÉNEMENT ONFLUSH

```php
/**
 * Log les modifications sur les entités
 *
 * @package AppBundle\EventListener
 */
class UpdatedEntityLoggerSubscriber implements EventSubscriber
{
    …
```

L'ÉVÉNEMENT ONFLUSH

```php
public function onFlush(OnFlushEventArgs $eventArgs)
{
    $em = $eventArgs->getEntityManager();
    $uow = $em->getUnitOfWork();

    foreach ($uow->getScheduledEntityInsertions() as $entity) {
        // ....
    }

    foreach ($uow->getScheduledEntityUpdates() as $entity) {
        // ...
    }

    foreach ($uow->getScheduledEntityDeletions() as $entity) {
        // ....
    }

    foreach ($uow->getScheduledCollectionDeletions() as $col) {
        // .....
    }

    foreach ($uow->getScheduledCollectionUpdates() as $col) {
        // .....
    }
}
```

L'ÉVÉNEMENT ONFLUSH

```php
foreach ($uow->getScheduledEntityUpdates() as $entity) {
    if (!$entity instanceof LoggableEntityInterface) {
        return;
    }

    $log = new EntityLog();
    $log->setUpdatedFields($uow->getEntityChangeSet($entity));
    $log->setTargetId($entity->getId());
    $log->setType($entity->getEntityName());

    $em->persist($log);
}

$uow->computeChangeSets();
```

3/ 4

/// Troisième partie

ASTUCES POUR AMÉLIORER LES PERFORMANCES

Base de données : MySQL
30 000 produits

id	title	pretax_price	in_stock	...
1	Produit x	120,53	16	…
…	…	…	…	…
30000	Produit y	27,17	39	…

CAS USUEL

```php
$results = $this->createQueryBuilder('p')
    ->getQuery()
    ->getResult();

foreach ($results as $result) {
    $result->setPretaxPrice(
          $result->getPretaxPrice() - $this->getProductDiscount($result->getId())
    );
}

$this->getEntityManager()->flush();
```

CAS USUEL - PERFORMANCES

23 435 ms	218,75 MB
TEMPS D'EXÉCUTION TOTAL	**MÉMOIRE UTILISÉE**

BATCHSIZE (20)

```php
$i = 0;
$batchSize = 20;

$results = $this->createQueryBuilder('p')
   ->getQuery()
   ->getResult();

foreach ($results as $result) {
    $result->setPretaxPrice(
        $result->getPretaxPrice() - $this->getProductDiscount($result->getId())
    );

    if (($i % $batchSize) === 0) {
        $this->getEntityManager()->flush();
    }

    ++$i;
}

$this->getEntityManager()->flush();
$this->getEntityManager()->clear();
```

BATCHSIZE (20) - PERFORMANCES

23 435

1 419 137 ms

TEMPS D'EXÉCUTION TOTAL

+ 5955 %

218,75

237,50 MB

MÉMOIRE UTILISÉE

+ 8 %

DESACTIVATION DES LOGS

```php
// Désactivation des logs SQL
$this->getEntityManager()->getConnection()->getConfiguration()->setSQLLogger(null);
```

BATCHSIZE (20) SANS LOGS
PERFORMANCES

23 435

1 020 377 ms

TEMPS D'EXÉCUTION TOTAL

+ 4254 %

218,75

117,25 MB

MÉMOIRE UTILISÉE

- 46 %

BATCHSIZE (1000) SANS LOGS
PERFORMANCES

23 435

28 705 ms

TEMPS D'EXÉCUTION TOTAL

+ 22 %

218,75

114.75 MB

MÉMOIRE UTILISÉE

- 48 %

ITERATE

```php
public function importPrice()
{
    $em = $this->getEntityManager();

    $connection = $em->getConnection();
    $connection->getConfiguration()->setSQLLogger(null);

    $results = $this->createQueryBuilder('p')->getQuery();


    foreach ($results->iterate() as $result) {

        $result[0]->setPretaxPrice(
             $result[0]->getPretaxPrice() - $this->getProductDiscount($result->getId())
        );

        $em->flush($result[0]);
        $em->detach($result[0]);
    }
}
```

