

Chapitre 11 : PHP et MySQL











Centre BTS ALKENDI CASABLANCA

Dr. Ayman NAIM

March 11, 2025

Contents

1	Introduction à PHP et MySQL	3
1.1	 Rôle de MySQL dans les applications web	3
1.2	 Fonctionnement de PHP avec MySQL	3
1.3	Comparaison entre bases de données relationnelles et non relationnelles	4
2	Installation et Configuration de MySQL	5
2.1	 Installation de MySQL sur un serveur local (XAMPP, MAMP, WAMP)	5
2.2	 Installation et configuration sur un serveur distant (Linux, cPanel)	5
2.3	 Gestion des utilisateurs et des privilèges MySQL	6
3	Connexion de PHP à MySQL	7
3.1	 Présentation des méthodes de connexion (PDO vs MySQLi)	7
3.2	 Connexion via MySQLi (procédural et orienté objet)	7
3.2.1	 Mode procédural	8
3.2.2	 Mode orienté objet	8
3.3	 Connexion via PDO et gestion des erreurs	9
3.4	Bonnes pratiques pour sécuriser la connexion	9
4	Création et Manipulation d'une Base de Données	10
4.1	 Création et suppression de bases de données avec PHP	10
4.2	 Sélection et changement de base de données	10
4.3	 Structure des tables et types de données en MySQL	11
4.4	 Exécution de requêtes SQL avec PHP	11
4.4.1	 Avec MySQLi procédural	12
4.4.2	 Avec PDO	12
5	CRUD (Create, Read, Update, Delete) avec PHP et MySQL	13
5.1	 Principe du CRUD et son importance	13
5.2	 Création d'enregistrements (CREATE)	13
5.3	 Lecture et affichage des données (READ)	14
5.4	 Mise à jour des données (UPDATE)	14

5.5	 Suppression des données (DELETE)	15
5.6	Utilisation des requêtes préparées pour renforcer la sécurité	15
6	Gestion Avancée des Données	16
6.1	Utilisation des jointures SQL (INNER JOIN, LEFT JOIN, RIGHT JOIN, FULL JOIN) . .	16
6.2	 Transactions SQL et contrôle d'intégrité (COMMIT, ROLLBACK)	16
6.3	 Optimisation des requêtes (indexation, analyse EXPLAIN)	17
7	Sécurité et Bonnes Pratiques en PHP/MySQL	18
7.1	 Protection contre les injections SQL	18
7.2	 Gestion des sessions et cookies sécurisés	19
7.3	 Authentification des utilisateurs et gestion des rôles	19
7.4	 Stockage sécurisé des mots de passe avec bcrypt	20
7.5	Bonnes pratiques pour éviter les failles de sécurité courantes	20
8	Conclusion et Ressources Complémentaires	21
8.1	 Récapitulatif des notions clés	21
8.2	 Ressources pour approfondir	21
8.3	 Prochaines étapes pour aller plus loin	22

Objectif du Chapitre

L'objectif de ce chapitre est de comprendre les principes du développement web dynamique en combinant PHP et MySQL. Vous apprendrez à installer et configurer MySQL, à vous connecter depuis PHP (via PDO ou MySQLi), à créer et manipuler une base de données, à réaliser des opérations CRUD et à implémenter des pratiques de sécurité et d'optimisation. Enfin, des ressources complémentaires vous guideront pour aller plus loin.



INTRODUCTION À PHP ET MYSQL



MySQL et PHP dans le développement web

MySQL et PHP sont deux technologies essentielles pour le développement web dynamique. MySQL permet de stocker et gérer les données, tandis que PHP offre un moyen puissant d'interagir avec ces données pour produire du contenu dynamique.

Rôle de MySQL dans les applications web

Qu'est-ce que MySQL ?

MySQL est un système de gestion de base de données relationnel (SGBDR) utilisé pour stocker, organiser et fournir l'accès aux données des sites et applications web.

-  **Base de données relationnelle** : Organisation en tables, lignes et colonnes.
-  **Performance** : Optimisé pour gérer un grand nombre de requêtes simultanées.
- **Sécurité** : Gestion avancée des utilisateurs et des privilèges.

Fonctionnement de PHP avec MySQL

Comment PHP et MySQL interagissent-ils ?

