

PDO - Accès à une base de donnée MySQL en PHP

PDO est un API de PHP qui permet d'accéder à des base de donnée MySQL, PostGre, Oracle ou encore SQLite.

<https://www.php.net/manual/en/book.pdo.php>

Pré-requis

- le module `php_mysqli` installé et activé dans le `php.ini`
- le module `php_pdo_mysql` installé et activé dans le `php.ini`
- le `php.ini` se trouve au même endroit que `php.exe`

Connexion

```
$dbname = "bdd_name";  
$user = "root";  
$pass = "root";  
$bdd = new PDO('mysql:host=localhost;dbname=$dbname', $user, $pass);
```

Si rien de s'affiche sur la page, tout fonctionne bien.

Les 3 classes de PDO

Dans PDO il existe trois classes :

- `PDO` , la classe qui permet d'instancier une connexion à la BDD, `new PDO()` . Elle permet de lancer des requête au serveur et renvoi des objet de la classe `PDOStatement`.
- `PDOStatement` , la classe qui contient le résultat et contient des méthodes pour naviguer dans les lignes du tableau de résultats renvoyé par `mysql`. La fonction la plsu importante est `PDOStatement::fetchAll()` qui permet de récupérer tout les résultats dans un array PHP.
- `PDOException` , la classe qui représente les erreurs de PDO (status code 500).

Envoyer une requête et récupérer un array

La fonction `PDOStatement::fetchAll()` permet de récupérer un array.

```
$name = "Jérôme";  
$stmt = $bdd->query("SELECT * FROM Product");  
$products = $stmt->fetchAll();
```

Envoyer une requête et récupérer un élément

La fonction `PDOStatement::fetch()` permet de récupérer le premier élément de l'array. A chaque appel de la fonction on récupère l'élément suivant. Très pratique dans une boucle `for` lorsque l'on ne veut pas lire tous les éléments.

```
$name = "Jérôme";  
$stmt = $bdd->query("SELECT * FROM Product WHERE name=$name");  
$user = $stmt->fetch();
```

La gestion d'erreur avec try..catch

En PHP si une erreur arrive on veut parfois la récupérer pour y réagir. Pour faire ceci on utilise `try{}catch{}`.

Syntaxe:

```
try{  
    // Code risqué  
}catch(Exception $error){  
    echo $error->getMessage();  
}
```

Avec PDO on veut toujours tester si la connexion à la PDO est réussie.

```
try{
    $dbname = "bdd_name";
    $user = "root";
    $pass = "root";
    $bdd = new PDO('mysql:host=localhost;dbname=$dbname', $user, $pass);
} catch(PDOException $error){
    echo $error->getMessage();
}
```

Les requêtes SQL préparées

Jusqu'à maintenant quand on écrivait une requête on utilisait la fonction query et si besoin on concaténait une variable pour faire un WHERE par exemple.

Seulement voilà si cette variable provient d'une autre source comme le client (variables POST ou GET) lors de la concaténation je risque une injection de code malicieux dans ma BDD.

Voilà pourquoi lorsque j'ai des paramètres dans ma requête j'utilise une méthode dite "préparée".

Etapas d'envoi d'une méthode préparée.

L'envoi d'une méthode préparée se fait en 2 étapes :

- `PDO::prepare()` qui va préparer la requête SQL et lui fournir les paramètres à utiliser pour les WHERE, etc ...
- `PDOStatement::execute` qui exécute la requête et fournit les paramètres.

Préparation de la requête

Syntaxe :

Il faut placer des points d'interrogations là où souhaitez mettre vos paramètres.

```
$requete = $dbh->prepare('REQUETE ? SUITE ? SUITE > ?');
$requete->execute([param1,param2,param3 ]);
```

Puis exécuter les requêtes en fournissant les paramètres dans un array , dans l'ordre l'apparition des points d'interrogation.

Exemple :

```
$request = $dbh->prepare('SELECT * FROM Produit WHERE price < ? AND stock > ?');  
$request->execute([99.99, 20]);  
// La requete finale est  
// SELECT * FROM Produit WHERE price < 99.99 AND stock > 20;
```

Note : Requête réutilisable

L'avantage des requêtes préparées c'est que l'objet request est réutilisable. La requête se lancera à chaque appel de `PDOStatement::execute` et en plus de ça chaque requête est paramétrable.

Les transactions SQL

Les transactions SQL permettent d'annuler les requêtes appliquées à la base de données. Les transactions sont incluses dans l'objet `pdo` et sont très pratiques couplées avec un `try catch`.

Les transactions sont des méthodes de l'objet `PDO` :

- `PDO::beginTransaction` : <https://www.php.net/manual/en/pdo.begintransaction.php>
- `PDO::commit` : <https://www.php.net/manual/en/pdo.commit.php>
- `PDO::rollBack` : <https://www.php.net/manual/en/pdo.rollback.php>
- `PDO::inTransaction` : <https://www.php.net/manual/en/pdo.intransaction.php>, cette méthode renvoie vrai si une transaction est en cours.

Exemple :

```
try {  
    // Je démarre ma transaction...  
    $dbh->beginTransaction();  
  
    $dbh->query("INSERT INTO employe (name, lastname) VALUES ('Jeff', 'Bezos')");  
    // Si tout c'est bien passé...je peux commit  
    $dbh->commit();  
  
} catch (Exception $e) {  
    // Un problème ! Vite j'annule avec rollback !  
    $dbh->rollBack();  
    echo "Erreur: " . $e->getMessage();  
}
```