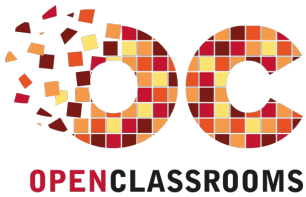
PDO : comprendre et corriger les erreurs les plus fréquentes

Par Guillaume Ch. (cGuille)



www.openclassrooms.com

Dernière mise à jour le 21/07/2011

Sommaire

PDO : comprendre et corriger les erreurs les plus fréquentes

Par 

Guillaume Ch. (cGuille)
Mise à jour : 21/07/2011
Difficulté : Facile

Bonjour à tous !

Vous venez d'apprendre le PHP grâce au cours de M@teo21 ? Ou bien vous connaissez déjà le PHP mais vous souhaitez abandonner l'API MySQL pour passer à PDO ? Vous vous embrouillez avec toutes ces nouveautés ?

Cet article a pour objectif de vous aider à comprendre votre code, et à corriger vous-même vos erreurs les plus simples.

Les prérequis sont les suivants :

- avoir suivi le cours de M@teo21 sur le PHP et terminé la partie qui concerne la base de données (ou niveau équivalent) ; ou
- avoir suivi le cours sur PDO écrit par Draeli.

PDO étant un ensemble de classes, quelques notions de POO pourront vous aider. Toutefois, je définirai les termes que j'utilise dans la première partie.

Cet article n'a pas pour vocation de vous apprendre à utiliser PDO, mais de vous apprendre à comprendre ce que vous faites avec. Vous devez **déjà** savoir utiliser les principales méthodes des classes de PDO.

Sommaire du tutoriel :

Un peu de vocabulaire

Introduction

Pour commencer, je vous propose un petit rappel sur le vocabulaire de base de la POO, **car PDO est composé de trois classes**. Notez que je ne vous ferai pas de cours sur la POO, d'autres s'en sont déjà chargés (cours de M@teo21, cours de vyk12). Je me contenterai de rappeler la définition de quelques mots-clés.

Si vous êtes à l'aise avec la POO, vous pouvez sauter cette partie. Si vous avez le moindre doute, je vous conseille de la lire tout de même. Vous n'en aurez que pour quelques minutes et ça ne pourra vous faire que du bien !

Quelques définitions

Classe

La classe est le moule qui permettra de créer des objets. Elle contient les attributs et les méthodes qui définissent l'objet. Par exemple, PDO est une classe. Il en va de même pour PDOStatement et PDOException.

Attribut

Un attribut, parfois nommé variable d'instance, est une variable propre à l'objet. Lorsque l'on conçoit une classe, il faut se poser la question « **De quoi est constitué mon objet ?** » pour trouver quels attributs lui donner. Par exemple, pour un objet "Client", on pourra répondre à cette question en disant qu'un client est constitué d'un numéro de client, d'un prénom, d'un nom, d'une adresse, d'un numéro de téléphone etc. On créera donc autant d'attributs.

Méthode

Une méthode est une fonction propre à l'objet. Lorsque l'on conçoit une classe, il faut se poser la question « **Que peut faire mon objet ?** » pour trouver quelles méthodes implémenter. Par exemple, un client peut changer de nom, d'adresse, de numéro de téléphone etc.

Instance

Une instance est tout simplement un exemplaire de notre objet. En PHP, le code suivant crée une instance de l'objet "Client" qui sera accessible via la variable `$client` :

Code : PHP

```php
<?php
$client = new Client();
```

Cette instruction utilise le moule (la classe Client) pour créer une instance (ou occurrence) de cette classe. Cette instance possède tous les attributs et toutes les méthodes propres à la classe Client.

