MySQL et PHP

Site: AMIO-FIT
Cours: B2 - PHP Laravel
Livre: MySQL et PHP

Imprimé par: Davy Blavette
Date: lundi 21 novembre 2022, 08:52

Description

Pour pouvoir manipuler nos bases de données MySQL en PHP (sans passer par phpMyAdmin), nous allons déjà devoir nous connecter à MySQL.

Table des matières

1. PDO

Pour pouvoir manipuler nos bases de données MySQL en PHP (sans passer par phpMyAdmin),nous allons déjà devoir nous connecter à MySQL.

Pour cela, le PHP met à notre disposition deux API (Application Programming Interface) :

- L'extension `MySQLi`
- L'extension `PDO` (PHP Data Object)

Chacune de ces deux API possède des forces différentes et comme vous vous en doutez ellesne sont pas forcément interchangeables.

Il existe notamment une différence notable entre ces deux API : l'extension MySQLi ne va fonctionner qu'avec les bases de données MySQL tandis que PDO va fonctionner avec 12 systèmes de bases de données différents.

Pour cette raison, nous préférerons généralement le PDO car si vous devez un jour utiliser unautre système de bases de données, le changement sera beaucoup plus simple que si vous avez tout codé en MySQLi auquel cas vous devrez réécrire le code dans son ensemble.

Dans ce module, nous utiliserons donc `PDO` et nous n'aborderons donc pas `MySQLi`.Vous trouverez la documentation complète de PDO sur le site php.net.

2. Connexion

Pour se connecter en utilisant PDO, nous allons devoir instancier la classe PDO en passant au constructeur la source de la base de données (DSN – Data Source Name) (serveur + nom de la base de données) ainsi qu'un nom d'utilisateur et un mot de passe.

Très souvent, pendant la phase de développement, nous accèderons à la bdd avec l'utilisateur root et sans mot de passe. Ceci est absolument interdit en production.

```
<!DOCTYPE html>
<html lang="en">

<head>
<meta charset="UTF-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>MySQL - PHP</title>
</head>

<body>
<h1>Base de données MySQL</h1>
<?php
 // Déclaration des variables
 $serverName = 'lab018.2isa.org';
 $dbName = 'db1';
 $userName = 'root';
 $port = '33018';
 $pwd = 'sqsddssqdxw';

 // Connexion à la base de données
 try{
        $dbCon = new PDO("mysql:host=$serverName;dbname=$dbName;port=$port;charset=UTF8", $userName, $pwd);
        echo "Connected to $serverName<br>";
        var_dump($dbCon);
 }catch(Exception $e){
        die('Erreur : ' . $e->getMessage());
 };
?>

</body>

</html>
```

2.1. TP Connexion

Exo 01 - Se connecter

3. Exceptions

Notre variable `$bdCon` contient un objet vide de type PDO. Nous sommes maintenant connectés à notre bdd.

On peut noter que pour se connecter à la bdd, il faut qu'elle existe. Il faut donc la créer avec un outil comme `phpMyAdmin`.

Notez également qu'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.

```
// tentative de connexion à la base de données
try {
$dbCon = new PDO("mysql:host=$serverName;dbname=$dbName;charset=UTF8", $userNam
e, $pwd);
//On définit le mode d'erreur de PDO sur Exception
$dbCon->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
echo 'Connexion réussie';
}
/*On capture les exceptions si une exception est lancée et on affiche les informations relatives à celle-ci*/
 catch(PDOException $e){
echo "Erreur : " . $e->getMessage();
}

echo '<br>La suite du script';
```

Ici, 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`.

4. Fin de connexion

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

Si on utilise 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.

```
// On ferme la connexion à la base de données
$dbCon = null;
```

5. Création d’une table

MySQL nous offre beaucoup de choix de types de données différent nous permettant de créer des tables de manière vraiment précise. Pour le moment, vous pouvez retenir qu’il existe quatre grands types de données principaux en MySQL : les données de type texte, les données de type nombre, les données de type date et les données de type spacial.

