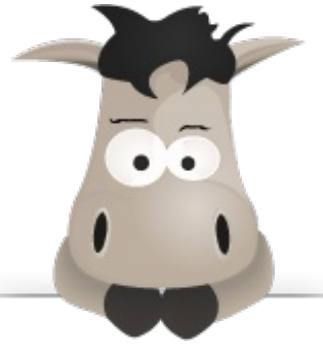

PDO : Interface d'accès aux BDD

Par Draeli

www.siteduzero.com

Dernière mise à jour le 7/08/2011

Sommaire

Sommaire 2
PDO : Interface d'accès aux BDD 3
Présentation 3
Activation - Php Windows 4
Connexion : création et gestion d'erreur 5
Méthodes : exec et query 6
Méthode : prepare 7
Sécuriser sans passer par des requêtes préparées 8
Partager 9

PDO : Interface d'accès aux BDD

Par  Draeli

Mise à jour : 07/08/2011
Difficulté : Facile

Ce tutoriel a pour but de vous présenter cette interface parue avec PHP 5 qui vise à utiliser des bases de données sans avoir à s'occuper du SGBD utilisé derrière.
Ainsi, il est tout à fait possible de faire un code qui marchera avec MySQL mais aussi avec Oracle, ODBC, etc.

On peut donc dire sans trop de risques que c'est l'avenir pour se connecter aux BDD avec PHP, un au revoir à mysql_query & co. ? **Oui**.

Vous trouverez ci-dessous une présentation de PDO, son activation et des exemples d'utilisation pour certaines des méthodes fournies par cette extension.
Sommaire du tutoriel :

- Présentation
- Activation - Php Windows
- Connexion : création et gestion d'erreur
- Méthodes : exec et query
- Méthode : prepare
- Sécuriser sans passer par des requêtes préparées

Présentation

PDO (***P**HP **D**ata **O**bjects*) vise à permettre la création d'un code comportant des accès aux BDD en faisant abstraction du moteur de SGBD utilisé.
C'est une interface qui, à ce titre, permet aux codeurs l'utilisant de travailler sans avoir à trop se poser de questions par la suite...
Enfin ceci dans l'idéal, car malheureusement cela dépend de certains détails que je vais développer.

Oui, mais alors quels sont-ils ces problèmes ?

Les problèmes, ou devrais-je dire LE problème (car finalement c'est du même problème que découlent tous les autres) concerne le respect des standards. Chaque SGBD a sa façon d'interpréter les standards et chaque SGBD dans sa façon de voir les choses essaye plus ou moins de s'y conformer, sans parler de tout ce qui peut être ajouté aux standards. Finalement, avec ces différents points, on se retrouve avec un tas de possibilités différentes de requêtes pour une même action.

Concrètement, ça veut dire quoi ?

Concrètement pour le développeur, ceci engendre le fait que selon le SGBD, des requêtes devront être réécrites spécifiquement.

Mais alors c'est nul, ton truc ?

Pas vraiment, car s'il est vrai que certaines requêtes devront être réécrites spécifiquement, beaucoup -de par le mécanisme qui sous-tend PDO- resteront valables ; en fait, l'intérêt de PDO n'est pas là mais que le code n'aura plus à être réécrit : c'est là l'avantage majeur de PDO.

Un autre avantage pour les familiers de la programmation orientée objet sous PHP 5, qui concerne les exceptions : PDO s'en sert.

Enfin, il faut signaler qu'à partir de PHP 6, c'est le système par défaut activé pour se connecter à une base de données, il est donc nécessaire de commencer à réécrire une partie de ses applications dans ce sens afin de préparer l'avenir (pas mal d'hébergeurs ont PHP 5 avec PDO et le *driver* pour MySQL installé donc n'hésitez pas à utiliser massivement PDO).

Avantages de PDO :

- *interface pour SGBD* : plus besoin de s'occuper de savoir quelle SGBD est derrière (en théorie) ;
- *orienté objet* : les objets PDO et PDOStatement peuvent être étendus, il est donc tout à fait possible de personnaliser et remodeler une partie du comportement initial ;
- *exception* : les objets de l'interface PDO utilisent des exceptions, il est donc tout à fait possible d'intégrer facilement un système de gestion des erreurs.

Activation - Php Windows

Avec PHP 5, cette extension n'est pas activée par défaut.

Afin de remédier au problème, ouvrez le '*php.ini*' que vous utilisez pour PHP 5 puis rendez-vous dans la partie où vous avez la liste des extensions. Faites une recherche sur '*; Windows Extensions*', vous devriez y arriver directement.
Bien : maintenant, vérifiez que vous avez la ligne suivante :

Code : Autre

```
;extension=php_pdo.dll
```

Si c'est le cas, décommentez la ligne afin d'avoir :

Code : Autre

```
extension=php_pdo.dll
```

Sinon, rajoutez la ligne à la main.

