OFPPT

Utiliser une base de données

- L'extension mysql
- L'extention mysqli
- L'extention PDO

Connexion à une base de données

La connexion au serveur MySQL s'effectue par la fonction msql_connect().

mysql_connect ('hôte', 'login', 'mot de passe')

retourne TRUE si la connexion est réussie et FALSE sinon.

mysql_select_db("base")

La fonction retourne aussi TRUE en cas de succès et FALSE en cas d'erreur.

Sélection des données

$result = **mysql_query**("SELECT id,nom,email from liste");
Toutefois, la variable **$result** n'est pas exploitable telle quelle.

la fonction **mysql_fetch_array()** Retourne **un tableau associatif** qui représente tous les champs d'une rangée dans le résultat. Chaque appel récupère la prochaine rangée et ce jusqu'à ce qu'il n'y en ait plus.

Chaque valeur de champ est stockée de deux façons: elle est **indexée** par un numéro qui commence par '**0**', et indexée par le nom **du champ**.

```
while ( $row = mysql_fetch_array($result)){
echo $row[id].' - '.$row[nom].' - '.$row[email] ;
}
```

Sélection des données

- **mysql_fetch_row()** Comme **mysql_fetch_array()** mais indexée uniquement par le numéro d'ordre du champ. C'est la méthode la plus rapide pour obtenir des résultats à partir d'une requête.

- **mysql_fetch_assoc()** Comme **mysql_fetch_array()** mais indexée uniquement par le nom du champ.

- **mysql_fetch_object()** semblable au deux premières. Mais à la place, elle retourne un **objet** pour chaque ligne.

Sélection des données

- **mysql_num_rows()** Retourne le nombre de rangées dans un résultat.

- **mysql_insert_id()** Après l'insertion d'un nouvel enregistrement avec un champ **auto_increment**, la fonction **mysql_insert_id()** retourne l'**ID** qui vient d'être affecté au nouvel enregistrement.

Gestion des erreurs

- **mysql_errno()** Retourne le numéro de l'erreur générée lors de la dernière action sur la base de donnée.

- **mysql_error()** Retourne la description textuelle d'une erreur générée par la dernière action sur une base de données.

Libération de ressources

PHP libère la mémoire allouée et fermera automatiquement la connexion lorsque le script arrivera à la fin. Cependant, il est conseillé de libérer et fermer la connexion, une fois, les requêtes sont exécutées :

- **mysql_free_result()** Libère la mémoire associée au résultat spécifié.
- **mysql_close()** ferme la connexion;

exemple

```
<?php
$hote = 'localhost';
$user = 'cyril';
$pass = 'motdepasse';
$base = 'publication';
// Étape 1 : connexion
$link = mysql_connect($hote, $user, $pass);
if (!$link) { die('Could not connect: ' . mysql_error());}
// On choisit la base ‘publication’
mysql_select_db($base);
// Étape 2 : création et exécution de la requête SQL
$pseudo = 'Cyril';
// On échappe notre variable :
$pseudo = mysql_real_escape_string($pseudo);
$query = "SELECT * FROM rmq WHERE pseudo = '$pseudo'";
// On exécute la requête SQL
$result = mysql_query($query);
// Étape 3 : traitement du résultat
while ($row = mysql_fetch_assoc($result)) {
  echo $row['pseudo'];
  echo $row['titre'];
  echo $row['texte'];
}
// Étape 4 : libération des ressources et fermeture de la connexion
mysql_free_result($result);
mysql_close($link);
?>
```

Extension php pour des SGBD

- MySQL
- mysqli — Extension mysqli
- Mssql — Serveur MS SQL
- OCI8 — Oracle OCI8
- PostgreSQL
- DB++
- filePro
- Firebird/InterBase
- IBM DB2 — Fonctions IBM DB2, Cloudscape et Apache Derby
- Informix
- Ingres — Ingres DBMS, EDBC, et Enterprise Access
- Mongo — Driver natif MongoDB

……..