Vous pouvez retrouver la liste complète des types de valeur ici.

```
<!DOCTYPE html>
<html lang="en">

<head>
<meta charset="UTF-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>MySQL - PHP</title>
</head>

<body>
<h1>Base de données MySQL</h1>
<?php
$serverName = 'localhost';
$dbName = 'pdodb';
$userName = 'root';
$pwd = '';

try {
$dbConn = new PDO("mysql:host=$serverName;dbname=$dbName;charset=UTF8", $userNa
me, $pwd);
$dbConn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

$sql = "CREATE TABLE Clients (
Id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
Nom VARCHAR(30) NOT NULL,
Prenom VARCHAR(30) NOT NULL, Adresse VARCHAR(70) NOT NULL, Ville VARCHAR(30) NOT NULL,
Codepostal VARCHAR(30) NOT NULL, Pays VARCHAR(30) NOT NULL,
Mail VARCHAR(50) NOT NULL,
DateInscription TIMESTAMP DEFAULT CURRENT_TIMESTAMP, UNIQUE(Mail))
DEFAULT CHARACTER SET utf8 DEFAULT COLLATE utf8_general_ci";

$dbConn->exec($sql);

echo 'Table bien créée !';
}
catch(PDOException $e){
echo "Erreur : " . $e->getMessage();
}
?>

</body>

</html>
```

Notez que ces opérations peuvent être effectuées en utilisant phpMyAdmin. Il en effet assez rare de créer des bdd et des tables dynamiquement.

6. Insérer des données

Reprenons par exemple notre table « Clients » créée dans la leçon précédente.

Cette table possède neuf colonnes dont une colonne Id avec un attribut AUTO_INCREMENT et une colonne DateInscription qui possède un attribut TIMESTAMP.

Essayons d'insérer une première entrée dans cette table en utilisant PHP et PDO :

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>MySQL - PHP</title>
  </head>
  <body>
    <h1>Base de données MySQL</h1>
<?php
$serverName = 'localhost';
$dbName = 'pdodb';
$userName = 'root';
$pwd = '';
try
{
    $dbConn = new PDO("mysql:host=$serverName;dbname=$dbName;charset=UTF8", $userName, $pwd);
    $dbConn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    $sql = "INSERT INTO Clients(Nom,Prenom,Adresse,Ville,Codepostal,Pays,Mail)            VALUES('Garcia','Philippe','St Ferréols','St
Rome','12490','France','philippe.g arcia@example.com')";
    $dbConn->exec($sql);
    echo 'Entrée ajoutée dans la table !';
}
catch(PDOException $e)
{
    echo "Erreur : " . $e->getMessage();
} ?>
  </body>
</html>
```

6.1. TP Données

TP 02 - Insertion

7. Transaction

La première idée serait d'enchainer plusieurs fois les insertions comme au chapitre précédent.

L'un des gros défauts de cette méthode est que s'il y a un problème d'exécution en cours du script, certaines entrées vont être insérées et pas d'autres et certaines entrées pourraient nepas avoir toutes leurs données insérées.

Pour éviter cela, nous pouvons ajouter les méthodes `beginTransaction()`, `commit()` et `rollBack()` dans notre code.

La méthode `beginTransaction()` permet de démarrer ce qu'on appelle une transaction et de désactiver le mode `autocommit`. Concrètement, cela signifie que toutes les manipulationsfaites sur la base de données ne seront pas appliquées tant qu'on ne mettra pas fin à la transaction en appelant `commit()`.

La méthode `commit()` sert donc à valider une transaction, c'est-à-dire à valider l'application d'une ou d'un ensemble de requêtes SQL. Cette méthode va aussi replacer la connexion en mode `autocommit`.

La méthode rollBack() sert à annuler une transaction si l'on s'aperçoit d'une erreur. Cette méthode restaure le mode autocommitaprès son exécution.

