

Secteur Tertiaire Informatique  
Filière « Etude et développement »

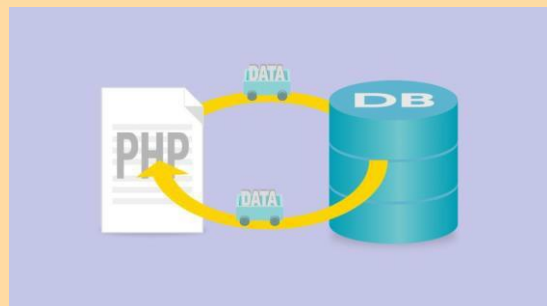
Séquence « Utiliser un composant objet d'accès  
aux données »

**Accéder à une Base de données en langage PHP  
à l'aide de PDO**

**Apprentissage**

Mise en pratique

Evaluation



## TABLE DES MATIERES

Table des matières .....	2
1. L'accès aux SGBD en PHP .....	4
2. PHP et MySQL.....	5
3. Le Framework PDO .....	5
4. Mise en œuvre de PDO pour MySQL en langage PHP .....	8
4.1 Se connecter à une base de données MySQL .....	8
4.2 Extraire des données d'une base MySQL .....	10
4.2.1 Exprimer une requête SQL .....	10
4.2.2 Lire le jeu de données en PHP .....	10
4.2.3 Exploiter le jeu de données retourné par PDO .....	11
4.2.4 Requêtes 'préparées' .....	12
4.2.5 Extraire les différentes valeurs des lignes d'enregistrements reçues.....	16
4.2.6 Pour faire le point sur l'extraction de données depuis une BDD .....	18
4.2.7 Mise à jour des données en base MySQL.....	20
4.3 PDO et la sécurité .....	21
4.3.1 Les transactions .....	21
4.3.2 Les erreurs d'exécution.....	22
4.4 Bonnes pratiques avec PDO .....	23
4.4.1 Isoler les instructions d'accès aux bases de données dans des scripts PHP séparés 23	
4.4.2 Définir les paramètres de connexion dans des variables ou constantes PHP .....	23
4.4.3 Désactiver le mode <i>silencieux</i> de PDO .....	23
4.4.4 Protéger tous les accès aux bases de données par des structures try/catch.....	23
4.4.5 Abuser des <i>alias SQL</i> et utiliser des syntaxes SQL génériques .....	24
4.4.6 Utiliser le système de requêtes préparées .....	24
4.4.7 Faire exécuter le maximum de travail sur les données par le SGBD lui-même .....	24
4.4.8 Gérer les transactions.....	24

Accéder à une Base de données en langage PHP à l'aide de PDO

Afpa © – Section Tertiaire Informatique – Filière « Etude et développement »

## Objectifs

Ce document a pour vocation de présenter comment accéder à une base de données depuis un script PHP et plus particulièrement en mettant en œuvre le framework PDO aujourd'hui incontournable.

## Pré requis

Connaître le principe des pages Web dynamiques et le langage de script PHP, y compris dans sa dimension '*orientée objet*'.

## Outils de développement

Tout éditeur ou IDE dédié au développement Web (NotePad++, CodeLobster, Brackets, NetBeans...).

## Méthodologie

Ce document se veut un guide de découverte et un aide-mémoire du framework PDO.

Il ne remplace pas la documentation de référence en ligne ; il la complète en précisant les rôles et utilités des objets du framework et en organisant la découverte par thèmes d'utilisation.

De nombreux liens renvoient vers la documentation en ligne en français très complète et bien maintenue à jour.

## Mode d'emploi

Symboles utilisés :



Renvoie à des supports de cours, des livres ou à la documentation en ligne constructeur.



Propose des exercices ou des mises en situation pratiques.



Point important qui mérite d'être souligné !

## Ressources

La documentation de référence en français disponible sur

<http://php.net/manual/fr/book.pdo.php>

## Lectures conseillées

Accéder à une Base de données en langage PHP à l'aide de PDO

Afpa © – Section Tertiaire Informatique – Filière « Etude et développement »

# 1. L'ACCES AUX SGBD EN PHP

Une page Web est constituée de scripts, en langages HTML (pour la structuration du contenu), CSS (pour la présentation) et JavaScript (pour gérer certains événements afin d'effectuer des contrôles de saisie et de gérer des interactions avec l'utilisateur).

**Un navigateur Web ne sait interpréter que ces langages et ne peut accéder qu'au contenu de la page en cours et aux bases de données locales embarquées dans les navigateurs ; en aucun cas, un navigateur ne peut « dialoguer » avec une base de données centrale située sur un serveur distant. Un navigateur Web ne dialogue qu'avec un serveur Web (serveur HTTP).**

En conséquence, **pour accéder à des données distantes à partir d'une page Web, il est nécessaire de passer par l'intermédiaire d'un script côté serveur Web qui se chargera de se connecter au serveur de base de données voulu, d'exprimer une requête de sélection ou de mise à jour de données, et de récupérer le résultat de cette requête afin de générer dynamiquement le contenu de la page Web** à transmettre au navigateur de l'utilisateur.

Chaque technologie côté serveur Web propose une ou plusieurs solutions pour réaliser ces traitements. Dans le « monde LAMP/WAMP » largement répandu (en particulier pour les sites d'hébergement), le langage de script est le **PHP** et le système de gestion de bases de données relationnel le plus souvent utilisé est **MySQL** (mais PHP peut s'interfacer avec un grand nombre de SGBD/R dont *SQL Server* et *Oracle* par exemple).

Pour réaliser cet interfaçage entre le langage PHP et un SGBD/R, il existe plusieurs solutions-types :

- des **bibliothèques de fonctions spécifiques à un SGBD** (« extensions PHP ») qui offrent alors de nouvelles instructions PHP (exemple : `mysql_connect(...)`),
- des **bibliothèques de classes spécifiques à un SGBD** prêtes à l'emploi (exemple : classe `mysqli`),
- des **bibliothèques de classes génériques et des pilotes spécifiques**, l'ensemble (« framework ») offrant une interface standard pour l'accès aux divers SGBD supportés ; l'exemple-type actuel est le **framework PDO** ; **son usage est recommandé.**

Quelle que soit la technologie utilisée, l'accès à des données centralisées depuis un script serveur nécessite :

- **d'établir une connexion** (ponctuelle ou persistante) avec le SGBD,
- **de préciser la base de données cible,**
- **de soumettre une requête** de sélection ou de mise à jour, **en langage SQL** (ou d'invoquer une procédure stockée)
- **de récupérer le résultat** de cette requête (jeu de données ou message),
- **de fermer la connexion** au serveur de données (afin de libérer des ressources).

Il s'agit maintenant de découvrir comment mettre en œuvre ces fonctionnalités à l'aide de PDO en ciblant une base MySQL.

Accéder à une Base de données en langage PHP à l'aide de PDO

Afpa © – Section Tertiaire Informatique – Filière « Etude et développement »

## 2. PHP ET MYSQL

**MySQL est un système de gestion de bases de données relationnel** gratuit, complet et performant. Il supporte maintenant toutes les fonctionnalités attendues d'un SGBD/R (contraintes déclaratives dont les clés étrangères, *procédures stockées* et *triggers*, *clusterisation* de serveurs...), au même titre que *SQL Server* ou *Oracle*, et présente des performances tout à fait acceptables lors de la montée en charge (voir

 <https://www.mysql.fr/why-mysql/benchmarks/>).

