Libs Docs

Bastien Helec

09/07/2025

PHP_BDD

```
Bastien Helec
Description : Explication des classes BDD de la section PHP

09-07-2025
```

Sommaire

Model

Description :

La partie BDD de PHP permet d'utiliser tous les outils de base pour pouvoir faire de la communications avec la base de données:

- libelle.php ⇒ Libelle
- pdo.php ⇒ DBPDO

Libelle **Description** :

La classe `Libelle` permet de gérer dynamiquement les valeurs d'une colonne spécifique d'une table dans la base de données. Elle facilite la récupération, l'ajout et la suppression de libellés (valeurs) dans une table donnée.

Arguments du constructeur :

- `$table` : Nom de la table à cibler.
- `$colonne` : Nom de la colonne à manipuler.
- `$pdo` : Instance PDO pour la connexion à la base de données.

Exemple d'utilisation:

```php
<?php
// Instanciation de la classe pour la table "categories" et la colonne "nom"
$libelle = new Libelle('categories', 'nom', $pdo);
```

```php
// Récupérer tous les libellés
$liste = $libelle->getLibelle();

// Ajouter un nouveau libellé
$libelle->setLibelle('Nouveau Libellé');

// Supprimer un libellé
$libelle->deleteLibelle('Ancien Libellé');
?>
```

Cela produira :

```sql
-- Récupérer les libellés
SELECT nom FROM categories

-- Ajouter un nouveau libellé
INSERT INTO categories (nom) VALUES ('Nouveau Libellé')

--Supprimer un libellé
DELETE FROM categories WHERE nom = 'Ancien Libellé'
```

Méthodes principales :

- `getLibelle()` : Récupère toutes les valeurs de la colonne spécifiée dans la table sous forme de tableau associatif `fetchAll`.
- `setLibelle($libelle)` : Ajoute une nouvelle valeur dans la colonne de la table.
- `deleteLibelle($libelle)` : Supprime une valeur spécifique de la colonne dans la table.

DBPDO Description :

La classe `DBPDO` encapsule la connexion à la base de données MySQL avec PDO, en simplifiant la configuration et la gestion de la connexion. Elle permet d'instancier une connexion PDO robuste et réutilisable.

Arguments du constructeur :

- `$HOST` : Adresse du serveur de base de données.
- `$DBNAME` : Nom de la base de données.
- `$USER` : Nom d'utilisateur pour la connexion.
- `$PASSWORD` : Mot de passe pour la connexion.
- `$PORT` : Port de connexion à la base de données.

Exemple d'utilisation :

```php
<?php
/ Instanciation de la classe avec les paramètres de connexion
$db = new DBPDO('localhost', 'ma_base', 'user', 'password', '3306');

// Connexion à la base de données
$pdo = $db->connect();
?>
```

Méthodes principales :

- `connect()` : Établit et retourne une connexion PDO configurée (mode exception, encodage utf8mb4, fetch associatif par défaut). Si la connexion existe déjà, elle est réutilisée.

SQL La section SQL contient le CRUDS (Create Replace Update Delete Select):

- Delete.php ⇒ Delete_SQL
- Insert.php ⇒ Insert_SQL
- Select.php ⇒ Select_SQL
- Update.php ⇒ Update_SQL

Delete_SQL :

Description :

La classe `Delete_SQL` permet de générer et d'exécuter des requêtes SQL de suppression (`DELETE`) sur une table donnée, avec des conditions simples ou complexes.

Arguments du constructeur :

- `$table` : Nom de la table sur laquelle effectuer la suppression.

Exemple d'utilisation :

```php
<?php
$delete = new Delete_SQL('categories');
// Suppression simple
$delete->execute_Simple_SQL("nom = 'Ancien Libellé'", $pdo);
// Suppression complexe (exemple)
$delete->execute_Cmplx_SQL('USING autre_table', 'WHERE categories.id = autre_table.cat_id', $pdo);
?>
```

Cela produira :

```sql
DELETE FROM categories WHERE nom = 'Ancien Libellé'
DELETE FROM categories USING autre_table WHERE categories.id = autre_table.cat_id
```

Méthodes principales :

- `execute_Simple_SQL($cdt, $pdo)` : Exécute une requête DELETE avec une condition simple (ex : `id=5`).
- `execute_Cmplx_SQL($cdt_plus, $cdts, $pdo)` : Exécute une requête DELETE avec des conditions ou jointures complexes.

Insert_SQL :

Description :

La classe `Insert_SQL` permet de générer et d'exécuter des requêtes SQL d'insertion (`INSERT`) dans une table, pour une ou plusieurs valeurs.

Arguments du constructeur :

`$columns` : Nom(s) de la ou des colonnes à remplir.