```php
<!DOCTYPE html>
<html lang="en">

<head>
<meta charset="UTF-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>MySQL - PHP</title>
</head>

<body>
<h1>Base de données MySQL</h1>
<?php
$serverName = 'localhost';
$dbName = 'pdodb';
$userName = 'root';
$pwd = '';

try {
$dbConn = new PDO("mysql:host=$serverName;dbname=$dbName;charset=UTF8", $userNa
me, $pwd);
$dbConn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

$dbConn->beginTransaction();

$sql1 = "INSERT INTO Clients(Nom,Prenom,Adresse,Ville,Codepostal,Pays,Mail) VALUES('Doe','John','rue des
trucs','Millau','12100','France','john.doe@example
.com')";

$dbConn->exec($sql1);

$sql2 = "INSERT INTO Clients(Nom,Prenom,Adresse,Ville,Codepostal,Pays,Mail)
VALUES('Bond','James','rue des gadgets','Toulouse','31','France','james.bond@ex ample.com')";

$dbConn->exec($sql2);

$dbConn->commit();

echo 'Entrées ajoutées dans la table !';
}
catch(PDOException $e){
$dbConn->rollBack();
echo "Erreur : " . $e->getMessage();
}

?>

</body>

</html>
```

Cette deuxième façon de procéder est déjà meilleure que la première. Cependant, en pratique, nous utiliserons plutôt les requêtes préparées pour insérer plusieurs entrées d'un coup dans nos tables, notamment lorsque les données seront fournies par les utilisateurs.

8. Requêtes préparées

Les requêtes préparées sont des requêtes qui vont être créées en trois temps : la préparation,

la compilation et l'exécution.

Tout d'abord, une première phase de préparation dans laquelle nous allons créer un template ou schéma de requête, en ne précisant pas les valeurs réelles dans notre requête mais en utilisant plutôt des marqueurs nommés (sous le forme :nom) ou des marqueurs interrogatifs (sous la forme ?).

Ces marqueurs nommés ou interrogatifs (qu'on peut plus globalement nommer marqueurs de paramètres) vont ensuite être remplacés par les vraies valeurs lors de l'exécution de la requête.

Une fois le template créé, la base de données va analyser, compiler, faire des optimisations sur notre template de requête SQL et va stocker le résultat sans l'exécuter.

Finalement, nous allons lier des valeurs à nos marqueurs et la base de données va exécuter la requête. Nous allons pouvoir réutiliser notre template autant de fois que l'on souhaite en liant de nouvelles valeurs à chaque fois.

Utiliser des requêtes préparées va nous offrir deux principaux avantages par rapport à

l'exécution directe de requêtes SQL :

- Nous allons gagner en performance puisque la préparation de nos requêtes ne va être faite qu'une fois quel que soit le nombre d'exécutions de notre requête.

- Le risque d'**injection SQL** est minimisé puisque notre requête est pré formatée et nous n'avons donc pas besoin de protéger nos paramètres ou valeurs manuellement.

8.1. Les méthodes

Pour exécuter une requête préparée, nous allons cette fois-ci devoir utiliser la méthode execute() et non plus exec() comme on utilisait depuis le début de cette partie.

En utilisant des marqueurs dans nos requêtes préparées, nous allons avoir deux grandes options pour exécuter la méthode **execute**() :

- On va pouvoir lui passer un tableau de valeurs de paramètres (uniquement en entrée).
- On va pouvoir d'abord appeler les méthodes **bindParam**() ou **bindValue**() pour respectivement lier des variables ou des valeurs à nos marqueurs puis ensuite exécuter **execute**().

Notez que passer un tableau à la méthode execute() , ececute(array(xxx,xxx, …)); est la façon de faire la plus simple qui va fonctionner dans la majorité des cas.

Les méthodes execute() bindParam() et bindValue() appartiennent toutes les trois à la classe PDOStatement (documentation) et non pas à la classe PDO (documentation). La classe PDOStatement représente une requête préparée et, une fois exécutée, l'ensemble des résultats associés.