ITERATE - PERFORMANCES

23 435

218,75

482 825 ms

8,50 MB

TEMPS D'EXÉCUTION TOTAL + 1960 %

MÉMOIRE UTILISÉE - 96 %

ITERATE / TRANSACTION

```php
public function importPrice()
{
    $em = $this->getEntityManager();

    $connection = $em->getConnection();
    $connection->getConfiguration()->setSQLLogger(null);

    $results = $this->createQueryBuilder('p')->getQuery();

    $connection->beginTransaction();

    foreach ($results->iterate() as $result) {

        $result[0]->setPretaxPrice(
             $result[0]->getPretaxPrice() - $this->getProductDiscount($result->getId())
        );

        $em->flush($result[0]);
        $em->detach($result[0]);
    }

    $connection->commit();
}
```

ITERATE / TRANSACTION - PERFORMANCES

23 435

18 373 ms

TEMPS D'EXÉCUTION TOTAL

- 22 %

218,75

173,08 MB

MÉMOIRE UTILISÉE

- 21 %

ITERATE / UPDATE / TRANSACTION

```php
public function importPrice()
{
    $em = $this->getEntityManager();
    $connection = $em->getConnection();

    $connection->getConfiguration()->setSQLLogger(null);

    $results = $this->createQueryBuilder('p')->getQuery();

    $connection->beginTransaction();

    foreach ($results->iterate() as $result) {
        $connection->update(
            $em->getClassMetadata('AppBundle:Product')->getTableName(),
            array(
                'pretax_price' => $result[0]->getPretaxPrice() -
                    $this->getProductDiscount($result->getId())
            ),
            array('id' => $result[0]->getId())
        );
        $em->detach($result[0]);
    }

    $connection->commit();
}
```

ITERATE / UPDATE / TRANSACTION PERFORMANCES

23 435

6 453 ms

TEMPS D'EXÉCUTION TOTAL

- 72 %

218,75

8,25 MB

MÉMOIRE UTILISÉE

- 96 %

UPDATE FULL SQL

```php
public function importPrice()
{
    $this->getEntityManager()->createQueryBuilder()
        ->update('AppBundle:Product', 'p')
        ->set('p.pretaxPrice', 'p.pretaxPrice * 0.9')
        ->getQuery()
        ->execute();
}
```

UPDATE FULL SQL - PERFORMANCES

23 435	218,75
158 ms	4,50 MB
TEMPS D'EXÉCUTION TOTAL	**MÉMOIRE UTILISÉE**
- 99 %	- 98 %

/// Troisième partie

ASTUCES POUR AMÉLIORER LES PERFORMANCES

Requêtes partielles

CAS USUEL

```php
public function getAllProductsNoPartial()
{
     return $this->createQueryBuilder('p')
          ->select('p, c')
          ->leftJoin('p.category', 'c')
          ->getQuery()
          ->getResult();
}
```

CAS USUEL - REQUÊTE SQL

```
SELECT
    p0_.id AS id_0,
    p0_.title AS title_1,
    p0_.vat_rate AS vat_rate_2,
    p0_.pretax_price AS pretax_price_3,
    p0_.in_stock AS in_stock_4,
    p0_.on_order AS on_order_5,
    p0_.created_at AS created_at_6,
    p0_.updated_at AS updated_at_7,
    p0_.deleted AS deleted_8,
    c1_.id AS id_9,
    c1_.title AS title_10,
    c1_.created_at AS created_at_11,
    c1_.updated_at AS updated_at_12,
    p0_.category_id AS category_id_13
FROM
    product p0_
LEFT JOIN
    category c1_ ON p0_.category_id = c1_.id;
```

