Les fiches récap de l'école O'clock

Bdd

PDO

Dernière modification: 9 janvier 2023

PDO

PDO est une extension (objet) de PHP (à partir de la v5.1).

Se connecter à la base de données

doc PHP

le DSN (Data Source Name)

= informations relatives à la BD à laquelle on veut se connecter. Composé de :

- nom du pilote (mysql)
- adresse du serveur
- nom de la base de données

Connexion

= instanciation de la classe PDO. Utilise:

- le DSN
- les identifiants à utiliser pour se connecter (username et password)

```php
<?php
$host = "localhost";
$dbname = "test";
$user = "user";
$pass = "pass";

try {
  $db_connect = new PDO("mysql:host=" . $host . ";dbname=" . $dbname,
$user, $pass);
}
catch (PDOException $e) {
    die("Erreur en se connectant à la BD: " . $e->getMessage());
}
```

Les classes PDO et PDOStatement

- la classe PDO gère la connexion à la base
- la classe PDOStatement gère une requête et son jeu de résultats

Exécuter une requête

PDO::query sur php.net

Lecture dans la base: `SELECT`

`public PDOStatement PDO::query($query)`

Renvoie un objet de classe PDOStatement.

utilisation:

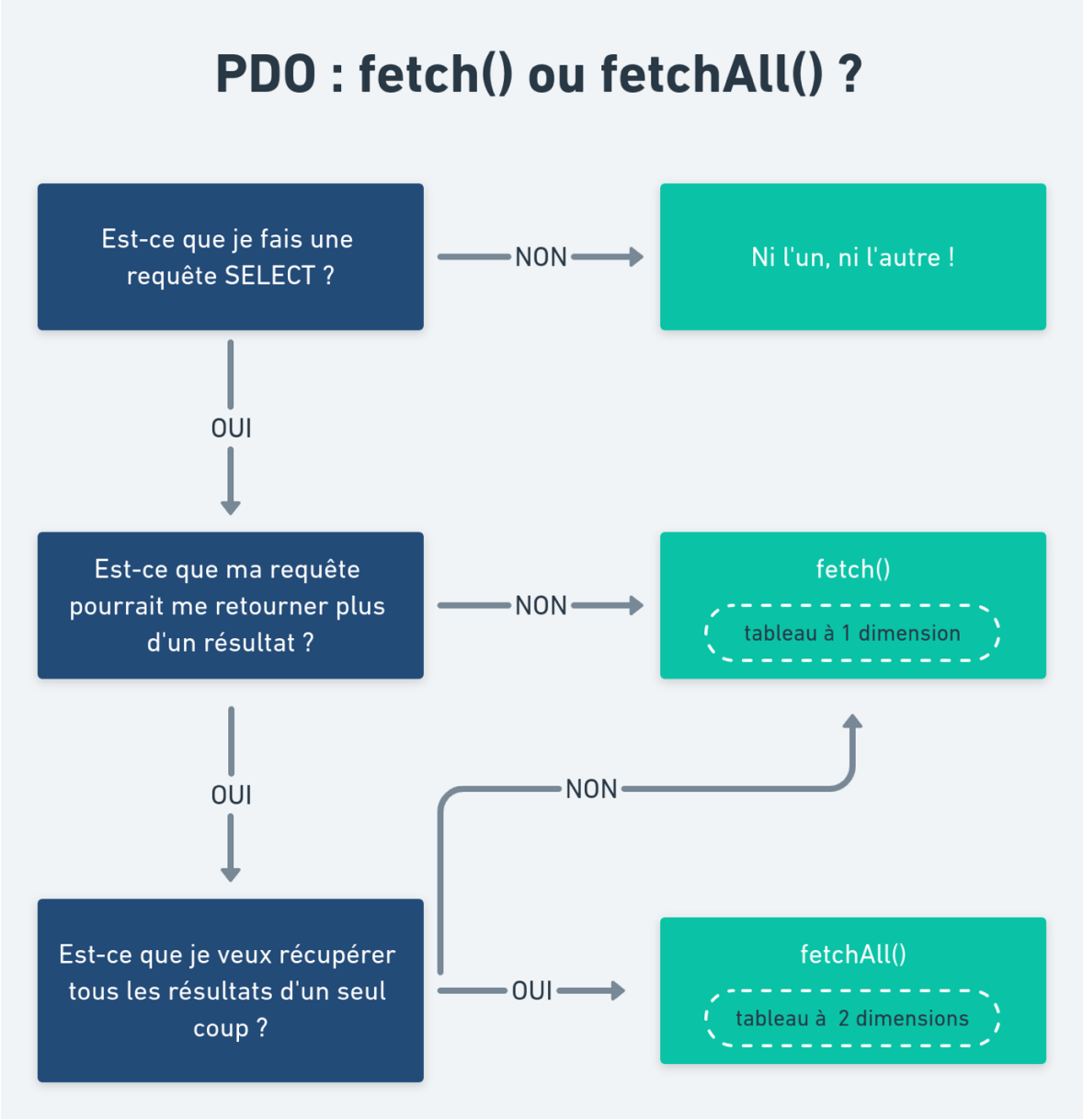
```php
$sql = "SELECT * FROM table";
$res_select = $db_connect->query($sql);
$resLine = $res_select->fetch();

// $resLine représente une ligne de la table de résultats de la requête
// C'est un tableau indexé (par les noms ET indices des champs par défaut)

echo $resLine['champ1'];
echo $resLine[0]; // même valeur
```

