TD2 – La persistance des données en PHP BDD, PDO

Dans le TDI vous avez appris à créer des classes et à instancier des objets de ces classes. Mais, comme vous l'avez constaté, la durée de vie des objets ainsi créés ne dépassait pas la durée de l'exécution du programme.

Dans ce TD, nous allons apprendre à rendre les objets persistants, en les sauvegardant dans une base de données. Ainsi, il sera possible de retrouver les objets d'une visite à l'autre du site web.

I. Connexion à la base de données

I.1 Les bases de PhpMyAdmin

Exercice 1

1. Connectez vous à votre base de données MySQL, à l'aide de l'interface PhpMyAdmin http://webinfo.iutmontp.univ-montp2.fr/my Le login est votre login IUT et votre mot de passe initial votre numéro INE.

2. Changez votre mot de passe (Page d'accueil > Paramètres généraux > Modifier le mot de passe) et reconnectez-vous. Si vous n'arrivez pas à vous connecter après avoir changé le mot de passe, essayer avec un autre navigateur ou bien videz le cache du navigateur (`Ctrl+F5`).

   Attention : N'utilisez pas un de vos mots de passe usuels, car nous allons bientôt écrire ce mot de passe dans un fichier qui sera sans doute vu par le professeur ou votre voisin.
   Donc vous avez deux possibilités :

   - (**recommandé**) Créez un mot de passe aléatoire à l'aide de https://www.random.org/passwords/ par exemple. Écrivez dès maintenant ce mot de passe dans un fichier.
   - Ou choisissez quelque chose de simple et de pas secret.

3. Créez une table `voiture` (sans majuscule) possédant 4 champs :

   - `immatriculationBDD` de type `VARCHAR` et de longueur maximale 8, défini comme la clé primaire (Index : `Primary`)
   - `marqueBDD` de type `VARCHAR` est de longueur maximale 25.
   - `couleurBDD` de type `VARCHAR` est de longueur maximale 12.
   - `nbSiegesBDD` de type `INT`.

   Important : Pour faciliter la suite du TD, mettez à la création de toutes vos tables `InnoDB` comme moteur de stockage, et `utf8_general_ci` comme interclassement (c'est l'encodage des données, et donc des accents, caractères spéciaux...).

   Attention : Les noms des champs sont comme des noms de variables, ils ne doivent pas contenir d'accents. Par ailleurs, et contrairement à Oracle, MySQL est sensible à la casse (minuscules/majuscules).

4. Insérez des données en utilisant l'onglet `Insérer` de PhpMyAdmin.

5. Dans la suite du TD, pensez à systématiquement tester vos requêtes SQL dans PhpMyAdmin avant de les inclure dans vos pages PHP.

I.2 Fichier de configuration en PHP

Pour avoir un code portable, il est préférable de séparer les informations du serveur du reste du code PHP.

Exercice 2

1. Créez un fichier `Conf.php`. Ce fichier contiendra une classe `Conf` possédant un attribut statique `$databases` comme suit (changez bien sûr les `a_remplir`).

   Notes :

   - Où doit-on enregistrer une page Web ? (Souvenez-vous du TD précédent)
   - Qu'est-ce qu'un attribut ou une méthode **statique** ? (Cours de Programmation Orientée Objet de l'an dernier ; voir aussi les compléments)

```php
<?php
class Conf {

  static private array $databases = array(
    // Le nom d'hote est webinfo a l'IUT
    // ou localhost sur votre machine
    //
    // ou webinfo.iutmontp.univ-montp2.fr
    // pour accéder à webinfo depuis l'extérieur
    'hostname' => 'a_remplir',
    // A l'IUT, vous avez une BDD nommee comme votre login
    // Sur votre machine, vous devrez creer une BDD
    'database' => 'a_remplir',
    // A l'IUT, c'est votre login
    // Sur votre machine, vous avez surement un compte 'root'
    'login' => 'a_remplir',
    // A l'IUT, c'est votre mdp (INE par defaut)
    // Sur votre machine personelle, vous avez creez ce mdp a l'installation
    'password' => 'a_remplir'
  );

  static public function getLogin() : string {
    // L'attribut statique $databases s'obtient avec la syntaxe
static::$databases
    // au lieu de $this->databases pour un attribut non statique
    return static::$databases['login'];
  }

}
?>
```