CAS USUEL - RESULTAT

```
array:2 [▼
  0 => Product {#660 ▼
    -id: 25001
    -title: "produit 25001"
    #vatRate: "20.00"
    #pretaxPrice: "25.37"
    -inStock: 718
    -onOrder: 108
    -createdAt: DateTime {#659 ▶}
    -updatedAt: DateTime {#627 ▶}
    #category: Category {#650 ▼
      -id: 1
      -title: "category 1"
      -createdAt: DateTime {#655 ▶}
      -updatedAt: DateTime {#656 ▶}
      -products: PersistentCollection {#631 ▶}
    }
    -deleted: null
  }
  1 => Product {#666 ▼
    -id: 25003
    -title: "produit 25003"
    #vatRate: "20.00"
    #pretaxPrice: "23.44"
    -inStock: 330
    -onOrder: 42
    -createdAt: DateTime {#630 ▶}
    -updatedAt: DateTime {#662 ▶}
    #category: Category {#667 ▼
      -id: 2
      -title: "cat 2 "
      -createdAt: DateTime {#664 ▶}
      -updatedAt: DateTime {#665 ▶}
      -products: PersistentCollection {#668 ▶}
    }
    -deleted: null
  }
]
```

UTILISATION

```php
public function getAllProductsNoPartial()
{
    return $this->createQueryBuilder('p')
        ->select('p.id, p.title, p.pretaxPrice, p.vatRate')
        ->addSelect('c.id, c.title')
        ->leftJoin('p.category', 'c')
        ->getQuery()
        ->getResult();
}
```

REQUÊTE SQL

```
SELECT
    p0_.id AS id_0,
    p0_.title AS title_1,
    p0_.vat_rate AS vat_rate_2,
    p0_.pretax_price AS pretax_price_3,
    c1_.id AS id_4,
    c1_.title AS title_5,
FROM
    product p0_
LEFT JOIN
    category c1_ ON p0_.category_id = c1_.id;
```

RESULTAT

```
array:2 [▼
  0 => array:5 [▼
    "id" => 25001
    "title" => "Produit 25001"
    "vatRate" => 20
    "pretaxPrice" => 25.37
    "category" => array:2 [▼
      "id" => 1
      "title" => "Category 1"
    ]
  ]
  1 => array:5 [▼
    "id" => 25002
    "title" => "Produit 25002"
    "vatRate" => 20
    "pretaxPrice" => 23.44
    "category" => array:2 [▼
      "id" => 2
      "title" => "Category 2"
    ]
  ]
]
```

CLASSE PRODUCT

```php
/**
 * Product
 *
 * @ORM\Table(name="product")
 * @ORM\Entity()
 */
class Product
{
    /**
     * Get fullTaxPrice
     *
     * @return float
     */
    public function getFullTaxPrice()
    {
        return $this->vatRate * $this->pretaxPrice / 100 + $this->pretaxPrice;
    }
```

UTILISATION DE PARTIAL

```php
public function getAllProductsPartial()
{
    return $this->createQueryBuilder('p')
            ->select('partial p.{id, title, vatRate, pretaxPrice}')
            ->addSelect('partial c.{id, title}')
             ->leftJoin('p.category', 'c')
            ->getQuery()
            ->getResult();
}
```

UTILISATION DE PARTIAL - REQUÊTE SQL

```
SELECT
    p0_.id AS id_0,
    p0_.title AS title_1,
    p0_.vat_rate AS vat_rate_2,
    p0_.pretax_price AS pretax_price_3,
    c1_.id AS id_4,
    c1_.title AS title_5,
    p0_.category_id AS category_id_6
FROM
    product p0_
LEFT JOIN
    category c1_ ON p0_.category_id = c1_.id ;
```

UTILISATION DE PARTIAL - RESULTAT

```
array:2 [▼
  0 => Product {#367 ▼
    -id: 25001
    -title: "produit 25001"
    #vatRate: "20.00"
    #pretaxPrice: "16.01"
    -inStock: null
    -onOrder: null
    -createdAt: null
    -updatedAt: null
    #category: Category {#363 ▼
      -id: 1
      -title: "category 1"
      -createdAt: null
      -updatedAt: null
      -products: PersistentCollection {#364 ▶}
    }
    -deleted: false
  }
  1 => Product {#360 ▼
    -id: 25003
    -title: "produit 25003"
    #vatRate: "20.00"
    #pretaxPrice: "23.44"
    -inStock: null
    -onOrder: null
    -createdAt: null
    -updatedAt: null
    #category: Category {#359 ▼
      -id: 2
      -title: "cat 2 "
      -createdAt: null
      -updatedAt: null
      -products: PersistentCollection {#356 ▶}
    }
    -deleted: false
  }
]
```

