Vade-mecum MySQL et PDO
V1.0.0

Document en ligne : www.mickael-martin-nevot.com

1 Utilisation de PHP et MySQL avec PDO

PHP Data Objects (PDO) est une extension PHP gérant la communication avec les bases de données. C'est une interface d'abstraction à l'accès de données offrant des fonctions génériques pour :

- la connexion aux bases de données ;
- le traitement des requêtes ;
- le traitement des tuples de ces requêtes ;
- la gestion des erreurs éventuelles.

PDO est disponible dans PHP depuis la version 5.1. Contrairement à des solutions comme PearDB ou AdoDB, qui ont la même ambition, PDO est réputée rapide puisqu'elle fait partie du noyau même de PHP, et qu'il s'agit d'une bibliothèque écrite en C/C++.

2 Configuration de PHP

Voici un exemple de configuration du fichier `php.ini` prenant en compte un serveur de développement et un de production :

```
[dev_server]

driver = mysql
host = localhost
username = root
password = "mysql"

[prod_server]

driver = mysql
host = 213.45.56.12
username = administrator
password = "secret"
```

3 La connexion avec PDO

On instancie la classe PDO en utilisant le DSN suivi de l'identifiant utilisateur et bien sûr du mot de passe :

```
// MySQL
$connection = new PDO('mysql:host=$host;dbname=$db', $user, $pwd);
// PostgreSQL
$connection = new PDO('pgsql:host=$host;dbname=$db', $user, $pwd);
```

Le DSN *(Data Source Name)* est une chaîne de caractères comportant les paramètres de connexion. Dans l'ordre, et divisés par des séparateurs, il y a le nom du pilote de base de données (`mysql`), le nom de l'hébergeur (`host`) et le nom de la base de données (`dbname`).

La fonction `parse_ini_file()` permet de traiter le contenu du fichier `php.ini` en renvoyant un tableau associatif. Il est donc possible de construire le DSN dynamiquement grâce à cette fonction :

```
define('PHP_INI_PATH', 'php.ini');
define('DB_NAME', 'database');

$debug = true;
$conf = parse_ini_file(PHP_INI_PATH, true);

if (!is_array($config))
{
    throw new DatabaseException('Erreur de chargement de configuration');
}
$conf = ($debug ? $conf['dev_server'] : $conf['prod_server']);
$dsn = $conf['driver'] . ':dbname=' . DB_NAME . ';host=' . $conf['host'];
$connection = new PDO($dsn, $conf['username'], $conf['password']);
```

4 Effectuer une requête

Il existe deux méthodes pour effectuer des requêtes avec un objet de classe `PDO` :

- `query()` ;
- `exec()` ;

On utilise communément la méthode `query()` pour des sélection, et `exec()` pour des insertions, des mises à jour et des suppressions.

4.1 Les sélections

```
// Instanciation de l'objet PDO.

if (!$statement = $connection->query('SELECT * FROM students')
{
     throw new Exception('Wrong query');
}

foreach ($statement as $row)
{
    echo '<pre>' . print_r($row) . '</pre>';
}
```

Il existe deux méthodes pour récupérer des données avec un objet de classe `PDOStatement` :

- `fetchAll()` : renvoie tous les résultats dans un tableau ;
- `fetch()` : renvoie les résultats ligne par ligne.

Si le nombre de résultats est de l'ordre de quelques centaines maximum, la fonction `fetchAll()` peut être utilisé. Au-delà, il est conseillé d'avoir plutôt recours à la fonction `fetch()`.

4.1.1 Les différents types d'analyse

Il est possible de récupérer les tuples sous différentes formes : tableaux, objets etc. Cela est possible en paramétrant l'appel de la fonction `fetch()` avec une constante de la classe `PDO` :

- **PDO::FETCH_NUM** : renvoie un tableau indexé ;
- **PDO::FETCH_ASSOC** : renvoie un tableau associatif ;
- **PDO::FETCH_OBJ** : renvoie un objet ;
- **PDO::FETCH_BOTH** : par défaut, renvoie un tableau associatif avec des clefs à la fois numériques et littérales.