2. Pour tester notre classe `Conf`, créons un fichier `testConf.php` que l'on ouvrira dans le navigateur.

   Souvenez-vous le TD dernier : Quelle est la bonne et la mauvaise URL pour ouvrir une page PHP ?

```php
<?php
  // On inclut les fichiers de classe PHP pour pouvoir se servir de la classe
Conf.
  // require_once évite que Conf.php soit inclus plusieurs fois,
  // et donc que la classe Conf soit déclaré plus d'une fois.
  require_once 'Conf.php';

  // On affiche le login de la base de donnees
  echo Conf::getLogin();
?>
```

3. Complétez `Conf.php` avec des méthodes statiques `getHostname()`, `getDatabase()` et `getPassword()`. Testez ces méthodes dans `testConf.php`.

   Remarque : Notez qu'en PHP, on appelle une méthode statique à partir du nom de la classe comme en Java, mais en utilisant `::` au lieu du `.` en Java. Souvenez-vous que les méthodes dynamiques (c'est-à-dire pas `static`) s'appellent avec `->` en PHP.

4. Enregistrez votre travail à l'aide de `git add` et `git commit`. Nous comptons sur vous pour penser à faire cet enregistrement régulièrement.

1.3 Initialiser un objet `PDO`

Pour se connecter à une base de données en PHP on utilise une classe fournie avec PHP qui s'appelle `PDO` (Php Data Object). Cette classe va nous fournir de nombreuses méthodes très utiles pour manipuler n'importe quelle base de donnée.

Exercice 3

1. Commençons par établir une connexion à la BDD. Créez un fichier `Model.php` déclarant une classe `Model`, qui possédera
   - un attribut `private $pdo`,
   - un constructeur sans argument qui ne fait rien pour l'instant (à générer avec PhpStorm),
   - un accesseur (getter) `getPdo()` à l'attribut `$pdo` (à générer avec PhpStorm).
2. Dans le constructeur, nous allons initialiser l'attribut `$pdo` en lui assignant un objet `PDO`. Procédons par étapes :
   i. Pour créer la connexion à notre base de donnée, il faut utiliser le constructeur de `PDO` de la façon suivante

      ```php
      new PDO("mysql:host=$hostname;dbname=$databaseName",$login,$password);
      ```

      Stockez ce nouvel objet `PDO` dans l'attribut `$pdo`.

   ii. Le code précédent a besoin que les variables `$hostname`, `$databaseName`, `$login` et `$password` contiennent les chaînes de caractères correspondant à l'hôte, au nom, au login et au mot de passe de notre BDD. Créez donc ces variables avant le `new PDO` en récupérant les informations à l'aide des fonctions de la classe `Conf`.

   iii. Comme notre classe `Model` dépend de `Conf.php`, ajoutez un `require_once 'Conf.php'` au début du fichier.

   iv. Testons dès à présent notre nouvelle classe. Créez le fichier `testModel.php` suivant. Vérifiez que l'exécution de `testModel.php` ne donne pas de messages d'erreur.

      ```php
      <?php
      require_once "Model.php";

      // On affiche un attribut de PDO pour vérifier  que la connexion est bien établie.
      // Cela renvoie par ex. "webinfo.iutmontp.univ-montp2.fr via TCP/IP"
      // mais surtout pas de message d'erreur
      // SQLSTATE[HY000] [1045] Access denied for user ... (mauvais mot de passe)
      // ou
      // SQLSTATE[HY000] [2002] php_network_getaddresses: getaddrinfo failed (mauvais hostname)
      $model = new Model();
      echo $model->getPdo()->getAttribute(PDO::ATTR_CONNECTION_STATUS);
      ?>
      ```

i. Patron de conception *Singleton*

Comme cela n'a pas de sens d'avoir plusieurs connexions à la BDD, nous allons utiliser le patron de conception *Singleton*. Il sert à assurer qu'il n'y ait qu'une et une seule instance possible de la classe `Model` dans l'application (et donc une seule connexion).