AVERTISSEMENT

- Les requêtes partielles sont **à utiliser avec précautions**
- Eviter que les instances partielles ne soient transmises à de grandes portions de code pour éviter tout effet de bord
- Limiter leurs utilisations sur des pages où les performances sont à privilégier.

4/4

/// Quatrième partie

DÉVELOPPER PLUS RAPIDEMENT GRÂCE À DOCTRINE

/// Quatrième partie

DÉVELOPPER PLUS RAPIDEMENT GRÂCE À DOCTRINE

Custom Hydrators

Les hydrators par défaut de Doctrine

- Query::HYDRATE_OBJECT
- Query::HYDRATE_ARRAY
- Query::HYDRATE_SCALAR
- Query::HYDRATE_SINGLE_SCALAR

/// Quatrième partie

DÉVELOPPER PLUS RAPIDEMENT GRÂCE À DOCTRINE

Hydrateur de type liste (clé/valeur)

HYDRATEUR DE TYPE LISTE (CLÉ/VALEUR)

```php
use Doctrine\ORM\Internal\Hydration\AbstractHydrator;
use PDO;

class KeyValueListHydrator extends AbstractHydrator
{
    protected function hydrateAllData()
    {
        $results = array();

        foreach ($this->_stmt->fetchAll(PDO::FETCH_ASSOC) as $row) {
            $this->hydrateRowData($row, $results);
        }

        return $results;
    }
```

HYDRATEUR DE TYPE LISTE (CLÉ/VALEUR)

```php
protected function hydrateRowData(array $data, array &$result)
{
    $keys = array_keys($data);
    $keyCount = count($keys);

    // La première colonne est considérée comme étant la clé
    $key = $data[$keys[0]];

    if ($keyCount == 2) {
        // Si deux colonnes alors
        // la seconde colonne est considéré comme la valeur
        $value = $data[$keys[1]];
    } else {
        // Sinon le reste des colonnes excepté la premières
        // sont considréré comme la talbeau de valeur
        array_shift($data);
        $value = array_values($data);
    }

    $result[$key] = $value;
}
```

HYDRATEUR DE TYPE LISTE (CLÉ/VALEUR)

```php
protected function hydrateRowData(array $data, array &$result)
{
    $keys = array_keys($data);
    $keyCount = count($keys);

    // La première colonne est considérée comme étant la clé
    $key = $data[$keys[0]];

    if ($keyCount == 2) {
        // Si deux colonnes alors
        // la seconde colonne est considéré comme la valeur
        $value = $data[$keys[1]];
    } else {
        // Sinon le reste des colonnes excepté la premières
        // sont considréré comme la talbeau de valeur
        array_shift($data);
        $value = array_values($data);
    }

    $result[$key] = $value;
}
```

HYDRATEUR DE TYPE LISTE (CLÉ/VALEUR)

```php
protected function hydrateRowData(array $data, array &$result)
{
    $keys = array_keys($data);
    $keyCount = count($keys);

    // La première colonne est considérée comme étant la clé
    $key = $data[$keys[0]];

    if ($keyCount == 2) {
        // Si deux colonnes alors
        // la seconde colonne est considéré comme la valeur
        $value = $data[$keys[1]];
    } else {
        // Sinon le reste des colonnes excepté la premières
        // sont considréré comme la talbeau de valeur
        array_shift($data);
        $value = array_values($data);
    }

    $result[$key] = $value;
}
```