Notez que pour assurer des accès simultanés massifs avec un bon niveau de performance, il est nécessaire d'acquérir des postes serveurs puissants et une version spécifique (et payante) de MySQL incluant de nombreux outils d'administration spécialisés. Néanmoins, toutes les fonctionnalités nécessaires sont présentes dans la version de base (gratuite) de MySQL qui ne nécessite aucune configuration matérielle ou système particulière.

Pour administrer une base de données MySQL, il existe différentes applications, sous forme de « client lourd » à installer sur le poste de travail (« *MySQL Workbench* » pour Windows) ou plus simplement, sous forme de pages Web dynamiques utilisables très simplement à distance ou en local. Les packages logiciels gratuits et populaires chez les développeurs, **WampServer** et **EasyPHP**, incluent une interface d'administration pour MySQL écrite en PHP nommée **phpMyAdmin**.

Les apports et exemples donnés dans ce document se basent sur des éditions standards de phpMyAdmin et de MySQL pour Windows telles que fournies dans le package WampServer.

 Pour en savoir plus sur MySQL, voir <http://www.mysql.com/> et <http://dev.mysql.com/>

Pour en savoir plus sur phpMyAdmin, voir <https://www.phpmyadmin.net/> et le manuel pratique **Z-manuel-pratique-phpMyAdmin.pdf**.


## 3. LE FRAMEWORK PDO

Le framework PDO (« *PHP Data Objects* ») est fourni comme une extension au langage PHP ; ainsi, dès lors qu'elle est activée sur le serveur Apache/PHP, le développeur peut utiliser les objets PDO dans ses scripts.

**PDO est constitué d'une bibliothèque de classes (contenant principalement une classe nommée « PDO ») et de pilotes (ou « drivers ») spécifiques à chaque SGBD supporté.** Dès lors que le pilote correspondant au SGBD cible est installé sur le serveur Apache/PHP, les objets PDO instanciés dans les scripts sauront dialoguer avec le SGBD selon le protocole spécifique à ce SGBD. Le pilote pour MySQL est bien évidemment fourni avec les packages WAMP/LAMP.


En cas de changement de SGBD, tout est transparent pour le développeur, aucune intervention n'est nécessaire dans le code PHP déjà écrit : il suffit d'installer le nouveau pilote ! On dit que le framework **PDO fournit une abstraction à l'accès aux données**.

PDO 'se limite' à cela (ce qui n'est déjà pas mal !) quand d'autres frameworks offrent de plus une *abstraction à la base de données* en générant eux-mêmes les requêtes SQL nécessaires ou même en automatisant la création des tables (voir **Hibernate** et ses variantes, **Entity**

 **Framework** en technologie .Net ou encore **Doctrine** dans le monde PHP).

Accéder à une Base de données en langage PHP à l'aide de PDO

Afpa © – Section Tertiaire Informatique – Filière « Etude et développement »

 Exercice : Questions pour faire le point sur PHP, PDO et MySQL ; pour chacune des phrases ci-dessous, cochez la bonne réponse :

	Vrai	Faux
Un (bon) navigateur Web peut interroger une base de données distante à l'aide d'un script en langage JavaScript	<input type="checkbox"/>	<input type="checkbox"/>
Le langage PHP contient déjà en standard des instructions spécifiques pour interroger une base de données MySQL	<input type="checkbox"/>	<input type="checkbox"/>
L'accès à une base de données depuis un script PHP nécessite d'établir une connexion à la base avant de pouvoir en exploiter les données	<input type="checkbox"/>	<input type="checkbox"/>
MySQL et phpMyAdmin, c'est la même chose, seulement phpMyAdmin est écrit en PHP	<input type="checkbox"/>	<input type="checkbox"/>
WAMPServer, LAMPServer et EasyPHP, c'est 'bonnet blanc et blanc bonnet', ça permet d'installer une configuration serveur Web complète sur son PC	<input type="checkbox"/>	<input type="checkbox"/>
PDO, plus complet, peut remplacer phpMyAdmin	<input type="checkbox"/>	<input type="checkbox"/>
MySQL, c'est juste pour développer et mettre au point le site Web ; il faudra bien passer à autre chose de plus 'sérieux' au moment de la publication par l'hébergeur	<input type="checkbox"/>	<input type="checkbox"/>
PDO permet d'accéder à une base de données MySQL, SQL Server ou Oracle mais comme les <i>logins</i> n'ont rien à voir entre eux, il faudra bien modifier les ordres de connexion dans tous les scripts en cas de changement de SGBD cible	<input type="checkbox"/>	<input type="checkbox"/>

*Réponses page suivante.*

Réponses et commentaires pour faire le point sur PHP, PDO et MySQL :

	Vrai	Faux
Un (bon) navigateur Web peut interroger une base de données distante à l'aide d'un script en langage JavaScript <b>JavaScript ne permet d'exploiter que les bases de données locales embarquées dans les navigateurs. Et encore, chaque navigateur dispose de sa propre base de données incompatible avec celles des autres navigateurs !</b>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Le langage PHP contient déjà en standard des instructions spécifiques pour interroger une base de données MySQL <b>Les gens qui connaissent les débuts de PHP seraient tentés de répondre 'vrai' car on utilise depuis toujours les fonctions <code>mysql_xxx(...)</code>, mais il faut bien comprendre qu'il s'agit de fonctions ajoutées au langage PHP par le biais de l'extension MySQL qui n'est pas nécessairement activée, selon les besoins.</b>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
L'accès à une base de données depuis un script PHP nécessite d'établir une connexion à la base avant de pouvoir en exploiter les données	<input checked="" type="checkbox"/>	<input type="checkbox"/>
MySQL et phpMyAdmin, c'est la même chose, seulement phpMyAdmin est écrit en PHP <b>Rien à voir ! MySQL est le système de gestion de base de données relationnel ; il s'agit d'un 'service' qui tourne en tâche de fond et qui répond à des requêtes en recherchant ou mettant à jour les informations dans les tables. phpMyAdmin est un ensemble (complexe) de pages Web dynamiques écrites en PHP et permettant d'administrer les bases de données MySQL (création et modification des tables, optimisation du fonctionnement, sauvegarde et restauration des données...).</b>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
WAMPServer, LAMPServer et EasyPHP, c'est 'bonnet blanc et blanc bonnet', ça permet d'installer une configuration serveur Web complète sur son PC <b>Globalement, vrai ; mais LAMPServeur s'installe sous Linux alors que les 2 autres sont conçus pour fonctionner sous Windows !</b>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
PDO, plus complet, peut remplacer phpMyAdmin <b>Là encore, rien à voir ! PDO est un ensemble d'outils pour aider le développeur à écrire des scripts PHP exploitant des données stockées en bases de données, alors que phpMyAdmin permet d'administrer les bases de données MySQL (voir plus haut).</b>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
MySQL, c'est juste pour développer et mettre au point le site Web ; il faudra bien passer à autre chose de plus 'sérieux' au moment de la publication par l'hébergeur <b>Même si l'hébergeur devra acquérir une version professionnelle de MySQL et l'optimiser pour ses serveurs, le moteur MySQL reste le même que celui des versions gratuites pour postes de travail ; ainsi le développeur peut mettre au point son application à l'aide de WAMPServer ou LAMPServer sur son poste de travail et exporter directement les bases de données MySQL chez l'hébergeur sans rien changer à ses développements.</b>	<input type="checkbox"/>	<input checked="" type="checkbox"/>

