XIII) Gérer une DB avec PDO

PHP avancé

Gérer une DB avec PDO

Il existe de très nombreux SGBD sur le marché et autant de méthodes spécifiques permettant de manipuler et gérer les bases de données. Heureusement est apparue avec la version 5.1 de PHP une extension désormais systématiquement installée et bien pratique : PDO (PHP Data Object).

PDO nous permet de gérer pour nous la plupart des connecteurs possibles existant (IBM, Informix, MySQL, ODBC et DB2, Postgre SQL, SQLite Oracle...). Certains sont encore en développement mais à terme devraient être implémentés. De plus il a été écrit en C, ce qui le rend rapide et efficace. Et enfin, il est orienté objet, facilitant sa manipulation et son utilisabilité.

Il est facile de vérifier si PDO est installé sur un serveur et quels sont les connecteurs disponibles avec respectivement les fonctions : **phpinfo()** et **PDO::getAvailableDrivers()**.

PHP avancé

Gérer une DB avec PDO

XIII.1) Les étapes

Pour toute manipulation, 5 étapes sont à respecter :

- Connexion
- Sélection de la DB
- Requête
- Exploitation des résultats
- Déconnexion

Note : Lorsque le moteur PHP arrive à la fin de l'exécution du script, il ferme automatiquement les éventuelles connexions, rendant la dernière étape optionnelle. Il peut cependant être judicieux de fermer la connexion dès qu'elle n'est plus nécessaire afin d'économiser des ressources. (Tout ce qui touche aux bases de données est en général extrêmement gourmand en terme de ressources pour le serveur)

PHP avancé

Gérer une DB avec PDO

XIII.2) Connexion

Avant de travailler avec un serveur de gestion de base de données comme MySQL, il faut ouvrir une connexion. C'est par elle que PHP et MySQL communiqueront.

Ouvrir une connexion se fait automatiquement lorsqu'on instancie un objet PDO, nécessaire à toutes les manipulations à venir.

Syntaxe

mon_objet = new PDO(DSN, utilisateur, mot_de_passe)

Le **DSN** permet de décrire la base de donnée à laquelle nous allons nous connecter et son type. Chaque SGBD a ses propres paramètres. Nous ne verrons que le DSN MySQL. Bien évidemment, un **utilisateur** et un **mot_de_passe** doivent être renseignés.

PHP avancé

Gérer une DB avec PDO

Exemple :

```
// préparation des informations nécessaires
$dsn = 'mysql:host=localhost;dbname=test';
$utilisateur = 'pascal';
$mdp = 'admin';

// création d'un objet PDO et connexion
$connDB = new PDO('mysql:host=localhost;dbname=test', $utilisateur, $mdp);
```

Dans le DSN, on retrouve en premier le nom du "pilote" suivi de 2 points. (ici **mysql:**) Viennent ensuite :

host	Nom ou adresse IP de l'hôte ("localhost" en local)
dbname	Le nom de la base de donnée
port	Le port TCP/IP (facultatif)
unix_socket	Adresse du socket unix (facultatif)

PHP avancé

Gérer une DB avec PDO

Etant donné que nous allons certainement utiliser une connexion à notre DB sur plusieurs pages différentes, il serait judicieux de créer un fichier de connexion et d'y faire appel à chaque fois que nous en avons besoin. De plus, si les données permettant de configurer notre accès changent, il nous suffira de mettre à jour ce simple fichier.

Voici un petit exemple reprenant le code précédent. Ici nous créons des constantes pour les paramètres à passer plutôt que des variables.

```
// préparation des informations nécessaires
define('DSN', 'mysql:host=localhost;dbname=test');
define('USER', 'pascal');
define('MDP', 'admin');

// création d'un objet PDO et connexion
$connDB = new PDO(DSN, USER, MDP);
```

PHP avancé

Gérer une DB avec PDO

Nous allons encore améliorer ce code en ajoutant une structure de test de type **try** / **catch** qui va nous permettre de récupérer le code d'erreur si la connexion devait échouer.

```
// préparation des informations nécessaires
define('DSN', 'mysql:host=localhost;dbname=test');
define('USER', 'pascal');
define('MDP', 'admin');

// création d'un objet PDO et connexion
try {
    $connDB = new PDO(DSN, USER, MDP);
} catch (Exception $erreur) {
    echo "Outch, erreur : ".$erreur -> getMessage();
    exit();
}
```

Note : Nous avons rajouté **exit()** en cas d'erreur pour arrêter l'exécution du script et ne pas enchaîner sur l'éventuel code qui pourrait suivre.