Récapitulatif

Code : PHP

```php
<?php
class Classe // Ici on déclare la classe.
{
 // Ici on déclare les attributs de la classe :
 private $attribut1 = 0;
 private $attribut2 = '';

 // Ceci est une méthode de la classe Classe :
 public function methode()
 {
  // Instructions de la méthode.
 }
}

$classe = new Classe(); // On construit une instance de la classe
Classe.
/*
* Cette instance possède tous les attributs et toutes les méthodes
* de la classe Classe : attribut1, attribut2 et methode().
*/
```

Une convention d'écriture à connaître

Pour désigner la méthode d'une classe, les programmeurs ont l'habitude de faire précéder son nom de celui de sa classe puis d'un double deux-points "**::**" , afin de savoir au premier coup d'œil à quelle classe elle appartient, et de ne pas la confondre avec une méthode homonyme d'une autre classe.
Par exemple, pour désigner la méthode methode() de la classe Classe j'écrirai **Classe::methode()**, et pour la méthode query() de la

classe PDO j'écrirai **PDO::query()**.

Ce sera tout pour le vocabulaire ! Si vous n'avez pas compris ces quelques termes, je vous conseille de jeter un œil à l'un des cours que j'ai cités tout à l'heure :

- la POO en PHP selon M@teo21 ;
- la POO en PHP selon vyk12.

Les options de configuration

Introduction

- Ma requête ne fonctionne pas, et pourtant je n'ai aucun message d'erreur !
- J'ai une erreur "Call to a member function fetch() on a non-object". Je ne comprends pas, pourtant j'appelle la méthode comme dans le tuto !

Hé oui ! Il faut savoir que par défaut, PDO n'affiche pas les erreurs SQL. Or, en tant que développeur, vous devez **aimer** les messages d'erreurs ! Si, si ! Sans eux, on n'a aucune idée de l'endroit où l'on s'est trompé. Parfois même, ils nous disent ce qui ne va pas.
La preuve qu'ils sont utiles : PDO n'affiche pas les erreurs, et vous êtes perdu. 

Des erreurs mystérieuses

Bon, j'ai compris pourquoi quand je faisais une erreur SQL, je n'avais pas de message d'erreur. Mais pourquoi est-ce que parfois j'ai une erreur sur le fetch(), alors ?

Hé bien tout d'abord, il faut se souvenir que l'on travaille avec deux langages :

- PHP ;
- SQL.

Or, PDO ne vous montre pas les erreurs SQL. Les erreurs PHP ne sont pas influencées par PDO, elles sont influencées par la configuration de PHP.
Autrement dit, l'erreur que vous pouvez voir sur le fetch() est une erreur PHP causée indirectement **par une erreur SQL que vous ne voyez pas**.

Prenons un code de base de sélection de données avec PDO :

Code : PHP

```php
<?php
$stmt = $pdo->query('SELECT id, auteur, contenu, date FROM
messages');
while($message = $stmt->fetch())
{
 // Utilisation des données.
}
```

Admettons que, pour une raison ou pour une autre, ma requête échoue. Que va-t-il se passer ? Regardons la documentation de la méthode PDO::query().

Citation : php.net

> PDO::query() retourne un objet PDOStatement, ou FALSE si une erreur survient.

Un objet de la classe PDOStatement ? Hé oui, si vous avez la curiosité de jeter un œil à la documentation de cette classe, vous vous apercevrez que c'est cette classe qui possède la méthode fetch(). Si ma requête réussit, `$stmt` sera donc une instance de la classe PDOStatement. Mais si elle échoue, elle contiendra uniquement… **`FALSE`** ! 

Vous avez compris ? Si la requête échoue, il n'y a pas d'objet PDOStatement (pas d'objet du tout même, on obtient un booléen !). Sans objet, pas de méthode. Par conséquent, PHP nous prévient qu'on essaie d'appeler une méthode sur quelque chose qui n'est pas un objet, et que c'est donc impossible !

Et pourquoi aucun message d'erreur n'est affiché parfois ?

Tout simplement parce que dans le cas d'une requête de type **UPDATE** ou **DELETE**, on n'utilise pas (ou rarement) la valeur de retour de la méthode.
Vous l'aurez compris, il devient **urgent** de demander poliment à PDO de nous prévenir en cas de problème.

