Interaction PHP-SGBD avec PDO

Alexandre Niveau

GREYC — Université de Caen

En partie adapté du cours de Jean-Marc Lecarpentier

Problématique

- Site web dynamique avec des données qui évoluent ⇒ le plus souvent, utilisation d'un système de gestion de base de données (SGBD)
- Divers SGBD existent : MySQL, PostGreSQL, Oracle, etc.
- PHP doit donc interagir avec divers SGBD
- Au départ, fonctions spécifiques à chacun :
  - `mysql_connect`, `mysql_query`...
  - `pg_connect`, `pg_query`...
- Pas de flexibilité dans le choix du SGBD

Couche d'abstraction

- Limiter la dépendance au SGBD choisi
- Diverses solutions existent pour PHP :
  - PEAR DB
  - PDO (PHP Data Object)
  - Zend Framework
  - Doctrine
  - etc.
- On va utiliser PDO dans ce cours
- Attention, abstraction de l'*accès* à la BD, pas d'abstraction du SGBD lui-même : PDO ne traduit pas les requêtes vers le dialecte SQL du SGBD choisi !

Pourquoi PDO

- Le standard de PHP
- Gestion des requêtes préparées
- Programmation objet + exceptions

- Gestion des transactions (si le SGBD le peut)
- Simple d'utilisation

Connexion au serveur

- Création d'une instance de PDO : `$pdo = new PDO($dsn, $user, $pass);`
- Le DSN (Data Source Name) donne des informations sur la BD utilisée.
- Le format général est `driver:`*`infos`*.
- `driver` dépend de la BD utilisée. Exemples : `mysql` ou `pgsql`
- Les informations et leur format dépendent du driver. Exemples de DSN pour plusieurs drivers :
  - Pour PostgreSQL, `pgsql:host=localhost;dbname=mydatabase`
  - Pour MySQL, `mysql:host=mysql.info.unicaen.fr;port=3306;dbname=jml_3;charset=utf8`
  - Pour Oracle, `oci:dbname=//localhost:1521/mydb`
  - Pour IBM, `ibm:DRIVER={IBM DB2 ODBC DRIVER};DATABASE=accounts; HOSTNAME=1.2.3,4;PORT=23456;PROTOCOL=TCPIP;`
  - etc.
- Dans ce cours on utilisera MySQL — voir slide suivant

Connexion à un serveur MySQL

- Pour MySQL, les informations du DSN ont la forme de couples *`clef=valeur`* séparés par des points-virgules. *Attention à ne pas mettre d'espaces !*
- Les informations attendues dans le DSN pour MySQL sont
  - `host`, le nom ou l'IP de la machine sur laquelle tourne le serveur
  - `port`, le port du host sur lequel le serveur MySQL écoute (optionnel, vaut 3306 par défaut)
  - `dbname`, le nom de la base de données à utiliser
  - `charset`, l'encodage de caractères de la base — **très important** pour éviter certaines failles de sécurité (voir plus loin)
- Serveur MySQL du département info : `mysql.info.unicaen.fr`
  Plusieurs remarques :
  - Attention, il est accessible depuis le serveur web et depuis les machines des salles TP, mais *pas depuis l'extérieur* (et donc notamment pas depuis eduroam) !
  - Outil graphique en ligne pour accéder à vos bases : `https://`*`NUMETU`*`.users.info.unicaen.fr/phpmyadmin`
  - Il est possible qu'aucune base ne soit créée dans votre compte au départ, vous devez le faire vous-même — attention, les noms des bases que vous avez le droit de créer sont très contraints !

Options de connexion

- L'objet PDO qu'on a créé est paramétrable avec un tas d'options [https://www.php.net/manual/fr/pdo.setattribute.php]
- Pour modifier ces paramètres : `$pdo->setAttribute($attribute, $value);`, ou passer les options dans le constructeur avec un tableau associatif
- Les paramètres peuvent être génériques ou spécifiques au SGBD utilisé
- Les paramètres (et certaines valeurs) sont des constantes définies dans la classe PDO
- Utilisé notamment pour définir la gestion des erreurs

[#gestion-des-erreurs]

Gestion des erreurs

- On positionne l'attribut `PDO::ATTR_ERRMODE`
  - `PDO::ERRMODE_SILENT` (option par défaut) : les erreurs sont signalées seulement par le code d'erreur renvoyé par la fonction ⇒ il faut absolument vérifier le code d'erreur à chaque appel

    ➤ **à changer absolument !**
  - `PDO::ERRMODE_WARNING` : émet un message de niveau E_WARNING en cas d'erreur
  - `PDO::ERRMODE_EXCEPTION` : renvoie une `PDOException` avec code et information sur l'erreur
- En pratique : `$pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);`

