Connection MySQL avec PDO

Nouvelle version du site sur WWW.JACKSAY.COM

Présentation

PDO est une **classe PHP** déstinée à permettre à PHP de communiquer avec un serveur de données. (PDO : Php Data Object)

PDO est ce qu'on appel une **couche d'abstraction**, c'est à dire qu'il va permet de communiquer avec n'importe quel server de base de données : MySQL, Oracle, Postgresql, etc... (En tout cas sur des requètes simples).

PDO va permettre (c'est son intérêt majeur) de **sécuriser** les requètes et de favoriser la réutilisation du code grâce aux **requètes préparées**.

1. Connecting people !

Voici la connection type à un serveur MySQL. Notez que la connection au serveur de données est établie au moment où l'objet PDO est instancier :

```php
// Connection au serveur
$dns = 'mysql:host=localhost;dbname=formation';
$utilisateur = 'db_rider';
$motDePasse = 'azerty';
$connection = new PDO( $dns, $utilisateur, $motDePasse );
```

La **DNS**, c'est en quelques sorte le point d'entrée pour accéder à notre base de données, il débute par le code du **moteur de base de données** (on appel ça l'**engine** parfois). Dans notre cas c'est **`mysql:`**

Ensuite on trouve l'adresse du serveur : **`host=localhost`**

Puis le nom de la base de données : **`dbname=formation`**

Et parfois on trouve aussi le port : **`port=3606`**, quand le port est spécifié, la variable **`$dns`** ressemble à ça :

```php
// DNS où le port est spécifié
$dns = 'mysql:host=localhost;dbname=formation;port=3606';
```

Quand le **port** est celui utilisé par défaut par le moteur de base de données, le spécifier est facultatif. Dans notre cas par exemle, le moteur de base de données est MySQL, pas conséquent, préciser le port 3606 est inutile (c'est le port par défaut).

L'accès au serveur est spécifier dans une chaine de caractère (la DNS), les paramètres s'expliment sour la forme **`parametre=valeur`** séparés par des point-virgules. Dans notre cas : **`"mysql:host=nomServeur;dbname=studio_formation"`**

Ensuite, le **2ème** et le **3ème paramètres** sont simplement les **identifiant** / **Mot de passe** MySQL, dans cet exemple on suppose que vous avez créez un utilisateur **formation** avec le mot de passe **azerty** ayant des droits d'accès à la base de données **`formation`**.

2. Erreur levée par PDO

PDO étant orienté objet, il lève des exceptions en cas de problème. Les erreurs levée sont des **PDOException**, voici les différents type d'erreurs que vous pouvez rencontrer lorsque vous instanciez un objet PDO.

2.1. Erreur "could not find driver"

Cette erreur surviens si vous avez **mal renseigné le moteur de base de données** dans la DNS ou si le drivers choisi n'est pas supporté par votre serveur. Dans le cas du drivers MySQL, il est généralement actif par défaut sur la majorité des hébergements (et biensur activé par défaut sur W/Mamp).

2.2. Erreur "Unknown MySQL server host"

Cette erreur surviens quand le nom du serveur est mal renseigné / indisponible (MySQL dans notre cas). Chez certains "gros" hébergeurs, le nom du serveur beb n'est généralement pas **`localhost`**.

2.3. Erreur "Can't connect to MySQL server"

Dans le cadres des accès distants (le serveur MySQL n'est pas sur la même machine que le serveur Web), Vous obtiendrez ce message si le **serveur est planté / indisponible**, ou si le serveur auquel vous tenter d'accéder n'est pas un serveur de données MySQL.

Le message d'erreur appraît généralement après un certains temps (assez long) de chargement, vous verrez que la page "mouline"

dans le vide. Il peut arrivé que votre serveur web lance un timeout avant que cette erreur ne survienne (votre page attend la réponse du serveur de données, mais ce dernier a mis trop de temps à répondre, du coup votre page se crash).

2.4. Erreur "Unknown database <db_name>"

Là le **nom de la base de données est incorrect**, ou plus grave, la base de données n'existe plus...

2.5. Erreur "Access denied for user"

Un grand classique, **vous n'avez pas le droit d'accéder au serveur**. Soit votre identifiant, soit votre mot de passe, soit les deux sont mal renseignés.