Accéder à une Base de données en langage PHP à l'aide de PDO

Afpa © – Section Tertiaire Informatique – Filière « Etude et développement »

PDO permet d'accéder à une base de données MySQL, SQL Server ou Oracle mais comme les <i>logins</i> n'ont rien à voir entre eux, il faudra bien modifier les ordres de connexion dans tous les scripts en cas de changement de SGBD cible <b><i>Faux dans le principe... mais la réalité réserve parfois quelques surprises et le développeur devrait 'isoler' le code de connexion dans un script séparé de manière à pouvoir le faire évoluer au besoin. De même, des adaptations des requêtes SQL pourraient être nécessaires si le développeur utilise des ordres ou options spécifiques à un SGBD en particulier...</i></b>	<input type="checkbox"/>	<input type="checkbox"/>
---	--------------------------	--------------------------

## 4. MISE EN ŒUVRE DE PDO POUR MYSQL EN LANGAGE PHP

### 4.1 SE CONNECTER A UNE BASE DE DONNEES MYSQL

Pour se connecter à une base de données depuis un script PHP, il est nécessaire, quel que soit le SGBD de préciser au moins :

- Le nom du serveur de données,
- Le port TCP sur lequel ce SGBD 'écoute' les requêtes et fournit les réponses,
- Le nom de la base de données à utiliser,
- Les informations de login conditionnant les droits d'accès aux données des bases, typiquement, un nom d'utilisateur et un mot de passe.

Comme PDO peut se connecter à de nombreux SGBD qui nécessitent de toutes aussi nombreuses variantes dans l'expression de ces paramètres, les auteurs du framework ont choisi de regrouper les premières informations dans une *chaîne de connexion* de format variable suivant le pilote de SGBD, appelée **Data Source Name** ou **DNS**. Pour MySQL, le format de la DSN inclut essentiellement les paramètres **host** (pour le nom du serveur), **port** (pour le port TCP), **dbname** (pour le nom de la BDD) et **charset** (pour le jeu de caractères utilisé).

Avec PDO, **le développeur précise toutes les informations nécessaires lors de l'instanciation d'un objet PDO**, en les passant en paramètre au *constructeur* de la classe PDO, tout d'abord le DSN, puis les données de *login*.

Exemple :

```
$connexion =
new PDO("mysql:host=localhost; port=3306; dbname=video; charset=utf8",
    "utilisateur", "motdepasse");
```

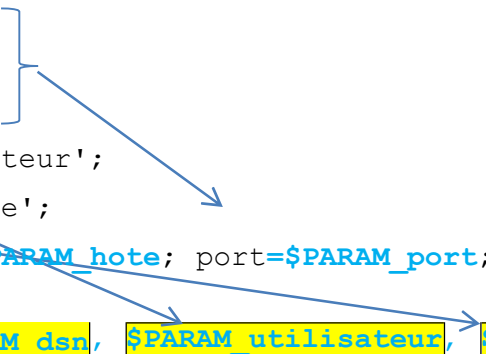
Voilà pour l'essentiel !



Pour offrir plus de souplesse en cas d'éventuel changement de SGBD cible, il est vivement recommandé de définir chaque paramètre dans une variable (ou constante) PHP :

Exemple avec des variables PHP :

```
$PARAM_hote='localhost';
$PARAM_port='3306';
$PARAM_nom_bd='video';
$PARAM_utilisateur='utilisateur';
$PARAM_mot_passe='motdepasse';
$PARAM_dsn = "mysql:host=$PARAM_hote; port=$PARAM_port; dbname=$PARAM_nom_bd;
charset=utf8" ;
$connexion = new PDO ($PARAM_dsn, $PARAM_utilisateur, $PARAM_mot_passe);
```



De plus, il est aussi vivement conseillé de stocker ce script de connexion dans un petit fichier distinct qui sera fusionné en tant que de besoin dans les scripts PHP qui nécessitent un accès à la base de données (fonction PHP `include()`, `require()` ou `include_once()`, `require_once()`).

Allez, un dernier paramètre est prévu pour préciser les modalités de fonctionnement en cas d'erreur (car tout ne vas pas toujours aussi bien que souhaité...) ou encore pour apporter d'autres précisions spécifiques au SGBD : **le constructeur de la classe PDO accepte en dernier paramètre optionnel un tableau contenant une ou plusieurs constantes PDO pré-définies.**

Exemple pour gérer les cas d'erreur en levant une Exception spécifique PDO :

```
$connexion = new PDO(....., array(PDO::ATTR_ERRMODE=> PDO::ERRMODE_EXCEPTION));
```

Bien entendu, il sera nécessaire de capturer l'Exception levée, grâce à une structure PHP `try/catch`. Ce point est détaillé en fin de document.

Exemple pour préciser les modalités d'encryptage avec MySQL :

```
$pdo = new PDO(
    'mysql:host=hostname;dbname=ssldb',
    'username',
    'password',
    array(
        PDO::MYSQL_ATTR_SSL_KEY => '/path/to/client-key.pem',
        PDO::MYSQL_ATTR_SSL_CERT => '/path/to/client-cert.pem',
        PDO::MYSQL_ATTR_SSL_CA => '/path/to/ca-cert.pem'
    )
);
```

**NB :** la déconnexion au serveur n'est pas nécessaire car PDO et PHP 'font le ménage' en fin de script.



Pour en savoir plus : <http://php.net/manual/fr/pdo.construct.php>  
et <http://php.net/manual/fr/ref.pdo-mysql.php>

Accéder à une Base de données en langage PHP à l'aide de PDO

Afpa © – Section Tertiaire Informatique – Filière « Etude et développement »

## 4.2 EXTRAIRE DES DONNEES D'UNE BASE MYSQL

Avec PDO, pour établir une connexion à une base de données, il a donc fallu instancier un objet PDO. Dès lors, **les traitements à effectuer sur cette base de données seront exécutés en invoquant des méthodes de cet objet PDO** (et d'autres objets connexes).

### 4.2.1 Exprimer une requête SQL

Avec PDO, **pour faire exécuter une requête SQL**, l'objet PDO expose une méthode **query()** qui attend en **paramètre le libellé de la requête SQL** à exécuter et **retourne un objet PDO particulier permettant l'accès au jeu d'enregistrements** (objet de type **PDOStatement**).

Exemple pour sélectionner tous les membres actifs d'une association :

```
$connexion = new PDO(...);  
$resultats = $connexion->query("SELECT nommembre  
FROM membres  
WHERE statut='actif'  
ORDER BY nommembre ASC");
```

Cette méthode `query()` soumet la requête SQL reçue en paramètre, en utilisant le protocole de communication adapté (défini par le pilote PDO utilisé), et ce, quelle que soit la base de données cible (et le SGBD qui la gère) ; facile !

Attention tout de même de respecter le langage SQL standard s'il est envisagé une évolution du SGBD car chaque SGBD propose ses extensions spécifiques. Ainsi, pour récupérer les 10 premiers enregistrements, MySQL utilise l'option `select ... limit 0,10` alors que SQL Server admet `select top 10 ...`

Là encore, une bonne pratique consiste à isoler tous les accès aux bases de données dans des scripts séparés de manière à pouvoir faire évoluer plus facilement l'application en cas de changement de SGBD.

