PDO ou PHP Data Objects

Définition:

PDO signifie **P**HP **D**ata **O**bjects, et c'est une extension de **PHP** qui fournit une interface pour accéder à une base de données de manière homogène, quelle que soit la base de données utilisée.

PDO est une alternative à l'extension **MySQLi**, qui est une autre extension de PHP pour interagir avec des bases de données de type MySQL ou MariaDB. **MySQLi** permettait de l'utiliser en procédural **ET** en orienté objet. Nous ne le verrons pas dans ce cours pour ne pas perdre du temps, car **PDO** est de loin le plus utilisé.

PDO constitue une **couche d'abstraction**. Elle intervient entre le serveur d'application et le serveur de base de données.

Informations : https://www.php.net/manual/fr/book.pdo.php

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.

PDO est un objet (une **classe**) en PHP, nous l'utiliserons avec MySQL ou MariaDB dans nos exemples.

PDO étant une classe, nous devons l'instancier avec le mot clef 'new'

```
$connect = new PDO('mysql:host=localhost;dbname=test;port=3306', $user, $pass);
```

La connexion se ferme seule (sauf en cas de connexion permanente), mais parfois il vaut mieux, pour la portabilité du code, fermer la connexion (après les requêtes évidemment !).

```
$connect = null;
```

Création d'une connexion PDO

```
<?php


// On peut créer des constantes avec define() ou const


const DB_TYPE='mysql'; // type de DB (identique pour MySQL ou MariaDB : mysql)
const DB_HOST='localhost'; // le chemin vers le serveur (souvant localhost, une adresse IP ou une URL)
const DB_PORT='3306'; // le chemin vers vers le port (3306 ou 3307 par défaut)
const DB_NAME='madb'; // le nom de votre base de données
const DB_USER='root'; // nom d'utilisateur pour se connecter
const DB_PWD=''; // mot de passe de l'utilisateur pour se connecter
const DB_CHARSET ='utf8mb4'; // charset


// connexion (sans les espaces après les ;)
$connexion = new PDO(DB_TYPE.':host='.DB_HOST.';port='.DB_PORT.';dbname='.DB_NAME;charset='.DB_CHARSET , DB_USER, DB_PWD);


// requêtes SQL


// fermeture de connexion
$connexion = null ;
?>
```

https://www.php.net/manual/fr/pdo.connections.php

Pour obtenir une connexion persistante :

```
$connexion = new
PDO(DB_TYPE.':host='.DB_HOST.';port='.DB_PORT.';dbname='.DB_NAME;charset=
'.DB_CHARSET , DB_USER, DB_PWD, array(
    PDO::ATTR_PERSISTENT => true
));
// !!! LE $connexion->setAttribute ne fonctionne pas la plupart du temps
pour mettre une connexion en “permanente”, il faut donc le faire au
moment de la connexion.
```

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 ré-utilisées 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.

Attention un nombre important de bases de données, sur des serveurs mutualisés, ont un nombre maximum de connexion !

Il vaut mieux ne pas activer les connexions permanentes sur ces DB car le risque d’avoir une erreur de type ‘too many connections’ est élevé !

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(PARAM_DB.':host='.PARAM_HOST.';port='.PARAM_PORT.';dbname='.PARAM_DBNAME;charset='.PARAM_CHARSET, PARAM_USER, PARAM_PWD);
	// activation de l'affichage des erreurs POUR les requêtes SQL (pas pour les erreurs de connexion, est déjà activé depuis PHP 8)
	$connexion->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
}
catch(PDOException $e) // vous pouvez utiliser catch(Exception $e), qui gère également les exceptions PDO (c’est une bonne pratique)
{
	echo 'Erreur : '.$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 apporter des changements (**INSERT**, **UPDATE**, **DELETE**) à la BDD, vous devrez utiliser '**exec**'.

Le ‘'**exec**'’ sera aussi utilisé pour des manipulations de tables au niveau de la DB, comme les **CREATE**, **DROP**, etc…

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";
```

Il en va de même pour les **INSERT** et **DELETE**

Et pour des SELECT

Utilisation du query, puis de fetch ou fetchAll pour récupérer les résultats au format attendu :

PDO::query

https://www.php.net/manual/fr/pdo.query.php

PDO::fetch

https://www.php.net/manual/fr/pdostatement.fetch.php

PDO::fetchAll

https://www.php.net/manual/fr/pdostatement.fetchall.php

// 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 tableau avec le fetchAll

```
    $resultats = $connexion->query("SELECT * FROM tabletest ORDER BY ladate DESC;");
        $result = $resultats->fetchAll(PDO::FETCH_ASSOC); // on fait un tableau indexé
des résultats qui contient des tableaux associatifs
      $resultats->closeCursor(); // on peut fermer le curseur des résultats car ils
sont déjà récupérés
        foreach($result as $item){
```

```
            echo $item['lemessage']."<hr/>";
        }
```

// 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 mélangables : 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) ;

https://www.php.net/manual/fr/pdo.prepare.php

et

https://www.php.net/manual/fr/pdostatement.execute.php

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 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 ! Ceci exécute un bindValue et non un bindParam (voir plus bas dans les feuilles)
$req->execute(array($an,$lettres));
// ou
// $req->execute([$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(1900,2022);
        $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()

Avec bindParam(), contrairement au bindValue(), la variable est liée comme **référence** et ne sera évaluée qu'au moment de l'appel de l' execute().

Le **bindParam()** permet donc de **garder le requête en mémoire et permet de l'exécuter plusieurs fois sans devoir redéclarer le prepare()**

https://www.php.net/manual/fr/pdostatement.bindparam.php

Le **bindValue()** ne le permet pas, il faut donc redéclarer le prepare() et le bindValue() si on souhaite l'exécuter plusieures fois

https://www.php.net/manual/fr/pdostatement.bindvalue.php

Ainsi, par exemple:

bindParam()