  ➤ **fortement recommandé** pour bénéficier de la gestion des exceptions

Exécution de requêtes

- Pour faire des requêtes « constantes » (qui n'utilisent pas de variables PHP), méthode `query`
- Elle renvoie le résultat sous forme d'objet `PDOStatement`, via lequel on peut récupérer les données sous divers formats
- Exemples :

```php
<?php
  /* requête select */
  $requete = 'SELECT * FROM users;';
  $stmt = $pdo->query($requete);
  /* récupération de la ligne courante */
  $ligne = $stmt->fetch();
  var_export($ligne);

  /* requêtes insert/update/delete */
  $requete = "INSERT INTO users VALUES ('Toto', 'Dupont');";
  $stmt = $pdo->query($requete);
  /* pas de lignes à récupérer dans le PDOStatement, mais
   * on peut vouloir savoir combien de lignes ont été affectées: */
  echo 'Nombre de lignes affectées :' . $stmt->rowCount();
```

```php
?>
```

Format des données

- Les objets `PDOStatement` obtenus suite à un `SELECT` ont en particulier deux méthodes permettant de récupérer les données :
  - `fetch` ne renvoie qu'**une seule** ligne de résultat
  - `fetchAll` renvoie un tableau avec tous les résultats

    ➤ *attention*, ne pas utiliser s'il y a potentiellement beaucoup de résultats !
- Différents modes pour `fetch` :
  - `PDO::FETCH_ASSOC` : tableau indexé avec les noms de colonnes
  - `PDO::FETCH_BOTH` (défaut) : retourne un tableau indexé par les noms de colonnes mais *aussi* par les numéros de colonnes
  - `PDO::FETCH_OBJ` : retourne un objet anonyme avec les propriétés qui correspondent aux noms des colonnes
  - `PDO::FETCH_INTO` : met à jour une instance existante de la classe demandée
- Marchent aussi avec `fetchAll`

Exemples de `fetch` et `fetchAll`

```php
<?php
$stmt = $pdo->query("SELECT name, species FROM animals");
$tableau = $stmt->fetchAll(PDO::FETCH_ASSOC);
foreach ($tableau as $ligne) {
    /* les champs correspondant à des indices */
  echo "Nom :" . $ligne['name'];
  echo "Espèce :" . $ligne['species'];
}

$stmt = $pdo->query("SELECT name, species FROM animals");
/* on peut préciser le mode globalement : */
$stmt->setFetchMode(PDO::FETCH_OBJ);
/* utilisation classique de fetch() :
   on récupère une ligne à la fois, tant qu'il en reste */
while (($ligne = $stmt->fetch()) !== false) {
    /* les champs correspondent à des attributs */
  echo "Nom :" . $ligne->name;
  echo "Espèce :" . $ligne->species;
}
?>
```

Attention !!

- La méthode `query` n'est à utiliser **que** pour faire des requêtes « constantes », c'est-à-dire avec une chaîne de caractères littérale, **sans variables**
- Ne pas respecter cette consigne rend votre site **vulnérable aux injections SQL**

Injection SQL

- Exemple typique de code vulnérable à une injection SQL :

```php
<?php
/* --- À NE PAS FAIRE --- */
  $login = $_GET['login'];
  $pdo-
>query("SELECT name FROM users WHERE login='$login'");  // très dangereux
?>
```

- Les internautes peuvent passer du code arbitraire à votre BD, par exemple `'; DROP TABLE users; --` (le grand classique) [https://www.xkcd.com/327/]
- La vulnérabilité vient du fait que les données n'ont pas été proprement *échappées*, ce qui est de toute façon nécessaire indépendamment de la sécurité (on peut légitimement essayer de mettre des apostrophes dans son login)
- Une solution : bien penser à toujours échapper les entrées qui viennent de l'utilisateur...
- Meilleure solution : utiliser des *requêtes préparées*

Requêtes préparées

- PDO permet la préparation de requêtes par le moteur SQL
- Objectifs :
  - Optimisation de l'exécution de requêtes répétées
  - Échappement *automatique* et propre des données dans les requêtes

    ➤ sécurité vis-à-vis des injections SQL

[#requetes-preparees-principe]
Requêtes préparées : principe

- La requête contient des champs non remplis, auxquels on peut donner des noms (*placeholders*) qui commencent par un caractère deux-points :

```php
$rq = "UPDATE users SET name=:nom, birthday=:naissance WHERE id=:id"
```

- Au lieu d'exécuter une requête, on la *prépare*, et on récupère un *statement* (`PDOStatement`) :

```php
$stmt = $pdo->prepare($rq);
```

- On a fixé la forme de la requête, impossible de la modifier (par exemple en modifiant la clause WHERE) ou d'en rajouter une à la suite
- On remplit les trous avec les données, et on exécute la requête :

```php
$data = array(
  ':id' => 23456,
  ':nom' => 'Rachel',
  ':naissance' => '2023-12-04',
);
$stmt->execute($data);
```

- Les caractères spéciaux dans les données sont échappées par PDO, donc pas de risque d'oubli ⇒ intégrité des données dans la base
- Attention : on ne peut pas mettre un *placeholder* à la place du nom de la table ou des champs. Seules les valeurs sont concernées.