### 4.2.2 Lire le jeu de données en PHP

Avec PDO, **pour lire les données reçues du SGBD**, un objet **PDOStatement** propose les méthodes **fetch()** et **fetchAll()** :

- **fetch()** effectue la **lecture d'un enregistrement et passe au suivant ou détecte la fin** du flot de données, et **retourne un tableau** contenant les données de l'enregistrement lu ;
- **fetchAll()** effectue la **lecture de tous les enregistrements et retourne un tableau à deux dimensions** où chaque ligne du tableau représente un enregistrement et chaque colonne du tableau correspond à une colonne/expression spécifiée en clause `select` de la requête SQL.

Accéder à une Base de données en langage PHP à l'aide de PDO

Afpa © – Section Tertiaire Informatique – Filière « Etude et développement »

#### Exemple de lecture pas à pas :

```
while($ligne = $resultats->fetch()) // récupère 1 à 1 les lignes lues
{
    // traitement de la ligne récupérée
} // fin de boucle après la dernière ligne
```

#### Exemple de lecture globale :

```
$records = $resultats->fetchAll(); // on lit tout ;
// $records est un tableau à 2 dimensions
```

Il devient donc très simple avec PDO d'extraire des données d'une base MySQL (ou autre) !

### 4.2.3 Exploiter le jeu de données retourné par PDO

Très bien, tout cela ; on sait maintenant se connecter à un serveur MySQL et exprimer une requête SQL. Mais comment, concrètement, exploiter les tableaux de données retournés par les méthodes de lecture d'enregistrements ?

Là encore, pour aider le développeur à exploiter les données reçues, les auteurs de PDO ont tout prévu : **PDO peut retourner les données lues** depuis la base de données principalement sous forme de :

- **Tableau PHP indicé** : `$data=$resultats->fetch(PDO::FETCH_NUM);`
- **Tableau PHP associatif** : `$data=$resultats->fetch(PDO::FETCH_ASSOC);`
- **Tableau PHP associatif et indicé** : `$data=$resultats->fetch();`
- **Objet standard PHP** : `$data=$resultats->fetch(PDO::FETCH_OBJ);`

Il suffit donc de préciser le type de résultat souhaité lors de la lecture des enregistrements, aussi bien en utilisant la méthode `fetch()` que la méthode `fetchAll()`.

Pour un **tableau indicé**, chaque colonne/expression de la clause `select` correspond à un **poste numéroté** (à partir de zéro).

#### Exemple :

```
echo $ligne[1] ; // restitue le contenu de la 2 colonne extraite par SQL
```

Pour un **tableau associatif**, chaque colonne/expression de la clause `select` correspond à un **poste nommé** selon le nom de colonne ou *d'alias* utilisé en langage SQL.

#### Exemple :

```
echo $ligne["nommembre"] ; // restitue le contenu de la colonne/expression
//extraite sous le nom nommembre
```

Accéder à une Base de données en langage PHP à l'aide de PDO

Afpa © – Section Tertiaire Informatique – Filière « Etude et développement »

Pour un **tableau associatif et indicé**, chaque colonne/expression de la clause `select` correspond à un **poste numéroté mais aussi nommé** ; au développeur de choisir comment il souhaite désigner le résultat recherché. Notez que ce cas correspond au fonctionnement par défaut.

Enfin, pour un **objet PHP standard**, PDO restitue la ligne lue sous forme **d'attributs publics encapsulés dans un objet anonyme** créé artificiellement (et qui reste assez pauvre...).

Exemple :

```
echo $ligne->nommembre ; // restitue le contenu de l'attribut issu
                        // de la conversion de la ligne en objet PHP
```

On vous l'a dit, les auteurs de PDO ont pensé à tout ; il existe encore d'autres variantes possibles mais celles-ci sont les plus utilisées.



Pour en savoir plus : <http://php.net/manual/fr/pdostatement.fetch.php>

#### 4.2.4 Requêtes 'préparées'

Tout cela commence à être opérationnel mais, comme la '*vraie vie*' n'est jamais aussi simple, il reste encore une difficulté pour le développeur : constituer un libellé de requête SQL variable d'une exécution à l'autre.

Reprenons notre exemple des membres de l'association. Les exemples précédents permettaient déjà de récupérer tous les membres actifs ; on voudrait maintenant sélectionner les membres actifs d'une ville particulière choisie par l'utilisateur et dont on connaît la valeur par une variable `$villevoulu`.

Le développeur devra choisir entre plusieurs stratégies.

Première solution : on récupère tous les membres actifs dans un tableau par la méthode `fetchAll()` par exemple et la boucle PHP de lecture du tableau contient un test pour ne prendre en compte que les membres de la ville voulue.

Exemple :

```
// on extrait tous les membres actifs
$resultats=$connexion->query("SELECT nommembre, ville
FROM membres
WHERE statut='actif'
ORDER BY nommembre ASC");
$records = $resultats->fetchAll(); // on lit tout ;
// on parcourt le tableau reçu, ligne à ligne
foreach ($records as $ligne) {
    if ($ligne["ville"] == $villevoulu) {
        // traitement des seuls membres de la ville
    } // fin de test sur membre
} // fin de parcours du jeu de lignes
```

Accéder à une Base de données en langage PHP à l'aide de PDO

Afpa © – Section Tertiaire Informatique – Filière « Etude et développement »

Cela fonctionnera correctement mais... quel gâchis de ressources ! En effet, avec cette solution, on a extrait toutes les lignes de la table des membres, on les a transmises du serveur de BDD vers le serveur Web afin que PHP sélectionne uniquement les membres souhaités. On risque fort d'écrouler les performances du serveur Web lors de la montée en charge...

Deuxième solution : on fait extraire les membres de la ville souhaitée par le serveur de base de données ; le serveur Web ne reçoit alors que les (quelques) enregistrements voulus et peut les restituer systématiquement à l'utilisateur.

```
// on extrait tous les membres actifs de la ville souhaitée
$resultats=$connexion->query("SELECT nommembre, ville
    FROM membres
    WHERE statut='actif'
    // condition supplémentaire ;
    AND ville = ' " . $villevoulue . "' // concaténation explicite
    ORDER BY nommembre ASC");
$records = $resultats->fetchAll(); // on lit tout ;
// on parcourt le tableau reçu, ligne à ligne, sans test
foreach ($records as $ligne){
    // traitement des membres de la ville voulue
} // fin de parcours du jeu de lignes
```

Cela fonctionne tout aussi correctement et on a allégé la charge du serveur Web (déjà bien sollicité par toutes les demandes simultanées de pages).

Très bien, mais il reste une difficulté technique pour le développeur dans les concaténations nécessaires pour reconstituer le libellé de la requête SQL en fonction du contenu des variables PHP, surtout si les paramètres sont nombreux.

On peut toujours simplifier l'écriture en utilisant le mécanisme de concaténation implicite de PHP :

```
// on extrait tous les membres actifs de la ville souhaitée
$resultats=$connexion->query("SELECT nommembre, ville
    FROM membres
    WHERE statut='actif'
    // condition supplémentaire ;
    AND ville = '$villevoulue' // concaténation implicite
    ORDER BY nommembre ASC");
```

Mais on vous le répète, les auteurs de PDO ont pensé à tout, et surtout à vous, développeurs :  
**PDO propose un système de 'requêtes préparées' qui prend en charge ces concaténations périlleuses** : il vous suffit :

- de **matérialiser l'emplacement des paramètres dans le libellé de la requête** SQL,
- **d'envoyer cet embryon de requête à la méthode `prepare()`** de l'objet PDO,
- de **déclencher l'exécution de la requête par la méthode `execute()`** de l'objet PDOStatement en **passant en paramètres les variables PHP** à fusionner.