```
$age = 20;
$s = $dbh->prepare('SELECT name FROM students WHERE age < :age');
// utilisez bindParam pour remplir une $variable
$s->bindParam(':age', $age, PDO::PARAM_INT);
$s->execute(); // exécute avec 'WHERE age < 20'
$a = $s->fetchAll(PDO::FETCH_ASSOC);
```

```
$age = 25;
$s->execute(); // exécute avec ‘WHERE age < 25’
$b = $s->fetchAll(PDO::FETCH_ASSOC);

$age = 30;
$s->execute(); // exécute avec ‘WHERE age < 30’
$c = $s->fetchAll(PDO::FETCH_ASSOC);
```

bindValue()

```
$age = 20;
$s = $dbh->prepare('SELECT name FROM students WHERE age < :age');
// utilisez bindValue pour remplir une valeur attribuée qu’une fois
$s->bindValue(':age', $age, PDO::PARAM_INT);
$s->execute(); // exécute avec ‘WHERE age < 20’
$a = $s->fetchAll(PDO::FETCH_ASSOC);

$age = 25;
$s->execute(); // exécute toujours avec ‘WHERE age < 20’
$b = $s->fetchAll(PDO::FETCH_ASSOC);

// redéclaration du prepare() et du bindValue()
$age = 30;
$s = $dbh->prepare('SELECT name FROM students WHERE age < :age');
$s->bindValue(':age', $age, PDO::PARAM_INT);
$s->execute(); // exécute avec ‘WHERE age < 30’
$c = $s->fetchAll(PDO::FETCH_ASSOC);
```

execute() avec un tableau de valeurs

Ceci est le mode raccourci utilisant bindValue() de manière implicite, la variable, valeur ou fonction n'est pas liée comme référence et elle sera évaluée immédiatement avant l’ execute().

En général c'est le plus utilisé, il n'y a pas de vérification de type, il faut donc les faire en amont. Il n'empêche que les injections SQL et garde la requête en cache.

```
// préparation de la requête avec des marqueurs nommés
    $prepare2 = $db->prepare("SELECT * FROM thearticle
    WHERE idthearticle
    BETWEEN :debut AND :fin
ORDER BY thearticletitle ASC");

    $prepare2->execute(array(
        'debut' => 5, // marqueur nommé comme clef, ! sans les ':'
        'fin' => 10,
    ));
    $result4 = $prepare2->fetchAll(PDO::FETCH_OBJ);
    var_dump($result4);


    // préparation de la requête avec des ?
    $prepare3 = $db->prepare("SELECT * FROM thearticle
    WHERE idthearticle
    BETWEEN ? AND ?
ORDER BY thearticletitle ASC");

    // on met les valeurs par ordre de lecture sans clef
    $prepare3->execute([
        10,
        mt_rand(12, 18),
    ]);
    $result5 = $prepare3->fetchAll(PDO::FETCH_OBJ);
    var_dump($result5);
```

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).

Attention le moteur de la base de donnée doit accepter les transactions:

*En MySQL c'est le cas de l'**InnoDB**, pas du MyISAM* !

```
// 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();
}
```

Méthodes les plus utiles de PDO

Venant de la classe PDO :

PDO::**beginTransaction** — Démarre une transaction
PDO::**commit** — Valide une transaction
PDO::**exec** — Exécute une requête SQL et retourne le nombre de lignes affectées
PDO::**lastInsertId** — Retourne l'identifiant de la dernière ligne insérée ou la valeur d'une séquence
PDO::**prepare** — Prépare une requête à l'exécution et retourne un objet
PDO::**query** — Prépare et Exécute une requête SQL sans marque substitutive
PDO::**rollBack** — Annule une transaction
PDO::**setAttribute** — Configure un attribut PDO

Venant de la classe PDOStatement :

PDOStatement::**bindParam** — Lie un paramètre à un nom de variable spécifique
PDOStatement::**bindValue** — Associe une valeur à un paramètre
PDOStatement::**closeCursor** — Ferme le curseur, permettant à la requête d'être de nouveau exécutée
PDOStatement::**execute** — Exécute une requête préparée
PDOStatement::**fetch** — Récupère la ligne suivante d'un jeu de résultats PDO
PDOStatement::**fetchAll** — Retourne un tableau contenant toutes les lignes du jeu d'enregistrements
PDOStatement::**rowCount** — Retourne le nombre de lignes affectées par le dernier appel à la fonction PDOStatement::execute()
PDOStatement::**setAttribute** — Définit un attribut de requête
PDOStatement::**setFetchMode** — Définit le mode de récupération par défaut pour cette requête

Venant de la classe PDOException (héritées de Exception) :

Exception::**getMessage** — Récupère le message de l'exception
Exception::**getCode** — Récupère le code de l'exception

Pour en savoir plus sur la PDO :

Documentation officielle PDO : la classe PDO complète :

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

Documentation officielle sur les transactions :

https://www.php.net/manual/fr/pdo.transactions.php

Documentation officielle sur les pilotes gérés par PHP :

https://www.php.net/manual/fr/pdo.drivers.php

Tutoriel complet sur la PDO de developpez.com

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