

Atelier Zend Framework : Implémentation de l'héritage de tables SQL sous Zend Framework

par Julien Pauli ([Tutoriels, article et conférences PHP et développement web](#)) ([Blog](#))

Date de publication : 20/04/2007

Dernière mise à jour : 02/02/2008

Je me suis amusé une fois de plus dans Zend Framework à essayer de lui faire accepter l'héritage de tables.

Comment traduire l'héritage d'une table, sous ZF ?

- I - Journée 1
- II - Journée 2

I - Journée 1

Je m'explique : si j'ai une table "membres" qui contient une colonne "is_admin" que l'on met à 0 pour un membre normal, et à 1 pour un admin; Je veux traduire ca dans ZF...

Lui dire que tout compte fait, un Admin est un Membre, même si il n'existe pas de table Admin à proprement parler.

Je traduis le fait qu'un Admin est un Membre, ils partagent la même table, mais un admin est caractérisé par is_admin=1 dans la table, vous suivez ?

En gros je veux pouvoir avoir 2 objets distincts Admin, et Membres, mais reliés à la même table, et dont la logique SQL serait gérée en arrière plan.

En UML c'est trop simple, extends et c'est fini (presque). Mais là, on est dans une base, et elle est relationnelle et non objet (d'où l'utilisation de la couche ORM de Zend Framework). Il faut pouvoir transformer en interne le mécanisme de l'ORM pour pouvoir lui faire comprendre un héritage de tables, de manière simple...

J'ai trouvé une solution, qui vaut ce qu'elle vaut, mais qui a pour but de montrer qu'il est très facile de développer dans ZF, et de monter sa sauce. ZF a été construit avec plusieurs étages objets bien conçus, et on peut donc intervenir ou l'on veut pour régler soit même sa logique.

Pour notre exemple, j'ai retenu qu'il faut donc créer une classe Membre extends Zend_Db_Table_Abstract, et Admin extends Membre.

Maintenant que niveau UML (objet) c'est fait, il faut revoir le mécanisme du coté de la base, pour qu'il interprète ca correctement, et quand je lui demande " trouve moi un admin qui a modéré ce post ", il cherche dans la table membres, les membres ayant modérés ce post, et étant admin :-)

Il faut donc rajouter la logique SQL obligatoire pour définir un Admin, toutes les requêtes d'appel (SELECT) devront se terminer par "AND is_admin = 1".

Il ne faut pas aussi oublier dans la classe membre, de spécifier "is_admin = 0" cette fois.

Ainsi il faut redéfinir l'appel le plus bas : **_fetch()** (hérité de Zend_Db_Table), pour modifier la requête au tout dernier moment.

C'est pas fini :-)

Il faut aussi redéfinir le conteneur Zend_Db_Table_Row en un AdminRow par exemple, de manière à ce que lorsqu'on fait un createRow(), on n'oublie pas de spécifier tout de suite la valeur de is_admin à 1. On se retrouve avec un conteneur vide, mais intègre, il représente bien un admin, la valeur de sa colonne "is_admin" sera automatiquement à 1.

Il faut enfin s'assurer lors d'un **save()**, de l'intégrité des données, dans **save()**, on vérifie que "is_admin" est à 1 dans l'objet, avant de le sauver, car ce paramètre peut très bien avoir été modifié dans la vie de l'objet; et on se retrouverait avec une instance d'Admin enregistrant un Membre, super pas top du tout ^^

L'ennui avec ce système, c'est qu'il est exclusif au niveau SQL. Un admin n'est PAS un membre, on ne peut plus chercher "tous les membres, qu'ils soient admin ou pas, qui ont écrit un message", en une seule requête. Ca sera ou un admin, ou un membre, ou la même requête sur les 2 objets, puis on englobe le tout (bofbof)

Il faudrait pour cela créer une troisième classe ensembliste, sans restriction SQL, une "personne" serait un groupe de "membres" et "d'admins", tout ça avec une seule table SQL, et 3 classes métiers pour gérer ça. A suivre éventuellement ...

Voici déjà mes idées :

```
<?php
class Admin extends Membre
{
    protected $_rowClass = 'AdminDbTableRow';

    protected function _fetch(Zend_Db_Table_Select $select)
    {
        $select->where('is_admin = ?', 1);
        return parent::_fetch($select);
    }
}
?>

<?php
class AdminDbTableRow extends Zend_Db_Table_Row_Abstract
{
    public function __construct(array $config = array())
    {
        parent::__construct($config);
        $this->_data['is_admin'] = 1;
    }

    public function save()
    {
        if ($this->_data['is_admin'] != 1){
            throw new Zend_Db_Table_Row_Exception('Données corrompues, vous tentez de sauver un admin qui n\'en est pas un');
        }
    }
}
```

Halte là! C'est toujours pas fini. Il faut enfin utiliser l'introspection pour pouvoir gérer les relations, et dire que si on cherche un Admin, par **findParentAdmin()**, alors qu'on a donné une referenceMap avec Membre, ben ça fonctionne aussi.

```
<?php
abstract class MyTables extends Zend_Db_Table_Abstract
{
    public function getReference($tableClassname, $ruleKey = null)
    {
        // code ... ..
        foreach ($this->_referenceMap as $reference) {
            $reflect = new ReflectionClass($tableClassname);
            if ($reference[self::REF_TABLE_CLASS] == $tableClassname || $reference[self::REF_TABLE_CLASS]
            == $reflect->getParentClass()->getName()) {
                return $reference;
            }
        }
        // code .. ..
    }
}
?>
```

Bon j'ai sorti ça comme ça, à première vue ça me semble tenir la route, mais rien n'est garanti, personnalisez l'idée ;-)

Ce cas là est concret et issu d'un projet sur lequel je bosse.