La **matérialisation des paramètres** peut se faire au choix :

- par un simple **caractère ?**
- par une **'étiquette'** nommée préfixée du **caractère :**

La **passation des variables PHP** se fait par la transmission d'un *tableau*, *simple* dans le premier cas, ou *associatif* dans le second cas.

Reprenons l'exemple précédent pour concrétiser tout cela :

```
$connexion = new PDO(...);  
// on extrait tous les membres actifs de la ville souhaitée  
$sql = "SELECT nommembre, ville  
        FROM membres  
        WHERE statut='actif'  
        // condition supplémentaire ;  
        AND ville = ?  
        ORDER BY nommembre ASC" ;  
// préparation requête (retourne un objet PDOStatement)  
$resultats = $connexion->prepare($sql) ;  
// exécution requête  
$resultats->execute(array($villevoulu)) ;  
$datas = $resultats->fetchAll(); // on récupère tout dans un tableau PHP
```

Accéder à une Base de données en langage PHP à l'aide de PDO

Afpa © – Section Tertiaire Informatique – Filière « Etude et développement »

Ou bien :

```
$connexion = new PDO(...);
// on extrait tous les membres actifs de la ville souhaitée
$sql = "SELECT nommembre, ville
FROM membres
WHERE statut='actif'
// condition supplémentaire ;
AND ville = :villevoulué
ORDER BY nommembre ASC" ;
// préparation requête (retourne un objet PDOStatement)
$resultats=$connexion->prepare($sql);
// exécution requête
$resultats->execute(array(':villevoulué'=>$villevoulué)) ;
$datas = $resultats->fetchAll(); // on récupère tout dans un tableau PHP ;
```

Aucune des 2 syntaxes n'est '*naturelle*' mais la deuxième s'avère bien plus facile à relire et à maintenir en cas de nombreux paramètres, car *le lien est fait sur les libellés des étiquettes et non sur leur ordre d'apparition dans la requête.*

#### Exemple :

```
$connexion = new PDO(...);
// on extrait tous les membres actifs de la ville souhaitée
$sql = 'select * from location where NUM_ADHERENT = :numadh and
ID_FILM = :codfil and DEBUT_LOCATION = :cejour' ;
// preparation requête
$resultats = $connexion->prepare($sql);
// execution requete
$resultats->execute(array(':numadh'=>$dataResa["numadherent"],
                        ':cejour'=>$dataResa["datejour"],
                        ':codfil'=>$dataResa["codfil"]
));
$datas = $resultats->fetchAll(); // on récupère tout dans un tableau PHP ;
```

Notons pour finir que la technique des requêtes préparées permet de reléguer à PDO les concaténations (souvent périlleuses) de guillemets ou autres *simple-quotes* nécessaires pour différencier en SQL les noms de colonnes des valeurs voulues. Merci PDO !



Pour en savoir plus sur `execute()` : <http://php.net/manual/fr/pdostatement.execute.php>

Pour en savoir plus sur `prepare()` : <http://php.net/manual/fr/pdo.prepare.php>

Accéder à une Base de données en langage PHP à l'aide de PDO

Afpa © – Section Tertiaire Informatique – Filière « Etude et développement »

#### 4.2.5 Extraire les différentes valeurs des lignes d'enregistrements reçues

Bien, nous savons maintenant nous connecter à une base de données (MySQL ou autre), exprimer une requête SQL (paramétrée ou non), et récupérer des enregistrements, sous forme de tableaux PHP (méthode `fetch()`) ou lignes de tableaux PHP à 2 dimensions (méthode `fetchAll()`).

Intéressons-nous maintenant à la récupération des différentes valeurs de ces tableaux PHP, de manière à pouvoir les injecter en lieu et place de la page HTML à retourner à l'utilisateur, car c'est bien l'aboutissement de tous ces traitements.

La méthode `fetch()` par défaut ou `fetch(PDO::FETCH_ASSOC)` retourne les données d'un enregistrement sous forme de tableau associatif PHP dont les indices sont les noms de colonnes/expressions de la clause `select` SQL. On peut donc déjà **accéder à chaque poste de ce tableau PHP par la notation entre crochets** comme on l'a vu précédemment en 4.2.3.

Exemple :

```
...
<select name="membrevoulu">
<?php
while($ligne = $resultats->fetch()) // récupère 1 à 1 les lignes lues
{
    // traitement de la ligne récupérée :
    // ajout du membre dans la liste HTML select
    echo '<option value="'. $ligne['idmembre'] . '>' . $ligne['nommembre'] . '</option>';
} // fin de boucle après la dernière ligne
?>
</select>
...
```

Mais les auteurs du langage PHP ont eux-aussi pensé à tout pour simplifier la vie du développeur !

Il existe une fonction 'magique' du langage PHP, `extract()`, qui permet de **créer automatiquement autant de variables indépendantes que de postes d'un tableau associatif** et, naturellement, **ces variables prennent les noms des indices du tableau associatif**.

Résultat : des variables PHP plus simples à manipuler, sans la syntaxe entre crochets et les concaténations explicites, mais aucun contrôle par le développeur sur l'existence de ces variables.



Le même exemple peut donc s'écrire :

```
...
<select name="membrevoulu">
<?php
while($ligne = $resultats->fetch()) // récupère 1 à 1 les lignes lues
{
    extract($ligne) ; // crée des variables à partir du tableau associatif
    // traitement de la ligne récupérée :
    // ajout du membre dans la liste HTML select
    echo "<option value=$idmembre>$nommembre</option>" ;
} // fin de boucle après la dernière ligne
?>
</select>
...
```

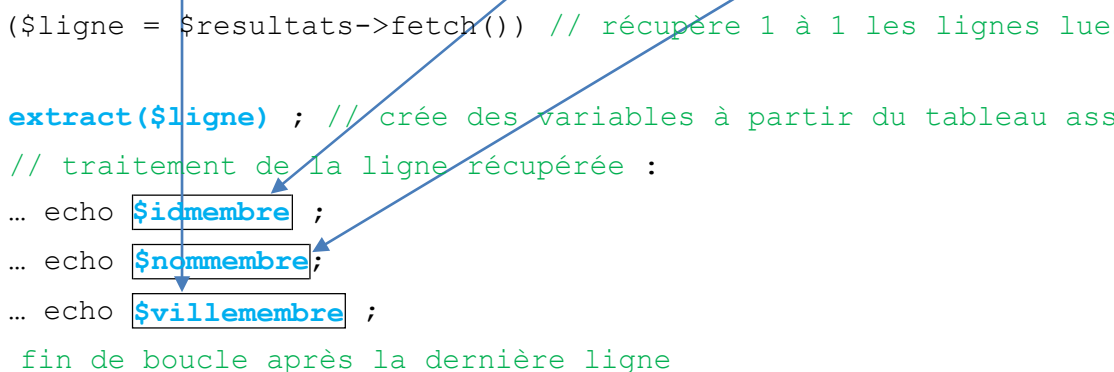
Notez que dans un cas comme dans l'autre, chaque information est accessible grâce à un nom qui est déterminé par le nom de colonne/expression de la clause `select` SQL ; **il est donc important de bien nommer les éléments de la clause `select` SQL en usant et abusant de l'option d'alias SQL `as`.**

Exemple :

```
$resultats=$connexion->query("SELECT id as idmembre, nom as nommembre,
ville as villemembre
FROM membres
WHERE statut='actif'
ORDER BY nommembre ASC");
```

Dès lors le script PHP de présentation en page HTML utilisera ces noms d'alias :

```
while($ligne = $resultats->fetch()) // récupère 1 à 1 les lignes lues
{
    extract($ligne) ; // crée des variables à partir du tableau associatif
    // traitement de la ligne récupérée :
    ... echo $idmembre ;
    ... echo $nommembre ;
    ... echo $villemembre ;
} // fin de boucle après la dernière ligne
```



On adoptera la même raisonnement en cas d'utilisation de la méthode `fetchAll()` qui retourne un tableau associatif PHP à 2 dimensions, car la lecture du tableau ligne à ligne retourne la même chose que la lecture des enregistrements un à un.