Autre exemple de requête préparée

- Avec un `SELECT`, on utilise aussi `execute`

```php
<?php
/*** préparation ***/
$rq = "SELECT * FROM animals WHERE species = :espece AND name = :nom";
$stmt = $pdo->prepare($rq);
/*** remplissage des paramètres ***/
$data = array(":espece" => "humain", ":nom" => "Jean-Michel");
/*** exécution du statement ***/
$stmt->execute($data);
/*** récupération du résultat ***/
$result = $stmt->fetchAll();
?>
```

- *Attention* à bien appeler `execute` avant de faire `fetch`/`fetchAll` !

Compléments : Guillemets autour des paramètres

- Pas de guillemets autour des paramètres dans la requête
- Ils sont ajoutés par PDO lors de l'exécution de la requête
- C'est le cas quel que soit le type du champ (même pour les nombres)
- En général ça ne pose pas de problème : le SGBD fait les conversions en utilisant le type des champs dans le schéma de BD
- Mais ça ne fonctionne pas pour les autres nombres. Par exemple `LIMIT` en SQL attend des entiers sans guillemets :

```php
<?php
$rq = "SELECT name FROM animals LIMIT :nb";
$stmt = $pdo->prepare($rq);
$data = array(":nb" => 10);
/* ne fonctionne pas : la requête exécutée est *
 *   SELECT name FROM animals LIMIT '10'       */
$stmt->execute($data);
$result = $stmt->fetchAll();
```

Compléments : Type des paramètres

- Pour éviter ça il faut préciser à PDO le type du paramètre
- Le plus simple est d'utiliser `bindValue`
- Trois principaux types possibles :
  - `PDO::PARAM_STR` (par défaut)

    - `PDO::PARAM_INT`
    - `PDO::PARAM_BOOL`
- Avec les deux derniers, les valeurs ne sont pas converties en chaînes et aucun guillemet n'est ajouté

```php
<?php
$rq = "SELECT name FROM animals LIMIT :nb";
$stmt = $pdo->prepare($rq);
$stmt->bindValue(":nb", 10, PDO::PARAM_INT);
$stmt->execute();
$result = $stmt->fetchAll();
```

- Attention, si la valeur ne correspond pas au type, il n'y aura pas de conversion ni d'erreur (sauf cas très particuliers). Ce n'est pas une sécurité supplémentaire, mais une indication de formatage de requête SQL.

Compléments : Émulation, encodage

- Par défaut, PDO ne fait qu'*émuler* la préparation des requêtes
    - Il se charge d'échapper les entrées ; mais il n'y a pas d'optimisation de vitesse
    - De cette façon, on peut profiter de la sécurité apportée par les requêtes préparées même pour les SGBD qui ne les supportent pas.
    - Pour désactiver l'émulation, et utiliser les requêtes préparées natives du SGBD, mettre l'option `PDO::ATTR_EMULATE_PREPARES` à `false`
- Il est important de préciser l'encodage de la base dans le DSN lors de la création de l'instance de PDO

  ➤ ne pas le faire peut créer une vulnérabilité dont même les requêtes préparées ne protègent pas [https://stackoverflow.com/questions/5741187/sql-injection-that-gets-around-mysql-real-escape-string/12118602#12118602] si elles ne sont qu'émulées.
- Au passage, il n'y a qu'**un seul encodage** à utiliser dans MySQL : **`utf8mb4`** (et pas `utf8`, qui n'est pas le vrai UTF-8 !)
- Une dernière chose : la méthode `lastInsertId()` [http://php.net/manual/en/pdo.lastinsertid.php] de PDO renvoie la valeur de la clef primaire autoincrémentée de la dernière entrée ajoutée (si le SGBD supporte cette fonction).

  ➤ peut être utile pour MySQL (pour PostgreSQL, utiliser `INSERT… RETURNING` semble plus simple)

Références et guides

- Manuel PHP sur PDO [http://php.net/manual/fr/book.pdo.php]

Tutoriels

- Un bon tutoriel PDO [https://phpdelusions.net/pdo] (en anglais)

Lectures complémentaires

- How to support full Unicode in MySQL databases [https://mathiasbynens.be/notes/mysql-utf8mb4] : car ce que MySQL appelle UTF-8 n'est pas de l'UTF-8…

 [http://creativecommons.org/licenses/by-nc-sa/4.0/]