PDO ou PHP Data Objects

Définition:

PDO (**P**HP **D**ata **O**bjects), est une extension définissant l'interface pour accéder à une base de données en PHP.

PDO constitue une couche d'abstraction. Elle intervient entre le serveur d'application et le serveur de base de données. C'est une interface (une sorte de classe) et est donc purement orientée objet.

La couche d'abstraction permet de séparer le traitement de la base de données. Ainsi on peut migrer vers un autre système de base de données (Oracle, ODBC, etc.) sans pour autant changer le code déjà développé en PHP.

Pour récupérer les enregistrements d'une table de la base de données, la méthode classique en PHP consiste à parcourir cette table ligne par ligne en procédant à des aller-retour entre le serveur d'application et le serveur de base de données.

Ceci risque d'alourdir le traitement surtout si les deux serveurs sont installés chacun sur une machine différente.

PDO remédie à ce problème en permettant de récupérer en une seule reprise tous les enregistrements de la table sous forme d'une variable PHP de type tableau à deux dimensions ce qui réduit visiblement le temps de traitement.

Connexion : création et gestion d'erreur

1) Création d'une connexion

```
<?php
$PARAM_HOST='localhost'; // le chemin vers le serveur
$PARAM_PORT='3306'; // le chemin vers vers le port (3306 par défaut)
$PARAM_NAME='madb'; // le nom de votre base de données
$PARAM_USER='root'; // nom d'utilisateur pour se connecter
$PARAM_PWD=''; // mot de passe de l'utilisateur pour se connecter
$connexion = new PDO('mysql:host='.$PARAM_HOST.';port='.$PARAM_PORT.';dbname='.$PARAM_NAME, $PARAM_USER, $PARAM_PWD); ?>
```

Gestion des erreurs

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.

Nous allons utiliser try catch pour cela.

```
<?php
try
{
     $connexion = new PDO('mysql:host='.$PARAM_HOST.';dbname='.$PARAM_NAME,   $PARAM_USER, $PARAM_PWD);
}

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

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'.

Un simple UPDATE:

```
$connexion->exec("UPDATE tabletest SET ladate = CURRENT_TIMESTAMP
WHERE id=2;");
```

Un UPDATE où on récupère le nombre d'entrées modifiées qu'il a modifié.

```
$combien = $connexion->exec("UPDATE tabletest SET ladate =
CURRENT_TIMESTAMP WHERE nom='Bon';");
echo "On a modifier $combien résultat(s) dans la table";
```

Et pour des SELECT

```
// mode tableau
$resultats = $connexion->query("SELECT * FROM tabletest ORDER BY
ladate DESC;");
$resultats->setFetchMode(PDO::FETCH_ASSOC); // on dit qu'on veut
que le résultat soit récupérable sous forme de tableau
while($rep = $resultats->fetch()){
     echo $rep['lemessage']."<hr/>";
}
$resultats->closeCursor(); // on ferme le curseur des résultats,
inutile pour mysql mais nécessaire pour la portabilité du code
```

```
// mode objet
        $resultats = $connexion->query("SELECT * FROM tabletest
ORDER BY ladate DESC;");
        $resultats->setFetchMode(PDO::FETCH_OBJ); // on dit qu'on
veut que le résultat soit récupérable sous forme d'objet
        while($rep = $resultats->fetch()){
```

```
            echo $rep->lemessage."<br/><strong>".$rep->nom." ".
$rep->prenom."</strong><hr/>";
        }
        $resultats->closeCursor();
```

```
// mode raccourci
        $resultats = $connexion->query("SELECT * FROM tabletest
ORDER BY ladate DESC;");
        while($rep = $resultats->fetch(PDO::FETCH_OBJ)){ // on
indique le type de résultat dans le fetch
            echo $rep->lemessage."<br/><strong>".$rep->nom." ".
$rep->prenom."</strong> - ".$rep->ladate."<hr/><hr/>";
        }
        $resultats->closeCursor();
```

Méthode: prepare

La plupart des bases de données supportent le concept des requêtes préparées.

Vous pouvez les voir comme une sorte de modèle compilé pour le SQL que vous voulez exécuter, qui peut être personnalisé en utilisant des variables en guise de paramètres.

La méthode 'prepare' ... 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 :

1) Optimisation des performances pour des requêtes appelées plusieurs fois ;
2) protection des injections SQL (plus besoin de le faire manuellement, même si certains contrôles restent nécessaires pour une sécurité maximale) ;

Exemple 1

```
// préparation de la requête
$req = $connexion->prepare("SELECT prenom FROM tabletest");
// exécution de celle-ci
$req->execute();
```