Afficher les erreurs SQL

Pour modifier les paramètres de PDO, nous avons deux solutions. Soit on donne les options directement au constructeur, soit on utilise la méthode PDO::setAttribute(). Je vais vous montrer ces deux méthodes, vous choisirez celle qui vous fait plaisir.

Il faut également savoir que l'on peut demander à PDO de nous avertir de deux façons :

- avec une erreur type "WARNING" ;
- avec une exception.

Vous connaissez déjà les erreurs WARNING. Un exemple ? Vous y avez droit si vous oubliez de donner un paramètre à une fonction : "**Warning: Wrong parameter count for intval()**". Si vous choisissez ce mode, vous aurez une erreur de ce type en cas d'erreur SQL.
En ce qui concerne les exceptions, vous en avez déjà vu au moins une ! Souvenez-vous, quand vous construisiez votre objet PDO :

Citation : Cours de PHP

Code : PHP

```php
<?php
try
{
 $bdd = new PDO('mysql:host=localhost;dbname=test', 'root', '');
}
catch (Exception $e)
{
        die('Erreur : ' . $e->getMessage());
}
```

Ici, `$e` est une exception. Les exceptions sont lancées par l'instruction **throw** et peuvent être interceptées par un bloc catch(). Je ne vous en dis pas plus, je ne compte pas vous faire un cours sur les exceptions. Si vous êtes curieux, vous chercherez de votre côté.

Demander un WARNING

Sans plus attendre, voici comment demander à PDO d'afficher des erreurs WARNING en cas de problème.
Vous pouvez passer un array en quatrième paramètre du constructeur. Cet array devra avoir pour clé le nom de l'attribut à modifier, et pour valeur… la valeur de l'attribut à modifier. En ce qui nous concerne, l'attribut à modifier est PDO::ATTR_ERRMODE et sa valeur PDO::ERRMODE_WARNING. On le comprend très bien : ATTR mis pour "attribut", et ERRMODE pour "error mode".
Sinon, vous pouvez utiliser la méthode PDO::setAttribute(). Elle prend deux paramètres : l'attribut à modifier et sa nouvelle valeur.

Voyez plutôt :

Code : PHP

```php
<?php
// Je donne les paramètres au constructeur :
$pdo = new PDO('mysql:host=localhost;dbname=DBNAME', 'root', '',
array(PDO::ATTR_ERRMODE => PDO::ERRMODE_WARNING));
```

Code : PHP

```php
<?php
// Je construis mon objet, puis je donne le paramètre à
PDO::setAttribute() :
$pdo = new PDO('mysql:host=localhost;dbname=DBNAME', 'root', '');
$pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_WARNING);
```

Demander une EXCEPTION

Allez, je suis sûr que vous pouvez trouver tout seul en cherchant la bonne valeur de l'attribut dans la liste des constantes prédéfinies de PDO.

Secret (cliquez pour afficher)

PDO::ERRMODE_EXCEPTION ? Oui, bravo ! Regardez ce que ça donne :

Code : PHP

```php
<?php
// Je donne les paramètres au constructeur :
$pdo = new PDO('mysql:host=localhost;dbname=DBNAME', 'root', '',
array(PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION ));
```

Code : PHP

```php
<?php
// Je construis mon objet, puis je donne le paramètre à
PDO::setAttribute() :
$pdo = new PDO('mysql:host=localhost;dbname=DBNAME', 'root', '');
$pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION );
```

Et voilà, vos erreurs ne sont plus silencieuses ! Si vous choisissez les exceptions, mettez vos requêtes dans un bloc try/catch pour pouvoir les attraper. Un exemple ? Le voici :

Code : PHP

```php
<?php
try
{
 $pdo = new PDO('mysql:host=localhost;dbname=DBNAME', 'root', '',
array(PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION ));
 $stmt = $pdo->query('SELECT id, auteur, contenu, DATE_FORMAT(date,
"%W %d %M %Y à %Hh%i") as date FROM messages');
 $messages = $stmt->fetchAll(PDO::FETCH_OBJ);
}
catch(Exception $e)
{
 exit('<b>Catched exception at line '. $e->getLine() .' :</b> '. $e-
>getMessage());
}
foreach($messages as $message)
{
 echo '<p>Le ', $message->date, ' par ', $message->auteur, ' : <br
/>', $message->contenu, '</p>';
}
```

D'autres attributs à modifier !

