PHP – PDO : Php Data Object

PDO est une **classe PHP** déstinée à permettre à PHP de communiquer avec un serveur de données.
PDO est ce qu'on appel une **couche d'abstraction**, c'est à dire qu'il va permettre de communiquer avec la plupart des serveurs de base de données : MySQL, Oracle, Postgresql, etc... (En tout cas sur des requêtes simples).
PDO va permettre (c'est son intérêt majeur) de **sécuriser** les requêtes et de favoriser la réutilisation du code grâce aux **requêtes préparées**.
mysql_connect(), mysql_query(), mysql_result(), mysql_fetch_array() etc... toutes ces fonctions que vous utilisez sont spécifiques au SGBD : Mysql, vous allez devoir modifier votre code si vous souhaitez déployer votre application sur un autre serveur, ou alors mettre en place une structure de programmation du type :

```
switch($typeDb)
{
    case 'mysql':
        mysql_query($query);
        break;
    case 'sqlite':
         sqlite_query($query);
         break;
    case 'mssql':
         mssql_query($query);
         break;
    //etc...
}
```

CONNEXION AU SERVEUR

Voici la connection type à un serveur MySQL :

```
// Connection au serveur
$host = 'mysql:host = localhost ; dbname = salaries';
$login = 'toto';
$password = 'toto';
$pdo = new PDO( $host, $login, $password );
```

La variable **$host**, représente le point d'entrée pour accéder à notre base de données, il débute par le code du **moteur de base de données, d**ans notre cas : **mysql**

- Ensuite on trouve l'adresse du serveur : host = localhost
- Puis le nom de la base de données : dbname = salaries

PDO étant orienté objet, il lève des exceptions en cas de problème. Le bloc try / catch est très pratique pour intercepter ce type d'erreur (on appelle ce genre d'erreur des **Exception**). La première étape va être d'indiquer à notre connexion que nous voulons que des erreurs soit émises :

```
try{
  $host = 'mysql:host = localhost ; dbname = salaries';
  $login = 'toto';
  $password = 'toto';
  $pdo = new PDO( $host, $login, $password ) ;
  $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
}
catch (Exception $e){
    die ($e->getMessage()) ;
}
```

EFFECTUER DES REQUÊTES

Une fois connecté au serveur MySQL avec notre objet PDO, nous allons pouvoir commencer à envoyer des requêtes SQL au serveur.

PDO distingue 2 types de requêtes :

- les requêtes de sélection : SELECT ->query($requete)
- les requêtes de modification - insertion : UPDATE – INSERT – DELETE ->exec($requete)

La différence entre ces deux méthodes est que query() retournera un jeu de résultats sous la forme d'un objet PDOStatement, alors que exec() retournera uniquement le nombre de lignes affectées. En d'autres termes, vous utiliserez query() pour des requêtes de sélection (SELECT) et exec() pour des requêtes d'insertion (INSERT), de modification (UPDATE) ou de suppression (DELETE).
(ref : http://fmaz.developpez.com/tutoriels/php/comprendre-pdo/)

Requête SQL retournant une seule ligne	

PDO
`$query = 'SELECT * FROM salaries WHERE id=1;';`
`$ligne = $pdo->query($query)->fetch();`
	mysql
	`$query = 'SELECT * FROM salaries where id=1;';`
	`$result = mysqli_query($conn,$query);`
	`$ligne = mysqli_fetch_assoc($result);`

Requête SQL retournant plusieurs lignes	

`$query = 'SELECT * FROM salaries;';`
`$liste = $pdo->query($query)->fetchAll();`
	`$query = 'SELECT * FROM salaries;';`
	`$result = mysqli_query($conn,$query);`
	`$liste = array();`
	`while($ligne=mysqli_fetch_assoc($result)){`
	`  $liste[] = $ligne ;`
	`}`

Requête d'insertion, de modification ou de suppression	

`$query = 'DELETE FROM salaries WHERE id=2;';`
`$rowCount = $pdo->exec($query);`
	`$query = 'DELETE FROM salaries WHERE id=2;';`
	`mysqli_query(sconn,$query);`
	`$rowCount = mysqli_affected_rows();`

Remarque : les méthodes **fetch()** et **fetchAll()** retourne les résultats en double : une fois indexés par le nom des colonnes et une deuxième fois indexés par leur numéro. Elles sont équivalentes à la fonction **mysql_fetch_array($result)** . Il est préférable de ne retourner qu'un résultat, dans ce cas vous pouvez passer un paramètre à la méthode :
exemple : la requête suivante **SELECT COUNT(*) as nb FROM salaries**

fetch(PDO::FETCH_NUM) ----- **mysqli_fetch_row($result)** tableau classique
Un simple **print_r** ou **var_dump** du résultat vous permet de visualiser les valeurs de retour
Array ([0] => 19) $result[0]

fetch(PDO::FETCH_ASSOC) ----- mysqli_fetch_assoc($result) tableau associatif
Array ([nb] => 19) $result['nb']

Si vous souhaitez retourner des objets, vous devez indiquer **fetch(PDO::FETCH_OBJ)**
qui retourne un objet avec les noms de propriétés qui correspondent aux noms des colonnes retournés dans le jeu de résultats
stdClass Object ([nb] => 19)
Pour accéder aux propiriétes de cet objet : $result → nb

Pour voir les différents modes : http://php.net/manual/fr/pdostatement.fetch.php

Remarque : Pour éviter d'avoir à indiquer à la chaque fois la méthode de retour vous pouvez choisir une seule méthode et l'indiquer dans le fichier de connexion :

$pdo->setAttribute(PDO::ATTR_DEFAULT_FETCH_MODE, PDO::FETCH_OBJ);

LES REQUÊTES PRÉPARÉES

Le principe est de préparer un « moule » de requête et de placer des *place holders* aux endroits où l'on voudra insérer nos valeurs dynamiques.
Vous avez une requête avec des *place holders* (les points d'interrogation) :

$requete ='*DELETE FROM salaries WHERE idsalaries = ?';*

Le SGBD va préparer (interpréter, compiler et stocker temporairement son "plan d'exécution logique") la requête.