```
while($row = $req->fetch(PDO::FETCH_OBJ)){
     echo $row->prenom." ";
}
```

La requête préparée ci-dessus utilise prepare() puis execute(), c'est le minimum pour une requête préparée

Exemple 2

```
// variables de test
 $an = 1985;
$lettres = "%comme%";

// préparation de la requête avec des emplacements (marqueurs) nommés
$req = $connexion->prepare("SELECT * FROM tabletest WHERE naissance < :annee AND lemessage LIKE :lettres ; ");

// attribution après vérification de variables aux emplacements avec bindParam()
$req->bindParam(':annee',$an,PDO::PARAM_INT); // doit être un entier
$req->bindParam(':lettres',$lettres,PDO::PARAM_STR);// doit être une chaîne de caractère

 // exécution de celle-ci
$req->execute();
while($row = $req->fetch(PDO::FETCH_OBJ)){
     echo "<h4>".$row->prenom." ".$row->nom." né en ".$row->naissance."</h4>";
     echo "<p>".$row->lemessage."</p><hr/>";
}
```

La requête ci-dessus utilise bindParam() pour attribuer des valeurs aux emplacements nommés (:emplacement). Les injections SQL ne sont pas possibles.

Voici une liste des paramètres les plus usités avec bindParam():

PDO::PARAM_BOOL (variable)

Représente le type de données booléen.

PDO::PARAM_NULL (variable)

Représente le type de données NULL SQL.

PDO::PARAM_INT (variable)

Représente le type de données INTEGER SQL.

PDO::PARAM_STR (variable)

Représente les types de données CHAR, VARCHAR ou les autres types de données sous forme de chaîne de caractères SQL.

!!! Les variables sont bien traitées pour éviter les injections SQL, mais doivent être vérifiées au préalable si vous souhaitez être plus strict. (par exemple un float sera transformé en int par PDO::PARAM_INT, si vous souhaitez éviter cela vous devez faire un ctype_digit() ou d'autres fonctions de vérification)

Exemple 3

```
// variables de test
$an = "1980";
$lettres = "%et%";

// préparation de la requête avec des emplacements marqués
$req = $connexion->prepare("SELECT * FROM tabletest WHERE
naissance < ? AND lemessage LIKE ? ; ");

// exécution avec un tableau de valeurs. Les variables sont à
mettre dans le même ordre que la requête
$req->execute(array($an,$lettres));

while($row = $req->fetch(PDO::FETCH_OBJ)){
     echo "<h4>".$row->prenom." ".$row->nom." né en ".$row-
>naissance."</h4>";
     echo "<p>".$row->lemessage."</p><hr/>";
}
```

La préparation de requêtes avec emplacement marqués utilise les ' ?' à la place des noms.

Le fait d'utiliser un tableau directement dans execute() ne permet pas de vérifier les types de données, mais empêche les injections SQL.

Les variables sont à mettre dans le même ordre que la requête.

Exemple 4

```
// variables de test
        $an = mt_rand(1900,2010);
        $lettres = "abceiop";
        $nom = str_shuffle($lettres);
        $prenom = str_shuffle($lettres);
        $mots = explode(" ","Lorem ipsum dolor sit amet
consectetur adipisicing elit sed do eiusmod tempor incididunt ut
labore et dolore magna aliqua Ut enim ad minim veniam quis nostrud
exercitation ullamco laboris nisi ut aliquip ex ea commodo
consequat");
        $has = mt_rand(6, 35);
        $nb_mots = count($mots)-1;

        $letexte="";
        for($i=0;$i<$has;$i++){
            $letexte.=$mots[mt_rand(0,$nb_mots)]." ";
        }
        echo $letexte;

        // préparation de la requête avec des emplacements marqués
        $req = $connexion->prepare("INSERT INTO tabletest VALUES
(NULL, ?, ?, ?, CURRENT_TIMESTAMP, ?) ; ");
        // utilisation de bindParam
        $req->bindParam(1,$nom,PDO::PARAM_STR);
        $req->bindParam(2,$prenom,PDO::PARAM_STR);
        $req->bindParam(3,$letexte,PDO::PARAM_STR);
        $req->bindParam(4,$an,PDO::PARAM_INT);
        $req2 = $connexion->query("SELECT * FROM tabletest; ");
        // exécution
        $req->execute();

        while($row = $req2->fetch(PDO::FETCH_OBJ)){
            echo "<h4>".$row->prenom." ".$row->nom." né en ".$row-
>naissance."</h4>";
            echo "<p>".$row->lemessage."</p><hr/>";
```

```
        }
```

Différence entre bindParam et bindValue :

bindParam :

Utilisez bindParam () pour lier les variables PHP aux marqueurs de paramètres: les variables liées passent leur valeur en entrée et reçoivent la valeur de sortie, le cas échéant, de leurs marqueurs de paramètres associés. Très pratique pour exécuter plusieurs requêtes avec des valeurs différentes sans devoir réécrire la requête préparée.

! Elle n'accepte qu'une variable ou constante.

