# Accès aux données dans PHP

## Principes

L’accès à des données est nécessaire pour assurer la persistance des informations manipulées, ainsi que leur cohérence.

Nous distinguerons :

* l’accès à des données structurées, dans une base de données. Nous traiterons de manière systématique ces points
* l’accès à des fichiers externes, par exemple CSV, le parcours de répertoire. Nous aborderons parfois ces éléments d’un point de vue pratique et moins systématique

## PHP et les moteurs de base de données

PHP est à l’origine très lié au moteur de base de données MySQL. Avec les évolutions apportées à PHP, les méthodes d’accès aux données ont permis des solutions plus génériques, pour utiliser :

* d’autres bases de données telles que Oracle, SQL Server, et
* des systèmes « No-SQL », c'est-à-dire pas seulement SQL, qui peuvent avoir des approches alternatives ou complémentaires
  + Document, comme MongoDB
  + Clef / valeur, comme redis

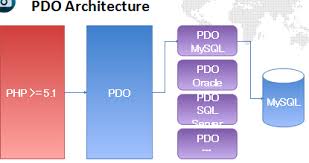
Par la suite, nous allons présenter et nous concentrer sur **PDO : PHP Data Objects**. En effet, l’approche alternative consistant à utiliser une extension de PHP spécifique à un SGBD, par exemple mysqli pour MySQL, ne nous satisfait pas.

# PDO – Définition, principes, utilisation

## Introduction à PDO. Architecture

PDO est une extension de PHP permettant de travailler de manière unifiée, quelle que soit le SGBD utilisé.

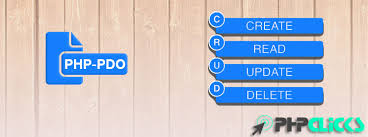
**Architecture des Drivers PDO**



PDO apporte une abstraction, et son approche Objet permet d’envisager d’étendre certaines fonctionnalités si besoin.

## Fonctionnalités

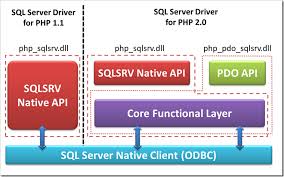
PDO supporte toutes les fonctionnalités CRUD.



Les données extraites peuvent être exploitées de différentes façons. Nous aborderons d’ailleurs dès aujourd’hui des méthodes génériques, pour se préparer à l’épreuve E5 (cf étude de cas).

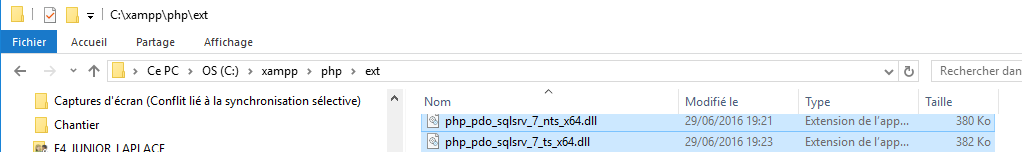
## Le driver PDO pour l’accès à SQL Server

**Schéma présentant les différents modes d’accès à SQL Server via PHP**



## Installation du driver PDO pour SQL Server

Etape 1 : on copie la bonne DLL dans le répertoire php\ext (par exemple C:\laragon\bin\php\php-7.1.14-Win32-VC14-x64\ext)



Pour être sûr des versions à installer et configurer dans PHP.ini, utiliser **phpinfo()** et examiner :

* Architecture : x86 ou x64
* Thread Safety : enabled (alors ts) ou non (nts)

Etape 2 : dans le fichier php.ini, on ajoute l’extension.

Exemple :

extension=php\_pdo\_sqlsrv\_71\_ts\_x86.dll

Avec ***Laragon*** : clic droit 🡪 PHP 🡪 Extension et cocher la DLL SQL Server

## Classes PDO

3 classes sont principalement liées à PDO :

* PDO : permet le lien et la connexion à la base de données
* PDOStatement :
  + gère les requêtes à soumettre, et
  + le résultat de ces requêtes
* PDOException : traite les erreurs générées par l’utilisation des classes précédentes

## Paramétrage de l’accès

Pour utiliser PDO pour le SGBD souhaité, on a besoin de définir 3 paramètres du constructeur de la classe PDO, qui sont : le DSN, le nom de l’utilisateur et le mot de passe.

Exemple de DSN pour SQL Server :

$dsn = "sqlsrv:Server=MUNIVIE\MUNIVIE,1433;Database=STAGE";

Dans le DSN, on a spécifié le nom du serveur, le port, et le nom de la base de données.

Concrètement, lors des TP, nous utiliserons une ***classe abstraite*** pour effectuer tout le paramétrage de notre base de données. Cf Exercices 1 et 2.

Question : Qu’est-ce qu’une classe abstraite ?