Exécuter des `SELECT` : les différents `fetch`

- `PDOStatement::fetch()`
- `PDOStatement::fetchColumn([$field])`
  utilisation:
  `$valChamp2 = $res_select->fetchColumn("champ2");`
- `PDOStatement::fetchAll()`
  `$res_array = $res_select->fetchAll();`

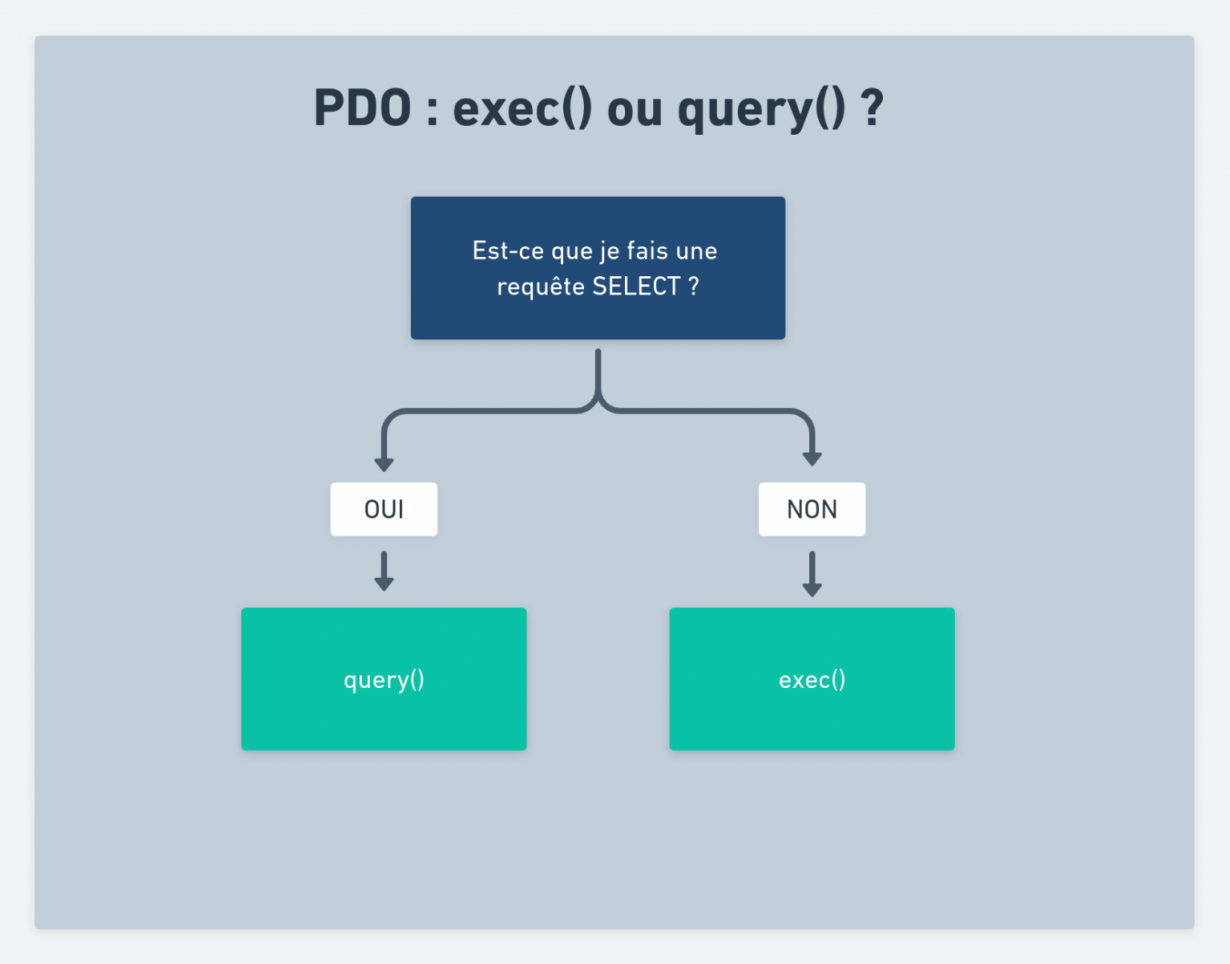


PDO::exec sur php.net

Pour modifier la base: `INSERT` , `UPDATE` , `DELETE`
`public int PDO::exec(string $statement)`
fonction de PDO qui renvoie le nombre de lignes affectées par la requête, ou false s'il y a eu une erreur dans l'éxécution de la requête.
(Si on utilise cette fonction, on ne bénéficie pas des fonctionnalités de protection des données de PDO.)

utilisation :

```
$res = $db_connect->exec('INSERT INTO table (champ1, champ2) VALUES ("value1", "value2")');
```



Les injections SQL

L'utilisateur malveillant « profite » d'une requête SQL du site qui utilise des `input` de visiteurs (formulaires: inscriptions, commentaires…) pour manipuler directement la base de données.

Les attaques *XSS – Cross Site Scripting*

Insertion de code malveillant (balises HTML + script JS) dans les pages consultées.

Type	Description	Contre-mesure
Persistante	Code inséré dans un champ texte. Destiné à être stocké, ce code sera ré-affiché sur les pages et	Traitement de tous les `input` utilisateurs

	consulté par d'autres visiteurs (commentaires par exemple)	
Non-persistante	Code inséré en paramètre d'une URL. Exécution immédiate pour agir sur la page (champ de recherche par exemple)	Encoder les paramètres utilisateurs affichés sur les pages. Comme `htmlspecialchars($_GET['searchString'])`

Requêtes préparées

Usage recommandé en cas de

- requêtes répétitives (Même requête avec changement de paramètres)
- requêtes utilisant des paramètres utilisateur (protection contre les injections)

Fonctionnement

1. Prépatation de la requête `PDO::prepare`, la requête SQL est enregistrée avec des paramètres nommés ou des marqueurs à la place des valeurs à utiliser
2. Exécution de la requête avec `PDOStatement::execute`, les paramètres sont remplacés par leurs valeurs, soit en passant par un tableau de valeurs dans la fonction, soit en ayant au préalable utilisé `bindParam` ou `bindValue`

Différence entre `bindParam` et `bindValue`

`bindValue`

Documentation

On relie notre token à la ***valeur*** de la variable utilisée dans `bindValue()`, au moment où elle est utilisée.

