Annexes - Fonctions PDO-MySQL

Annexe 1 Liste des fonctions

Extrait de la documentation officielle accessible sur le site

bool **PDO::beginTransaction** (void)	désactive le mode *autocommit*.
bool **PDO::commit** (void)	valide une transaction
mixed **PDO::errorCode** (void)	retourne un SQLSTATE, un identifiant alphanumérique de cinq caractères défini dans le standard AINSI SQL.
array **PDO::errorInfo** (void)	retourne un tableau contenant des informations sur l'erreur survenu lors de la dernière opération exécutée par ce gestionnaire de base de données. Le tableau contient les [éléments] suivants : [0] Code d'erreur SQLSTATE (un identifiant alphanumérique de cinq caractères défini dans le standard ANSI SQL). [1] Code d'erreur spécifique au driver. [2] Message d'erreur spécifique au driver.
int **PDO::exec** (string `$statement`)	exécute une requête SQL dans un appel d'une seule fonction, retourne le nombre de lignes affectées par la requête. **PDO::exec()** ne retourne pas de résultat pour une requête SELECT. Pour une requête SELECT dont vous auriez besoin une seule fois dans le programme, utilisez plutôt la fonction PDO::query(). Pour une requête dont vous auriez besoin plusieurs fois, préparez un objet PDOStatement avec la fonction PDO::prepare() et exécutez la requête avec la fonction PDOStatement::execute().
mixed **PDO::getAttribute** (int `$attribute`)	Cette fonction retourne la valeur d'un attribut d'une connexion à une base de données.
bool **PDO::inTransaction** (void)	Vérifie si une transaction est actuellement active dans le driver. Cette méthode ne fonctionne que pour les drivers de base de données qui supportent les transactions.
string **PDO::lastInsertId** ([string `$name` = **`NULL`**])	Retourne l'identifiant de la dernière ligne insérée, ou la dernière valeur d'une séquence d'objets
PDOStatement **PDO::prepare** (string `$statement` [, array `$driver_options` = array()])	Prépare une requête SQL à être exécutée par la méthode PDOStatement::execute(). La requête SQL peut contenir zéro ou plusieurs noms (*:nom*) ou marqueurs (*?*) pour lesquels les valeurs réelles seront substituées lorsque la requête sera exécutée.

PDOStatement **PDO::query** (string `$statement`)	exécute une requête SQL en appelant une seule fonction, retourne le jeu de résultats (s'il y en a). Pour une requête que vous devez exécuter plusieurs fois, vous réaliserez de meilleurs performances si vous préparez l'objet PDOStatement en utilisant la fonction PDO::prepare() et exécutez la requête via plusieurs appels à la fonction PDOStatement::execute(). **PDO::query()** retourne un objet PDOStatement, ou **`FALSE`** si une erreur survient.
string **PDO::quote** (string `$string` [, int `$parameter_type` = PDO::PARAM_STR])	place des guillemets simples autour d'une chaîne d'entrée, si nécessaire et protège les caractères spéciaux présents dans la chaîne d'entrée, en utilisant le style de protection approprié au pilote courant. Si vous utilisez cette fonction pour construire des requêtes SQL, vous êtes *vivement* invités à utiliser PDO::prepare() pour préparer les requêtes SQL avec des paramètres liés au lieu d'utiliser **PDO::quote()**
bool **PDO::rollBack** (void)	Annule la transaction courante, initiée par la fonction PDO::beginTransaction(). Une exception PDOException sera lancée si aucune transaction n'est active.
bool **PDO::setAttribute** (int `$attribute` , mixed `$value`)	Configure un attribut du gestionnaire de base de données.
bool **PDOStatement::bindColumn** (mixed `$column` , mixed `&$param` [, int `$type` [, int `$maxlen` [, mixed `$driverdata`]]])	fait en sorte qu'une variable PHP soit liée à une colonne données dans le jeu de résultats dans une requête.
bool **PDOStatement::closeCursor** (void)	libère la connexion du serveur, permettant ainsi à d'autres requêtes SQL d'être exécutées, mais quitte la requête, permettant ainsi qu'elle soit de nouveau exécutée.
int **PDOStatement::columnCount** (void)	retourne le nombre de colonnes dans le jeu de résultats représenté par l'objet PDOStatement.
bool **PDOStatement::execute** ([array `$input_parameters`])	Exécute une requête préparée.
mixed **PDOStatement::fetch** ([int `$fetch_style` [, int `$cursor_orientation` = PDO::FETCH_ORI_NEXT [, int `$cursor_offset` = 0]]])	Récupère une ligne depuis un jeu de résultats associé à l'objet *PDOStatement*. Le paramètre *`fetch_style`* détermine la façon dont PDO retourne la ligne.
array **PDOStatement::fetchAll** ([int `$fetch_style` [, mixed	Retourne un tableau contenant toutes les lignes du jeu d'enregistrements