XIII.3) Déconnexion

Pour se déconnecter, il suffit de détruire l'objet PDO que nous avons crée. Affecter la valeur **null** à cet objet suffira.

```
// fermeture de la connexion
if ($connDB) {
    $connDB = NULL;
}
```

Note : Ici nous vérifions l'existence de notre objet avant de changer sa valeur, évitant ainsi la génération d'une erreur si la connexion avait déjà été précédemment fermée.

XIII.4) Effectuer une requête

Une fois la connexion établie, nous allons pouvoir communiquer avec notre DB pour modifier, créer, supprimer ou lire des données. Il n'y à pas de fonctions d'écriture/lecture comme avec les fichiers, tout passe par le langage SQL.

XIII.4.1) Préparer sa requête

Avant d'envoyer notre requête il est préférable de la préparer et de la "ranger" dans une variable. Ce n'est pas nécessaire mais fortement conseillé pour conserver une bonne segmentation/lisibilité de votre code.

```
// préparation de la requête
$requete = "SELECT * FROM stagiaires ORDER BY sta_nom";
```

PHP avancé

Gérer une DB avec PDO

XIII.4.2) Envoyer sa requête

Il existe 2 fonctions permettant d'envoyer notre requête au serveur.
Si nous faisons une sélection et avons donc besoin de récupérer des données en retour, nous utiliserons **query()**. Dans tous les autres cas, nous utiliserons **exec()**.

XIII.4.2.1) exec()

Cette fonction nous renvoie quand même une information : le nombre de lignes qui ont été modifiées suite à l'exécution de la requête.

```
// préparation de la requête
$requete = "DELETE FROM stagiaires WHERE sta_nom='Toto'";
// envoi de la requête au serveur et récupération du résultat
$resultat = $connDB->exec($requete);
// affichage du résultat
echo $resultat;
```

PHP avancé

Gérer une DB avec PDO

Attention : Si aucune modification n'a été faite suite à la requête, le résultat renvoyé sera **0**. Dans le cas où la requête a générée une erreur, c'est **false** qui est renvoyé. Or avec un opérateur de comparaison classique (**==**) ces 2 valeurs sont identiques !
Il nous faut donc utiliser l'opérateur de comparaison de type (**===**).

Exemple :

```
// préparation de la requête
$requete = "DELETE FROM stagiaires WHERE sta_nom='Toto'";
// envoi de la requête au serveur et récupération du résultat
$resultat = $connDB->exec($requete);
// interprétation du résultat
if ($resultat===false) {
    $reponse = "Problème dans la requête";
} else if ($resultat==0) {
    $reponse = "Aucune modification dans la DB";
} else {
    $reponse = "Requête effectuée, ".$resultat." lignes ont été affectées";
}
```

PHP avancé
Gérer une DB avec PDO

XIII.4.2.2) query()

Pour toutes les requêtes renvoyant des données autres que le nombre d'enregistrements concernés, nous allons utiliser **query()**.
Cette fonction ne renvoit pas directement les données prêtes à être affichées mais un sous objet PDO qui dispose de 2 méthodes permettant de manipuler ces données.

Méthode **fetchAll()** : Toutes les données sont rangées dans un tableau et le SGBD est libéré. Avantage : nous avons toutes les données d'un coup et pouvons travailler dessus. Inconvénient : toutes les données sont présentes en mémoire.

