Cours détaillé sur PDO en PHP et connexion à MySQL

1. Introduction à PDO

PDO (**PHP Data Objects**) est une extension de PHP qui permet d'accéder à des bases de données de manière **sécurisée et orientée objet**.

PDO offre plusieurs avantages :

- **Sécurité** : Protection contre les injections SQL grâce aux requêtes préparées.
- **Portabilité** : Il permet de se connecter à différentes bases de données (MySQL, PostgreSQL, SQLite, etc.).
- **Performances** : Il est optimisé et plus performant que l'extension `mysqli`.

2. Connexion à une base de données MySQL avec PDO

2.1. Paramètres de connexion

Pour se connecter à une base de données, nous avons besoin des informations suivantes :

- **Hôte** : `localhost` (ou une adresse IP)
- **Nom de la base de données** : Exemple `test_db`
- **Nom d'utilisateur** : Exemple `root`
- **Mot de passe** : Exemple `password`

2.2. Exemple de connexion

Voici un exemple de connexion à une base de données MySQL en utilisant PDO :

```
<?php
$host = "localhost";
$dbname = "test_db";
$username = "root";
$password = "";

try {
    // Création de la connexion PDO
    $pdo = new PDO("mysql:host=$host;dbname=$dbname;charset=utf8",
```

```
$username, $password);

    // Configuration pour afficher les erreurs PDO
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    // Mode de récupération en tableau associatif
    $pdo->setAttribute(PDO::ATTR_DEFAULT_FETCH_MODE,
PDO::FETCH_ASSOC);

    echo "Connexion réussie !";
} catch (PDOException $e) {
    die("Erreur de connexion : " . $e->getMessage());
}
?>
```

Explication du code

- `new PDO(...)` : Initialise une connexion à MySQL.
- `setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION)` : Active l'affichage des erreurs sous forme d'exceptions.
- `setAttribute(PDO::ATTR_DEFAULT_FETCH_MODE, PDO::FETCH_ASSOC)` : Permet de récupérer les résultats sous forme de tableau associatif.
- `catch (PDOException $e)` : Capture les erreurs et les affiche.

3. Exécuter des requêtes SQL avec PDO

PDO permet d'exécuter des requêtes de deux manières :

1. **Requêtes directes** (`query()`)
2. **Requêtes préparées** (`prepare()` et `execute()`)

3.1. Exécuter une requête directe (SELECT)

```
<?php
$sql = "SELECT * FROM users";
$statement = $pdo->query($sql);
$users = $statement->fetchAll(); // Récupère tous les résultats

foreach ($users as $user) {
    echo $user['id'] . " - " . $user['name'] . "<br>";
```

```
}
?>
```

Ne pas utiliser cette méthode avec des données provenant de l'utilisateur (risque d'injection SQL).

4. Requêtes préparées avec PDO (sécurisées)

Les **requêtes préparées** permettent d'éviter les injections SQL en utilisant des **paramètres liés**.

4.1. Sélectionner des données avec `prepare()`

```
<?php
$sql = "SELECT * FROM users WHERE email = :email";
$stmt = $pdo->prepare($sql);
$stmt->execute(['email' => 'exemple@email.com']);
$user = $stmt->fetch();

if ($user) {
    echo "Utilisateur trouvé : " . $user['name'];
} else {
    echo "Aucun utilisateur trouvé.";
}
?>
```

Ici, `:email` est un **paramètre lié** qui est sécurisé.

5. Insérer des données avec PDO

5.1. Requête d'insertion

```
<?php
$sql = "INSERT INTO users (name, email, password) VALUES (:name,
:email, :password)";
$stmt = $pdo->prepare($sql);
```

```
$stmt->execute([
    'name' => 'Aicha diop',
    'email' => 'aicha@example.com',
    'password' => password_hash('123456', PASSWORD_DEFAULT) //
Hachage du mot de passe
]);

echo "Utilisateur ajouté avec succès !";
?>
```

6. Mettre à jour des données

```
<?php
$sql = "UPDATE users SET name = :name WHERE id = :id";
$stmt = $pdo->prepare($sql);
$stmt->execute([
    'name' => 'Mariama modifié’,
    'id' => 1
]);

echo "Utilisateur mis à jour.";
?>
```

7. Supprimer des données

```
<?php
$sql = "DELETE FROM users WHERE id = :id";
$stmt = $pdo->prepare($sql);
$stmt->execute(['id' => 1]);

echo "Utilisateur supprimé.";
?>
```

8. Transactions avec PDO

Une transaction est utile pour assurer l'intégrité des données en cas d'erreur.

8.1. Exemple d'utilisation des transactions

```
<?php
try {
    $pdo->beginTransaction();

    $pdo->prepare("INSERT INTO users (name, email) VALUES ('Aicha',
'aicha@example.com')")->execute();
    $pdo->prepare("INSERT INTO users (name, email) VALUES ('Diop',
'diop@example.com')")->execute();

    $pdo->commit(); // Valider les changements
    echo "Transaction réussie !";
} catch (Exception $e) {
    $pdo->rollBack(); // Annuler les changements en cas d'erreur
    echo "Échec de la transaction : " . $e->getMessage();
}
?>
```

9. Gérer les erreurs PDO

9.1. Mode d'affichage des erreurs

PDO propose trois modes de gestion des erreurs :

1. **ATTR_ERRMODE, PDO::ERRMODE_SILENT** → Mode silencieux (aucune erreur affichée).
2. **ATTR_ERRMODE, PDO::ERRMODE_WARNING** → Affiche un avertissement (`E_WARNING`).
3. **ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION** → Lève une exception (recommandé).

Exemple :

```
$pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
```

10. Fermer la connexion PDO

En PHP, PDO **ferme automatiquement la connexion** à la base de données lorsqu'il n'est plus utilisé.

Cependant, on peut explicitement fermer la connexion en assignant `null` à l'objet PDO :

```
$pdo = null;
```

Résumé du cours

Fonction	Description
`new PDO(...)`	Créer une connexion à la base de données
`setAttribute(...)`	Configurer PDO (erreurs, mode de récupération)
`query()`	Exécuter une requête SQL simple
`prepare()` + `execute()`	Exécuter une requête préparée sécurisée
`fetch()`	Récupérer une ligne de résultats
`fetchAll()`	Récupérer plusieurs lignes de résultats
`beginTransaction()` + `commit()`	Gérer les transactions
`rollBack()`	Annuler une transaction en cas d'erreur

Ce cours couvre les bases de **PDO en PHP** pour une connexion sécurisée avec **MySQL**.