Vous venez donc d'activer PDO mais cela n'est pas suffisant : maintenant, il vous faut activer le *driver* correspondant au type de BDD que vous souhaitez utiliser.
Pour cela rien de plus simple : imaginons que vous vouliez activer MySQL, rajoutez pour cela la ligne suivante (ou activez-la si elle est présente) :

Code : Autre

```
extension=php_pdo_mysql.dll
```

Bien, enregistrez, relancez le serveur et vous voilà prêts à travailler avec l'extension PDO qui utilise MySQL.

Ouais, c'est cool mais moi, je n'utilise pas MySQL...

Ce n'est pas un problème : pour savoir si votre SGBD peut être utilisé et activé, allez dans le répertoire d'extensions de PHP '*Ext*' et regardez les fichiers portant l'extension *.dll* et commençant par : *php_pdo_*.*.
Si vous trouvez votre bonheur, il ne vous reste plus qu'à activer l'extension.
Ainsi, si vous ne voulez pas MySQL :

Code : Autre

```
extension=php_pdo_mysql.dll
```

mais plutôt Oracle :

Code : Autre

```
extension=php_pdo_odbc.dll
```

Vous pouvez trouver le tableau de correspondance des DLL avec le SGBD sur la documentation officielle.

Drivers : tout comme les *drivers* logiciels, ils permettent d'implémenter le système de la BDD avec ses spécifications, c'est ceci qui permettra que tout fonctionne.

Connexion : création et gestion d'erreur

Si vous êtes ici, c'est que normalement vous avez activé les extensions nécessaires au bon fonctionnement de PDO.

Tous les exemples qui seront traités à partir d'ici utiliseront MySQL comme SGBD (php_pdo_mysql.dll à activer si cela n'est pas fait) ; par la suite, rien ne vous empêche d'utiliser autre chose, la forme sera toujours la même (seuls les paramètres de connexion peuvent varier d'un SGBD à l'autre, tout comme les requêtes).

Création d'une connexion :

Code : PHP

```php
<?php
$PARAM_hote='localhost'; // le chemin vers le serveur
$PARAM_port='3306';
$PARAM_nom_bd='sdz'; // le nom de votre base de données
$PARAM_utilisateur='root'; // nom d'utilisateur pour se connecter
$PARAM_mot_passe=''; // mot de passe de l'utilisateur pour se connecter
$connexion = new
PDO('mysql:host='.$PARAM_hote.';port='.$PARAM_port.';dbname='.$PARAM_nom_bd,
$PARAM_utilisateur, $PARAM_mot_passe);
?>
```

Comme vous le voyez, ceci se fait très simplement, vous indiquez comme d'habitude les paramètres et ça marche.

Si vous tentez une connexion via un code similaire à celui-ci, et qu'une erreur se produit, vous verrez que l'erreur affichée expose le DSN. Il est nécessaire d'utiliser un gestionnaire d'erreur, même simple. Ceci n'est pas difficile, et présenté juste à la suite.

Oui mais moi j'ai une erreur, comment je fais ?

Vous pouvez avoir une erreur pour plusieurs raisons ; vous avez donné de mauvais paramètres, ou vous avez spécifié des paramètres qui ne correspondent pas au *driver* que vous souhaitez utiliser.
Afin de savoir d'où vient le problème, le mieux est encore d'afficher 'proprement' les erreurs qui peuvent apparaître.

Code : PHP

```php
<?php
try
{
        $connexion = new
PDO('mysql:host='.$PARAM_hote.';dbname='.$PARAM_nom_bd,
$PARAM_utilisateur, $PARAM_mot_passe);
}

catch(Exception $e)
{
        echo 'Erreur : '.$e->getMessage().'<br />';
        echo 'N° : '.$e->getCode();
```

```
}
?>
```

Vous aurez alors l'affichage de l'erreur, le catch est chargé d'intercepter une éventuelle erreur apparue dans le try : vous pouvez donc très bien mettre un arrêt du script comme ceci :

Code : PHP

```
<?php
try
{
        $connexion = new
PDO('mysql:host='.$PARAM_hote.';dbname='.$PARAM_nom_bd,
$PARAM_utilisateur, $PARAM_mot_passe);
}

catch(Exception $e)
{
        echo 'Une erreur est survenue !';
        die();
}
?>
```

Vous pouvez mettre un traitement de l'erreur comme vous voulez.

Méthodes : exec et query

PDO fait la distinction entre deux familles de requêtes : ceci est un peu déroutant au début, mais finalement assez simple quand on en a compris le principe et l'intérêt.
Comme vous le savez, il est possible sur une BDD de récupérer des informations, mais aussi d'effectuer des changements (qui peuvent prendre la forme d'ajout, suppression ou modification).