```
$value = 'foo';

$s = $dbh->prepare('SELECT name FROM bar WHERE baz = :baz');

$s->bindParam(':baz', $value); // use bindParam to bind the
variable

$a = $s->execute(); // executed with WHERE baz = 'foo'
$value = 'foobarbaz';
$b = s->execute(); // executed with WHERE baz = 'foobarbaz'
```

bindValue :

Contrairement à PDOStatement :: bindValue (), la variable est liée comme une référence et ne sera évaluée qu'au moment où PDOStatement :: execute () est appelé.

! Elle n'accepte qu'une simple valeur, une variable ou constante.

```
$value = 'foo';

$s = $dbh->prepare('SELECT name FROM bar WHERE baz = :baz');

$s->bindValue(':baz', $value); // use bindValue to bind the
variable's value

$value = 'foobarbaz';
$s->execute(); // executed with WHERE baz = 'foo'
```

Pour obtenir une connexion persistante :

```
$connexion = @new PDO('mysql:host='.$PARAM_HOST.';port='.
$PARAM_PORT.';dbname='.$PARAM_NAME.';charset=utf8', $PARAM_USER,
$PARAM_PWD, array(
    PDO::ATTR_PERSISTENT => true
));
```

On obtient une connexion permanente en passant après le mot de passe un tableau contenant l'attribut persistant.

Beaucoup d'applications web utilisent des connexions persistantes aux serveurs de base de données. Les connexions persistantes ne sont pas fermées à la fin du script, mais sont mises en cache et réutilisées lorsqu'un autre script demande une connexion en utilisant les mêmes paramètres.

Le cache des connexions persistantes vous permet d'éviter d'établir une nouvelle connexion à chaque fois qu'un script doit accéder à une base de données, rendant l'application web plus rapide.

Les transactions

Maintenant que vous êtes connecté via PDO, vous devez comprendre comment PDO gère les transactions avant d'exécuter des requêtes.

Si vous n'avez jamais utilisé les transactions, elles offrent 4 fonctionnalités majeures : Atomicité, Consistance, Isolation et Durabilité (**ACID**).

En d'autres termes, n'importe quel travail mené à bien dans une transaction, même s'il est effectué par étapes, est garanti d'être appliqué à la base de données sans risque, et sans interférence pour les autres connexions, quand il est validé.

Le travail des transactions peut également être automatiquement annulé à votre demande (en supposant que vous n'avez encore rien validé), ce qui rend la gestion des erreurs bien plus simple dans vos scripts.

Les transactions sont typiquement implémentées pour appliquer toutes vos modifications en une seule fois ; ceci a le bel effet d'éprouver drastiquement l'efficacité de vos mises à jour. Dans d'autres termes, les transactions rendent vos scripts plus rapides et potentiellement plus robustes (vous devez les utiliser correctement pour avoir ces bénéfices).

```
// on essaye le code
try {
     // on active l'affichage des erreurs
     $connexion->setAttribute(PDO::ATTR_ERRMODE,
PDO::ERRMODE_EXCEPTION);
     // on commence une transaction
     $connexion->beginTransaction();


     // on insert 3 entrées dont 1 erronée
     $connexion->exec("INSERT INTO tabletest VALUES (NULL, 'aaa',
'bbb', 'du blabla', CURRENT_TIMESTAMP, 1988) ; ");
     $connexion->exec("INSERT INTO tabletest VALUES (NULL, 'ccc',
'ddd', 'du blabla2', CURRENT_TIMESTAMP, 1989) ; ");
     $connexion->exec("INSERT INTO tabletest222 VALUES (NULL,
'eee', 'fff', 'du blabla3',  CURRENT_TIMESTAMP, 1990) ; ");
     // si les 3 n'ont pas d'erreurs on insert les 3 lignes
     $connexion->commit();
     echo '<br/>Insertions ok' ;
// si erreur
} catch (Exception $e) {
     // on efface les requêtes
     $connexion->rollBack();
     // on affiche le message d'erreur
     echo "Erreur: " . $e->getMessage();
}
```

Pour en savoir plus sur la PDO :

Documentation officielle PDO : la classe pdo

http://php.net/manual/fr/class.pdo.php

Tutoriel complet sur la PDO

http://fmaz.developpez.com/tutoriels/php/comprendre-pdo/