## Constantes pré définies et gestion des erreurs

Les constantes prédéfinies de la classe PDO sont documentées sur le lien suivant :

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

La gestion des erreurs peut être affinée en paramétrant certaines constantes : cf par exemple <http://php.net/manual/fr/pdo.error-handling.php>

Suivant les options configurées, on peut :

* Ne pas afficher les erreurs (par défaut)
* Utiliser le mode d’erreur classique, où des avertissements de niveau E\_WARNING seront émis en cas d’erreur
* Lever des exceptions (blocs try / catch à utiliser donc) pour chaque erreur de requête

Le mode le plus pertinent en phase de développement est le 3ème , qu’on paramètre de la façon suivante :

$conn->setAttribute( PDO::ATTR\_ERRMODE, PDO::ERRMODE\_EXCEPTION );

## Paramètres spécifiques de connexion

La liste des paramètres spécifiques de connexion dépend de la base de données utilisée. Par exemple pour SQL Server, la liste des constantes pré définies est disponible sur le lien suivant :

<http://php.net/manual/fr/ref.pdo-sqlsrv.php>

## Exécution de requêtes

~~Une fois que la connexion est disponible et configurée, on va pouvoir envoyer des requêtes au serveur, en utilisant une des 2 méthodes de l’objet PDO permettant d’effectuer des requêtes : query() ou exec()~~

* On préférera systématiquement des requêtes préparées : voir plus loin
* **Interdiction d’utiliser autre chose que PDO** (Mysqli …). Ne mélangez pas les 2.

## Choix de la nature et du type de retour des résultats

Par défaut, PDO utilise le mode PDO ::FETCH\_BOTH, ce qui veut dire que le résultat est disponible à la fois sous forme de :

* Tableau associatif où la clef est le nom de la colonne dans la base de données et la valeur la valeur associée de la colonne pour chaque ligne
* Tableau simple indexé (de 0 au nombre de colonnes du résultat – 1)

Exemple : cf dans le TP l’exercice: Affichage de données de la table SOFTWARE.REF\_LIB\_JEUX

Le lien ci-dessous donne la liste des options pour récupérer les données, via les constantes de la classe PDO permettant de choisir le type de retour : <http://php.net/manual/fr/pdostatement.fetch.php>

Un mode particulièrement intéressant est le mode FETCH\_CLASS : il permet de lier les colonnes du résultat de la requête à des attributs d’une classe PHP !

Cf l’exemple du TP, pour une utilisation qui reprend une table utilisée pour développer une procédure stockée SQL Server (ELEVES).

## Requête préparée

Comme en Java avec les classes JDBC, il est possible d’utiliser des requêtes préparées (à condition que le SGBD le permette), ce qui permet une optimisation et d’éviter l’injection SQL (sécurité).

Les requêtes préparées sont le mode préféré en Java et PHP pour récupérer ou modifier des données dans la base de données.

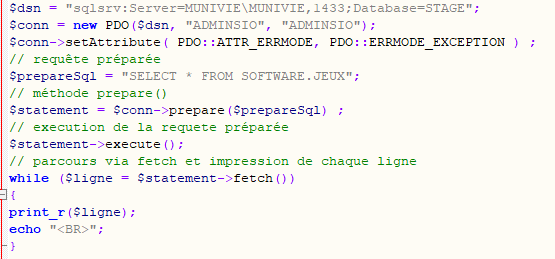
**Syntaxe** :

Pour préparer une requête, on utilise la méthode ***prepare()*** de la classe PDO

Pour exécuter une requête, on utilise la méthode ***execute()***

Ci-dessous un exemple complet de requête préparée.

Exemple de requête préparée



## Insertion de données avec une requête préparée

L’insertion des données via une requête préparée se fait en 3 étapes :

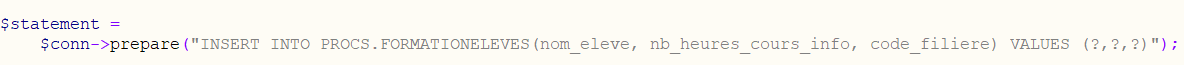
**Etape 1** : préparation de la requête selon une des formes suivantes :

1. Paramètre nommé
2. Paramètre positionnel

Exemple 1 : avec des paramètres nommés



Exemple 2 : avec des paramètres positionnels

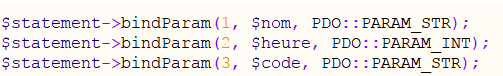


**Etape 2** : liaison (bind) des paramètres ou des positions avec une donnée PHP

Exemple 1 : avec des paramètres nommés



Exemple 2 : avec des paramètres positionnels



Utilisation de la méthode ***bindParam(***). Le 3ème paramètre, PDO ::PARAM\_STR etc, est optionnel et permet de préciser le type PHP pour effectuer le mapping avec la donnée dans la table du SGBD.

**Etape 3** : exécution du statement

Appel de la méthode ***execute(***)

Exemple :



## Un site de référence de haut niveau sur PDO

<https://phpdelusions.net/pdo>

Beaucoup d’exemples et de bons conseils.

## Annexe

## Requêtes avec résultat

Une requête renvoyant des résultats, comme SELECT (mais aussi DESC, SHOW ou EXPLAIN), nécessite l’utilisation de la méthode ***query()***

Pour une requête d’insertion, mise à jour ou suppression (INSERT, UPDATE ou DELETE), il faut utiliser la méthode ***exec()*** (qui exécute la requête sous jacente et renvoie juste le nombre de lignes concernées par la requête).

Après un ***query()*** , les données ne sont pas directement affichées. Le résultat est une *instance* de la classe PDOStatement.

Il faut utiliser une méthode comme ***fetchAll()*** ou ***fetch(***) pour parcourir et utiliser un objet de type PDOStatement

Exemple :

