PHP : Bases de données

Quand tu développes un site web, tu dois souvent enregistrer et retrouver des informations. Pour faire cela de manière efficace et organisée, on utilise une base de données. En PHP, on peut communiquer avec une base de données MySQL pour **ajouter**, **lire**, **modifier** ou **supprimer** des informations (*CRUD*). Dans ce cours, tu vas apprendre pas à pas comment connecter ton code PHP à une base de données, comment envoyer des requêtes, et comment afficher ou enregistrer des résultats — tout cela en suivant les bonnes pratiques du métier.

🎯 Objectif de la leçon

À la fin de cette leçon, tu seras capable de :

- Te connecter à une base de données MySQL depuis un script PHP
- Lire et écrire des données avec des requêtes SQL
- Utiliser de bonnes pratiques (sécurité, structure du code)
- Comprendre ce que tu fais, et pourquoi tu le fais

🧠 Notions essentielles à retenir

- Une base de données permet de **stocker et organiser des données**.
- Pour interagir avec MySQL en PHP, on utilise souvent **PDO** (PHP Data Objects).
- Il est **obligatoire de sécuriser** les données utilisateurs (injections SQL).
- On **centralise** la connexion pour **éviter les répétitions** et faciliter la maintenance du code.

⚙️ Préparer la base de données

Tu dois d'abord avoir :

- Un **serveur local** installé (XAMPP, MAMP, WAMP, etc.)
- Une base de données créée avec ce SQL :

```sql
CREATE DATABASE cours_php;

USE cours_php;

CREATE TABLE utilisateurs (
  id INT AUTO_INCREMENT PRIMARY KEY,
  nom VARCHAR(100),
  email VARCHAR(100)
);
```

Cette table `utilisateurs` va stocker un `id`, un `nom`, et un `email`.

🧩 Étape 1 : Se connecter à la base (fichier `config/db.php`)

```php
<?php
$host = 'localhost';              // Nom du serveur MySQL (localhost si c'est sur ton PC)
$dbname = 'cours_php';            // Nom de ta base de données
$username = 'root';               // Nom d'utilisateur MySQL par défaut (root)
$password = '';                   // Mot de passe (souvent vide en local)

try {
    $pdo = new PDO(
        "mysql:host=$host;dbname=$dbname;charset=utf8",
        $username,
        $password
    );
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
} catch (PDOException $e) {
    die("Connexion échouée : " . $e->getMessage());
}
?>
```

🔍 Pourquoi ?

- **PDO** est une classe PHP qui permet de se connecter à plusieurs types de bases de données, dont MySQL.
- On utilise un bloc `try/catch` pour **gérer les erreurs** (par exemple : mauvais mot de passe).
- Le `charset=utf8` permet de **gérer les accents** et caractères spéciaux.
- `setAttribute(...ERRMODE_EXCEPTION)` : si une requête échoue, **une erreur claire est générée**.

🎓 Bonnes pratiques

- Ce fichier est à **inclure dans tous les scripts** PHP qui ont besoin de la base.
- **On ne duplique pas la connexion** dans chaque page. Cela évite les erreurs, rend le code plus lisible et facilite les changements futurs.

🧩 Étape 2 : Lire les données (fichier `liste_utilisateurs.php`)

```php
<?php
require_once 'config/db.php';     // On inclut le fichier de connexion

$sql = "SELECT * FROM utilisateurs";      // Requête SQL : on récupère toutes les lignes de la
$stmt = $pdo->query($sql);                // Exécution de la requête
$utilisateurs = $stmt->fetchAll(PDO::FETCH_ASSOC); // On récupère les résultats sous forme de
?>
```

```php
<ul>
<?php foreach ($utilisateurs as $utilisateur): ?>
    <li><?= htmlspecialchars($utilisateur['nom']) ?> (<?= htmlspecialchars($utilisateur['email
<?php endforeach; ?>
</ul>
```

🔍 Pourquoi ?

- `require_once` évite les erreurs si le fichier est déjà inclus ailleurs.
- `$pdo->query(...)` : exécute une requête **en lecture seule**.
- `fetchAll(PDO::FETCH_ASSOC)` : transforme les résultats en tableau avec les **noms des colonnes** comme clés.
- `foreach` permet d'**afficher dynamiquement** tous les utilisateurs.
- `htmlspecialchars()` est essentiel pour **empêcher le HTML ou JavaScript malicieux** (protection XSS).

🧩 Étape 3 : Ajouter un utilisateur

📋 Formulaire (HTML)

```html
<form method="post" action="ajouter_utilisateur.php">
    <input type="text" name="nom" placeholder="Nom" required>
    <input type="email" name="email" placeholder="Email" required>
    <button type="submit">Ajouter</button>
</form>
```

⚙️ Traitement du formulaire

```php
<?php
require_once 'config/db.php';

if (!empty($_POST['nom']) && !empty($_POST['email'])) {
    $sql = "INSERT INTO utilisateurs (nom, email) VALUES (:nom, :email)";
    $stmt = $pdo->prepare($sql); // Préparation de la requête avec des "placeholders"
    $stmt->execute([
        ':nom' => $_POST['nom'],
        ':email' => $_POST['email']
    ]);
    echo "Utilisateur ajouté !";
}
?>
```

🔍 Pourquoi ?

- `method="post"` : on **ne montre pas les données dans l'URL**.
- `required` : évite les envois vides.
- `prepare(...)` + `execute(...)` permet d'**éviter les injections SQL** (ex. : si quelqu'un met `' OR 1=1`).
- Les `:nom` et `:email` sont des **paramètres nommés** qui seront automatiquement sécurisés.

🧰 Bonnes pratiques résumées

Bonne pratique	Pourquoi ?
Centraliser la connexion (`db.php`)	Gain de temps, moins d'erreurs, facile à modifier
Requêtes préparées (`prepare + execute`)	Sécurise les données, empêche les attaques
`htmlspecialchars()`	Évite les failles XSS (scripts malicieux dans les champs)
Ne pas afficher d'erreur brute à l'utilisateur	Protéger les informations du serveur
Tester `$_POST` ou `$_GET` avec `empty()`	Évite les erreurs et le traitement d'informations vides

✅ À retenir

- Pour communiquer avec une base SQL, PHP utilise **PDO**
- La connexion à la base doit être **réutilisable** dans tout le projet
- Les données doivent être **protégées à chaque étape**
- Les requêtes **INSERT, UPDATE, DELETE** doivent toujours être **préparées**
- L'affichage doit être **sécurisé avec** `htmlspecialchars`