Voici le squelette d'un singleton :

```php
class Model {
    private static $instance = null;

    private $pdo;

    public static function getPdo() {
        return static::getInstance()->pdo;
    }

    private function __construct () {
        // Code du constructeur
    }

   // getInstance s'assure que le constructeur ne sera
   // appelé qu'une seule fois.
   // L'unique instance crée est stockée dans l'attribut $instance
    private static function getInstance() {
        // L'attribut statique $pdo s'obtient avec la syntaxe static::$pdo
        // au lieu de $this->pdo pour un attribut non statique
        if (is_null(static::$instance))
            // Appel du constructeur
            static::$instance = new Model();
        return static::$instance;
    }
}
```

Remarques :

- Quand un attribut est statique, il s'accède par une syntaxe `NomClasse::$nomVar` comme indiqué précédemment.
- Quand un attribut statique est appelé avec `NomClasse::$nomVar`, on peut récupérer le type `NomClasse` à l'aide du mot clé `static`.

Exercice 4

1. Mettez à jour votre classe `Model` pour qu'elle suive le design pattern *Singleton*.
2. Mettez à jour `testModel.php` et vérifiez que tout marche bien.
3. Pour que PhpStorm comprenne que `Model::getPdo()` renvoie un objet de la classe `PDO`, et qu'il puisse nous proposer l'autocomplétion des méthodes de cette classe, nous devons déclarer le type de retour.
   Si ce n'est pas déjà fait, **déclarez** que l'attribut `$pdo` et la valeur de retour de `Model::getPdo()` sont de type `PDO`.
   Vérifiez que l'autocomplétion de PhpStorm s'est améliorée dans `testModel.php`.
4. **Déclarez** que l'attribut `$instance` et la valeur de retour de `Model::getInstance()` sont de type `Model`.
   L'IDE indique un problème : L'attribut `$instance` est initialisé à `null`, qui n'est pas de type `Model` en PHP (contrairement à Java), mais de type `null`.
   Corrigez ce problème en indiquant le type `?Model` pour l'attribut `$instance`. En effet, `?Model` est un raccourci pour le type `Model|null`, qui veut dire `Model` ou `null`.

ii. Gestion des erreurs

Nous allons maintenant améliorer la gestion des erreurs de `PDO`.

Exercice 5

Pour avoir plus de messages d'erreur de `PDO` et qu'il gère mieux l'UTF-8, **mettez à jour** la connexion dans `Model` en remplaçant `$this->pdo = new PDO(...);` par

```
// Connexion à la base de données
// Le dernier argument sert à ce que toutes les chaines de caractères
// en entrée et sortie de MySql soit dans le codage UTF-8
$this->pdo = new PDO("mysql:host=$hostname;dbname=$databaseName", $login,
$password,
                    array(PDO::MYSQL_ATTR_INIT_COMMAND => "SET NAMES utf8"));

// On active le mode d'affichage des erreurs, et le lancement d'exception en cas
d'erreur
$this->pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
```

2. Opérations sur la base de données

Voyons maintenant comment les objets `PDO` servent à effectuer des requêtes SQL. Nous allons nous servir de deux méthodes fournies par `PDO` :

1. La méthode `query($SQL_request)` de la classe `PDO`
   - prend en entrée une requête SQL (chaîne de caractères)
   - et renvoie la réponse de la requête dans une représentation interne pas immédiatement lisible (un objet `PDOStatement`).
2. La méthode `fetch()` de la classe `PDOStatement` s'appelle sur les réponses de requêtes et renvoie la réponse de la requête dans un format lisible par PHP. Plus précisément, elle renvoie une entrée SQL formatée comme un tableau. Ce tableau est indexé par les noms des champs de la table de données, et aussi par les numéros des champs. Les valeurs du tableau sont celles de l'entrée SQL.

