TP PHP/MySQL n°4

I. Truck, Université Paris 8, 2019–2020

1 Connecter PHP à la base MySQL

a. On a vu dans le TP précédent comment créer une base de données, des tables, des tables liées avec l'outil **phpmyadmin**. Mais ceci nécessite un traitement "à la main". Comment automatiser ce travail ? ⇒ en connectant PHP à la base de données MySQL.

b. D'abord, il faut avoir un moyen de connexion entre PHP et MySQL, autrement dit, il faut une API (*Application Programming Interface*). Nous allons utiliser l'extension PDO (*PHP Data Objects*) qui est "universelle" car elle permet de se connecter à MySQL, mais aussi à PostgreSQL ou encore à Oracle. Elle est orientée objet et contient une couche d'abstraction très pratique pour la programmation. Elle supporte également les procédures stockées et les commandes préparées (qu'est-ce que c'est ?). Noter que les fonctions commençant par `mysql_` sont obsolètes et ont été remplacées par des fonctions commençant par `mysqli_` ('i' comme *improved*). PDO est simple et portable mais a tout de même des inconvénients : elle ne permet pas d'utiliser toutes les fonctionnalités avancées des dernières versions de MySQL (par exemple, pas possible de faire certaines requêtes multiples). Pour autant, nous nous en contenterons.

c. Créer un fichier `connexion.php` et se connecter à la base de données MySQL du TP 3 grâce à PDO. Il faut créer une instance de la classe de base de PDO. Le constructeur prend en entrée les paramètres qui spécifient la source de la base et, éventuellement, le nom d'utilisateur et le mot de passe. Exemple :

```
<?php
  $bd = new PDO('mysql:host=localhost;dbname=test', $user, $pass);
?>
```

Remplacer les paramètres par des variables initialisées auparavant.

d. Modifier les paramètres comme le mot de passe (`$pass`) en mettant volontairement une valeur erronnée. Que se passe-t-il ?

Le but est de **traiter** les erreurs en les **capturant** avec `try` et `catch`. Il faut donc entourer la création de l'instance de base de PDO par un `try` et ajouter ensuite un `catch (Exception $e)` dans lequel on appelle la fonction `die()`. `die()` est équivalente à `exit()` et peut afficher le contenu des erreurs en appelant les méthodes `getMessage()` ou `getCode()` de la classe `Exception`. Chercher la syntaxe exacte et ajouter un test pour traiter les erreurs de connexion, dans le fichier `connexion.php`.

e. Ajouter, après la phase de connexion, une requête (méthode `query()` de la classe PDO) qui affiche la totalité des données contenues dans la table `client`. Une fois que la requête est écrite, il faut afficher correctement son résultat à l'écran. On utilise la méthode `fetch()` qui renvoie la première ligne de réponse de la requête. L'objet renvoyé par `fetch()` est un tableau associatif contenant chaque nom de champ demandé et sa valeur (exemple le tableau `$tab['nom']` contiendra `Dupond`). Pour lire toutes les lignes résultats de la requête, il faut bien sûr mettre `fetch()` dans un `while`. Après avoir traité une requête, on **ferme les résultats** avec `closeCursor()`. On doit donc le faire **après chaque requête**. Par ailleurs, la connexion à la base de données n'a besoin d'être faite qu'**une seule fois**, en début de page PHP.

f. Aide avec explications sur fetch, fetchAll, etc.

```
<?php
// la table fruit contient 3 champs : id, nom, couleur. Avec les valeurs suivantes :
// id   nom     couleur
// 1    pomme   rouge
// 2    kiwi    vert
// 3    orange  orange
// 4    raisin  vert
```

```php
// on suppose que la connexion à la base s'est faite correctement via $dbh
$sth = $dbh->prepare("SELECT nom, couleur FROM fruit");
$sth->execute();

/* comment récupérer les résultats ?
=> fetch() : */
// Attention : soit on utilise fetch, soit while (fetch), soit fetchAll... mais on ne peut
// pas tout faire à la suite comme ici. Sinon, on n'obtient pas grand chose d'intéressant
// car les résultats sont extraits (comme "effacés") dès le 1er fetch(). $sth contient
// une ligne de moins à chaque fetch().

$result = $sth->fetch(); // retourne dans un tableau associatif la 1ere ligne résultat
// de la requete. Donc, ici :
// $result['nom'] : pomme
// $result['couleur'] : rouge

/* OU BIEN DANS UN WHILE : */
while ($row = $sth->fetch()) { //retourne une ligne résultat, tant qu'il y a des résultats
    //$row['nom'] vaut d'abord 'pomme' (1er tour de boucle), puis kiwi, puis orange, etc.
    //$row['couleur'] vaut d'abord 'rouge' (1er tour de boucle), puis vert, puis orange, etc.
}

/* OU BIEN
=> fetchAll() et foreach : */
$tabResult = $sth->fetchAll(PDO::FETCH_ASSOC); // retourne le tableau de toutes les lignes
// résultat
foreach ($tabResult as $row) {
    //$row['nom'] vaut d'abord 'pomme' (1er tour de boucle), puis kiwi, puis orange, etc.
    //$row['couleur'] vaut d'abord 'rouge' (1er tour de boucle), puis vert, puis orange, etc.
}
?>
```