2.6. Intercepter les erreurs de connection avec un try { catch }

Le bloc try / catch est très pratique pour intercepter ce type d'erreur (on appelle ce genre d'erreur des **Exception**) :

```php
// Connection au serveur
try {
  $dns = 'mysql:host=localhost;dbname=formation';
  $utilisateur = 'db_rider';
  $motDePasse = 'azerty';
  $connection = new PDO( $dns, $utilisateur, $motDePasse );
} catch ( Exception $e ) {
  echo "Connection à MySQL impossible : ", $e->getMessage();
  die();
}
```

La fonction **`die()`** porte bien son nom, elle va tuer le script PHP (toutes les lignes situées après le die ne seront jamais executées).

Il est fortement recommandé de se créer un fichier php (par exemple connection.php) contenant ces quelques lignes. En effet, si l'une de vos pages necessite alors un accès à la base de données, il vous suffira d'inclure ce fichier avec **`require_once`**.

```php
require_once('conf/connection.php');
```

3. Envoyer des requètes

Une fois connecter au serveur MySQL avec notre objet PDO, nous allons pouvoir commencer à envoyer des requètes au serveur. Il existe plusieurs façon d'envoyer des requètes avec PDO, d'abord, tout dépend du type de requètes.

On distingue 2 types de requètes :

- Les requètes qui selectionnent des enregistrements : **`SELECT`**
- Les requètes qui transforment les enregistrements : **`UPDATE`** / **`INSERT`** / **`DELETE`**

Pour tester les requètes qui suivent, vous pouvez télécharger la même table que dans les exemples qui suivent :

http://exemples.jacksay.com/PHP/PDO/createurs.sql : Fichier SQL à importer pour tester les requètes qui suivent (à importer dans votre base de données).

3.1. Envoyer des SELECT

Pour **selectionner des enregistrements**, nous utiliserons la méthode **`query($requete)`**.

```php
// On établis la connection
require_once('conf/connection.php');

// On envois la requète
$select = $connection->query("SELECT * FROM createurs");
```

La variable **`$select`** contiend maintenant le résultat de la requète, mais sous une forme un peu particulière : un **PDOStatement**. Vous pourrez allé consulter la documentation pour obtenir des informations complémentaires sur cette classe.

http://www.php.net/manual/fr/class.pdostatement.php : Documentation officiel pour PDOStatement

Cet objet contiend la réponse du serveur de données à la requète que nous lui avons envoyé. Cette objet va nous permettre de gérer l'affichage des données reçues, pour cela il existe plusieurs méthodes :

3.1.1. Avec setFetchMode / fetch

PDO nous offre la libertée d'utiliser la réponse au format que nous voulons, nous pouvons dire que nous voulons traiter les enregistrements reçus comme des tableaux, comme des objets typés, etc... Comme nous débutons, nous allons traiter les enregistrements comme des objets simples.

Donc juste après avoir envoyé la requète (ou avant peu importe), nous allons "configurer" notre objet PDOStatement pour qu'ils nous livre les enregistrements comme des objets :

```php
// On indique que nous utiliserons les résultats en tant qu'objet
$select->setFetchMode(PDO::FETCH_OBJ);
```

Notre variables **`$select`** contiend maintenant un objet pour chaque enregistrement obtenu, pour traiter tous les résultats nous allons utilise la boucle TANT QUE (la boucle while) :

```php
// Nous traitons les résultats en boucle
while( $enregistrement = $select->fetch() )
```

```
{
  // Affichage d'un des champs
  echo '<h1>', $enregistrement->nom, ' ', $enregistrement->prenom, '</h1>';
}
```

Normalement, vous verrez une liste de nom de personnes connues appraitre dans un titre H1...

Si la réponse ne contiens pas de résutats, **`$select->fetch()`** va retourner **`NULL`**/**`FALSE`**. (Et donc l'éxécution de la boucle sera interrompue).

Si vous faites une requète qui ne doit retourner qu'un seul enregistrement, vous n'avez pas besoin d'utiliser une boucle pour traiter les résultats, mais utilisez quand même une condition pour éviter un message d'erreur disgracieux au moment de l'affichage :

```
// Traitement d'un seul résultat
$enregistrement = $select->fetch();

// On test si la variable $enregistrement, au cas
// ou elle serait vide.
if( $enregistrement ) {
  echo '<h1>', $enregistrement->nom, ' ', $enregistrement->prenom, '</h1>';
}
// La requète n'a pas retournée de résultat
else {
  echo "Aucun résultat";
}
```