Accéder à une Base de données en langage PHP à l'aide de PDO

Afpa © – Section Tertiaire Informatique – Filière « Etude et développement »

### Exemple :

```
$records = $resultats->fetchAll(); // on lit tout dans un tableau
// on parcourt le tableau reçu, ligne à ligne
foreach ($records as $ligne){
    extract($ligne) ; // crée des variables à partir du tableau associatif
    // traitement de la ligne récupérée :
    ... echo $idmembre ;
    ... echo $nommembre;
    ... echo $villemembre ;
} // fin de boucle après la dernière ligne du tableau
```

Pour en savoir plus sur `fetch()` et `fetchAll()` :



<http://php.net/manual/fr/pdostatement.fetch.php>

<http://php.net/manual/fr/pdostatement.fetchall.php>

### 4.2.6 Pour faire le point sur l'extraction de données depuis une BDD



Exercice : Questions pour faire le point sur l'extraction de données à l'aide de PDO ; pour chacune des phrases ci-dessous, cochez la bonne réponse :

	Vrai	Faux
On débute l'accès par PDO à une base de données en établissant une connexion au serveur de données grâce à une instance d'objet PDO	<input type="checkbox"/>	<input type="checkbox"/>
Un objet PDO dispose d'une méthode <code>query()</code> qui permet d'adresser directement une requête SQL au serveur	<input type="checkbox"/>	<input type="checkbox"/>
Les méthodes <code>query()</code> et <code>prepare()</code> de l'objet PDO retournent un objet <code>PDOStatement</code>	<input type="checkbox"/>	<input type="checkbox"/>
Un objet PDO dispose d'une méthode <code>fetch()</code> qui permet de lire une ligne de donnée reçue du serveur	<input type="checkbox"/>	<input type="checkbox"/>
Un objet <code>PDOStatement</code> dispose d'une méthode <code>fetchAll()</code> qui permet de lire toutes les lignes de donnée reçues du serveur	<input type="checkbox"/>	<input type="checkbox"/>
La méthode <code>prepare()</code> s'applique à un objet PDO alors que la méthode <code>execute()</code> s'applique à un objet <code>PDOStatement</code>	<input type="checkbox"/>	<input type="checkbox"/>
Pour paramétrer une requête SQL, il est préférable d'utiliser la technique des requêtes préparées	<input type="checkbox"/>	<input type="checkbox"/>
Dans une requête préparée, un paramètre peut rester anonyme, seule sa place dans la requête est matérialisée par un symbole	<input type="checkbox"/>	<input type="checkbox"/>
Dans une requête préparée, il est préférable de nommer les paramètres grâce à des étiquettes	<input type="checkbox"/>	<input type="checkbox"/>
Les méthodes <code>fetch()</code> et <code>fetchAll()</code> retournent par défaut un tableau indicé et associatif	<input type="checkbox"/>	<input type="checkbox"/>
La fonction <code>extract()</code> peut remplacer avantageusement un ordre de lecture par <code>fetch()</code> ou même <code>fetchAll()</code>	<input type="checkbox"/>	<input type="checkbox"/>

Réponses page suivante

Accéder à une Base de données en langage PHP à l'aide de PDO

Afpa © – Section Tertiaire Informatique – Filière « Etude et développement »

Réponses et commentaires pour faire le point sur l'extraction de données à l'aide de PDO.

	Vrai	Faux
On débute l'accès par PDO à une base de données en établissant une connexion au serveur de données grâce à une instance d'objet PDO <b>Et tout est dit lors de l'instanciation de cet objet : serveur et base de données cibles, jeu de caractères, informations d'identification</b>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Un objet PDO dispose d'une méthode <code>query()</code> qui permet d'adresser directement une requête SQL au serveur <b>Un objet PDO dispose déjà de méthodes permettant d'adresser des requêtes simples ou de préparer des requêtes paramétrées</b>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Les méthodes <code>query()</code> et <code>prepare()</code> de l'objet PDO retournent un objet <code>PDOStatement</code> <b>Et cet objet <code>PDOStatement</code> permettra, lui, d'effectuer la lecture des enregistrements.</b>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Un objet PDO dispose d'une méthode <code>fetch()</code> qui permet de lire une ligne de donnée reçue du serveur <b>L'objet PDO correspond à la connexion avec le serveur de données alors que l'objet <code>PDOStatement</code> représente le jeu de données reçues</b>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Un objet <code>PDOStatement</code> dispose d'une méthode <code>fetchAll()</code> qui permet de lire toutes les lignes de donnée reçues du serveur	<input checked="" type="checkbox"/>	<input type="checkbox"/>
La méthode <code>prepare()</code> s'applique à un objet PDO alors que la méthode <code>execute()</code> s'applique à un objet <code>PDOStatement</code> <b>L'objet PDO correspond à la connexion avec le serveur de données alors que l'objet <code>PDOStatement</code> représente le jeu de données reçues</b>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Pour paramétrer une requête SQL, il est préférable d'utiliser la technique des requêtes préparées <b>... à moins d'être un champion des concaténations...</b>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Dans une requête préparée, un paramètre peut rester anonyme, seule sa place dans la requête est matérialisée par un symbole <b>Oui, mais cela reste risqué si plusieurs paramètres sont nécessaires ; d'où la réponse suivante</b>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Dans une requête préparée, il est préférable de nommer les paramètres grâce à des étiquettes <b>Solution bien plus lisible et maintenable (mais au prix d'une variante de syntaxe car les étiquettes sont préfixées du caractère :) !</b>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Les méthodes <code>fetch()</code> et <code>fetchAll()</code> retournent par défaut un tableau indicé et associatif <b>Ainsi, on peut aussi bien les exploiter à l'aide d'un indice (dans une boucle par exemple) que grâce à leur nom (pour les adresser individuellement)</b>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
La fonction <code>extract()</code> peut remplacer avantageusement un ordre de lecture par <code>fetch()</code> ou même <code>fetchAll()</code> <b>Cette fonction <code>extract()</code> se limite à créer des variables autonomes à partir d'un tableau associatif, ce qui est pratique pour exploiter les différentes informations dans la page HTML en construction ; mais ce tableau associatif reste à alimenter par des opérations de lecture</b>	<input type="checkbox"/>	<input checked="" type="checkbox"/>

Accéder à une Base de données en langage PHP à l'aide de PDO

Afpa © – Section Tertiaire Informatique – Filière « Etude et développement »

#### 4.2.7 Mise à jour des données en base MySQL

Une application ne se limite pas à extraire des données de la BDD pour les injecter dans des pages HTML ; les formulaires HTML permettent à l'utilisateur distant de saisir des données et de les soumettre à un script PHP qui se chargera de mettre à jour la base de données.

Rappelons que les données saisie en formulaire HTML sont à disposition du développeur dans les '*tableaux associatifs supra-globaux*' \$\_GET et \$\_POST.