L'extension MySQLI

- l'extension MySQL *améliorée* (i pour *improved* en anglais), a été développée pour exploiter les nouvelles fonctionnalités des systèmes MySQL version 4.1.3 et plus récent.

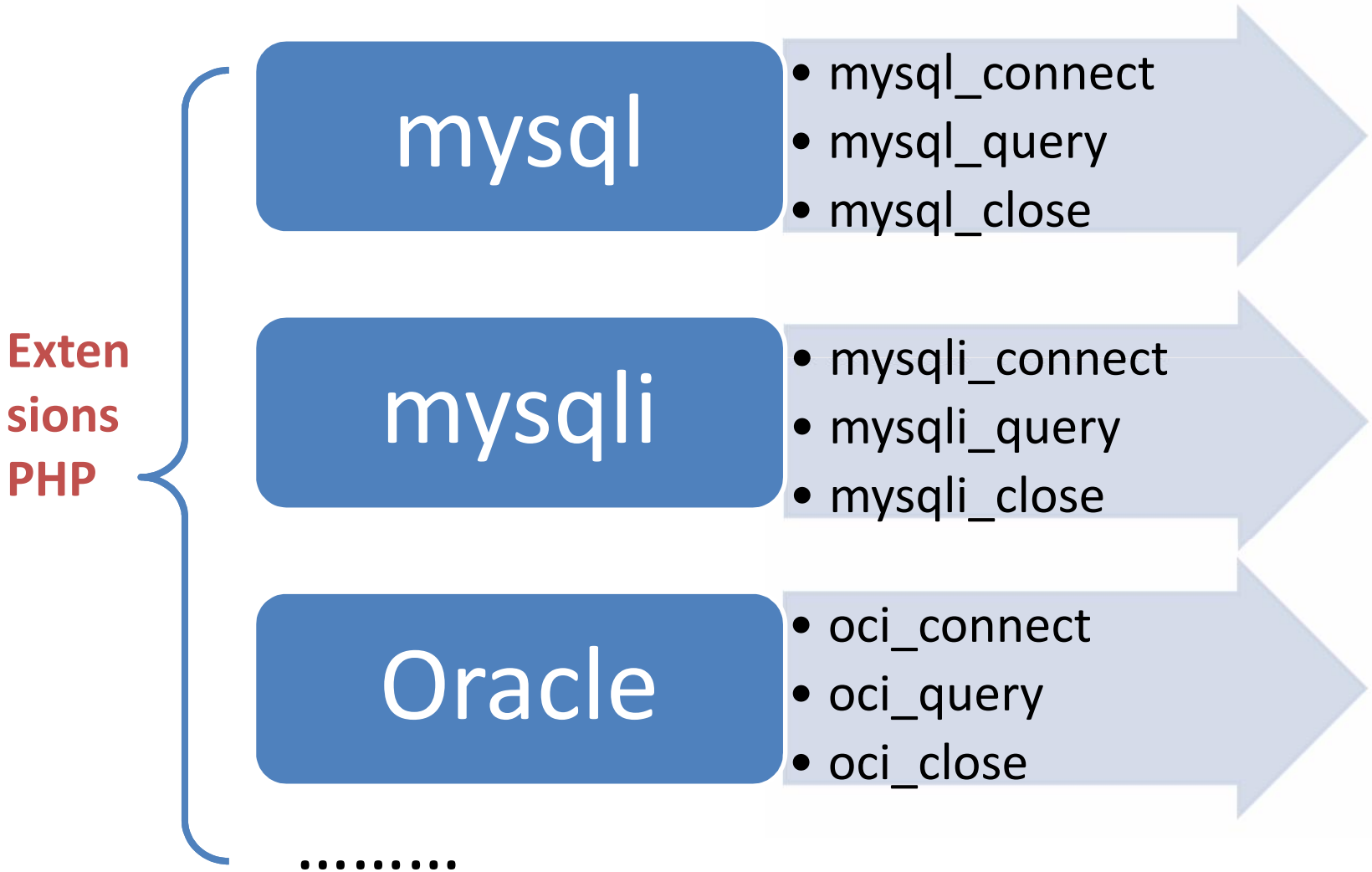
- L'extension *mysqli* est inclue dans **PHP depuis les versions 5**.