Si vous regardez plus en détail la liste des constantes de PDO que je vous ai montrée tout à l'heure, vous vous apercevrez que ERRMODE n'est pas le seul attribut modifiable. Par exemple, j'aime bien modifier le DEFAULT_FETCH_MODE, plutôt que de le donner à chaque fois à mes méthodes PDOStatement::fetch() ou PDOStatement::fetchAll().
Si vous êtes attentif, vous remarquerez que toutes les constantes qui commencent par PDO::ATTR_ sont des attributs que l'on peut modifier. Par exemple pour le fetch mode par défaut : PDO::ATTR_DEFAULT_FETCH_MODE fera l'affaire. Les valeurs possibles pour cet attribut sont les constantes qui commencent par PDO::FETCH_.

En bonus, voici comment je construis mon propre objet PDO. À vous d'adapter ce code selon vos goûts et vos besoins :

Code : PHP

```php
<?php
try
{
 $db_config = array();
 $db_config['SGBD'] = 'mysql';
 $db_config['HOST'] = 'localhost';
 $db_config['DB_NAME'] = 'tests';
 $db_config['USER'] = 'root';
 $db_config['PASSWORD'] = '';
 $db_config['OPTIONS'] = array(
  // Activation des exceptions PDO :
  PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
  // Change le fetch mode par défaut sur FETCH_ASSOC ( fetch()
retournera un tableau associatif ) :
  PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC
 );

 $pdo = new PDO($db_config['SGBD'] .':host='. $db_config['HOST']
.';dbname='. $db_config['DB_NAME'],
 $db_config['USER'],
 $db_config['PASSWORD'],
 $db_config['OPTIONS']);
 unset($db_config);
}
catch(Exception $e)
{
 trigger_error($e->getMessage(), E_USER_ERROR);
}
```

N'hésitez pas à parcourir la documentation des trois classes qui composent PDO pour apprendre à les utiliser.

Sécuriser ses requêtes

Introduction

On parle beaucoup des injections SQL. Cette technique consiste à écrire du code SQL dans un champ de texte pour détourner l'utilisation d'une requête *a priori* inoffensive pour une utilisation malveillante.
Sur les forums, je lis souvent « avec PDO, pas besoin de sécuriser ses requêtes, ça le fait tout seul ».
Ceci est **faux**.

Cette affirmation résulte d'une confusion entre PDO et les requêtes préparées. Comme je l'ai déjà dit, cet article n'a pas vocation à vous apprendre à utiliser ce genre de chose. D'autres s'en chargent bien mieux que moi dans le tutoriel de M@teo21 ou celui de Draeli.
En revanche, je vais vous expliquer dans quel cas il faut se protéger, et comment.

Une histoire de types

Il y a deux catégories de variables que l'on utilise dans une requête SQL : les nombres (int, float, double…) et les chaînes de caractères.
Or, une injection SQL est constituée de code SQL, et donc de texte. Cela signifie qu'une variable qui contient **uniquement** un nombre n'est pas dangereuse.

Seulement voilà, il existe une règle très importante en informatique : **never trust user input**, soit en français "ne faites jamais confiance aux données entrées par l'utilisateur". En effet, celui-ci peut se tromper ou pire : tenter d'insérer un code malveillant ! Avant d'utiliser une variable utilisateur (c'est-à-dire les variables dont la valeur peut être influencée par l'utilisateur) **censée** contenir un nombre, il faut vérifier qu'elle en contient effectivement un.