HYDRATEUR DE TYPE LISTE (CLÉ/VALEUR)

```php
protected function hydrateRowData(array $data, array &$result)
{
    $keys = array_keys($data);
    $keyCount = count($keys);

    // La première colonne est considérée comme étant la clé
    $key = $data[$keys[0]];

    if ($keyCount == 2) {
        // Si deux colonnes alors
        // la seconde colonne est considéré comme la valeur
        $value = $data[$keys[1]];
    } else {
        // Sinon le reste des colonnes excepté la premières
        // sont considréré comme la talbeau de valeur
        array_shift($data);
        $value = array_values($data);
    }

    $result[$key] = $value;
}
```

HYDRATEUR DE TYPE LISTE (CLÉ/VALEUR)

```php
protected function hydrateRowData(array $data, array &$result)
{
    $keys = array_keys($data);
    $keyCount = count($keys);

    // La première colonne est considérée comme étant la clé
    $key = $data[$keys[0]];

    if ($keyCount == 2) {
        // Si deux colonnes alors
        // la seconde colonne est considéré comme la valeur
        $value = $data[$keys[1]];
    } else {
        // Sinon le reste des colonnes excepté la premières
        // sont considréré comme la talbeau de valeur
        array_shift($data);
        $value = array_values($data);
    }

    $result[$key] = $value;
}
```

DECLARATION

app/config/config.yml

```yaml
doctrine:
    orm:
        hydrators:
            KeyValueListHydrator: AppBundle\Doctrine\Hydrators\KeyValueListHydrator
```

UTILISATION

```php
public function getCategoriesList()
{
    return $this->createQueryBuilder('c')
        ->select('c.id, c.title')
        ->getQuery()
        ->getResult('KeyValueListHydrator');
}
```

RÉSULTAT

```
array:2 [▼
  1 => "category 1"
  2 => "category 2"
]
```

/// Quatrième partie

DÉVELOPPER PLUS RAPIDEMENT GRÂCE À DOCTRINE

Filtres

Table « news »

id	title	body	accreditation_level
1	Changement des codes nucléaires	273 …	9
2	Opération militaire en cours	Urg …	7
3	Fuite dans les toilettes du RDC	Une fui …	1

REQUÊTE HABITUELLE

```php
$query = $this->createQueryBuilder('n')
    ->where('n.accreditationLevel <= :userLevel')
    ->setParameter('userLevel', $user->getAccreditationLevel())
    ->getQuery();
```

CREATION D'UN FILTRE

```php
namespace AppBundle\Doctrine\Filter;

use Doctrine\ORM\Query\Filter\SQLFilter;
use Doctrine\ORM\Mapping\ClassMetadata;

/**
* Filtre les actualités selon le niveau d'accréditation du lecteur
*/
class NewsAccreditationFilter extends SQLFilter
{
    /**
     * Gets the SQL query part to add to a query.
     *
     * @param ClassMetaData $targetEntity
     * @param string        $targetTableAlias
     *
     * @return string The constraint SQL if there is available, empty string otherwise.
     */
    public function addFilterConstraint(ClassMetadata $targetEntity, $targetTableAlias)
    {

    }
}
```

CREATION D'UN FILTRE

```php
public function addFilterConstraint(ClassMetadata $targetEntity, $targetTableAlias)
{
    if ($targetEntity->name !== 'AppBundle\Entity\News') {
        return '';
    }

    try {
        $accreditationLevel = $this->getParameter('accreditationLevel');
    } catch (\InvalidArgumentException $e) {
        return '';
    }

    return sprintf(
        '%s.%s <= %s',
        $targetTableAlias,
        $targetEntity->getColumnName('accreditationLevel'),
        $accreditationLevel
    );
}
```

CREATION D'UN FILTRE

```php
public function addFilterConstraint(ClassMetadata $targetEntity, $targetTableAlias)
{
    if ($targetEntity->name !== 'AppBundle\Entity\News') {
        return '';
    }

    try {
        $accreditationLevel = $this->getParameter('accreditationLevel');
    } catch (\InvalidArgumentException $e) {
        return '';
    }

    return sprintf(
        '%s.%s <= %s',
        $targetTableAlias,
        $targetEntity->getColumnName('accreditationLevel'),
        $accreditationLevel
    );
}
```