- l'extension propose en plus de l'interface procédurale, une **interface orientée objet.**

L'extension MySQLI

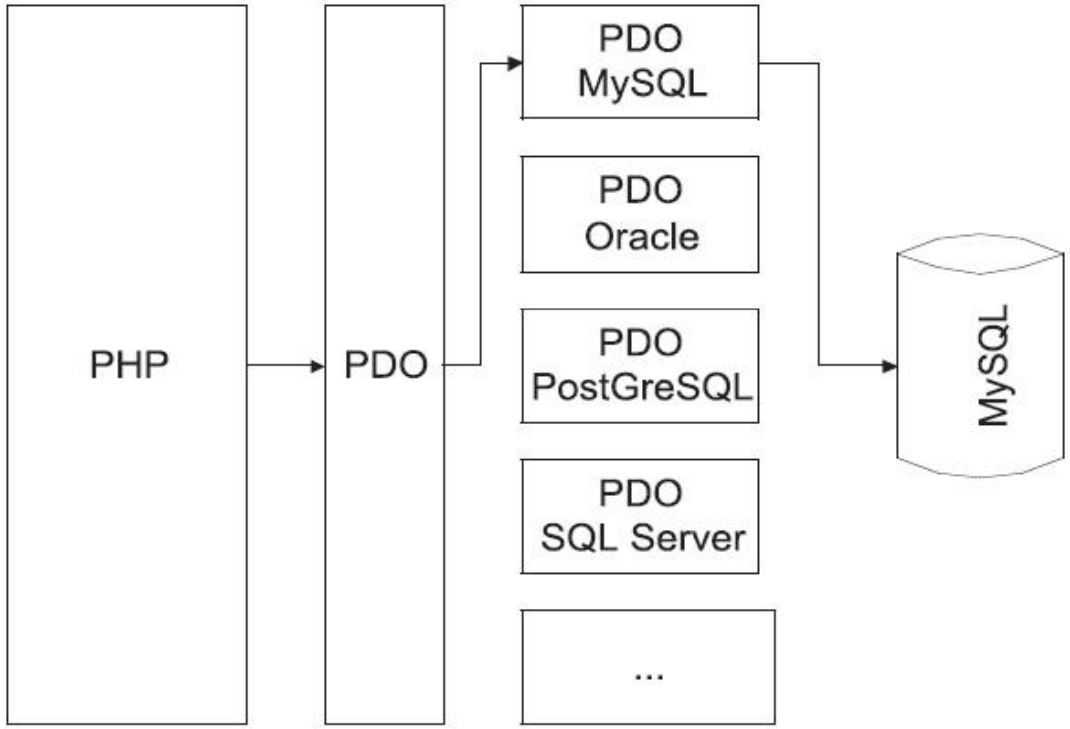
```
$mysqli = new mysqli("localhost", "my_user", "my_password", "t
est");
 $mysqli->select_db("world");
if ($result = $mysqli->query("SELECT * FROM client WHERE
…")) {
    $row = $result->fetch_row();
    echo "Le nom du client sélectionné est", $row[0]);
    $result->close();
}



$mysqli->close();
```