`$table` : Nom de la table cible

Exemple d'utilisation:

```php
<?php
$insert = new Insert_SQL('nom', 'categories');
// Insertion simple
$insert->execute_Simple_SQL('Nouveau Libellé', $pdo);
// Insertion multiple
$insert->execute_Simple_SQL(['Libellé 1', 'Libellé 2'], $pdo);
?>
```

Cela produira alors :

```
INSERT INTO categories (nom) VALUES ('Nouveau Libellé')
INSERT INTO categories (nom) VALUES (?, ?) -- puis binding des valeurs
```

Méthodes principales:

`execute_Simple_SQL($val, $pdo)` : Exécute une requête INSERT simple (une ou plusieurs valeurs).

`execute_Cmplx_SQL($val, $cdt_plus, $cdts, $pdo)` : Exécute une requête INSERT complexe (avec options supplémentaires).

`cdt_plus` étant l'argument après les colonnes.

`cdts` toutes les autres conditions comme : `where id=1` etc etc.

Select_SQL :

Description :

La classe `Select_SQL` permet de générer et d'exécuter des requêtes SQL de sélection (`SELECT`) sur une table, avec des conditions simples ou complexes.

Arguments du constructeur :

`$field` : Nom(s) du ou des champs à sélectionner.

`$table` : Nom de la table cible.

Exemple d'utilisation:

```php
<?php
$select = new Select_SQL('nom', 'categories');
// Sélection simple
$result = $select->execute_Simple_SQL("nom LIKE 'N%'", $pdo);
// Sélection complexe
$resultAll = $select->execute_Cmplx_fetchAll_SQL('WHERE nom IS NOT NULL', $pdo);
?>
```

Cela produira :

```
SELECT nom FROM categories WHERE nom LIKE 'N%'
SELECT nom FROM categories WHERE nom IS NOT NULL
```

Méthodes principales :

`execute_Simple_SQL($where, $pdo)` : Exécute une requête SELECT avec une condition WHERE simple.

`execute_Cmplx_fetch_SQL($cdts, $pdo)` : Exécute une requête SELECT complexe et retourne le dernier résultat uniquement.

`execute_Cmplx_fetchAll_SQL($cdts, $pdo)` : Exécute une requête SELECT complexe **Identique a** `execute_Cmplx_fetch_SQL` et retourne tous les résultats.

Update_SQL :

Description :

La classe `Update_SQL` permet de générer et d'exécuter des requêtes SQL de mise à jour (`UPDATE`) sur une table, pour des cas simples (plusieurs colonnes) ou complexes (avec jointures ou conditions avancées).

Arguments du constructeur :

`$table` : Nom de la table à mettre à jour.

`$after_set` : Tableau des colonnes à mettre à jour (cas simple) ou chaîne de la clause SET (cas complexe).

`$values` : Tableau ou chaîne des valeurs à appliquer.

Exemple d'utilisation :

```php
<?php
// Mise à jour simple
$update = new Update_SQL('categories', ['nom'], ['nom' => 'Libellé Modifié']);
$update->execute_Simple_SQL("nom = 'Ancien Libellé'", $pdo);

// Mise à jour complexe
$updateC = new Update_SQL('categories', 'nom = "Libellé Modifié"', []);
$updateC->execute_Cmplx_SQL('INNER JOIN autre_table ON ...', 'WHERE ...', $pdo);
?>
```

Cela produira :

```sql
UPDATE categories SET nom = 'Libellé Modifié' WHERE nom = 'Ancien Libellé'
UPDATE categories INNER JOIN autre_table ON ... SET nom = "Libellé Modifié"
```

Méthodes principales :

`execute_Simple_SQL($cdt, $pdo)` : Exécute une requête UPDATE simple avec une condition WHERE.

`execute_Cmplx_SQL($join, $sup_cdts, $pdo)` : Exécute une requête UPDATE complexe (avec jointure, etc.).

`$join` A rentrer si présence d'une jointure sinon laissé vide. `$sup_cdts` Les conditions supplémentaires.

Vue-Implémentation

La globalités des classes présentes ne sont pas implémentables en vue étant donnés la nature SQL de ce qui est générer.

Néanmoins pour `Libelle.php` il existe une vue `Libelles/afficher.view.php`

- Si le tableau `$Libelle` n'est pas vide, on parcourt chaque ligne pour afficher un `<li>`.
- Chaque `<li>` possède un **id** et un **attribut** `data-value` correspondant à la valeur du libellé, ce qui facilite la sélection ou l'interaction côté JavaScript.
- Les underscores (`_`) dans les libellés sont remplacés par des espaces pour un affichage plus lisibles.