Pour cela plusieurs solutions s'offrent à vous. Soit vous faites une condition avec une fonction `is_type()`, comme par exemple is_int(). Soit vous convertissez (on peut aussi dire "transtyper") cette valeur en nombre.
Voici deux façons de procéder pour réaliser cette conversion (ou "cast" en anglais) :

Code : PHP

```php
<?php
// Première méthode : en précisant le type de destination.
$id = (int) $_GET['id'];

// Seconde méthode : en utilisant la fonction intval().
$id = intval($_GET['id']);
```

Si la variable ne contenait pas un nombre entier, alors sa nouvelle valeur sera 0. C'est très pratique lorsque l'on sait que les champs auto-incrémentés commencent à 1. Si après la conversion vous obtenez 0 : soit la variable ne contenait pas un entier, soit elle contenait zéro. Quoi qu'il arrive nous n'aurions pas pu trouver d'identifiant correspondant.

Lorsque vous voulez récupérer un identifiant auto_increment, vous pouvez donc procéder comme ceci :

Code : PHP

```php
<?php
$id = (int) $_GET['id'];
if($id <= 0)
{
 /* L'identifiant est invalide. Vous pouvez faire une redirection,
* un message d'erreur… bref ce que vous voulez !
*/
}
else
{
 // Faites votre requête en toute confiance : un nombre entier,
c'est inoffensif !
}
```

Voilà pour les nombres. Ce n'est pas très compliqué, il suffit de vérifier que la variable sur laquelle on travaille est effectivement un nombre.
En revanche, pour les chaînes de caractères… cela va dépendre de comment vous procédez !

Les requêtes préparées

Bon nombre de Zéros comprennent mal le fonctionnement des requêtes préparées. On leur a dit "avec une requête préparée, on ne peut pas subir d'injection SQL". C'est vrai, **mais seulement si on utilise correctement les requêtes préparées** !

Pour faire simple, toutes les données que vous envoyez à la requête via PDOStatement::bindValue(), PDOStatement::bindParam() ou PDOStatement::execute() ne risquent pas de provoquer une injection.
Si vous voulez comprendre pourquoi, je vous conseille cet article de Draeli qui explique le mécanisme des requêtes préparées.

En revanche, toutes les variables utilisateur qui sont présentes dans la chaîne de caractères donnée en paramètre à PDO::prepare() peuvent causer une injection SQL ! En théorie, aucune variable utilisateur ne devrait se trouver ici. Vous devriez utiliser une des trois fonctions que j'ai citées plus haut. Toutefois, si cela s'avère indispensable, vous pouvez utiliser la même méthode que celle que je vais vous donner pour PDO::query() et PDO::exec().

Un exemple de **faille** :

Code : PHP

```php
<?php
$stmt = $pdo->prepare("SELECT * FROM membres WHERE pseudo =
$pseudo");
$stmt->execute();
```

Voici une vraie requête préparée :

Code : PHP

```php
<?php
$stmt = $pdo->prepare("SELECT * FROM membres WHERE pseudo =
:pseudo");
$stmt->execute(array('pseudo' => $pseudo));
```

Et encore plus joli !

Code : PHP

```php
<?php
$stmt = $pdo->prepare("SELECT * FROM membres WHERE pseudo =
:pseudo");
$stmt->bindValue('pseudo', $pseudo, PDO::PARAM_STR);
$stmt->execute();
```

Les méthodes PDO::query() et PDO::exec()

Si pour une raison ou pour une autre, vous n'utilisez pas une requête préparée, alors toutes vos variables qui contiennent des chaînes de caractères doivent être protégées à l'aide de la méthode PDO::quote().
Comment ça "encore une méthode" ? Allez, ne faites pas la tête, cette méthode s'occupe de tout ! Elle entoure même votre chaîne de caractères avec des guillemets !