Méthode **fetch()** : Les résultats sont lus de manière séquentielle, un par un. Avantage : parfait pour traiter des résultats très nombreux sans surcharger la mémoire. Inconvénient : on ne peut pas faire d'autres requêtes pendant le traitement de ces résultats.

PHP avancé

Gérer une DB avec PDO

Exemples :

```
// extraction de données avec fetchAll()
$requete = "SELECT * FROM stagiaires";
$resultat = $connDB->query($requete);
$donnees = $resultat->fetchAll();
foreach($donnees as $ligne) {
    echo $ligne['sta_prenom']." ".$ligne['sta_nom']."<br>";
}
// extraction de données avec fetch()
$requete = "SELECT * FROM stagiaires";
$resultat = $connDB->query($requete);
while ($ligne = $resultat->fetch()) {
    echo $ligne['sta_prenom']." ".$ligne['sta_nom']."<br>";
}
```

Notes : Ces 2 extraits de code vont donner exactement le même résultat pour l'utilisateur. Si l'on ne souhaite manipuler que les noms des champs, on passera **PDO::FETCH_ASSOC** en paramètre pour optimiser les requêtes. (**PDO::FETCH_BOTH** par défaut)

PHP avancé
Gérer une DB avec PDO

XIII.4.3) Sécuriser sa requête

Le SQL a ses propres caractères spéciaux et délimiteurs, rendant par là possible les mauvaises interprétations ou, pire, les injections.
Il faudra donc éviter au possible d'insérer directement des données sans les avoir au préalable vérifié/formaté.

XIII.4.3.1) quote()

Une méthode de notre objet PDO permet d'échapper automatiquement tous les caractères génants et ajouter des délimiteurs à notre requête.

```
// texte original
$texte_brut = "L'insertion dans la DB peut être problématique";
// sécurisation du texte pour le SQL
$texte_ok = $connDB->quote($texte_brut);
// Le résultat sera : 'L\insertion dans la DB peut être problématique'
```

Note : La fonction s'est chargée de remplacer les quotes.

XIII.5) Méthodes avancées

XIII.5.1) Requêtes préparées

Lorsqu'on exécute une requête, la base de données va l'analyser, la compiler, l'optimiser puis l'exécuter. Le but des requêtes préparées est de ne pas répéter toutes ces actions lorsqu'on exécute des requêtes identiques. Nous allons donc créer un "modèle", qui sera supprimé à la fermeture de la connexion. Etapes de la préparation :

- Création de la requête et substitution des valeurs
- Lecture des données
- Exécution de la requête
- Exploitation des résultats

La première étape n'est pas répétée à chaque exécution, rendant cette méthode avantageuse si elles sont nombreuses.

PHP avancé

Gérer une DB avec PDO

XIII.5.1.1 Valeurs passées dans un tableau

Remplacement des valeurs par des paramètres nommés. Ceux-ci doivent être précédés de 2 points (**:**)

```
$requete="INSERT INTO stagiaires (sta_prenom,sta_nom) VALUES (:unPrenom,:unNom)";
```

Préparation de la requête

```
$modele = $connDB -> prepare($requete);
```

Exécution en associant des valeurs aux paramètres nommés

```
$modele -> execute(
    array(
        ':unPrenom' => 'Cémoi',
        ':unNom' => 'Toto'
    )
);
```

Cette méthode est un peu lourde, voyons les suivantes...

PHP avancé

Gérer une DB avec PDO

XIII.5.1.2 Valeurs définies par BindValue

Le principe est de lier une variable ou une valeur à un paramètre nommé sans passer par un tableau. Les premières étapes ne changent pas.

Préparation de la requête

```
$requete="INSERT INTO stagiaires (sta_prenom,sta_nom) VALUES (:unPrenom,:unNom)";
$modele=$connDB -> prepare($requete);
```

Association des valeurs avec **bindValue**

```
$modele -> bindValue('unPrenom' , 'Cémoi');
$modele -> bindValue('unNom' , 'Toto');
```

Exécution de la requête

```
$modele -> execute();
```

PHP avancé

Gérer une DB avec PDO

XIII.5.1.3 Valeurs définies par BindParam

Une variante où les valeurs ne sont pas passées directement, mais au sein de variables.