Extensions PHP



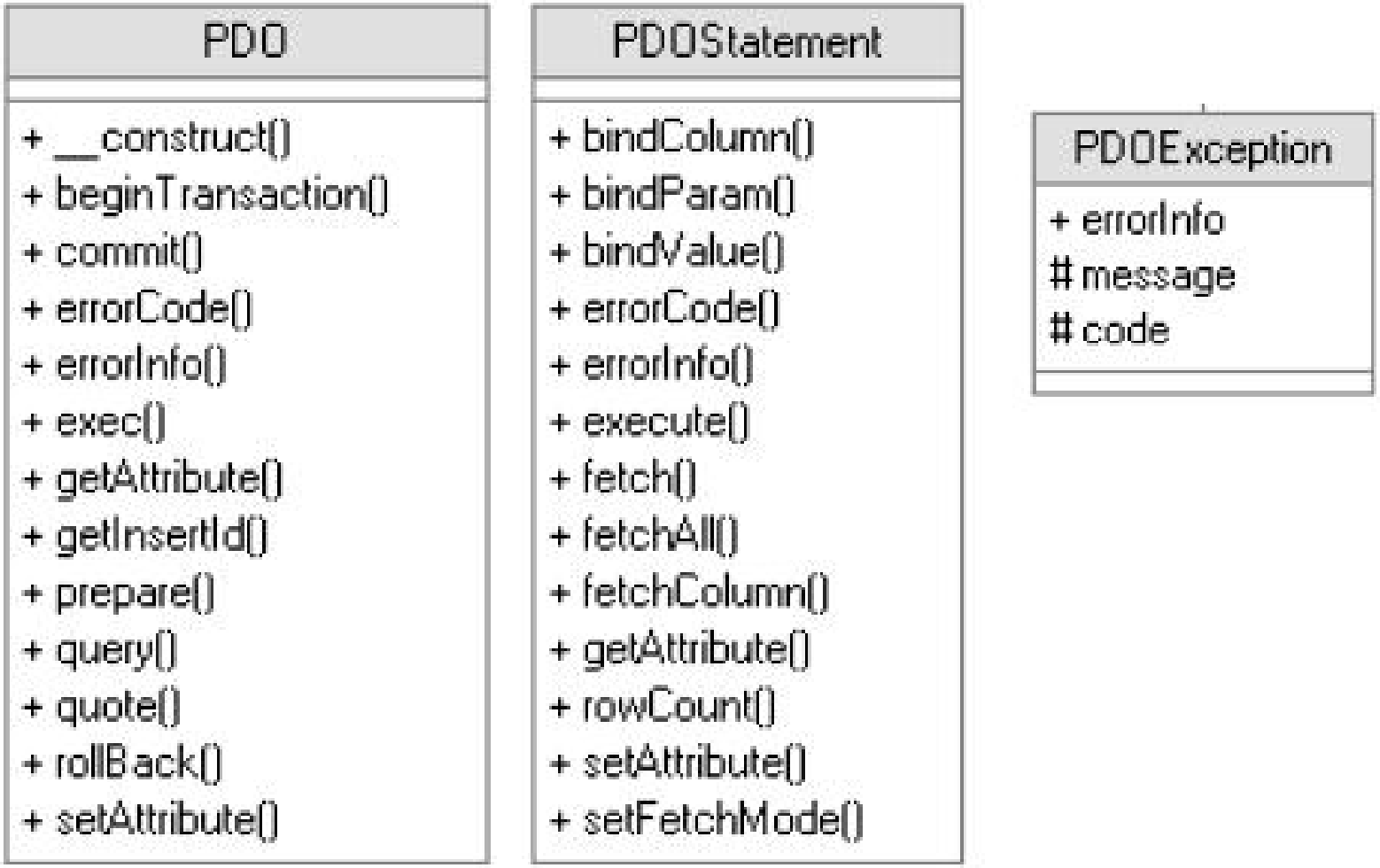
PDO, PHP Data Object

PDO (PHP 5) constitue une **couche d'abstraction**. Elle intervient entre le serveur d'application et le serveur de base de données.



La couche d'abstraction permet de **séparer** le traitement de la base de données.

Structure de PDO



Classe PDO

- commit : Valide une transaction
- __construct : Crée une instance PDO qui représente une connexion à la base
- errorCode : Retourne le SQLSTATE associé avec la dernière opération sur la base de données
- errorInfo : Retourne les informations associées à l'erreur lors de la dernière opération sur la base de données
- exec : Exécute une requête SQL et retourne le nombre de lignes affectées
- getAttribute : Récupère un attribut d'une connexion à une base de données
- PDO::getAvailableDrivers : Retourne la liste des pilotes PDO disponibles
- inTransaction : Vérifie si nous sommes dans une transaction
- lastInsertId : Retourne l'identifiant de la dernière ligne insérée ou la valeur d'une séquence
- prepare : Prépare une requête à l'exécution et retourne un objet
- query : Exécute une requête SQL, retourne un jeu de résultats en tant qu'objet PDOStatement
- quote : Protège une chaîne pour l'utiliser dans une requête SQL PDO
- rollBack : Annule une transaction (même si le script php plante)
- setAttribute : Configure un attribut PDO
- beginTransaction : Démarre une transaction

