Accès aux données avec PHP

1 – Connexion à une base de données

Nous allons utiliser dans ce cours une architecture permettant de séparer
- **l'accès aux données** (géré uniquement en **PHP**)
- **l'affichage de ces données** (géré avec **HTML** et **PHP**).

Remarque : Cette **séparation données/affichage** devra **toujours** être respectée !

A RETENIR

La connexion à la BD et les dialogues avec la BD (c'est-à-dire les requêtes SQL) seront faites dans un dossier appelé "**persistance**"

L'affichage des données sera fait dans un dossier appelé "**presentation**".

On parlera de
- ***la couche "Persistance"***
- ***la couche "Présentation".***

Création d'un nouveau projet :

➔ Créez un **nouveau projet** nommé : **votrenom-TP-BDD-PHP**

➔ Sous ce projet, créez un **dossier** nommé "**persistance**"

➔ Dans ce dossier "**persistance**", créez un fichier php nommé "**connexion.php**" qui va contenir **la classe nommée "Connexion".**

Copiez le code suivant dans ce fichier :
*Fichier **connexion.php***

Attention de bien respecter les **majuscules et minuscules** tout au long de ce cours pour mieux comprendre !

connexion.php pour le fichier
Connexion pour la classe

```
<?php
class Connexion

    private $cnx = null;
    private $dbhost;
    private $dbbase;
    private $dbuser;
    private $dbpwd;

    public static function getConnexion() {

        $dbhost = '127.0.0.1';
        $dbbase = 'personnel';
        $dbuser = 'usersio';
        $dbpwd = 'sio';

        try {
            $cnx = New PDO("mysql:host=$dbhost;dbname=$dbbase",$dbuser,
$dbpwd);
            $cnx->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
            $cnx->exec('SET CHARACTER SET utf8');
        } catch (PDOException $e) {
            $erreur = $e->getMessage();
        }
        return $cnx;
    }

    public static function deConnexion() {

        try {
            $cnx = null;
        } catch (PDOException $e) {
            $erreur = $e->getMessage();
        }
}
```

Explication connexion : Observez votre code

La classe `Connexion` permet de se connecter à la base de données grâce à la méthode (fonction) `getConnexion()`.

Cette opération est réalisée de manière orientée objet, en **instanciant** un objet de la classe PDO par le biais de son constructeur.

Le nouvel objet est affecté à la variable `$cnx` qui est retourné par la fonction.

```
$cnx = new PDO("mysql:host=$dbhost;dbname=$dbbase", $dbuser, $dbpwd);
```

Le constructeur utilisé ici comporte 3 paramètres :
- Le premier paramètre ("mysql:...") définit le nom de la source de données. Ce nom contient le nom et le type du SGBD (ici `MySQL` sur la machine **$dbhost** qui correspond à `localhost`), le nom de la base de données **($dbbase** ici `personnel`)
- Le deuxième paramètre **$dbuser** est le login utilisé pour se connecter à la BD. Ici `usersio`
- Le troisième paramètre **$dbpwd** est le mot de passe utilisé pour se connecter à la BD. Ici `sio`.

Vous pouvez remarquer que pour se connecter à une base de données avec PDO, vous devez passer son nom dans le constructeur de la classe PDO (ici personnel).
Cela implique donc qu'il faut que la base ait déjà été créée au préalable (avec phpMyAdmin par exemple) ou qu'on la crée dans le même script.

Explication du bloc "Try Catch" pour la gestion des erreurs :

Avec PDO il est véritablement indispensable que votre script gère et capture les exceptions (erreurs) qui peuvent survenir durant la connexion à la base de données.
En effet, si votre script ne capture pas ces exceptions, l'action par défaut va être de terminer le script et d'afficher une trace.
Cette trace contient tous les détails de connexion à la base de données (nom d'utilisateur, mot de passe, etc.).
Nous devons donc la capturer pour éviter que des utilisateurs malveillants tentent de la lire.

Pour faire cela, nous allons utiliser des blocs try et catch qui est un mécanisme de gestion des erreurs utilisant les **exceptions**.

Sans rentrer dans des détails inutiles, son fonctionnement est le suivant :
- Les instructions situées dans la partie **try** sont exécutées normalement.
- Si l'une des instructions du bloc **try** provoque une erreur, elle est interceptée (*catchée*) par le bloc **catch**, dont les instructions sont alors exécutées.
- Si aucune instruction du bloc **try** n'échoue, le contenu du bloc **catch** n'est pas exécuté.

Pour résumer les choses :

- `try` délimite un bloc de code dans lequel des erreurs peuvent se produire.
- `catch` délimite un bloc de code qui intercepte et gère les erreurs apparues dans le bloc **try**.

Ici, le bloc **try** contient toutes les instructions d'accès à la BD. En cas d'erreur pendant la connexion, le bloc **catch** construit un message d'erreur (variable `$erreur`).

Remarque plus technique :
Nous utilisons également la méthode **setAttribute()** en lui passant deux arguments **PDO::ATTR_ERRMODE** et **PDO::ERRMODE_EXCEPTION**.

La méthode setAttribute() sert à configurer un attribut PDO. Dans ce cas précis, nous lui demandons de configurer l'attribut PDO::ATTR_ERRMODE qui sert à créer un rapport d'erreur et nous précisons que l'on souhaite qu'il émette une exception avec PDO::ERRMODE_EXCEPTION.

Plus précisément, en utilisant PDO::ERRMODE_EXCEPTION on demande au PHP de lancer une exception issue de la classe PDOException (classes étendue de Exception) et d'en définir les propriétés afin de représenter le code d'erreur et les informations complémentaires.

Ensuite, nous n'avons plus qu'à capturer cette exception PDOException et à afficher le message d'erreur correspondant. C'est le rôle de notre bloc catch.

Fermer la connexion à la base de données

Une fois la connexion à la base de données ouverte, celle-ci reste active jusqu'à la fin de l'exécution de votre script.

Avec PDO, il faudra détruire l'objet représentant la connexion et effacer toutes ses références. Nous pouvons faire cela en assignant la valeur null à la variable gérant l'objet (**$cnx = null**)