SQL propose les ordres `insert`, `update` et `delete` pour réaliser ces mises à jour de données. Leur résultat n'est plus un jeu d'enregistrements mais un simple message de succès ou d'échec et le nombre d'enregistrements affectés. La logique de l'extraction de données ne peut donc pas s'appliquer directement pour les besoins de mise à jour.

Là encore, PDO a pensé à tout : **un objet PDO expose une méthode `exec()` qui reçoit un paramètre une requête SQL de mise à jour**. Cette méthode `exec()` –à ne pas confondre avec `execute()` de l'objet `PDOStatement`– **retourne le nombre d'enregistrements affectés par l'opération de mise à jour**.

Exemple de base :

```
$connexion = new PDO (.....) ;  
$connexion->exec ('update membres set mot_pass=' . $_POST['mdp'] . ' where...');
```

Ou encore, en récupérant le nombre d'enregistrements affectés :

```
$connexion = new PDO (.....) ;  
$affectes = $connexion->exec ('update membres set mot_pass=' . $_POST['mdp'] . ' where...');
```

Et si le développeur préfère utiliser des requêtes paramétrées, cela reste encore possible et l'objet `PDOStatement` peut récupérer lui aussi le nombre d'enregistrements affectés !

Exemple :

```
$connexion = new PDO (.....) ;  
$sql = "update membres set mot_pass = :mdp where..." ;  
$resultats = $connexion->prepare ($sql) ;  
$resultats->execute (array (':mdp' => $_POST ['mdp'], ...)) ;  
$affectes = $resultats->rowCount () ; // nombre de lignes affectées
```

Bien entendu, les exemples donnés ici avec une commande SQL `update` s'appliquent de la même manière avec les commandes `insert` et `delete`.

Notez enfin que la méthode `exec()` permet aussi d'invoquer des *procédures stockées*.



Pour en savoir plus sur la méthode `exec()` : <http://php.net/manual/fr/pdo.exec.php>

Accéder à une Base de données en langage PHP à l'aide de PDO

Afpa © – Section Tertiaire Informatique – Filière « Etude et développement »

## 4.3 PDO ET LA SECURITE

### 4.3.1 Les transactions

Bien souvent plusieurs interventions de mise à jour sur des tables sont nécessaires pour assurer la mémorisation d'un événement. Le cas typique est celui de notre comptabilité '*en partie double*' pour laquelle une opération comptable se traduit par une ou plusieurs insertions de lignes de débit contrebalancées par une ou plusieurs opérations d'insertion de lignes de crédit.

Le problème qui peut alors se poser est celui d'un incident d'exécution pendant les mises à jour (panne serveur, disque plein...), alors que les écritures ne sont pas enregistrées en totalité. Résultat : la comptabilité est '*fausse*' et le comptable risque de passer quelques nuits blanches à pointer ses livres de comptes...

**Le mécanisme de gestion des transactions permet de s'assurer que les mises à jour formant un lot sont toutes passées ou bien toutes annulées en cas de mise à jour incomplète.**

La gestion des transactions est assurée par tous les SGBD modernes et MySQL n'est pas en reste, à condition d'utiliser le '*moteur InnoDB*' (qui est maintenant le moteur par défaut pour MySQL). PDO prend donc en relai la gestion des transactions.

**Par défaut, avec PDO, toute requête est exécutée et validée unitairement, immédiatement.**

On parle de validation automatique ou de mode '*auto-commit*', mais le développeur peut insérer des ordres spécifiques pour délimiter des lots de requêtes SQL formant un tout indissociable, donc une transaction.

Pour ce faire, la classe PDO expose 3 méthodes *statiques* :

- **PDO::beginTransaction()** : marque un **point de synchronisation avant** le lot de requêtes de mise à jour ;
- **PDO::commit()** : **suit la dernière commande SQL de mise à jour et demande la validation du lot** complet (ou l'annulation de ce qui aurait déjà été exécuté en partie depuis le dernier point de synchronisation) ;
- **PDO::rollback()** : **demande expressément l'annulation des mises à jour** exécutées depuis le dernier point de synchronisation ; cette dernière méthode est à utiliser en cas de détection d'erreur par des tests `if` ou (mieux) des structures `try/catch`.

La gestion des transactions peut être mise en œuvre aussi bien dans un script PHP que dans des *procédures stockées* ou *triggers* (qui sont des programmes à base de requêtes SQL exécutés par le SGBD lui-même).

Pour des exemples de mise en œuvre en PHP et pour approfondir la question :

<http://php.net/manual/fr/pdo.transactions.php>

<https://www.grafikart.fr/tutoriels/mysql/procedures-triggers-fonctions-593>



Une série de requêtes SQL sont logiquement liées entre elles et on voudrait qu'elles soient **toutes exécutées** ou **aucune**. En effet dans certains cas, la prise en compte d'une partie des requêtes seulement peut conduire à une incohérence dans le système d'information. La base de données peut ainsi être corrompue et très difficile à rectifier par la suite. Par exemple, si on a 2 requêtes qui se suivent et qui sont liées :

```
<?php

$pdo=new PDO('mysql:host=localhost;port=3308;dbname=Banque;charset=utf8',
'root', '', array(PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION));

$stmt1 = $pdo->prepare('
UPDATE compte
SET solde = solde - :montant
WHERE nom = :nom
');

$stmt2 = $pdo->prepare('
UPDATE compte
SET solde = solde + :montant
WHERE nom = :nom
');

// Retrait du Compte1
$cpte1 = 'Compte1';
$montant = 50;
$stmt1->bindParam(':nom', $cpte1);
$stmt1->bindParam(':solde', $montant, PDO::PARAM_INT);
$stmt1->execute();

// Crédit du Compte2
$cpte2 = 'Compte2';
$depot = 50;
$stmt2->bindParam(':nom', $cpte2);
$stmt2->bindParam(':montant', $depot, PDO::PARAM_INT);
$stmt2->execute();

?>
```

Ceci peut conduire à un problème en cas d'interruption de cette séquence. En particulier le Compte1 peut avoir été débité sans que le Compte2 soit crédité. On peut résoudre cette fragilité en utilisant une transaction :

```
<?php
```

```

$pdo=new PDO('mysql:host=localhost;port=3308;dbname=Banque;charset=utf8',
'root', '', array(PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION));

$stmt1 = $pdo->prepare('
UPDATE compte
SET solde = solde - :solde
WHERE nom = :nom
');

$stmt2 = $pdo->prepare('
UPDATE compte
SET solde = solde + :montant
WHERE nom = :nom
');

// On commence la transaction
$pdo->beginTransaction();

// Retrait du Compte1
$cpte1 = 'Compte1';
$montant = 100;
$stmt1->bindParam(':nom', $cpte1);
$stmt1->bindParam(':solde', $montant, PDO::PARAM_INT);
$stmt1->execute();

// Crédit du Compte2
$cpte2 = 'Compte2';
$depot = 50;
$stmt2->bindParam(':nom', $cpte2);
$stmt2->bindParam(':montant', $depot, PDO::PARAM_INT);
$stmt2->execute();

//on termine la transaction
$connexion -> commit();

?>

```

#### 4.3.2 Les erreurs d'exécution

En ce qui concerne le **traitement des erreurs**, le développeur a toujours 2 stratégies à sa disposition :

- **Eviter les erreurs**, en faisant des **tests avant** de lancer une opération risquée (if...) ou en **empêchant** cette opération (commande ou objet graphique non-disponible ou masqué par exemple) ;
- **Gérer l'erreur**, en protégeant une opération risquée dans un **bloc try/catch** afin de capturer l'erreur et de réagir en conséquence

Accéder à une Base de données en langage PHP à l'aide de PDO

Afpa © – Section Tertiaire Informatique – Filière « Etude et développement »

Avec PDO, il existe aussi une troisième voie : poursuivre son chemin comme si de rien n'était... Et c'est le mode de fonctionnement par défaut !