Code : PHP

```php
<?php
$stmt = $pdo->query('SELECT id FROM membres WHERE pseudo = '. $pdo-
>quote($_POST['pseudo']));
```

Code : PHP

```php
<?php
$pdo->exec('INSERT INTO livre_or(pseudo, message, date_post)
VALUES('. $pdo->quote($_POST['pseudo']) .', '. $pdo-
>quote($_POST['message']) .', NOW())');
```

PDOStatement::execute() et le type INT

Les symptômes

Une erreur très fréquente est basée sur le comportement de PDOStatement::execute().
Essayez donc ce genre de requête :

Code : PHP

```php
<?php
$offset = 5;
$stmt = $pdo->prepare('SELECT * FROM messages LIMIT 0, ?');
$stmt->execute(array($offset));
```

Le but de cette requête est de récupérer un certain nombre d'entrées dans la table messages. Ce nombre est défini par une variable (ici : $offset).
Si vous avez correctement configuré PDO, vous devez avoir ce message d'erreur :

Citation : Erreur

> SQLSTATE[42000]: Syntax error or access violation: 1064 You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near '5' at line 1

Et pourtant, cette requête fonctionne :

Code : PHP

```php
<?php
$stmt = $pdo->prepare('SELECT * FROM messages LIMIT 0, 5');
$stmt->execute();
```

Alors quoi ? Normalement, execute() remplace le marqueur **?** par la valeur de la variable, donc **5**. Le résultat est le même, non ?

Non. Ça vous en bouche un coin, hein ? 
En fait, si on prend la peine de lire la documentation de PDOStatement::execute(), on peut y trouver cette phrase qui décrit l'array passé en paramètre :

Citation : php.net

> Un tableau de valeurs avec autant d'éléments qu'il y a de paramètres à associer dans la requête SQL qui sera exécutée. **Toutes les valeurs sont traitées comme des constantes PDO::PARAM_STR**.

C'est la partie en gras qui est importante. Pour execute(), **tout est chaîne de caractères**. Par conséquent, il va entourer de *quotes* toutes les valeurs qu'on lui donne. La requête exécutée ressemble à cela : `SELECT * FROM messages LIMIT 0, '5'` , ce qui produit une erreur de syntaxe.

Pourtant j'ai fait cette requête et je n'ai pas de problème de type, alors que mon champ `id` est de type INT !

Code : SQL

```sql
SELECT * FROM `table` WHERE `id` = '1';
```

Comment ça se fait ?

MySQL est très permissif. Il va transtyper '1' en entier, et cela suffira (tandis que sur d'autres SGBD, cela pourrait ne pas fonctionner). En revanche, pour un LIMIT, il ne fera pas cette opération et signalera une erreur de syntaxe.

Alors, je ne dois plus donner de paramètre à execute() ? 

Si. Vous le pouvez tant que les paramètres que vous donnez sont des chaînes de caractères !
En pratique, j'ai pris l'habitude de ne plus donner aucun paramètre à execute(), car je trouve que l'autre solution rend le code plus lisible.

PDOStatement::bindValue() et PDOStatement::bindParam()

Je vous conseille d'aller voir ce que dit la documentation à propos de ces deux méthodes. Elles s'utilisent toutes les deux de la même façon. Il y a trois paramètres à donner :

- le nom du marqueur à remplacer ;
- la valeur à utiliser ;
- le type de valeur à envoyer. On utilise pour cela les constantes de PDO. En pratique, ce sera presque toujours PDO::PARAM_INT (pour un entier) ou PDO::PARAM_STR (pour une chaîne de caractères).

Si on reprend notre exemple avec le LIMIT, voici le résultat avec bindValue() :

Code : PHP

```php
<?php
$offset = 5;
$stmt = $pdo->prepare('SELECT * FROM messages LIMIT 0, :offset');
$stmt->bindValue('offset', $offset, PDO::PARAM_INT);
$stmt->execute();
```

Hourra, cela fonctionne cette fois-ci !

Y a-t-il une différence entre bindValue() et bindParam() ?

Oui ! Traduisons le nom de ces méthodes en français : "bind value" signifie "lier une valeur" tandis que "bind param" signifie "lier un paramètre". La différence est simple, mais subtile : bindValue() lie **la valeur d'une variable**, tandis que bindParam() lie **la variable elle-même**.
N'oubliez pas qu'une requête préparée peut être exécutée plusieurs fois en appelant plusieurs fois execute(). Ainsi, si l'on utilise bindParam(), il suffira de changer la valeur de la variable liée puis de rappeler execute() pour faire une nouvelle requête.