PHP est un langage de script côté serveur qui permet de communiquer avec MySQL via des requêtes SQL. Voici comment fonctionne l'échange entre PHP et MySQL :

- ➔ **Le client** (navigateur) envoie une requête HTTP vers un fichier `.php`.
- ➔ **Le serveur web** exécute le code PHP, qui interagit avec MySQL.
- ➔ **MySQL** traite la requête SQL et renvoie les résultats à PHP.
- ➔ **PHP** génère du contenu dynamique (HTML, JSON...) en fonction des données reçues.

Exemple de connexion entre PHP et MySQL





```
<?php
$mysqli = new mysqli("localhost", "user", "password", "database");
if ($mysqli->connect_error) {
    die("Échec de connexion: " . $mysqli->connect_error);
}
echo "Connexion réussie !";
?>
```

Comparaison entre bases de données relationnelles et non relationnelles





SQL vs NoSQL

Les bases de données se divisent en deux grandes catégories : relationnelles (SQL) et non relationnelles (NoSQL). Chacune a ses avantages et ses domaines d'application spécifiques.

Bases de données relationnelles (SQL)

-  Organisation structurée en tables avec relations entre les données.
-  Garantissent l'intégrité et la cohérence des données.
-  Utilisent le langage SQL (Structured Query Language).
-  Exemples : MySQL, PostgreSQL, Oracle, SQL Server.




Bases de données non relationnelles (NoSQL)

-  Structure flexible : stockage sous forme de documents JSON, colonnes dynamiques, clé-valeur...
-  Excellentes performances et scalabilité horizontale.
-  Moins strictes en termes d'intégrité référentielle.
-  Exemples : MongoDB (documents), Redis (clé-valeur), Cassandra (colonnes).

Quand choisir SQL ou NoSQL ?

Utilisez SQL si vos données sont bien structurées et nécessitent des relations fortes. **Utilisez NoSQL** si vous avez de gros volumes de données non structurées et que la scalabilité est prioritaire.

Résumé de la section

-  MySQL est un SGBD performant pour gérer des bases de données relationnelles.
-  PHP utilise MySQL pour stocker et récupérer des données dynamiques.
-  Les bases de données relationnelles conviennent aux structures rigides, alors que NoSQL est plus flexible et scalable.



INSTALLATION ET CONFIGURATION DE MYSQL



Pourquoi installer MySQL ?

MySQL est un système de gestion de base de données relationnel (SGBDR) essentiel pour développer des applications web dynamiques. Il permet de stocker, gérer et interroger des données efficacement. Cette section vous guidera à travers son installation en local et sur un serveur distant.



Installation de MySQL sur un serveur local (XAMPP, MAMP, WAMP)



Pourquoi utiliser un environnement local ?

Installer MySQL en local permet aux développeurs de tester leurs applications sans avoir besoin d'un serveur distant. Des outils comme XAMPP, WAMP et MAMP regroupent Apache (serveur web), PHP et MySQL en un seul package facile à configurer.

Étapes d'installation avec XAMPP :

- ✓ Téléchargez et installez **XAMPP** (ou WAMP/MAMP).
- ✓ Ouvrez le panneau de contrôle **XAMPP**.
- ✓ Activez les services **Apache** et **MySQL**.
- ✓ Accédez à l'interface **phpMyAdmin** via : <http://localhost/phpmyadmin/>.



Vérification de l'installation

Ouvrez un terminal ou l'invite de commande et entrez :

```
mysql --version
```

Vous devriez voir un message indiquant la version de MySQL installée.



Installation et configuration sur un serveur distant (Linux, cPanel)



Installation sur un VPS ou un serveur dédié

Si vous déployez une application en production, vous devez installer MySQL sur un serveur distant (VPS ou dédié). L'installation diffère selon que vous utilisez un serveur Linux (Ubuntu/Debian) ou un panneau d'administration comme cPanel.

Installation via la ligne de commande (Ubuntu/Debian) :

```
1 sudo apt-get update
2 sudo apt-get install mysql-server
```

Sécurisation de MySQL (fortement recommandé) :

```
1 sudo mysql_secure_installation
```

Ce script vous guidera à travers plusieurs étapes de sécurisation :

- Définition d'un mot de passe root sécurisé.
- Suppression des utilisateurs anonymes.
- Désactivation des connexions root distantes.
- Suppression des bases de test inutiles.