CREATION D'UN FILTRE

```php
public function addFilterConstraint(ClassMetadata $targetEntity, $targetTableAlias)
{
    if ($targetEntity->name !== 'AppBundle\Entity\News') {
        return '';
    }

    try {
        $accreditationLevel = $this->getParameter('accreditationLevel');
    } catch (\InvalidArgumentException $e) {
        return '';
    }

    return sprintf(
        '%s.%s <= %s',
        $targetTableAlias,
        $targetEntity->getColumnName('accreditationLevel'),
        $accreditationLevel
    );
}
```

n.accreditation_level <= 5

DECLARATION DU FILTRE

```yaml
# Doctrine Configuration
doctrine:
    # ..
    orm:
        # ..
        filters:
            news_accreditation_filter:
                class: AppBundle\Doctrine\Filter\NewsAccreditationFilter
                enabled: false
```

DECLARATION DU LISTENER

```yaml
# Doctrine Configuration
doctrine:
    # ..
    orm:
        # ..
        filters:
            news_accreditation_filter:
                class: AppBundle\Doctrine\Filter\NewsAccreditationFilter
                enabled: false
```

```yaml
# Services Declaration
services:
    app.filter.configurator:
        class: AppBundle\EventListener\FilterConfiguratorListener
        arguments:
            - "@doctrine.orm.entity_manager"
            - "@security.token_storage"
        tags:
            - { name: kernel.event_listener, event: kernel.request }
```

CREATION DU LISTENER

```php
namespace AppBundle\EventListener;

use Symfony\Component\Security\Core\Authentication\Token\Storage\TokenStorageInterface;
use Doctrine\Common\Persistence\ObjectManager;
use Doctrine\Common\Annotations\Reader;

/**
* Configuration des filtres Doctrine
*/
class FilterConfiguratorListener
{
    /**
     * @var \Doctrine\Common\Persistence\ObjectManager
     */
    protected $em;

    /**
     * @var
\Symfony\Component\Security\Core\Authentication\Token\Storage\TokenStorageInterface
     */
    protected $tokenStorage;

    public function __construct(ObjectManager $em, TokenStorageInterface $tokenStorage)
    {
        $this->em = $em;
        $this->tokenStorage = $tokenStorage;
    }
}
```

CREATION DU LISTENER

```php
/**
* Configuration des filtres Doctrine
*/
class FilterConfiguratorListener
{
    // ...

    public function onKernelRequest()
    {
        $filter = $this->em->getFilters()->enable('news_accreditation_filter');

        if ($user = $this->getUser()) {
            $accreditationLevel = $user->getAccreditationLevel();
        }
        else {
            $accreditationLevel = 0;
        }

        $filter->setParameter('accreditationLevel', $accreditationLevel);
    }

    private function getUser()
    {
        $token = $this->tokenStorage->getToken();

        if (!$token || !$user = $token->getUser()) {
            return NULL;
        }

        return $user;
    }
}
```

CREATION DU LISTENER

```php
/**
* Configuration des filtres Doctrine
*/
class FilterConfiguratorListener
{
    // ...

    public function onKernelRequest()
    {
        $filter = $this->em->getFilters()->enable('news_accreditation_filter');

        if ($user = $this->getUser()) {
            $accreditationLevel = $user->getAccreditationLevel();
        }
        else {
            $accreditationLevel = 0;
        }

        $filter->setParameter('accreditationLevel', $accreditationLevel);
    }

    private function getUser()
    {
        $token = $this->tokenStorage->getToken();

        if (!$token || !$user = $token->getUser()) {
            return NULL;
        }

        return $user;
    }
}
```