Les deux codes suivants produisent les mêmes requêtes :

Code : PHP

```php
<?php
$id_membre = 1;
$message = 'Bonjour :)';

$stmt = $pdo->prepare('INSERT INTO minichat(id_membre, message)
VALUES(:id_membre, :message)');
$stmt->bindValue('id_membre', $id_membre, PDO::PARAM_INT);
$stmt->bindValue('message', $message, PDO::PARAM_STR);
$stmt->execute();

$id_membre = 2;
$message = 'Salut !';

$stmt->bindValue('id_membre', $id_membre, PDO::PARAM_INT);
$stmt->bindValue('message', $message, PDO::PARAM_STR);
$stmt->execute();
```

Code : PHP

```php
<?php
$id_membre = 1;
$message = 'Bonjour :)';

$stmt = $pdo->prepare('INSERT INTO minichat(id_membre, message)
VALUES(:id_membre, :message)');
$stmt->bindParam('id_membre', $id_membre, PDO::PARAM_INT);
$stmt->bindParam('message', $message, PDO::PARAM_STR);
$stmt->execute();

$id_membre = 2;
$message = 'Salut !';
$stmt->execute();
```

Là, je n'ai modifié les valeurs qu'une seule fois. Mais admettons que j'aie besoin de les modifier 10 fois ou 100 fois ! Dans ce cas, cela reste plus pratique que de rappeler bindValue() à chaque fois.

Q.C.M.

Le premier QCM de ce cours vous est offert en libre accès.
Pour accéder aux suivants

Quelle classe ne fait pas partie des trois classes qui composent PDO ?

- ◯ PDOStatement.
- ◯ PDOException.
- ◯ PDOResult.
- ◯ PDO.

Que faire lorsqu'une de mes requêtes reste sans effet, alors que je n'ai aucun message d'erreur ?

- ◯ Demander poliment de l'aide sur les forums.
- ◯ Utiliser les constantes PDO::ATTR_* pour demander à PDO d'être plus précis.
- ◯ Laisser tomber PDO. C'est trop compliqué de toute façon.

Y a-t-il un problème avec ce code ?

Code : PHP

```php
<?php
// $pdo est une instance de la classe PDO.
$stmt = $pdo->query('SELECT * FROM membres WHERE pseudo = "'. $pdo->quote($_POST['pseudo']) .'"');
```

- ◯ Oui, il faut utiliser obligatoirement une requête préparée.
- ◯ Oui, la variable est mal insérée dans la requête.
- ◯ Non, la méthode PDO::quote() fait du bon boulot.

Quelles seront les valeurs insérées par le code suivant ?

Code : PHP

```php
<?php
$foo = 1;
$bar = 1;

$stmt = $pdo->prepare('INSERT INTO ma_table(foo, bar) VALUES(:foo, :bar)');
$stmt->bindParam('foo', $foo, PDO::PARAM_INT);
$stmt->bindValue('bar', $bar, PDO::PARAM_INT);
$stmt->execute();

$foo = 2;
$bar = 2;
$stmt->execute();
```

- ◯ foo: 1, bar: 1, puis foo: 1, bar: 1
- ◯ foo: 1, bar: 1, puis foo: 2, bar: 1
- ◯ foo: 1, bar: 1, puis foo: 2, bar: 2

Correction !

Statistiques de réponses au QCM

Ce sera tout pour les erreurs typiques de ceux qui débutent avec PDO ! Si vous en voyez d'autres ou que vous pensez que certains points nécessitent d'être clarifiés ou développés, n'hésitez pas à me contacter.

Si vous avez des questions, je répondrai avec plaisir à vos MP. Préférez un message sur les forums ou dans les commentaires, ainsi la réponse profitera à tout le monde et vous aurez d'autres points de vue que le mien.

J'insiste une nouvelle fois sur ce point : utilisez la documentation (doc PHP, doc PDO) ! Lisez la description des méthodes, de leurs paramètres et de leurs valeurs de retour. C'est comme cela que vous comprendrez au mieux comment les utiliser.

Ce tutoriel a été corrigé par les zCorrecteurs.