2 Exercice : interroger une table *via* PHP

a. Importer sous l'interface phpMyAdmin la base `ufr_erites.sql` se trouvant à l'adresse `http://isis.truck.free.fr/Site/ens/fichiers/handi/ufr_erites.sql`

b. Faire un **schéma** de la base de données ainsi importée. Quelles sont les différentes tables en présence ? Quelles sont les clefs primaires et étrangères ?

c. Créer un nouveau fichier `connexion_ufr_erites.php` qui fait une requête sur la table `etudiant` et qui affiche sous la forme d'un tableau les informations suivantes (il faut donc revoir l'affichage de tableaux en HTML ⇒ balises `<table>`, `<th>`, `<td>`, `<tr>`...) :

idEtud	nom	prenom	classe
1	Turing	Alan	M1Handi
2	Lovelace	Ada	M1Handi
3	Baggage	Charles	M1Handi
4	Meyer	Bertrand	M2Handi
5	Wirth	Niklaus	M2CNA
6	TALON	Achille	M2Handi
7	Berner-Lee	Tim	M1CNA

d. On va afficher le contenu de la table `enseignant` sous forme de `<select>...<option value="...">...` Vous ferez une requête en remplissant cette fois les nœuds `<option>` d'un nœud `<select>` et en les affichant sous le tableau de la question précédente. Pensez à utiliser la fonction `trim()` avant d'utiliser `<select>`, pour "nettoyer" les données remontées de la table, et, en particulier, supprimer les espaces ou autres caractères invisibles.

e. Maintenant, **triez** les données de liste (de la question précédente) sur l'identifiant de la table `enseignant` (mot-clef `ASC` ou `DESC`). Limitez-vous ensuite aux 6 premières lignes de réponse (mot-clef `LIMIT`).

f. Supposons maintenant que la requête contient une erreur (par exemple, un nom de champ erroné). Ajoutez volontairement une erreur dans votre requête. Que se passe-t-il ?
On peut connaître l'erreur dans une requête en utilisant la méthode `errorInfo()` de PDO. Regardez son prototype dans le manuel. Ainsi, pour traiter les erreurs, il faut **prendre l'habitude** de faire suivre toutes ses requêtes par `die(print_r(...))`. Faites-le.

g. Affichez la liste des notes et des modules associés de l'étudiant `Meyer` sous forme de tableau :

nom	prenom	intitule	Note
Meyer	Bertrand	PHP	12
Meyer	Bertrand	LSF	14
Meyer	Bertrand	PHP	12
Meyer	Bertrand	PHP	8
Meyer	Bertrand	PHP	8

h. Utilisation de variables dans les requêtes : on souhaite maintenant pouvoir demander à l'internaute le contenu d'une variable. Par exemple, il faut récupérer le nom de l'étudiant (dans la requête précédente) *via* un formulaire en méthode POST, le champ `input` de type `text` étant nommé par exemple `nomEtudiant`. Ensuite, il faut concaténer la requête avec la variable `$_POST['nomEtudiant']`. Faites-le.

Est-ce sécurisé ? Pourquoi ?

i. On va maintenant faire des **requêtes préparées**. On remplace la méthode `query` par les deux méthodes `prepare()` et `execute()`. Le principe est de préparer la requête, sans la partie variable (qui est remplacée par des `'?'`). Ensuite, avec `execute()`, on précise les `'?'` et le contenu des variables est automatiquement protégé. Les variables sont récupérées par `execute()` dans un `array` simple (tableau numéroté). S'il y a plusieurs variables dans la requête, il suffit de les indiquer successivement à `execute()`, en respectant l'ordre indiqué dans `prepare()`. Regarder leur fonctionnement.

j. En utilisant les requêtes préparées, afficher la liste des étudiants ayant eu une note égale à au moins une certaine valeur passée en paramètre dans un formulaire et étant dans une certaine classe (M1Handi, M2Handi, etc.) passée en paramètre dans le même formulaire.

k. A la place des points d'interrogation dans la méthode `prepare()`, on peut mettre des marqueurs **nominatifs** commençant par le symbole ':'. Cette fois-ci, les variables sont récupérées par `execute()` dans un `array` associatif. Par exemple, on peut écrire :

```
<?php
$req = $connexion->prepare('SELECT nom FROM etudiant WHERE nom = :nom');
$req->execute(array('nom' => $_POST['nomEtudiant']));
?>
```

Faire le test en reprenant la question j. Pour vérifier qu'il n'y a pas d'erreur, n'oubliez pas d'utiliser `die()`.

l. On peut également organiser une requête préparée de la façon suivante : `prepare()`, `bindParam()` ou bien `bindValue()` puis `execute()`. Faites le test.