2.1 Faire une requête SQL sans paramètres

Commençons par la requête SQL la plus simple, celle qui lit tous les éléments d'une table (`voiture` dans notre exemple) :

```
SELECT * FROM voiture
```

Exercice 6

1. Créez un fichier `lireVoiture.php`
2. Incluez le fichier contenant la classe `Model` pour pouvoir se connecter à la BDD.
3. Appelez la fonction `query` de l'objet `PDO Model::getPdo()` en lui donnant la requête SQL. Stockez sa réponse dans une variable `$pdoStatement`.
4. Comme expliqué précédemment, pour lire les réponses à des requêtes SQL, vous pouvez utiliser

   ```
   $voitureFormatTableau = $pdoStatement->fetch()
   ```

   qui, dans notre exemple, renvoie un tableau avec 8 cases :

   - `immatriculationBDD`, `couleurBDD`, `marqueBDD` et `nbSiegesBDD` (les champs de la BDD).
   - `0`, `1`, `2` et `3` qui correspondent aux champs de la BDD dans l'ordre. Ces cases sont donc un peu redondantes.

   Utilisez l'un des affichages de débogage (*e.g.* `var_dump`) pour afficher ce tableau.

5. Créez une `$voiture` de classe `Voiture` à l'aide de `$voitureFormatTableau` en appelant le constructeur. Affichez la voiture en utilisant la méthode adéquate de `Voiture`.

6. On souhaite désormais afficher toutes les voitures dans la BDD. On pourrait faire une boucle `while` sur `fetch` tant qu'on n'a pas parcouru toutes les entrées de la BDD.

   Heureusement, il existe une syntaxe simplifiée qui fait exactement cela :

   ```php
   foreach($pdoStatement as $voitureFormatTableau){
       // ...
   }
   ```

   Note :

   - chaque tour de boucle agit comme si on avait fait un fetch

     ```php
     $voitureFormatTableau = $pdoStatement->fetch()
     ```

   - on peut faire foreach car PDOStatement implémente l'interface Traversable. C'est similaire à Java qui permettait la boucle `for(xxx : yyy)` pour les objets implémentant l'interface `Iterable`.

   Utilisez la boucle `foreach` dans `lireVoiture.php` pour afficher toutes les voitures.

7. Avez-vous pensé à enregistrer régulièrement votre travail sous Git ?

Exercice 7

Nous allons maintenant isoler le code qui retourne toutes les voitures et en faire une méthode de `Voiture`.

1. Isolez le code qui construit l'objet `Voiture` à partir du tableau donné par `fetch` (*e.g.* `$voitureFormatTableau`) dans une méthode

   ```php
   public static function construire(array $voitureFormatTableau) : Voiture
   {
   // ...
   }
   ```

2. Créez une fonction statique `getVoitures()` dans la classe `Voiture` qui ne prend pas d'arguments et renvoie le tableau d'objets de la classe `Voiture` correspondant à la BDD.

   Rappel : On peut rajouter facilement un élément "à la fin" d'un tableau avec

   ```php
   $tableau[] = "Nouvelle valeur";
   ```

3. Mettez à jour `lireVoiture.php` pour appeler directement `getVoitures()`.

4. Maintenant que vous avez bien compris où les noms de colonnes (`immatriculationBDD`, `couleurBDD`, ...) de la table `voiture` interviennent dans le tableau `$voitureFormatTableau`, nous allons leur redonner des noms plus classiques :
   i. Changer les noms des colonnes pour `immatriculation`, `couleur`, `marque` et `nbSieges`. Pour ceci, dans PhpMyAdmin, cliquez sur l'onglet "Structure" de la table `voiture`, puis "Modifier" sur chaque colonne.
   ii. Modifiez le code PHP à l'endroit où interviennent ces noms de colonnes.

2.2 Format de retour de `fetch()`

Rappelons que la méthode `fetch($fetchStyle)` s'appelle sur les réponses de requêtes et renvoie la réponse de la requête dans un format lisible par PHP. Le choix du format se fait avec la variable `$fetchStyle`. Les formats les plus communs sont :