$prep = $pdo->prepare($requete) ;

Il faudra ensuite associer des valeurs aux *place holders*, qui agissent un peu comme des variables .
Par exemple si vous souhaitez associer la valeur de 10 au premier place holder

$prep->bindValue(1, 10, PDO::PARAM_INT);

(1) : pour le premier place holder rencontré
(10) : pour la valeur associée
(PDO::PARAM_INT) : Par défaut, si le 3e paramètre de la méthode bindValue() est ignoré, le type utilisé sera PDO::PARAM_STR. Ce qui signifie que les valeurs auront des guillemets de part et d'autre pour les délimiteurs. En général, ça ne pose pas problème, mais dans le cas de la clause LIMIT, c'est problématique .

Ensuite on exécute la requête préparée :

$prep->execute() ;

Nommage des place holders
Un autre avantage de PDO est qu'il permet l'utilisation de *place holders* nommés, c'est-à-dire d'attribuer un nom au *place holder*, plutôt que d'utiliser des points d'interrogation et un indicateur de position numérique. L'exemple précédent devient :

```
$requete ='DELETE FROM salaries WHERE idsalaries =:id';
$prep = $pdo->prepare($requete) ;
$prep->bindValue(':id', 10, PDO::PARAM_INT);
$prep->execute() ;
```

Avantages et inconvénients des requêtes préparées :

Le principe des requêtes préparées est, comme son nom l'indique, de préparer les requêtes pour ensuite les utiliser. Ainsi, au lieu de recalculer la requête à chaque fois qu'on lui envoi, la base de données va la calculer une seule fois. Les fois suivantes, la base de données n'aura qu'à les exécuter.

Le second principe des requêtes préparées c'est de ne plus traiter les données comme faisant partie de la requête mais vraiment comme des données. Lors de la préparation d'une requête la base de données crée en réalité des cases dans lesquelles elle va « binder » les valeurs qu'on lui donne. Ce qui limite les possibilités d'injection SQL c'est à dire l'envoi de données frauduleuses.

Par contre les requêtes préparées sont plus longues à écrire et le débogage d'une requête est légèrement plus complexe . **$prep->debugDumpParams();**

EXEMPLE D'EXÉCUTION :

connexionPDO.php

```
<?php

 $host='localhost';
 $bd='salaries';
 $login='root';
 $password='sio';

 try
{
 $pdo = new PDO('mysql:host='.$host.';dbname='.$bd, $login, $password);
 $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
 $pdo->setAttribute(PDO::ATTR_DEFAULT_FETCH_MODE, PDO::FETCH_OBJ);
}
 catch (Exception $e) //Le catch est chargé d'intercepter une éventuelle erreur
 {
  die ($e->getMessage());
 }

 global $pdo;
?>
```

fonctionPDO.php (extrait)

```
require_once('connexionPdo.php') ;

function getNbSalaries(){
 global $pdo;
 $query = "SELECT count(*) as nb FROM salaries ;";
 try {
  $result = $pdo->query($query)->fetch();
  return $result->nb ;
 }
 catch ( Exception $e ) {
  die ("erreur dans la requete ".$e->getMessage());
 }
}
```

```
function getAllSalaries(){
    global $pdo;
    $query = 'SELECT * FROM salaries ';

    try {
      $result = $pdo->query($query)->fetchAll(PDO::FETCH_OBJ);
      return $result;
    }
    catch ( Exception $e ) {
      die ("erreur dans la requete ".$e->getMessage());
    }
}
```

listeSalariesPDO.php

```
<?php
  require_once('fonctionsPDO.php');

 $listeSalaries = getAllSalaries();
 $nbSalaries = getNbSalaries() ;
<table>
   <?php foreach ($listeSalaries as $leSalarie ) :?>
    <tr>
          <td><?php echo $leSalarie->idsalaries; ?></td>
          <td><?php echo $leSalarie->nom; ?></td>
    </tr>
    <?php endforeach; ?>
</table>

<p>Nombre de salariés : <?php echo $nbSalaries ; ?> </p>
```

Annexe : http://php.net/manual/fr/language.variables.scope.php

La portée d'une variable dépend du contexte dans lequel la variable est définie. Pour la majorité des variables, la portée concerne la totalité d'un script PHP. Mais, lorsque vous définissez une fonction, la portée d'une variable définie dans cette fonction est locale à la fonction. De la même manière la fonction ne pourra pas accéder à la variable définie en dehors. Pour accéder aux variables définies en dehors des fonctions, vous pouvez en PHP comme dans de nombreux langages de programmation, utiliser le mot clé **global suivi du nom de la variable.**

Travail à faire :

1 – Téléchargerl'archive **cor-mvc**, la décompresser dans votre répertoire **public_html.** Analyser le code.

3 - Modifier cette application , en utilisant la techologie **PDO.**