Il y a aussi le cas "non ensembliste", ou "exclusif" : une table évènements, possède une colonne "type_evenement". Egale à 1, il s'agit d'un Voyage, égale à 2 il s'agit d'une Visite.

Je veux 2 classes, mappées sur la même table, dans la même logique qu'auparavant, et cette fois ci, exclusive, un Evènement est finalement abstrait, il est OU de type voyage et donnera naissance à un objet Voyage, ou de type visite, donnant naissance à un objet Visite...

II - Journée 2

Une journée de réflexion de plus m'amène à vous donner une solution plus poussée, et mieux testée :

Une (et en tout une seule) table évènements avec une colonne type. Si type=0 alors l'évènement est un voyage, si type=1 alors c'est une visite.

Pour l'ORM, j'ai créé 3 classes, Evenement, Visite, et Voyage.

Evenement définit le mapping. Visite et Voyage héritent d'évènement.

Dans visite on redéfinit `_fetch()`, en lui rajoutant `$select->where('type = ?', '1');`

Dans voyage on redéfinit `_fetch()`, en lui rajoutant `$select->where('type = ?', '0');`

On laisse évidemment `_fetch()` de évènement indemne.

Evenement n'hérite pas directement de `Zend_Db_Table`, mais de 'AbstractTables' que j'ai créée.

AbstractTables redéfinit `getReference()`, qui va servir à référencer les tables entre elles.

Lorsque je définis le mapping, un membre par exemple, gère un évènement. Mais sur ce membre, je veux aussi pouvoir trouver les visites qu'il gère, les voyages, ou encore n'importe : les évènements de manière générale.

Ainsi avec cette structure je peux faire

```
<?php
$membre->findEvenement(); // je récupère les visites ET les voyages qu'a organisés ce membre
$membre->findVisite(); // je récupère les visites qu'a organisées ce membre
$membre->findVoyage(); // je récupère les voyages qu'a organisés ce membre
?>
```

Le tout avec une seule table de base de données.

En redéfinissant aussi des résultats (`Zend_Db_Table_Row`) pour chacun, je peux m'assurer de l'intégrité des résultats manipulés.

Voici les codes :

```
<?php
abstract class AbstractTables extends Zend_Db_Table_Abstract
{
    public function getReference($tableName, $ruleKey = null)
    {
        $thisClass = get_class($this);
        if ($ruleKey != null) {
            if (!isset($this->_referenceMap[$ruleKey])) {
                require_once "Zend/Db/Table/Exception.php";
                throw new Zend_Db_Table_Exception("No reference rule \"\$ruleKey\" from table $thisClass to table $tableName");
            }
            if ($this->_referenceMap[$ruleKey][self::REF_TABLE_CLASS] != $tableName) {
                require_once "Zend/Db/Table/Exception.php";
            }
        }
    }
}
```

```
throw new Zend_Db_Table_Exception("Reference rule \"\$ruleKey\" does not reference table
\$tableClassName");
}
return $this->_referenceMap[$ruleKey];
}
foreach ($this->_referenceMap as $reference) {
$reflect = self::getreflect($tableClassName);
if ($reference[self::REF_TABLE_CLASS] == $tableClassName ||
in_array($reference[self::REF_TABLE_CLASS], $reflect)) {
return $reference;
}
}
require_once "Zend/Db/Table/Exception.php";
throw new Zend_Db_Table_Exception("No reference from table \$thisClass to table \$tableClassName");
}

private static function getreflect($tableClassName){
static $depTables;
$reflect = new ReflectionClass($tableClassName);
if ($reflect->getParentClass() instanceof ReflectionClass &&
strpos($reflect->getParentClass()->getName(), 'Zend_') === FALSE){
$depTables[] = $reflect->getParentClass()->getName();
return self::getreflect($reflect->getParentClass()->getName());
}else{
return $depTables;
}
}
}

<?php
class Voyage extends Evenement
{
protected $_rowClass = 'VoyageResultat';

protected function _fetch(Zend_Db_Table_Select $select)
{
$select->where('type = ?', '0');
return parent::_fetch($select);
}
}

<?php
class Visite extends Evenement
{
protected $_rowClass = 'VisiteResultat';

protected function _fetch(Zend_Db_Table_Select $select)
{
$select->where('type = ?', '1');
return parent::_fetch($select);
}
}

<?php
class Evenement extends AbstractTables
{
protected $_name = 'evenement';

protected $_primary = 'id';

protected $_dependentTables = array('Membre');

protected $_referenceMap = array(
'auteur' => array(
'columns' => array('idmembre'));
}
```

Les résultats maintenant :

```
<?php
class VoyageResultat extends Zend_Db_Table_Row_Abstract
{
    public function __construct(array $config = array())
    {
        parent::__construct($config);
        $this->_data['type'] = '0';
    }

    public function save()
    {
        if ($this->_data['type'] != '0'){
            throw new Zend_Db_Table_Row_Exception('Données corrompues, vous tentez de sauver un voyage
            qui n\'en est pas un');
        }
    }
}

<?php
class Evenement_VisiteResultat extends Zend_Db_Table_Row_Abstract
{
    public function __construct(array $config = array())
    {
        parent::__construct($config);
        $this->_data['type'] = '1';
    }

    public function save()
    {
        if ($this->_data['type'] != '1'){
            throw new Zend_Db_Table_Row_Exception('Données corrompues, vous tentez de sauver une visite
            qui n\'en est pas un');
        }
    }
}
```

Voilà qui devrait être satisfaisant, enfin on sent déjà que de la refactorisation s'impose, pour généraliser le comportement, à vos éditeurs ! ;-)

