PDO - Accès à une base de donnée MySQL en PHP

PDO est un API de PHP qui permet d'accèder 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` le php.ini
- le module php `pdo_mysql` dans le php.ini

Le php.ini se trouve au même endroit que php.exe

> Si vous utiliser une image lamp docker pas besoin tout est déjà installés.

Connexion

Après avoir crée une BDD dans phpmyadmin !

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

> Remplacez db_name par le nom de la BDD crée sur phpMyAdmin
> 'tiger' est le mot de passe du compte root du container Docker LAMP, le mot de passe peut changer en fonction de votre installation.

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.
- `PDOExeception` , la classe qui représente les erreurs de PDO (status code 500).

Envoyer une requête et récupérer un array

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

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

Envoyer une requête et récupérer un élement

La fonction `PDOStatement::fetch()` permet de récpuérer le première element de l'array. A chaque appel de la fonction on récupére l'élement suivant. Très pratique dans une boucle for lorsque l'on ne veux pas lire tout les élements.

```
$name = "Jérome";
$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(Exeception $error){
    echo $error->getMessage();
}
```

Avec PDO on veut toujouts tester si la connexion à la PDO est réussi.

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

Si la connexion à échouée on récupère le message d'erreur.

Les requêtes SQL préparées

Jusqu'à maintenant quand on écriait une requête on utilisait la fonction query et si besoin on concatènait 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 requête dite "préparée".

Les étapes d'envoi d'une requête préparée.

L'envoi d'une requête préparé 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::bindValue()` qui va attribuer à chaque à chaque paramètre une valeur.
- `PDOStatement::execute` qui execute la requête.

Préparation de la requête

Syntaxe :
Il faut placer `:` suivi du `nom` de votre paramètre dans la `string` de la requête SQL.

```php
$requete = $bdd->prepare('REQUETE :param1 SUITE :param2 SUITE > :param3');
```

Puis attribuer une valeur et un type à chaque paramètre, par défaut le type est `PDO::PARAM_STR` soit une `string` ce qui convient pour la plupart des types y compris les `string` et les `float` .

Voir la liste des types dans la doc : https://www.php.net/manual/en/pdo.constants.php

```
$requete->bindValue("param1",value);
$requete->bindValue("param2",value);
$requete->bindValue("param3",value,PDO::PARAM_INT); // Si la valeur est un int.
```

> Attention cependant, si vous avez un `int` il faudra preciser `PDO::PARAM_INT` , comme pour `LIMIT 50` par exemple.

Et enfin executer la requête.

```
$requete->execute();
```

Exemple complet:

```
$request = $bdd->prepare('SELECT * FROM Produit WHERE price < :price AND stock > :stock
$request->bindValue("price",99);
$request->bindValue("stock",20);
$request->bindValue("limit",20,PDO::PARAM_INT);
$request->execute();
// La requete finale est
// SELECT * FROM Produit WHERE price < 99 AND stock > 20;
```

> N'oubliez pas d'attacher vos paramètre à votre requêtes avec `PDOStatement::bindValue` avant d'executer la requête.

> **Note : Requête réutilisable**
> L'avantage des requêtes préparées, c'est la réutilisation de l'objet $request. La requête se relancera à chaque appel de `PDOStatement::execute` .

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 à `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 :

```php
try {
    // Je démarre ma transaction...
    $bdd->beginTransaction();

    $bdd->query("INSERT INTO Produit (name, price,image) VALUES ('Jeff', 38.7,'/public/d
    $bdd->query("INSERT IN Produit (name, price,image) VALUES ('Jeff', 38.7,'/public/de1
    // La deuxième requête à echoué !
    // J'ai écrit IN au lieu de INTO !
    // Le catch c'est donc activé.
    $bdd->commit();

} catch (Exception $e) {
    // Un problème ! Vite j'annule avec rollback !
    $bdd->rollBack();
    // Aucune produit n'a été ajouté, pas même la première requête.
    echo "Erreur: " . $e->getMessage();
}
```