Préparation de la requête

```
$requete="INSERT INTO stagiaires (sta_prenom,sta_nom) VALUES (:unPrenom,:unNom)";
$modele=$connDB -> prepare($requete);
```

Association des valeurs avec **bindParam**

```
$modele->bindParam('unPrenom', $prenom);
$modele->bindParam('unNom', $nom);
```

Exécution de la requête

```
$prenom = 'Cémoi';
$nom = 'Toto';
$modele -> execute();
```

PHP avancé
Gérer une DB avec PDO

XIII.5.2) Transactions

Une transaction est un bloc d'instructions que l'on souhaite exécuter de manière unie. L'ensemble doit être réalisé sans erreur sans quoi toutes les opérations sont annulée (notion d'atomicité) ce qui ne serait pas le cas si nous exécution classiquement nos opérations, de manière séquentielle.

Etapes d'une transaction :

- Déclaration de début de transaction
- Ecriture de toutes les opérations souhaités
- Exécution de la transaction

Attention : Tous les moteurs de stockage ne gèrent pas les transactions. (InnoDB est souvent conseillé si l'on souhaite utiliser des opérations assez évoluées telles que celle-ci)

PHP avancé

Gérer une DB avec PDO

Voici un exemple où 2 opérations sont demandées :

```
// déclaration de transaction
$connDB -> beginTransaction();
// première opération
$requete = "INSERT INTO stagiaires (sta_prenom,sta_nom) VALUES ('Cémoi', 'Toto')";
$connDB -> exec($requete);
// deuxième opération
$requete = "INSERT INTO stagiaires (sta_prenom,sta_nom) VALUES ('Cétoi', 'Titi')";
$connDB -> exec($requete);
// exécution de la transaction (et annulation en cas d'échec)
If (!$connDB-> commit()) {
     $connDB->rollBack();
     echo "Echec des opérations";
} else {
    echo "Opérations exécutées avec succès";
}
```

Note : Le fait de déclarer une transaction empêche les exécutions séquentielles (mode *auto-commit*). Une fois la transaction finie, ce mode est remis par défaut.

PHP avancé

Gérer une DB avec PDO

XIII.5.3) Gestion des erreurs

D'une base de donnée à une autre, la gestion des erreurs peut être inexistante comme extrêmement poussée. PDO là encore va nous aider de manière transparente et nous permettre d'avoir des messages d'erreurs unifiés. Il existe 3 modes de gestion des erreurs :

ERRMODE_SILENT	Pas d'affichage des erreurs (par défaut)
ERRMODE_WARNING	Mode d'erreur classique
ERRMODE_EXCEPTION	Utilisation des exceptions

On utilise la méthode **setAttribute()** de notre objet PDO pour modifier le mode.

Exemple pour le mode "exception" :

```
$connDB -> setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
```

PHP avancé

Gérer une DB avec PDO

Plutôt que de laisser les erreurs s'afficher ou non à l'écran, là encore il est préférable de gérer cela avec une structure adaptée **try** / **catch**. Modifions notre précédent exemple de transaction :

```
// déclaration de transaction
$connDB -> beginTransaction();
// structure try / catch
try {
    // première opération
    $requete = "INSERT INTO stagiaires (sta_prenom,sta_nom) VALUES ('Cémoi', 'Toto')";
    $connDB -> exec($requete);
    // deuxième opération
    $requete = "INSERT INTO stagiaires (sta_prenom,sta_nom) VALUES ('Cétoi', 'Titi')";
    $connDB -> exec($requete);
    // exécution de la transaction (et annulation en cas d'échec)
    $connDB-> commit();
    echo "Opérations exécutées avec succès";
} catch (Exception $erreur) {
     $connDB->rollBack();
    echo "Echec des opérations. Message d'erreur : ".$erreur -> getMessage();
}
```