Par conséquent, si elle est modifiée par la suite, ça n'a pas d'impact sur la requête SQL réellement exécutée.

Ex :

```
$email = 'toto@oclock.io'; // on déclare $email
$sql = 'SELECT * FROM `app_user` WHERE `email` = :email';
$pdoStatement = $pdo->prepare($sql);
$pdoStatement->bindValue(':email', $email, PDO::PARAM_STR);
$email = 'tata@oclock.io'; // on modifie $email
$pdoStatement->execute();
// ici, on exécute "SELECT * FROM `app_user` WHERE `email` =
'toto@oclock.io'"
// la modification de $email n'a eu aucun impact
```

`bindParam`

Documentation

Cette fois-ci, c'est la variable qu'on associe au token. La conséquence, c'est que si la valeur de cette variable est modifiée avant l'appel à la méthode `PDOStatement::execute()`, cela aura un impact sur la requête réellement exécutée.

Ex :

```
$email = 'toto@oclock.io'; // on déclare $email
$sql = 'SELECT * FROM `app_user` WHERE `email` = :email';
$pdoStatement = $pdo->prepare($sql);
$pdoStatement->bindParam(':email', $email, PDO::PARAM_STR);
$email = 'tata@oclock.io'; // on modifie $email
$pdoStatement->execute();
// ici, on exécute "SELECT * FROM `app_user` WHERE `email` =
'tata@oclock.io'"
```

`bindParam` assigne une variable (passage par référence) dont la valeur ne sera évaluée que lors de l'appel de `execute`

PDO::prepare sur php.net

Paramètres nommés

> la fonction retourne un obj. PDOStatement (comme exec & query)

```php
$calories = 150;
$couleur = 'rouge';

$sth = $db_connect->prepare('SELECT nom, couleur, calories
  FROM fruit
  WHERE calories < :calories AND couleur = :couleur');

$sth->bindParam(':calories', $calories, PDO::PARAM_INT);
$sth->bindParam(':couleur', $couleur, PDO::PARAM_STR, 12);

$sth->execute();
```

Marqueurs

```php
$calories = 150;
$couleur = 'rouge';

$sth = $db_connect->prepare('SELECT nom, couleur, calories
  FROM fruit
  WHERE calories < ? AND couleur = ?');

$sth->bindValue(1, $calories, PDO::PARAM_INT);
$sth->bindValue(2, $couleur, PDO::PARAM_STR);

$sth->execute();
```

Choisir la bonne méthode PDO pour les requêtes

query et prepare renvoient toujours un objet `PDOStatement` lecture des resultats par fetch et fetchAll

exec()

Pour les requetes qui MODIFIENT la base de données : `INSERT` , `UPDATE` , `DELETE` avec des données « **sûres**« , que l'on maîtrise.

Par exemple pour la sauvegarde d'une table

query()

Pour les requêtes de LECTURE : `SELECT` qui ne prennent **aucun paramètre**

Par exemple pour lister tous les enregistrements d'une table

> Renvoie un objet `PDOStatement` qui pourra être utilisé pour lire les résultats avec `fetch()` ou `fetchAll()`

prepare()