3.1.2. Méthode fetch(PDO::FETCH_OBJ)

Petite variante avec une ligne de moins, dans cette exemple, le format de récupération (Format Objet) est précisé au moment du **`fetch()`** :

```
// Nous traitons les résultats en boucle
// C'est lors de l'utilisation de fetch() que nous spécifions
// le format de récupération pour le traitement.
while( $enregistrement = $select->fetch(PDO::FETCH_OBJ) )
{
  // Affichage d'un des champs
  echo '<h1>', $enregistrement->nom, ' ', $enregistrement->prenom, '</h1>';
}
```

3.1.3. Méthode fetchAll

Cette méthode va convertir notre objet de résultats en un tableau d'objet, ensuite nous traitons le tableau comme un tableau classique, ce cas d'application est surtout utilisé pour traiter des listes de résultats (Lorsque l'on s'attend à plusieurs résultats) :

```
// On transforme les résultats en tableaux d'objet
$createurs = $select->fetchAll(PDO::FETCH_OBJ);

// On traite le tableau $créateur
while( $enregistrement = next($createurs) )
{
  // Affichage d'un des champs
  echo '<h1>', $enregistrement->nom, ' ', $enregistrement->prenom, '</h1>';
}
```

Je n'ai pas fait de test de performance sur ces différentes méthodes, pour ma part j'aime bien la dernière car cela me permet potentiellement de manipuler le tableau... Sinon j'utilise la précédente quand je n'attend qu'un seul résultat.

3.1.4. C'est quoi ces caractères bizarres ???

Si vous avez mis des caractères spéciaux dans l'une des données affichées, c'est le drame.

Rien ne sert de vérifier le format de document ça ne viens pas de lui :P (Mais faites le quand même ça ne coute rien).

En effet, lors d'un echange avec le serveur web, l'**encodage de transmission n'est pas formement l'UTF8** !!!

Pour corriger/forcer ça nous avons 2 solutions :

- Une solution standard mais pas très lisible
- Une solution moins standard mais plus facile à lire.

3.2. Gérer d'encodage de la connection

Pour forcer l'échange en **UTF8** avec MySQL, nous devons envoyer une requète au serveur pour lui expliquer que nous allons communiquer avec lui en **UTF8**, de façon standard, cela se joue au moment de la connection, et Ô bonheur, l'Object PDO est prévus pour ça (Quel heureux hazard !).

En effet, la ligne :

```
$connection = new PDO( $dns, $utilisateur, $motDePasse );
```

Nous permet d'ajouter un **4ème paramètres** pour préciser des options de connection, comme par exemple une requète à exécuter au moment ou la connection est établie, cela donne :

```
// Connection au serveur
try {
  $dns = 'mysql:host=localhost;dbname=ma_base_de-donnees';
  $utilisateur = 'sergio';
  $motDePasse = 'azerty';

  // Options de connection
  $options = array(
```

```
    PDO::MYSQL_ATTR_INIT_COMMAND    => "SET NAMES utf8"
  );

  // Initialisation de la connection
  $connection = new PDO( $dns, $utilisateur, $motDePasse, $options );
} catch ( Exception $e ) {
  echo "Connection à MySQL impossible : ", $e->getMessage();
  die();
}
```

Notez l'ajout de la variable **$options** (un array) qui va contenir une option **'MYSQL_ATTR_INIT_COMMAND'**, cette options permet d'envoyer une requètes au moment ou la connection est établie, la requètes en question **"SET NAMES utf8"** indique simplement à MySQL que nous allons echanger nos données en UTF8.

Bon la version moins standard consiste simplement à faire cette requète juste après la connection :

```
// Connection au serveur
try {
  $dns = 'mysql:host=localhost;dbname=ma_base_de-donnees';
  $utilisateur = 'sergio';
  $motDePasse = 'azerty';
  $connection = new PDO( $dns, $utilisateur, $motDePasse );
  $connection->query("SET NAMES utf8");
} catch ( Exception $e ) {
  echo "Connection à MySQL impossible : ", $e->getMessage();
  die();
}
```

Ca reviens au même, mais disons que la première version, certes plus verbeuse, est plus "élégante" (et accessoirement, nous allons par la suite ajouter des paramètres dans les options).

3.3. Pas d'erreur houston ?

Autre problème, vous avez peut-être (plus ou moins volontairement) commis des erreur en saisissant la requète, si ça n'est pas le cas allez y, introduisez une erreur de syntaxe dans la ligne :

```
$select = $connection->query("SELECT * FROM mauvaisetable");
```

La requète ne produit aucune erreur ! C'est très contrariant...