Si vous souhaitez récupérer une information (`SELECT`), vous devrez alors utiliser '`query`' ; si c'est pour un apporter des changements (`INSERT`, `UPDATE`, `DELETE`) à la BDD, vous devrez utiliser '`exec`'.


Hum, je ne comprends pas trop !

A ce stade, si vous n'avez pas compris, ce n'est pas grave, je vais détailler tout ça par des exemples. Ces exemples étant fictifs, vous devrez les adapter avec vos propres tables. 

Utilisation de la méthode : **`exec`**
Supposons que nous souhaitons modifier le mot de passe des membres :

Code : PHP

```
<?php
$connexion = new PDO("mysql:host=$PARAM_hote;dbname=$PARAM_nom_bd",
$PARAM_utilisateur, $PARAM_mot_passe); // connexion à la BDD

$connexion->exec("UPDATE membres SET mot_pass='toto'"); // on
modifie le mot de passe de tous les utilisateurs (oui, ça n'a aucun
intérêt mais c'est pour l'exemple)
?>
```

A ce stade, la modification a bien eu lieu mais ceci peut ne pas être suffisant.
Imaginons que nous souhaitons savoir le nombre de lignes qui ont été affectées par ce changement, nous aurons alors ceci :

Code : PHP

```
<?php
// création d'une connexion

$nombre_changement=$connexion->exec("UPDATE membres SET
```

```
mot_pass='toto'"); // on modifie le mot de passe de tous les
utilisateurs (oui, ça n'a aucun intérêt mais c'est pour l'exemple)
echo "La requête à modifié : $nombre_changement lignes.";
?>
```

Comme vous le voyez c'est très simple.

Utilisation de la méthode : **query**
Imaginons que nous souhaitons connaître tous les membres :

Code : PHP

```
<?php
$connexion = new PDO("mysql:host=$PARAM_hote;dbname=$PARAM_nom_bd",
$PARAM_utilisateur, $PARAM_mot_passe); // connexion à la BDD

$resultats=$connexion->query("SELECT membre FROM membres ORDER BY
membre ASC"); // on va chercher tous les membres de la table qu'on
trie par ordre croissant
$resultats->setFetchMode(PDO::FETCH_OBJ); // on dit qu'on veut que
le résultat soit récupérable sous forme d'objet
while( $ligne = $resultats->fetch() ) // on récupère la liste des
membres
{
        echo 'Utilisateur : '.$ligne->membre.'<br />'; // on affiche
les membres
}
$resultats->closeCursor(); // on ferme le curseur des résultats
?>
```

Comme vous le voyez, ça ressemble sensiblement à ce qui se fait habituellement, avec cependant deux détails que je me dois de soulever.

Le premier point concerne le '`setFetchMode`' qui permet d'indiquer sous quel format on souhaite récupérer les résultats de la requête précédente. Il existe plusieurs méthodes qui sont les suivantes : `PDO::FETCH_ASSOC`, `PDO::FETCH_BOTH`, `PDO::FETCH_OBJ`, etc., le mieux étant encore de lire la documentation pour savoir lequel vous préférerez utiliser.
Personnellement étant un habitué du `mysql_fetch_object`, j'ai une préférence pour celui-ci (dans l'exemple) qui permet une lecture claire.
Toujours concernant le mode, sachez qu'il existe une méthode plus courte d'écriture pour spécifier le mode, en le spécifiant directement dans `fetch()`, ce qui donne :

Code : PHP

```
<?php
$ligne = $resultats->fetch(PDO::FETCH_OBJ)
?>
```

Je vous conseille vivement de toujours indiquer une méthode (sous cette forme), ce qui permet une relecture du code plus claire par vous, ou toute personne passant derrière vous.

Le second point de remarque concerne `closeCursor` : il permet de fermer le curseur associé à un jeu de résultats, il doit donc être fermé une fois que vous avez fini de récupérer les résultats, et également être fermé avant toute autre requête.
Il vous est donc fortement conseillé de le mettre, sinon vous risquez des erreurs.

A remarquer qu'en cas d'erreur de la requête, le résultat ne sera pas un objet mais un booléen valant `false` (cette erreur étant alors soit due à une mauvaise requête, soit à un curseur qui n'a pas été fermé).

Méthode : prepare

Ceux qui connaissent les requêtes préparées et / ou qui ont lu mon tuto là-dessus en verront tout de suite l'intérêt ; aux autres : n'hésitez pas à aller lire le tuto sur les requêtes préparées, et à revenir.