Pour toutes les autres requêtes

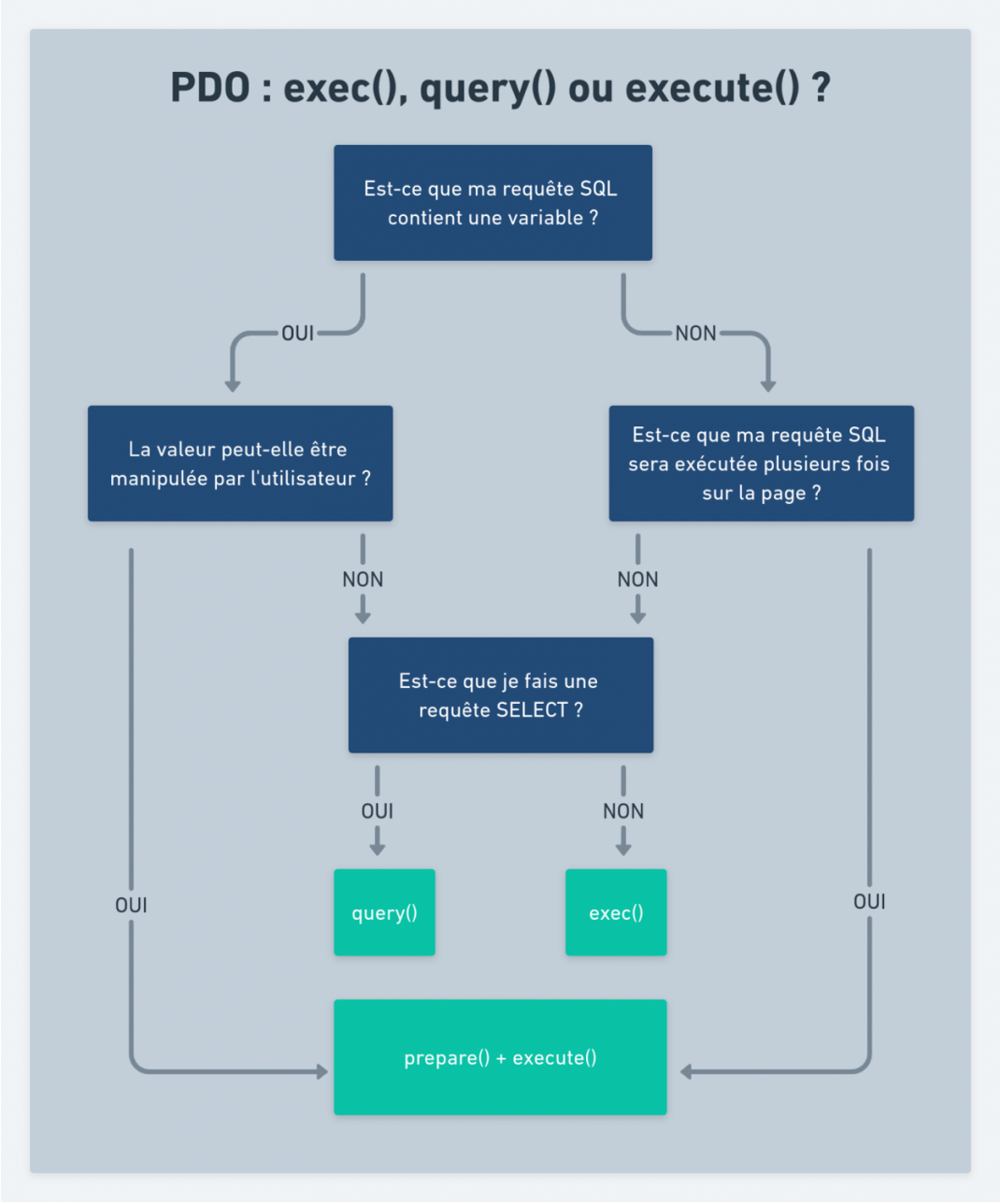
- LECTURE `SELECT` avec des paramètres 'utilisateur'

Par exemple trouver un enregistrement après une recherche d'un utilisateur

- MODIFICATION `INSERT` , `UPDATE` , `DELETE` avec des paramètres 'utilisateur'

Par exemple insérer un commentaire en base de données

> Renvoie un objet `PDOStatement` qui pourra être utilisé pour lire les résultats avec `fetch()` ou `fetchAll()`


PDO : exec(), query() ou execute() ?
Est-ce que ma requête SQL contient une variable ?
OUI
NON
La valeur peut-elle être manipulée par l'utilisateur ?
Est-ce que ma requête SQL sera exécutée plusieurs fois sur la page ?
NON
NON
Est-ce que je fais une requête SELECT ?
OUI
NON
OUI
query()
exec()
OUI
prepare() + execute()