Cela est un réél problème car par la suite, nous executons une serie de traitement pour les résultats de cette requète en partant du principe que cette dernière a fonctionné... Si elle échoue à cause d'une erreur de syntaxe, nous risquons de mettre du temps à comprendre que le problème se situe au niveau de la requète et pas au niveau de son traitement...

Pour régler le soucis, la première étape va être d'indiquer à notre connection que nous voulons que des erreurs soit émises, pour cela modifier notre fichier de connection de cette façon :

```
// Connection au serveur
try {
  $dns = 'mysql:host=localhost;dbname=ma_base_de-donnees';
  $utilisateur = 'sergio';
  $motDePasse = 'azerty';

  // Options de connection
  $options = array(
    PDO::MYSQL_ATTR_INIT_COMMAND => "SET NAMES utf8",
    PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION
  );
  $connection = new PDO( $dns, $utilisateur, $motDePasse, $options );
} catch ( Exception $e ) {
  echo "Connection à MySQL impossible : ", $e->getMessage();
  die();
}
```

En testant l'envois de la requète, vous pourrez constater qu'une belle erreur s'affiche, bien entenu, la récupération des requètes et leur traitement vont maintenant être placées des des blocs try / catch :

```
// Connection au serveur
require_once('conf/connection.php');

// Récupération des données
try {
  // On envois la requète
  $select = $connection->query("SELECT * FROM createurs");

  // On indique que nous utiliserons les résultats en tant qu'objet
  $select->setFetchMode(PDO::FETCH_OBJ);

  // Nous traitons les résultats en boucle
  while( $enregistrement = $select->fetch() )
  {
    // Affichage des enregistrements
    echo '<h1>', $enregistrement->nom, ' ', $enregistrement->prenom, '</h1>';
  }
} catch ( Exception $e ) {
  echo "Une erreur est survenue lors de la récupération des créateurs";
}
```

3.4. UPDATE et DELETE

La méthode query est utilisée pour faire des **SELECT**, pour réaliser des **UPDATE/INSERT** ou des **DELETE**, nous utiliserons plutôt la méthode **exec**.

Notez que je donne l'utilisation de cette méthode à titre indicatif sans m'étendre comme j'aime le faire. En effet, dans la partie suivante, j'aborderai les requêtes préparées qui sont simplement incontournables pour les **UPDATE** / **DELETE**.

```php
// Connection au serveur
require_once('conf/connection.php');

// Suppression de données
try {
  // On envois la requête (Suppression arbitraire de 2 enregistrements)
  $nombreDeSuppression = $connection->exec("DELETE FROM createurs LIMIT 2");

  // On affiche le nombre d'enregistrements supprimés
  echo $nombreDeSuppression, " ont été supprimé.";

} catch ( Exception $e ) {
  echo "Une erreur est survenue lors de la suppression";
}
```

A chaque actualisation, vous verrez s'afficher 2, et à un moment 1 (si vous avez un nombre impair d'enregistrements) et enfin 0. Vous vennez de vider la table :) Pensez à réimporter les données avant de passer à la suite.

4. Les requètes préparées

Voilà la toute puissance de PDO... Les fameuses requètes préparées. Réutilisables, plus sûr, plus classe.

Le principe est simple, vous n'allez pas exécuter directement les requètes, vous allez définir un **gabarit de requète** avec des "options".

Par exemple plutot que de dire "Supprimer l'enregistrement numéro 1", vous aller préparer la requète avant "Supprimer l'enregistrement numéro %onVerraApres%", ensuite dès que vous avez besoin de supprimer un enregistrement avec la requète préparée, vous aller executer la requète préparée en lui précisant au moment de l'execution quel sont les valeurs à utiliser.

4.1. Paramètres mystères

Il y a 2 façon de définir une requète préparée, la première utilise le caractère **?** pour transmettre des valeurs au moment de l'execution, concrètement ça ressemble à ça :

```php
// Préparation de la requète
$selectionPrepa = $connection->prepare('SELECT * FROM createurs WHERE id=? LIMIT 1');
try {
  // On envois la requète
  $selectionPrepa->execute(array(1));

  // Traitement
  if( $enregistrement = $selectionPrepa->fetch(PDO::FETCH_OBJ)){
    echo '<h1>', $enregistrement->nom, ' ', $enregistrement->prenom,  '</h1>';
  }
} catch( Exception $e ){
  echo 'Erreur de suppression : ', $e->getMessage();
}
```

Ce système permet de **réutiliser** la requète préparée par la suite en modifiant la valeur utilisée lors de l'execution par une autre.