Voici un exemple d'utilisation avec la fonction `fetchAll()` :

```
// Instanciation de l'objet PDO.
// Envoie de la requête de sélection.

$rows = $statement->fetchAll(PDO::FETCH_ASSOC));
foreach ($rows as $row)
{
    echo $row['name']. ' ' . $row['firstname'] . PHP_EOL . '<br>';
}
```

Voici un exemple d'utilisation avec la fonction `fetch()` :

```
// Instanciation de l'objet PDO.
// Envoie de la requête de sélection.

while ($row = $statement->fetch(PDO::FETCH_OBJ))
{
    echo $row->name . ' ' . $row->firstname . PHP_EOL . '<br>';
}
```

4.2 Les insertions, mises à jour et suppressions

La méthode `exec()` renvoie le nombre de lignes modifiées (à tester avec `===` pour éviter de confondre le nombre de 0 tuple renvoyé et la valeur `false` renvoyée), elle est donc utile lors de modifications des données. Il est également possible d'utiliser la méthode `lastInsertId()` pour obtenir l'identifiant du dernier tuple inséré. Voici un exemple d'utilisation :

```
// Instanciation de l'objet PDO.

if (!$count = $connection->exec('INSERT INTO students (name, firstname,
grade) VALUES (\'smith\', \'john\', 10)')
{
     throw new Exception('Wrong query');
}

echo $count . ' insertion(s) effectuée(s)' . PHP_EOL . '<br>';
echo 'Dernier id inséré : ' . $connection ->lastInsertId() . PHP_EOL . '<br>';
```

4.3 Les requêtes préparées

Les requêtes préparées sont des modèles de requêtes. C'est utile pour la réutilisation de requête mais aussi pour la sécurisation contre les injections SQL. Le serveur MySQL traite une requête de la manière suivante :

1. Vérification dans le cache de la présence de la requête, si oui son résultat est renvoyé.
2. Analyse de la syntaxe de la requête, pré-compilation et optimisation avec élaboration d'un plan d'exécution.
3. Lecture et exécution du plan par le moteur.
4. Renvoie du résultat au client.

Ainsi, avec ce fonctionnement, le bénéfice de réutiliser des requêtes préparées est évident.

Pour une requête préparée, l'emplacement réservé aux variables est signalé par un `?`. La requête est initialisée avec l'appel de la méthode `prepare()`, puis exécutée avec l'appel de la méthode `execute()` qui récupère des paramètres, dans l'ordre, sous forme de tableau.

Voici un exemple de requête de sélection sous la forme d'une requête préparée :

```
// Instanciation de l'objet PDO.

$gradeInf = 10;
$gradeSup = 18;

$statement = $connection->prepare('SELECT DISTINCT(name), firstname from
students WHERE grade >= ? AND grade <= ? ORDER BY name DESC');
$statement->execute(array($gradeInf, $gradeSup));

while ($row = $statement->fetch(PDO::FETCH_OBJ))
{
    echo $row->name . PHP_EOL . '<br>';
}
```

Voici un autre exemple, où une requête d'insertion se fait en itérant dans un tableau de tableaux :

```
// Instanciation de l'objet PDO.

$students[] = array('name' => 'Smith', 'firstname' => 'John', 'grade' => 10);
$students[] = array('name' => 'Kumar', 'firstname' => 'Ashok', 'grade' => 15);

$query = 'INSERT INTO students (name, firstname, grade) VALUES (?, ?, ?)';
$statement = $connection->prepare($query);

foreach($students as $stu)
{
    $statement->execute(array($stu['name'], $stu['firstname'], $stu['note']));
}
```

Il existe une autre façon d'écrire une requête préparée, en nommant les variables de communication (préfixées par `:`). Avec cette utilisation, l'ordre de spécification des arguments n'est pas déterminant :

```
// Instanciation de l'objet PDO.

$statement = $connection->prepare('INSERT INTO teacher (name, firstname, grade)
VALUES (:name, :firstname, :grade)');
```

```
foreach($teachers as $teacher)
{
    $statement->execute(array(':firstname' => $teacher['firstname'],
                              ':grade' => $teacher['grade'],
                              ':name' => $teacher['name']));
}
```

Voici enfin une dernière façon, très courante, d'utiliser les requêtes préparées. Avec cet usage, pas de passage de paramètres avec un tableau à la fonction `execute()`, les paramètres sont attachés à leurs valeurs respectives dans la requête de la manière suivante :

```
// Instanciation de l'objet PDO.

$statement = $connection-
>prepare('INSERT INTO students (name, firstname, grade) VALUES (:name, :firstna
me, :grade)');

foreach($students as $student)
{
    $statement->bindValue(':name', $student['nom']);
    $statement->bindValue(':firstname', $student['prenom']);
    $statement->bindValue(':grade', $student['note']);

    $statement->execute();
}
```