Formation Développement Web : Backend : PHP

Partie : 4

I. Connexion PHP à MySQL

La connexion entre PHP et MySQL est essentielle pour interagir avec des bases de données dans le développement web. Deux méthodes couramment utilisées sont MySQLi et PDO, offrant des fonctionnalités sécurisées et avancées. MySQLi propose une interface orientée objet et procédurale, tandis que PDO offre une abstraction de base de données uniforme. Évitant la désormais obsolète extension `mysql_`, ces approches garantissent une meilleure sécurité et sont compatibles avec les dernières versions de PHP. La connexion MySQL-PHP facilite le stockage, la récupération et la manipulation des données, constituant ainsi un élément fondamental du développement web dynamique.

A. MySQLi (MySQL Improved)

MySQLi est une extension améliorée de MySQL pour PHP, introduite pour offrir une interface orientée objet et procédurale. Voici un exemple de connexion avec MySQLi :

1. Exemple

```
<?php
    $serveur = "localhost";
    $utilisateur = "nom_utilisateur";
    $mot_de_passe = "mot_de_passe";
    $base_de_donnees = "nom_base_de_donnees";

    // Connexion à MySQLi
    $connexion = new mysqli($serveur, $utilisateur, $mot_de_passe, $base_de_donnees);

    // Vérifier la connexion
    if ($connexion->connect_error) {
       die("Échec de la connexion : " . $connexion->connect_error);
    }

    // Utilisation de la connexion...

    // Fermer la connexion
    $connexion->close();
?>
```

B. PDO (PHP Data Objects)

PDO est une extension PHP qui fournit une interface uniforme pour accéder à des bases de données, y compris MySQL. Voici un exemple de connexion avec PDO :

```
<?php
    $serveur = "localhost";
    $utilisateur = "nom_utilisateur";
    $mot_de_passe = "mot_de_passe";
    $base_de_donnees = "nom_base_de_donnees";

    try {
       // Connexion à PDO
       $connexion = new PDO("mysql:host=$serveur;dbname=$base_de_donnees", $utilisateur,
    $mot_de_passe);

       // Définir le mode d'erreur sur Exception
       $connexion->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

       // Utilisation de la connexion...

    } catch (PDOException $e) {
       die("Échec de la connexion : " . $e->getMessage());
    }

    // Fermer la connexion (PDO se ferme automatiquement à la fin du script)
?>
```

II. Compréhension de PDO

A. Présentation de PDO (PHP Data Objects)

PDO, ou PHP Data Objects, constitue une interface d'accès aux bases de données pour PHP. Cette extension offre une approche orientée objet, fournissant une abstraction de base de données uniforme pour interagir avec divers systèmes de gestion de bases de données, dont MySQL.

B. Avantages de PDO par rapport à MySQLi

- **Polyvalence** : PDO prend en charge plusieurs types de bases de données, offrant ainsi une flexibilité accrue pour travailler avec différentes plateformes.
- **Interface Uniforme** : PDO offre une interface cohérente pour accéder aux bases de données, facilitant la transition entre différents systèmes de gestion.
- **Sécurité Renforcée** : Utilisation de requêtes préparées pour prévenir les attaques par injection SQL, renforçant la sécurité des applications.
- **Gestion des Erreurs Améliorée** : PDO utilise le mécanisme d'exceptions, simplifiant la gestion des erreurs et améliorant la lisibilité du code.
- **Support des Transactions** : PDO prend en charge les transactions, permettant d'effectuer plusieurs opérations de base de données de manière atomique.

III. Configuration de la Connexion

A. Paramètres de Connexion

Avant d'établir une connexion avec PDO, il est essentiel de définir les paramètres nécessaires :

- **Hôte ($serveur)** : Adresse du serveur de base de données.
- **Utilisateur ($utilisateur)** : Nom d'utilisateur pour accéder à la base de données.
- **Mot de passe ($mot_de_passe)** : Mot de passe associé à l'utilisateur.
- **Base de Données ($base_de_donnees)** : Nom de la base de données à laquelle se connecter.

```
<?php
        $serveur = "localhost";
        $utilisateur = "nom_utilisateur";
        $mot_de_passe = "mot_de_passe";
        $base_de_donnees = "nom_base_de_donnees";

        try {
           // Connexion à PDO
           $connexion = new PDO("mysql:host=$serveur;dbname=$base_de_donnees", $utilisateur,
        $mot_de_passe);

           // Définir le mode d'erreur sur Exception
           $connexion->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

           // Utilisation de la connexion...

        } catch (PDOException $e) {
           die("Échec de la connexion : " . $e->getMessage());
        }

        // Fermer la connexion (PDO se ferme automatiquement à la fin du script)
?>
```