`$fetch_argument` [, array `$ctor_args` = array()]]])	
string **PDOStatement::fetchColumn** ([int `$column_number` = 0])	Retourne une colonne depuis la ligne suivante d'un jeu de résultats ou **`FALSE`** s'il n'y a plus de ligne.
mixed **PDOStatement::fetchObject** ([string `$class_name` = "stdClass" [, array `$ctor_args`]])	Récupère la prochaine ligne et la retourne en tant qu'objet.
bool **PDOStatement::nextRowset** (void)	Quelques bases de données supportent les procédures stockées qui retournent plus d'une ligne de résultats (aussi connu comme des jeux de résultats). **PDOStatement::nextRowset()** vous permet d'accéder à la seconde et suivantes lignes de résultats associées avec l'objet PDOStatement. Chaque ligne de résultats a des jeux différents de colonnes depuis la ligne de résultats.
int **PDOStatement::rowCount** (void)	retourne le nombre de lignes affectées par la dernière requête DELETE, INSERT ou UPDATE exécutée

Exemple de requêtes préparées

Si vous exécutez une requête une seule fois, la méthode query() convient parfaitement. Si vous avez besoin d'exécuter plusieurs fois la même requête (et c'est bien souvent le cas), il est fortement conseillé d'utiliser des requêtes préparées (méthodes : prepare() et execute()). Voici quelques exemples de leur utilisation :

```
    /* Exécution de requêtes préparées */

    // requête préparée avec marqueur nominatif
    print "<h1>requête préparée avec marqueur nominatif</h1>";

    $str_requete = "SELECT id,titre,webmaster FROM liens WHERE id>=:idMin
and id<=:idMax ";

    // preparation de la requête
    $pstmt = $db->prepare($str_requete);

    // 1) Exécution de la requête en renseignant les marqueurs
    print "<h2>1) Exécution de la requête en renseignant les
marqueurs</h2>";
```

```
    $pstmt->execute(array('idMin'=>1,'idMax'=>15));
```

```
    $lien = $pstmt->fetch(PDO::FETCH_OBJ);
    print $lien->id . " - " . $lien->titre . "<br />";
    $result->closeCursor();

    $str_requete = "SELECT id,titre,webmaster FROM liens WHERE id>=:id and
titre LIKE :titre ";

    // preparation de la requête
    $pstmt = $db->prepare($str_requete);

    // 2) Exécution de la requête en liant les variables
    print "<h2>2) Exécution de la requête en liant les variables</h2>";
    $id = 4;
    $titre = "%PHP%";
    $pstmt->bindParam(':id', $id, PDO::PARAM_INT);
    $pstmt->bindParam(':titre', $titre, PDO::PARAM_STR);
    $pstmt->execute();

    $lien = $pstmt->fetch(PDO::FETCH_OBJ);
    print $lien->id . " - " . $lien->titre . "<br />";
    $result->closeCursor();

    // requête préparée avec marqueur interrogatif
    print "<h1>requête préparée avec marqueur interrogatif</h1>";

    $str_requete = "SELECT id,titre,webmaster FROM liens WHERE id>=? and id
<=?";

    // preparation de la requête
    $pstmt = $db->prepare($str_requete);

    // 1) Exécution de la requête en renseignant les marqueurs
    print "<h2>1) Exécution de la requête en renseignant les
marqueurs</h2>";
    $pstmt->execute(array(1,15));

    while ($lien = $pstmt->fetch(PDO::FETCH_OBJ)){
        print $lien->id . " - " . $lien->titre . "<br />";
    }
    $result->closeCursor();
```

Elles sont plus rapides à s'exécuter et de plus elles protègent naturellement le code de l'injection de SQL. Même si du SQL est envoyé dans les marqueurs, la requête étant déjà compilée, il sera vu comme une chaîne de caractères.