Installation via cPanel

Si vous utilisez un hébergement mutualisé avec **cPanel**, suivez ces étapes :

1. Accédez à cPanel et ouvrez **Bases de données MySQL**.
2. Créez une nouvelle base de données.
3. Ajoutez un utilisateur MySQL et attribuez-lui des privilèges.
4. Notez les informations de connexion pour votre script PHP.

Gestion des utilisateurs et des privilèges MySQL

Pourquoi gérer les utilisateurs MySQL ?

Par défaut, MySQL crée un utilisateur **root** avec tous les droits. Cependant, pour des raisons de sécurité, il est préférable de créer des utilisateurs ayant uniquement les privilèges nécessaires.

Création d'un utilisateur sécurisé pour une application web :

```
1 CREATE DATABASE my_app;
2 CREATE USER 'webuser'@'localhost' IDENTIFIED BY 'MotDePasseSecurise';
3 GRANT ALL PRIVILEGES ON my_app.* TO 'webuser'@'localhost';
4 FLUSH PRIVILEGES;
```




Limiter les privilèges d'un utilisateur :

```
1 GRANT SELECT, INSERT, UPDATE, DELETE ON my_app.* TO 'webuser'@'localhost';
```

Bonnes pratiques de sécurité

- Ne jamais utiliser le compte **root** pour une application web.
- Toujours définir un mot de passe fort pour les utilisateurs MySQL.
- Limiter les privilèges des utilisateurs selon leurs besoins.
- Désactiver l'accès root distant si non nécessaire.
- Effectuer des sauvegardes régulières des bases de données.

Résumé de la section

-  **Local** : XAMPP/MAMP/WAMP facilitent l'installation de MySQL sur un PC.
-  **Serveur distant** : Sur Linux, installation via `apt-get install mysql-server`.
-  **Sécurité** : Créez des utilisateurs dédiés avec des privilèges limités.





CONNEXION DE PHP À MYSQL

Pourquoi connecter PHP à MySQL ?

Les applications web dynamiques ont besoin d'une base de données pour stocker et gérer les informations des utilisateurs, des produits ou toute autre donnée essentielle. PHP propose deux principales méthodes pour interagir avec MySQL : **MySQLi** et **PDO**.

Présentation des méthodes de connexion (PDO vs MySQLi)

Comparaison entre MySQLi et PDO

-  **MySQLi** (*MySQL Improved*) est spécifique à MySQL et permet une interaction via une interface procédurale ou orientée objet.
-  **PDO** (*PHP Data Objects*) est une interface plus flexible qui prend en charge plusieurs SGBD (MySQL, PostgreSQL, SQLite...).

MySQLi :

- Spécifique à MySQL.
- Interface procédurale et orientée objet disponibles.

PDO :

- Compatible avec plusieurs bases de données.
- Gestion avancée des erreurs et requêtes préparées.

Quand utiliser MySQLi ou PDO ?

Si vous travaillez uniquement avec MySQL, **MySQLi** peut suffire. Si vous voulez une solution plus flexible pour plusieurs SGBD ou bénéficier d'une meilleure gestion des erreurs, **PDO** est recommandé.

Connexion via MySQLi (procédural et orienté objet)

> </> Mode procédural

Exemple de connexion MySQLi (procédural)

```
<?php
$host = 'localhost';
$user = 'webuser';
$pass = '12345';
$dbname = 'my_app';

$conn = mysqli_connect($host, $user, $pass, $dbname);

if (!$conn) {
    die("Connexion échouée : " . mysqli_connect_error());
}
echo "Connexion réussie !";
?>
```

> ⚙ Mode orienté objet

Exemple de connexion MySQLi (objet)

```
<?php
$host = 'localhost';
$user = 'webuser';
$pass = '12345';
$dbname = 'my_app';

$mysqli = new mysqli($host, $user, $pass, $dbname);

if ($mysqli->connect_error) {
    die("Connexion échouée : " . $mysqli->connect_error);
}
echo "Connexion réussie !";
?>
```


</> Connexion via PDO et gestion des erreurs

Exemple de connexion avec PDO

```
<?php
$dsn = 'mysql:host=localhost;dbname=my_app;charset=utf8';
$user = 'webuser';
$pass = '12345';




try {
    $pdo = new PDO($dsn, $user, $pass);
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    echo "Connexion réussie via PDO !";
} catch (PDOException $e) {
    echo "Erreur de connexion : " . $e->getMessage();
}
?>
```

Pourquoi PDO est-il plus sécurisé ?