RESULTATS SUR UN FIND ALL

```
$news = $em->getRepository('AppBundle:News')->findAll();
```

```
SELECT t0.id AS id_1, t0.title AS title_2, t0.body AS body_3,
t0.accreditation_level AS accreditation_level_4
FROM news t0
WHERE ((t0.accreditation_level <= '1'))
```

```
array:1 [▼
  0 => News {#321 ▼
    -id: 3
    -title: "Fuite dans les toilettes du RDC"
    -body: "Une fui..."
    -accreditationLevel: 1
  }
]
```

```
SELECT t0.id AS id_1, t0.title AS title_2, t0.body AS body_3,
t0.accreditation_level AS accreditation_level_4
FROM news t0
WHERE ((t0.accreditation_level <= '9'))
```

```
array:3 [▼
  0 => News {#321 ▼
    -id: 1
    -title: "Changement des codes nucléaires"
    -body: "5645454"
    -accreditationLevel: 9
  }
  1 => News {#323 ▼
    -id: 2
    -title: "Opération militaire en cours"
    -body: "Urgent : Une .."
    -accreditationLevel: 7
  }
  2 => News {#324 ▼
    -id: 3
    -title: "Fuite dans les toilettes du RDC"
    -body: "Une fui..."
    -accreditationLevel: 1
  }
]
```

ALLER PLUS LOIN : COMPORTEMENT GENERIQUE

```php
<?php

namespace AppBundle\Doctrine\Annotation;

use Doctrine\Common\Annotations\Annotation;
use Doctrine\Common\Annotations\Annotation\Target;

/**
* Décrit une entité qui peut être supprimée logiquement
*
* @Annotation
* @Target("CLASS")
*/
final class SoftDeletable
{
   public $deletionField = 'deleted';
}
```

APPLICATION SUR UNE ENTITE

```php
namespace AppBundle\Entity;

use Doctrine\ORM\Mapping as ORM;
use AppBundle\Doctrine\Annotation\SoftDeletable;

/**
 * @SoftDeletable()
 * @ORM\Table(name="product")
 * @ORM\Entity(repositoryClass="AppBundle\Repository\ProductRepository")
 */
class Product
{
    // ...

    /**
     * Statut de suppression logique du produit
     * @var boolean $deleted
     *
     * @ORM\Column(name="deleted", type="boolean", nullable=true)
     */
    private $deleted = false;

    // ...
}
```

DECLARATION DE NOTRE FILTRE GENERIQUE

```yaml
# Doctrine Configuration
doctrine:
    # ..
    orm:
        # ..
        filters:
            soft_deleted_filter:
              class: AppBundle\Doctrine\Filter\SoftDeletedFilter
              enabled: false
```

DECLARATION DE NOTRE FILTRE GENERIQUE

```yaml
# Doctrine Configuration
doctrine:
    # ..
    orm:
        # ..
        filters:
            soft_deleted_filter:
              class: AppBundle\Doctrine\Filter\SoftDeletedFilter
              enabled: false
```

```yaml
# Services Declaration
services:
    app.filter.configurator:
        class: AppBundle\EventListener\FilterConfiguratorListener
        arguments:
            - "@doctrine.orm.entity_manager"
            - "@security.token_storage"
            - "@annotation_reader"
        tags:
            - { name: kernel.event_listener, event: kernel.request }
```

DECLARATION DE NOTRE FILTRE GENERIQUE

```yaml
# Doctrine Configuration
doctrine:
    # ..
    orm:
        # ..
        filters:
            soft_deleted_filter:
              class: AppBundle\Doctrine\Filter\SoftDeletedFilter
              enabled: false
```

```yaml
# Services Declaration
services:
    app.filter.configurator:
        class: AppBundle\EventListener\FilterConfiguratorListener
        arguments:
            - "@doctrine.orm.entity_manager"
            - "@security.token_storage"
            - "@annotation_reader"
        tags:
            - { name: kernel.event_listener, event: kernel.request }
```

MODIFICATION DU LISTENER

```php
/**
 * Configuration des filtres Doctrine
 */
class FilterConfiguratorListener
{
    public function onKernelRequest()
    {
        // Configuration du filtre SoftDeletable
        $filter = $this->em->getFilters()->enable('soft_deleted_filter');
        $filter->setAnnotationReader($this->annotationReader);
```