Classe PDOStatement

- bindColumn : Lie une colonne à une variable PHP
- bindParam : Lie un paramètre à un nom de variable spécifique
- bindValue : Associe une valeur à un paramètre
- closeCursor : Ferme le curseur, permettant à la requête d'être de nouveau exécutée
- columnCount : Retourne le nombre de colonnes dans le jeu de résultats
- debugDumpParams : Détaille une commande préparée SQL
- errorCode : Récupère le SQLSTATE associé lors de la dernière opération sur la requête
- errorInfo : Récupère les informations sur l'erreur associée lors de la dernière opération sur la requête
- execute : Exécute une requête préparée
- fetch : Récupère la ligne suivante d'un jeu de résultat PDO
- fetchAll : Retourne un tableau contenant toutes les lignes du jeu d'enregistrements
- fetchColumn : Retourne une colonne depuis la ligne suivante d'un jeu de résultats
- fetchObject : Récupère la prochaine ligne et la retourne en tant qu'objet
- getAttribute : Récupère un attribut de requête
- getColumnMeta : Retourne les métadonnées pour une colonne d'un jeu de résultats
- nextRowset : Avance à la prochaine ligne de résultats d'un gestionnaire de lignes de résultats multiples
- rowCount : Retourne le nombre de lignes affectées par le dernier appel à la fonction PDOStatement::execute()
- setAttribute : Définie un attribut de requête
- setFetchMode : Définit le mode de récupération par défaut pour cette requête

La classe PDOException

- PDOException::getMessage — Récupère le message de l'exception
- PDOException::getPrevious : Retourne l'exception précédente
- PDOException::getCode : Récupère le code de l'exception
- PDOException::getFile : Récupère le fichier dans lequel l'exception est survenue
- PDOException::getLine : Récupère la ligne dans laquelle l'exception est survenue
- PDOException::getTrace : Récupère la trace de la pile
- PDOException::getTraceAsString : Récupère la trace de la pile en tant que chaîne
- PDOException::__toString : Représente l'exception sous la forme d'une chaîne
- PDOException::__clone : Clone l'exception

Utiliser PDO

```php
<?php
// Définition des variables de connexion
$user = 'lptw';
$pass = 'lptw00';
$dsn = 'mysql:host=localhost;dbname=lptwdb';
// Connexion à la base de données
try {
$dbh = new PDO($dsn, $user, $pass);
} catch (PDOException $e) {
die( "Erreur ! : " . $e->getMessage() );
}
```

DNS (*Data Source Name)*

DSN pour MySQL

- **host :** adresse du serveur distant (ex: localhost) ;
- **dbname :** nom de la base de données à utiliser ;
- **port :** donnée facultative indiquant le port TCP/IP utilisé pour la connexion ;
- **unix_socket** : donnée facultative indiquant l'adresse de la socket unix pour la connexion locale.

```
<?php
$hote = 'localhost';
$base = 'test';
$port = '3307';
$socket = '/tmp/mysql.sock';
$dsn = "mysql:host=$hote;port=$port;dbname=$base";
$dsn2 = "mysql:unix_socket=$socket;dbname=$base";
?>
```