Voyons comment gérer tout cela de manière professionnelle et moderne.

Le langage PHP propose depuis toujours une **clause 'or die(...)'** disponible pour des opérations risquées afin de terminer proprement le script PHP en cas d'erreur (la fonction `die(...)` admet en paramètre du code HTML qui sera restitué à l'utilisateur).

Le langage PHP a introduit ensuite la **structure `try/catch`** (un grand '*classique*' des langages '*modernes*'), qui permet un '*débranchement*' vers un bloc d'instructions de gestion de l'erreur survenue dans le bloc `try{}.` En complément, l'interpréteur PHP instancie un objet `Exception` en cas d'erreur et cet objet est accessible dans le bloc `catch(){}.` et utilisable grâce principalement à ses méthodes `getCode()` et `getMessage()`.

Exemple de structure `try/catch` utilisant l'objet `Exception` :

```
try {  
    $connection = new PDO(...);  
    $connection->query(...);  
}  
catch (Exception $e) {  
    echo 'Échec lors de la connexion : ' . $e->getMessage();  
}  
... // la suite
```

Mais cela ne s'arrête pas là ! Un bloc `try {}` peut être suivi de plusieurs blocs `catch() {}`, chacun capturant un *type* d'erreur différent, la notion de *type* étant ici celle de *l'orienté objet* qui tient compte de la *hiérarchie d'héritage entre classes*. La classe `Exception` peut bien entendu être dérivée par le développeur. Ainsi, le premier bloc `catch() {}` doit capturer le type d'erreur le plus spécialisé, le dernier bloc `catch() {}` prenant en charge le type d'erreur le plus générique, soit le type `Exception`. Cette structure de la gestion d'erreur, plus complexe à mettre en œuvre, permet au script de réagir différemment selon l'erreur rencontrée lors de l'exécution.

**Pour aider le développeur à gérer les erreurs dues aux accès à la base de données, PDO introduit déjà une classe `PDOException` dérivée directement de la classe `Exception`.**



### Exemple de mise en œuvre avec capture de 2 types d'erreurs :

```
try {  
    $connection = new PDO(...);  
    $connection->query(...);  
}  
  
catch (PDOException $e) { // erreurs dues à l'accès à la BDD  
    echo 'Échec lors de la connexion : ' . $e->getMessage();  
}  
  
catch (Exception $e) { // autres erreurs  
    echo 'Erreur inattendue : ' . $e->getMessage();  
}  
  
... // la suite
```

Dans cet exemple simplifié, le script s'interrompra dans tous les cas d'erreur mais le message restitué à l'utilisateur sera différent. Il est bien évident que chaque bloc `catch() {}` devrait tenter de résoudre le problème avant d'afficher un message d'erreur final ; un bloc `catch() {}` peut donc contenir lui aussi une structure `try/catch...`

## 4.4 BONNES PRATIQUES AVEC PDO

Pour terminer, voici quelques points de repères pour programmer en PHP, de manière '*propre et pro*', les accès aux bases de données à l'aide du Framework PDO.

### 4.4.1 Isoler les instructions d'accès aux bases de données dans des scripts PHP séparés

Cette première recommandation, citée en 4.1, favorise la maintenabilité des applications.

### 4.4.2 Définir les paramètres de connexion dans des variables ou constantes PHP

Cette recommandation, citée aussi en 4.1, favorise de même la maintenabilité des applications.

### 4.4.3 Désactiver le mode *silencieux* de PDO en cas d'erreur

Comme le mode de fonctionnement par défaut de PDO masque les erreurs d'exécution, il est impératif d'activer la levée des `Exceptions` en cas de problème :

```
$connexion = new PDO($dsn, $user, $password);  
$connexion->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
```

Ou bien

```
$connexion = new PDO($dsn, $user, $password, array(PDO::ATTR_ERRMODE=>  
PDO::ERRMODE_EXCEPTION));
```

### 4.4.4 Protéger tous les accès aux bases de données par des structures `try/catch`

Ecrire au moins un bloc `catch() {}` capturant l'erreur générique `Exception`. Pour aller plus loin et si le problème peut être résolu, écrire au moins 2 blocs `catch() {}` comme cité plus haut en 4.3.3.

#### 4.4.5 Abuser des *alias SQL* et utiliser des syntaxes SQL génériques

Afin de nommer sans ambiguïté les données récupérées pour les insérer dans la page Web en cours de construction, il est nécessaire de donner des noms clairs et précis aux colonnes/expressions extraites par SQL grâce au mot-clé `as`.

Toujours afin de favoriser la maintenabilité des applications, il est vivement conseillé de fuir les instructions ou options spécifiques à un SGBD en particulier. Rappelons que cette recommandation pose problème dans les applications Web où il est courant de lister des données page par page, ce qui nécessite avec MySQL l'utilisation de la clause `limit x,y` spécifique à ce SGBD (rien n'est parfait...).

#### 4.4.6 Utiliser le système de requêtes préparées

Pour paramétrer les requêtes SQL, il est vivement recommandé d'utiliser des requêtes préparées plutôt que des concaténations plus ou moins hasardeuses.

#### 4.4.7 Faire exécuter le maximum de travail sur les données par le SGBD lui-même

Afin d'alléger la charge du serveur Web qui exécute les scripts PHP, il est vivement conseillé de faire exécuter sélections, calculs et autres regroupements par le SGBD, en langage SQL, afin qu'il restitue des informations précises directement exploitables par PHP. Attention de bien utiliser les alias SQL pour nommer correctement les résultats des calculs.

#### 4.4.8 Gérer les transactions

La base de données est souvent considérée comme le donjon du château fort médiéval, c'est-à-dire le '*dernier rempart*' où l'on stocke et protège les données sensibles des applications, à l'abri des erreurs de programmation et des attaques de *hackers*.

Identifier et bien gérer les transactions permet d'assurer une bonne intégrité des données stockées en BDD.

Une combinaison de ces 3 derniers principes conduit certaines équipes à invoquer systématiquement des procédures stockées au lieu d'exprimer les requêtes SQL dans les scripts PHP ; la maintenabilité des applications et la sécurité des données s'en trouvent alors renforcées.

## **CREDITS**

### **ŒUVRE COLLECTIVE DE l'AFPA**

**Sous le pilotage de la DIIP et du centre d'ingénierie sectoriel Tertiaire-Services**

### **Equipe de conception (IF, formateur, mediatiseur)**

Benoit Hézard - Formateur

Chantal. Perrachon – Ingénieure de formation

## **Reproduction interdite**

Article L 122-4 du code de la propriété intellectuelle.

« Toute représentation ou reproduction intégrale ou partielle faite sans le consentement de l'auteur ou de ses ayants droits ou ayants cause est illicite. Il en est de même pour la traduction, l'adaptation ou la reproduction par un art ou un procédé quelconque. »

Accéder à une Base de données en langage PHP à l'aide de PDO

Afpa © – Section Tertiaire Informatique – Filière « Etude et développement »