CREATION DU FILTRE

```php
namespace AppBundle\Doctrine\Filter;

use Doctrine\ORM\Query\Filter\SQLFilter;
use Doctrine\ORM\Mapping\ClassMetadata;
use Doctrine\Common\Annotations\Reader;

/**
 * Filtre les entités supprimées logiquement
 *
 * @package AppBundle\Doctrine\Filter
 */
class SoftDeletedFilter extends SQLFilter
{
    /**
     * Lecteur d'annotation
     * @var \Doctrine\Common\Annotations\Reader
     */
    protected $annotationReader;


    public function addFilterConstraint(ClassMetadata $targetEntity, $targetTableAlias)
    {

    }

    public function setAnnotationReader(Reader $annotationReader)
    {
        $this->annotationReader = $annotationReader;
    }
```

CREATION DU FILTRE

```php
public function addFilterConstraint(ClassMetadata $targetEntity, $targetTableAlias)
{
    if (empty($this->annotationReader)) {
        return '';
    }

    // La requête en cours concerne t'elle une entité supprimable logiquement ?
    $softDeletable = $this->annotationReader->getClassAnnotation(
        $targetEntity->getReflectionClass(),
        'AppBundle\\Doctrine\\Annotation\\SoftDeletable'
    );

    if (!$softDeletable || empty($softDeletable->deletionField)) {
        return '';
    }

    return sprintf('%s.%s IS NULL', $targetTableAlias, $softDeletable->deletionField);
}
```

CREATION DU FILTRE

```php
public function addFilterConstraint(ClassMetadata $targetEntity, $targetTableAlias)
{
    if (empty($this->annotationReader)) {
        return '';
    }

    // La requête en cours concerne t'elle une entité supprimable logiquement ?
    $softDeletable = $this->annotationReader->getClassAnnotation(
        $targetEntity->getReflectionClass(),
        'AppBundle\\Doctrine\\Annotation\\SoftDeletable'
    );

    if (!$softDeletable || empty($softDeletable->deletionField)) {
        return '';
    }

    return sprintf('%s.%s = 0', $targetTableAlias, $softDeletable->deletionField);
}
```

p.deleted = 0

DESACTIVER LE FILTRE A LA DEMANDE

```php
namespace AppBundle\Repository;

class ProductRepository extends EntityRepository
{
    /**
     * Retourne les produits supprimés
     * @return array
     */
    public function getDeletedProducts()
    {
        $em = $this->getEntityManager();

        $em->getFilters()->disable('soft_deleted_filter');

        $products = $this->findBy(array('deleted' => true));

        $em->getFilters()->enable('soft_deleted_filter');

        return $products;
    }
}
```

DESACTIVER LE FILTRE A LA DEMANDE

```php
class SoftDeletedFilter extends SQLFilter
{
    /**
     * Permet de désactiver le filtre temporairement
     * @var bool
     */
    protected $disabled = false;

    public function addFilterConstraint(ClassMetadata $targetEntity, $targetTableAlias)
    {
        if ($this->disabled || empty($this->annotationReader)) {
            return '';
        }

        // ...
    }

    public function disable()
    {
        $this->disabled = true;
    }

    public function enable()
    {
        $this->disabled = false;
    }
}
```

DESACTIVER LE FILTRE A LA DEMANDE

```php
namespace AppBundle\Repository;

class ProductRepository extends EntityRepository
{
    /**
     * Retourne les produits supprimés
     * @return array
     */
    public function getDeletedProducts()
    {
        $em = $this->getEntityManager();

        if ($em->getFilters()->isEnabled('soft_deleted_filter')) {
            $em->getFilters()->getFilter('soft_deleted_filter')->disable();
        }

        $products = $this->findBy(array('deleted' => true));

        if ($em->getFilters()->isEnabled('soft_deleted_filter')) {
            $em->getFilters()->getFilter('soft_deleted_filter')->enable();
        }

        return $products;
    }
}
```

Merci
pour votre attention,
et bon appétit ☺

RETROUVEZ WEBNET