Cette méthode prépare une requête SQL à être exécutée en offrant la possibilité de mettre des marqueurs qui seront substitués lors de l'exécution.
Il existe deux types de marqueurs qui sont respectivement **?** et les marqueurs nominatifs. Ces marqueurs ne sont pas mélangeables : donc pour une même requête, il faut choisir l'une ou l'autre des options.

Avantages de cette méthode :

- optimisation des performances pour des requêtes appelées plusieurs fois ;
- protection des injections SQL (plus besoin de le faire manuellement) ;

Exemple de cette méthode sans marqueur :
Imaginons que nous voulons récupérer la liste des membres.

Code : PHP

```php
<?php
// ouverture d'une connexion ...

$requete_prepare_1=$connexion->prepare("SELECT identifiant FROM
membres"); // on prépare notre requête
$requete_prepare_1->execute();
while($lignes=$requete_prepare_1->fetch(PDO::FETCH_OBJ))
{
        echo $lignes->identifiant.'<br />';
}
?>
```

Comme on peut le voir, c'est très simple.

Exemple de cette méthode avec marqueur nominatif :
Imaginons maintenant que nous voulons un identifiant bien précis.

Code : PHP

```php
<?php
// ouverture d'une connexion ...

$requete_prepare_1=$connexion->prepare("SELECT identifiant FROM
membres WHERE ID_membre = :id"); // on prépare notre requête
$requete_prepare_1->execute(array( 'id' => 1 ));
$lignes=$requete_prepare_1->fetch(PDO::FETCH_OBJ);
echo $lignes->identifiant.'<br />';
?>
```

Il est possible d'avoir plus de marqueurs : dans ce cas, il faut les rajouter dans la requête et dans `execute`, le tableau contiendra autant de clés et de valeurs qu'il y a de marqueurs dans la requête préparée.

Exemple de cette méthode avec marqueur **?** :
Nous voulons toujours notre identifiant unique.

Code : PHP

```php
<?php
// ouverture d'une connexion ...

$requete_prepare_1=$connexion->prepare("SELECT identifiant FROM
membres WHERE ID_membre = ? "); // on prépare notre requête
$requete_prepare_1->execute(array( 1 ));
$lignes=$requete_prepare_1->fetch(PDO::FETCH_OBJ);
echo $lignes->identifiant.'<br />';
?>
```

Comme vous le voyez, c'est encore plus simple.

Sécuriser sans passer par des requêtes préparées

Comme nous l'avons vu précédemment, il est possible de préparer une requête, ce qui va la sécuriser de facto.
Cependant certains usages (soit par facilité, soit par certaines limitations imposées par des requêtes) nécessitent de passer par des requêtes non préparées, dans ce cas, on est en droit de vouloir que la requête soit sécurisée.

Pour faire cela, rien de plus simple, il existe la méthode *quote*. Cette méthode s'utilise directement avec la ressource de connexion.

Exemple :

Code : PHP

```php
<?php
$RessourceDeConnection = new PDO(...); // ouverture d'une connection
$RessourceDeConnection->query("SELECT id_membre FROM membres WHERE
nom = ".$RessourceDeConnection->quote($nom, PDO::PARAM_STR));
?>
```

Ici nous ouvrons une nouvelle connexion, ensuite nous faisons une requête de type SELECT avec la méthode query prévue à cet effet.
Remarquez que pour appeler le nom, nous utilisons directement la ressource de connexion à laquelle nous ajoutons la méthode *quote*, en premier argument on met la valeur qui devra être protégée et en deuxième argument (qui est optionnel), on met le type de valeur à protéger.

Valeurs possibles pour le deuxième argument :
- **PDO::PARAM_STR** : pour une chaîne de caractères ;
- **PDO::PARAM_INT** : pour le type 'integer' de SQL ;
- **PDO::PARAM_NULL** : pour le type NULL de SQL ;
- **PDO::PARAM_BOOL** : pour un booléen ;
- **PDO::PARAM_LOB** : pour le type 'objet large' de SQL.

Attention, lorsque vous utilisez cette méthode, n'englobez aucune valeur avec " (pour une chaîne de caractère), la méthode se chargera de le faire pour vous si c'est nécessaire.

Par défaut, si vous ne spécifiez pas de deuxième argument, la valeur PDO::PARAM_STR sera utilisée.

Comme vous le voyez cette méthode est facile à utiliser. Cependant en terme de performances, elle est un peu moins rapide.
Finalement, la conclusion de tout ceci est que l'extension PDO est vraiment très pratique, et que même si elle demande un peu d'apprentissage, elle permettra à terme de ne plus avoir à modifier un code mais seulement les requêtes dans le cas éventuel où il y ait besoin d'une adaptation.

Bref, PDO, mangez-en : c'est bon.

La documentation officielle est disponible ici : http://fr.php.net/pdo.

Remerciements à :

Ce tutoriel a été corrigé par les zCorrecteurs.