Utiliser PDO

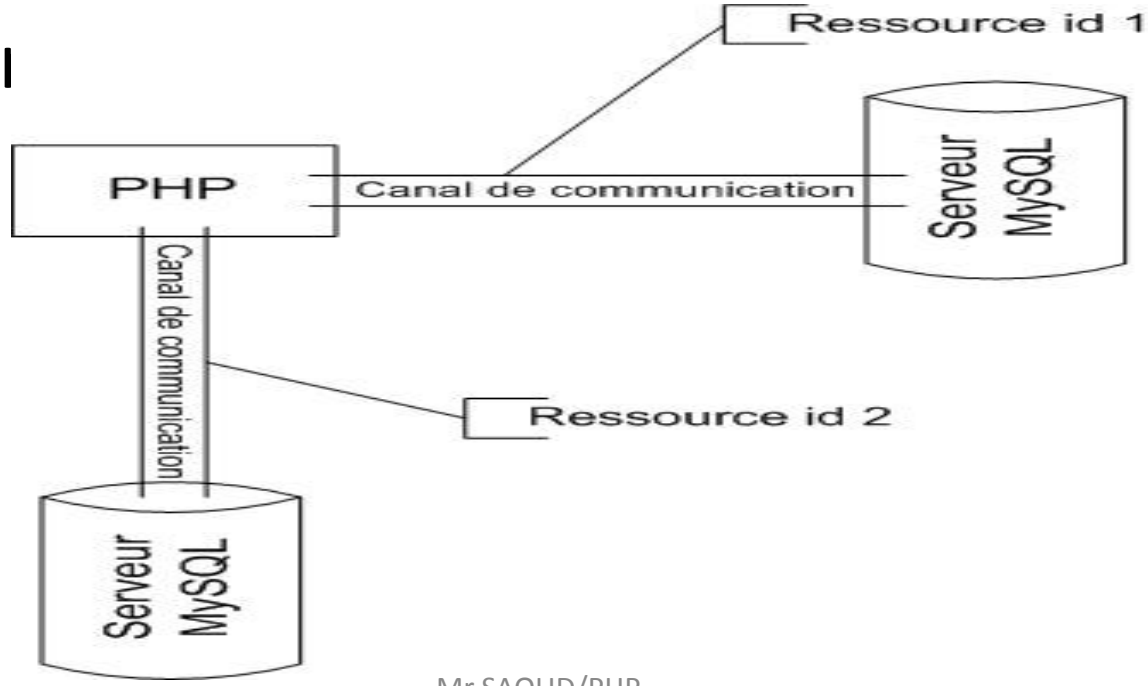
```
// Insertion d'un enregistrement
$sql = "INSERT INTO user VALUES ('toto','titi')";
$dbh->exec($sql);

// Lecture d'enregistrements
$sql = "SELECT login FROM user";
$resultat = $dbh->query($sql);
while ($row = $resultat->fetch()) {
Echo $row[0]."<br>";
}
//
Fermeture de la connexion
$dbh = NULL;
?>
```

Connexion à plusieurs bases de données

il suffit de créer plusieurs instances de la classe PDO.

$dbh1 = new PDO($dsn1, $user1, $pass1);

$dbh2 = I



Sélection des données : **query ou exec**

Requête SQL	Méthode PDO à utiliser
INSERT	exec()
UPDATE	exec()
DELETE	exec()
SELECT	query()
EXPLAIN	query()
SHOW	query()
DESC	query()

- ✓ **Exec()** renvoie que le nombre de lignes modifiées ou insérées
- ✓ **query()** renvoie une instance de l'objet PDOStatement contenant les résultats.

Sélection des données : fetchAll ou fetch

- **fetchAll()** retourne l'ensemble des données sous forme d'un tableau PHP et libère le SGBD.

- **fetch()** permet une lecture séquentielle du résultat. À un instant t, vous ne lisez qu'un seul résultat et la mémoire du système n'est pas occupée avec les autres entrées.

```
$sql = "SELECT login, nom FROM auteur ";
$sth = $dbh->query($sql);
$result = $sth->fetchAll(PDO::FETCH_ASSOC);
print_r($result);
```

Sélection des données : fetchAll ou fetch

- **PDO::FETCH_ASSOC** retourne un tableau associatif indexé par le nom de la colonne.
- **PDO::FETCH_BOTH** (par défaut) retourne un tableau indexé par les noms de colonnes et aussi par les numéros de colonnes (commençant à l'indice 0)
- **PDO::FETCH_OBJ** retourne un objet anonyme avec les noms de propriétés qui correspondent aux noms des colonnes.