IV. Exécution de Requêtes SQL avec PDO

A. Préparation et exécution d'une requête SQL

On prépare la requête SQL en utilisant la méthode prépare de l'objet PDO ($connexion). La requête contient deux paramètres nommés :valeur1 et :valeur2. Ces paramètres serviront à accueillir des valeurs lors de l'exécution.

```
< ?php
        $requeteSql = $connexion->prepare("REQUETTE SQL AVEC :valeur1 ET :valeur2") ;
?>
```

B. Liaison des valeurs aux paramètres :

On utilise la méthode bindValue pour lier les valeurs aux paramètres de la requête préparée.

- **:valeur1** est lié à la variable $valeur1 avec le type de données PDO::PARAM_STR (chaîne de caractères).
- **:valeur2** est lié à la variable $valeur2 avec le type de données PDO::PARAM_INT (entier).

1. Les type de paramètre

- **PDO::PARAM_STR** : Représente le type chaîne de caractères.
- **PDO::PARAM_INT** : Représente le type entier.
- PDO::PARAM_BOOL : Représente le type booléen.
- **PDO::PARAM_NULL** : Représente le type NULL.

```
< ?php
        $requeteSql ->bindValue(':valeur1', $valeur1, PDO::PARAM_STR);
        $requeteSql ->bindValue(':valeur2', $valeur2, PDO::PARAM_INT);
?>
```

C. Exécution de la requête :

On exécute la requête préparée avec les valeurs liées en utilisant la méthode **execute()**.

```
< ?php
        $requeteSql->execute() ;
?>
```

V. Exécution des Requêtes SQL (INSERT, SELECT, UPDATE, DELETE)

A. Requête INSERT

```
< ?php
        $requeteInsert = $connexion->prepare("INSERT INTO ma_table (nom, age, ville) VALUES
        (:nom, :age, :ville)");

        $requeteInsert->bindValue(':nom', $nom, PDO::PARAM_STR);
        $requeteInsert->bindValue(':age', $age, PDO::PARAM_INT);
        $requeteInsert->bindValue(':ville', $ville, PDO::PARAM_STR);

        $requeteInsert->execute();
?>
```

B. Requête SELECT

```
< ?php
    $requeteSelect = $connexion->prepare("SELECT * FROM ma_table");

    $requeteSelect->execute();

    // Récupération des résultats
    $resultats = $requeteSelect->fetchAll(PDO::FETCH_ASSOC);
    // Utilisez $resultats comme nécessaire
?>
```

C. Requête UPDATE

```
< ?php
    $requeteUpdate = $connexion->prepare("UPDATE ma_table SET ville = :nouvelleVille
    WHERE id = :id");

    $requeteUpdate->bindValue(':nouvelleVille', $nouvelleVille, PDO::PARAM_STR);
    $requeteUpdate->bindValue(':id', $id, PDO::PARAM_INT);

    $requeteUpdate->execute();
?>
```

D. Requête DELETE

```
< ?php
    $requeteDelete = $connexion->prepare("DELETE FROM ma_table WHERE id =
    :idToDelete");

    $requeteDelete->bindValue(':idToDelete', $idToDelete, PDO::PARAM_INT);

    $requeteDelete->execute();
?>
```

VI. Sécurité et Bonnes Pratiques

A. Prévention des Injections SQL

Utilisez des requêtes préparées pour éviter les injections SQL. Ne concaténez jamais directement des valeurs dans vos requêtes.

1. Mauvaise pratique

```
< ?php
    $requete = "SELECT * FROM utilisateurs WHERE nom = '" . $nom . "'";
?>
```

2. Utilisez une requête préparée

```
< ?php
    $requetePrep = $connexion->prepare("SELECT * FROM utilisateurs WHERE nom = :nom");
    $requetePrep->bindValue(':nom', $nom, PDO::PARAM_STR);
    $requetePrep->execute();
?>
```

B. Fermeture Appropriée des Connexions

Fermez les connexions de base de données correctement pour éviter des problèmes potentiels.

```
< ?php
    // Fermeture de la connexion
    $connexion = null;

    // Ou utilisez unset
    unset($connexion);
?>
```