On peut biensûr définir plusieurs paramètres mystères :

```php
$selectionPrepa = $connection->prepare('SELECT * FROM createurs WHERE YEAR(date_naiss)=? AND nationalit
try {
  // On envois la requète
  $selectionPrepa->execute(array(1925, 'fr'));

  // Traitement
  while( $enregistrement = $selectionPrepa->fetch(PDO::FETCH_OBJ)){
    echo '<h1>', $enregistrement->nom, ' ', $enregistrement->prenom,  '</h1>';
  }

} catch( Exception $e ){
  echo 'Erreur de requète : ', $e->getMessage();
}
```

Les paramètres sont utilisées dans l'ordre d'apparition dans la requète.

4.2. Paramètres nommés

Supposons maintenant que nous ayons beaucoup de paramètres (ou des paramètres que nous voudrions utiliser plusieurs fois). Vous pouvez alors utiliser les paramètres nommées.

Premier cas, un paramètre utilisé plusieurs fois :

```php
$selectionPrepa = $connection->prepare('SELECT * FROM createurs WHERE nom LIKE :search OR prenom LI
try {
  // On envois la requète
  $selectionPrepa->execute(array('search'=>'%gi%'));

  // Traitement
```

```
  while( $enregistrement = $selectionPrepa->fetch(PDO::FETCH_OBJ)){
    echo '<h1>', $enregistrement->nom, ' ', $enregistrement->prenom,  '</h1>';
  }

} catch( Exception $e ){
  echo 'Erreur de requête : ', $e->getMessage();
}
```

Autre cas, beaucoup de paramètres :

```
$insert = $connection->prepare('INSERT INTO createurs VALUES(
NULL, :nom, :prenom, :date_naiss, :date_mort, :nationalite, :pseudo)');
try {
  // On envois la requète
  $success = $insert->execute(array(
    'nom'=>'Dus',
    'prenom'=>'Jean-Claude',
    'date_naiss'=>date('Y-m-d'),
    'date_mort'=>NULL,
    'nationalite'=>'fr',
    'pseudo'=>NULL
  ));

  if( $success ) {
    echo "Enregistrement réussi";
  }
} catch( Exception $e ){
  echo 'Erreur de requête : ', $e->getMessage();
}
```

Au moment de l'execution, vous transmettez un tableau indexé avec les clefs portant le nom des paramètres nommés (sans le 2 points).

Les données transmise peuvent provenir de variables :P

PDO se chargera automatiquement des échapements et des formats de transmission. Mais vous pouvez allé plus loin en utilisant la méthode bindParam :)

4.3. Paramètres Bindés

La méthode **bindParam()** permet de remplir la requète préparée avec le contenu d'une variable, on a également la possibilité de préciser un type de donnée et une taille.

Important : L'utilisation de bindParam est très puissante, vous pouvez préparer votre requète bien avant l'envois et la rééxécuter plusieurs fois, si des changements ont lieu dans les variables bindées, elle seront prises en compte au moment de l'exécution.

```
$insert = $connection->prepare('INSERT INTO createurs VALUES(
NULL, :nom, :prenom, :date_naiss, :date_mort, :nationalite, :pseudo)');
try {
  // On rempli les paramètres
  $insert->bindParam(':nom', $nom, PDO::PARAM_STR, 100);
  $insert->bindParam(':prenom', $prenom, PDO::PARAM_STR, 100);
  $insert->bindParam(':date_naiss', date('Y-m-d'));
  $insert->bindParam(':nationalite, $nationalite, PDO::PARAM_STR, 2);
  $insert->bindParam(':pseudo', $pseudo, PDO::PARAM_STR);

  // On exécute
  $insert->execute();

  if( $success ) {
    echo "Enregistrement réussi";
  }
} catch( Exception $e ){
  echo 'Erreur de requête : ', $e->getMessage();
}
```

La méthode **bindParam()** a une petite soeur : **bindValue()**. Cette méthode est une version light de **bindParam()** qui présente l'avantage de proposer de contrôle des données transmises à la requète préparée.

5. Ressources complémentaires :

http://php.developpez.com/faq/?page=pdo : une FAC en français dédiée à PDO

http://www.siteduzero.com/tutoriel-3-34790-pdo-interface-d-acces-aux-bdd.html : Un tutoriel qui présente PDO de façon très simple.