Requêtes renvoyant beaucoup de résultats

fetch() cherche les enregistrements les uns après les autres

```
$sql = "SELECT * FROM table1";
$sth = $dbh->query($sql);
while($row = $sth->fetch(PDO::FETCH_ASSOC)){
Echo "$row [nom]" ;
}
```

L’opérateur ===

```
$sql = "DELETE FROM rmq WHERE pseudo='John'";
$retour = $dbh->exec($sql);
if($retour === FALSE){
    die('Erreur dans la requête') ;
}elseif($retour === 0){
    echo 'Aucune modification effectuée';
}else{
    echo $retour . ' lignes ont étés affectées.';
}
```

Remarque : l’opérateur de comparaison « === » vérifie l’égalité de valeur et de type. Ainsi, FALSE ne sera pas considéré de la même manière que 0.

Sécurité et échappements

- Oublier **d'échapper des caractères spéciaux** peut entraîner de graves problèmes de sécurité (on parle souvent de faille par injection SQL).

- **PDO** propose la méthode **quote()** de l'objet de connexion pour formate la chaine passé en argument de façon à pouvoir l'utiliser directement dans une requête SQL : des délimiteurs de chaîne sont insérés autour et les caractères spéciaux sont échappés.

Requêtes préparées

Pourquoi les requêtes préparées ?

- Inconvénients:
  - plus de lignes de code que pour une requête simple
  - exécution unique : performances moindres par rapport à une exécution directe
  - résultat final identique : exécuter une requête
  - débogage de la requête légèrement plus complexe
- Avantages:
  - impose une certaine rigueur de programmation
  - plus grande sécurité au niveau des requêtes
  - séparation requête SQL des données

Requêtes préparées

```php
<?php
    $stmt = $dbh->prepare("INSERT INTO REGISTRY (name, value) VALUES (:name, :value)");
    $stmt->bindParam(':name', $name);
    $stmt->bindParam(':value', $value);

    // insertion d'une ligne
    $name = 'one';
    $value = 1;
    $stmt->execute();

    // insertion d'une autre ligne avec des valeurs différentes
    $name = 'two';
    $value = 2;
    $stmt->execute();
    ?>
```

Requêtes préparées

Récupération des données en utilisant des requêtes préparées

```
<?php

$stmt = $dbh->prepare("SELECT * FROM REGISTRY where name = ?");

    if ($stmt->execute(array($_GET['name']))) {
      while ($row = $stmt->fetch()) {
        print_r($row);
      }
    }
    ?>
```

Requêtes préparées

Appel d'une procédure stockée avec un paramètre de sortie

```php
<?php
    $stmt = $dbh->prepare("CALL sp_returns_string(?)");
    $stmt->bindParam(1, $return_value, PDO::PARAM_STR, 4000);

    // Appel de la procédure stockée
    $stmt->execute();

    print "La procédure a retourné : $return_value\n";
    ?>
```

Requêtes préparées

Appel d'une procédure stockée avec un paramètre d'entrée/sortie

```
<?php
    $stmt = $dbh->prepare("CALL sp_takes_string_returns_string(?)");
    $value = 'hello';

    $stmt->bindParam(1, $value, PDO::PARAM_STR|PDO::PARAM_INPUT_OUTPUT, 4000);

    // appel de la procédure stockée
    $stmt->execute();

    print "La procédure a retourné : $value\n";
    ?>
```

Les transactions

```
try {
 $dbh->setAttribute(PDO::ATTR_ERRMODE,
    PDO::ERRMODE_EXCEPTION);
 $dbh->beginTransaction();
 $dbh->exec("insert into staff (id, first, last) values (23, 'Joe',
    'Bloggs')");
 $dbh->exec("insert into salarychange (id, amount, changedate)
 values (23, 50000, NOW())");
 $dbh->commit();
 } catch (Exception $e) {
 $dbh->rollBack();
 echo "Failed: " . $e->getMessage();
 }
```