- `PDO::FETCH_ASSOC` : Chaque entrée SQL est un tableau indexé par les noms des champs de la table de la BDD ;
- `PDO::FETCH_NUM` : Chaque entrée SQL est un tableau indexé par le numéro de la colonne commençant à 0 ;
- `PDO::FETCH_BOTH` (valeur par défaut si on ne donne pas d'argument `$fetchStyle`) : combinaison de `PDO::FETCH_ASSOC` et `PDO::FETCH_NUM`. Ce format retourne un tableau indexé par les noms de colonnes et aussi par les numéros de colonnes, commençant à l'index 0, comme retournés dans le jeu de résultats
- `PDO::FETCH_OBJ` : Chaque entrée SQL est un objet dont les noms d'attributs sont les noms des champs de la table de la BDD ;
- `PDO::FETCH_CLASS` : De même que `PDO::FETCH_OBJ`, chaque entrée SQL est un objet dont les noms d'attributs sont les noms des champs de la table de la BDD. Cependant, on peut dans ce cas spécifier le nom de la classe des objets. Pour ce faire, il faut avoir au préalable déclaré le nom de la classe avec la commande suivante :

  ```php
  $pdoStatement->setFetchMode( PDO::FETCH_CLASS, 'class_name');
  ```

  Note : Ce format qui semble très pratique a malheureusement un comportement problématique :

  - il crée d'abord une instance de la classe demandée (sans passer par le constructeur !) ;
  - il écrit les attributs correspondants aux champs de la BDD (même s'ils sont privés ou n'existent pas !) ;
  - **puis** il appelle le constructeur *sans arguments*.

Dans les TDs, nous vous recommandons d'utiliser au choix :

- le format par défaut `PDO::FETCH_BOTH` en appelant `fetch()` sans arguments,
- le format `PDO::FETCH_ASSOC` pour ne pas avoir de cases redondantes (*e.g* `immatriculationBDD` et `0`).
  Dans ce cas, appelez `$pdoStatement->setFetchMode(PDO::FETCH_ASSOC)` avant d'appeler `fetch()`.

3. Site de covoiturage

Appliquez ce que l'on a fait pendant ce TD aux classes `Trajet` et `Utilisateur` du TP précédent (exercice sur le covoiturage) :

Exercice 8

1. Dans votre PhpMyAdmin, créez une table `utilisateur` avec les champs suivants :
   - `login` : VARCHAR 32, clé primaire
   - `nom` : VARCHAR 32
   - `prenom` : VARCHAR 32

   Important : Avez-vous bien pensé à `InnoDB` et `utf8_general_ci` comme précédemment ?
2. Insérez quelques utilisateurs.
3. Créez une table `trajet` avec les champs suivants :
   - `id` : INT, clé primaire, qui s'auto-incrémente (voir en dessous)
   - `depart` : VARCHAR 32
   - `arrivee` : VARCHAR 32
   - `date` : DATE
   - `nbPlaces` : INT

- `prix` : INT
- `conducteurLogin` : VARCHAR 32

Note : On souhaite que le champ primaire `id` s'incrémente à chaque nouvelle insertion dans la table. Pour ce faire, sélectionnez cocher la case `A_I` (auto-increment) pour le champ `id`.

4. Insérez quelques trajets en prenant soin de ne pas remplir la case `id` (pour que l'auto-incrément marche) et en mettant dans `conducteurLogin` des login d'utilisateurs valides (pour éviter des problèmes par la suite).
5. Créez les fonctions statiques `getTrajets()` et `getUtilisateurs()` qui listent tous les trajets / utilisateurs.

4. (Optionnel) Pour utiliser une base de données locale

Actuellement, votre code PHP se connecte au serveur MySql de l'IUT. Cela marche très bien tant que vous avez une connexion internet.

Si vous souhaitez utiliser une base de données `MySQL` en local, voici quelques instructions :

- Dans une installation XAMPP sous Linux, il y a un compte `root` sans mot de passe pour la base de données et PhpMyAdmin. Voici comment configurer le tout :
  - PhpMyAdmin est accessible à l'adresse http://localhost/phpmyadmin. Normalement, il ne nécessite pas de connexion par login / mdp. En effet, vous êtes directement connecté en tant que `root`.
    Pour avoir la même configuration qu'à l'IUT, vous pouvez créer une base de données portant votre nom qui servira pour les TDs de PHP.
  - Pour se connecter à votre BDD dans PHP :
    - Dans `Conf.php`, l'hôte est `localhost`, la base de données est celle que vous venez de créer. Pour le login, indiquez `root`. Le mot de passe ne sera pas nécessaire.
    - Dans `Model.php`, changer l'appel au constructeur `new PDO(...)` pour donner la valeur `null` à l'argument `password`. Ceci a pour effet de vous connecter sans mot de passe.