- Il gère les erreurs via des exceptions.
- Il supporte les requêtes préparées, protégeant ainsi contre les injections SQL.
- Il permet une migration facile vers d'autres bases de données sans changer tout le code.

</> Bonnes pratiques pour sécuriser la connexion

Conseils de sécurité

-  Ne stockez jamais les mots de passe en clair dans le code (**utilisez des variables d'environnement**).
- Utilisez des requêtes préparées pour prévenir les injections SQL.
-  Limitez les privilèges de l'utilisateur MySQL à uniquement ce qui est nécessaire.
-  Activez SSL/TLS pour chiffrer les connexions en production.

Résumé de la section

-  PHP permet de se connecter à MySQL via **MySQLi** ou **PDO**.
-  **MySQLi** est spécifique à MySQL, tandis que **PDO** supporte plusieurs SGBD.
- La sécurité est primordiale : utilisez les requêtes préparées et protégez vos identifiants.



CRÉATION ET MANIPULATION D'UNE BASE DE DONNÉES



Introduction

Les bases de données sont au cœur des applications web modernes. MySQL permet de stocker, organiser et interroger efficacement des données. Dans cette section, nous verrons comment créer et gérer une base de données avec PHP et MySQL.

➤ Création et suppression de bases de données avec PHP



Astuce

Avant de créer une base de données, assurez-vous que votre serveur MySQL est en cours d'exécution et que vous avez les droits d'administration.



Code PHP : Création et suppression d'une base de données

```
<?php
$conn = mysqli_connect('localhost', 'root', 'motdepasse');

if (!$conn) {
    die("Connexion échouée : " . mysqli_connect_error());
}

// Créer une base de données
$sql = "CREATE DATABASE IF NOT EXISTS ma_nouvelle_base";
if (mysqli_query($conn, $sql)) {
    echo "Base de données créée avec succès.";
} else {
    echo "Erreur : " . mysqli_error($conn);
}

// Supprimer une base de données
$sql = "DROP DATABASE IF EXISTS ma_nouvelle_base";
if (mysqli_query($conn, $sql)) {
    echo "Base de données supprimée avec succès.";
} else {
    echo "Erreur : " . mysqli_error($conn);
}

mysqli_close($conn);
?>
```



➤ Sélection et changement de base de données

Avec MySQLi :

Sélectionner une base de données avec MySQLi

```
mysqli_select_db($conn, 'my_app');
```



Avec SQL :

Commande SQL : Sélectionner une base de données

```
USE my_app;
```

Structure des tables et types de données en MySQL

Les bases de données MySQL sont constituées de tables, qui contiennent des champs de différents types :

- **# INT, BIGINT** : nombres entiers.
- **A VARCHAR, TEXT** : chaînes de caractères.
- **DATE, DATETIME, TIMESTAMP** : dates et heures.
-  **DECIMAL, FLOAT, DOUBLE** : nombres décimaux.
-  **BOOLEAN** (souvent TINYINT(1) en MySQL).

Exemple : Création d'une table utilisateurs

```
CREATE TABLE utilisateurs (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nom VARCHAR(100) NOT NULL,
    email VARCHAR(100) NOT NULL UNIQUE,
    mot_de_passe VARCHAR(255) NOT NULL,
    date_creation TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Exécution de requêtes SQL avec PHP

> Avec MySQLi procédural

Exemple : Sélectionner des données avec MySQLi

```
<?php
$conn = mysqli_connect('localhost', 'webuser', '12345', 'my_app');

$query = "SELECT * FROM utilisateurs";
$result = mysqli_query($conn, $query);

while ($row = mysqli_fetch_assoc($result)) {
    echo $row['nom'] . " - " . $row['email'] . "<br>";
}

mysqli_close($conn);
?>
```

> Avec PDO

Exemple : Sélectionner des données avec PDO





```
<?php
try {
    $pdo = new PDO('mysql:host=localhost;dbname=my_app', 'webuser', '12345');
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    $query = "SELECT * FROM utilisateurs";
    $stmt = $pdo->query($query);

    while ($row = $stmt->fetch(PDO::FETCH_ASSOC)) {
        echo $row['nom'] . " - " . $row['email'] . "<br>";
    }

} catch (PDOException $e) {
    echo "Erreur : " . $e->getMessage();
}
?>
```

Résumé de la section





-  Création et suppression de bases de données avec MySQL et PHP.
-  Sélection et manipulation d'une base de données existante.
-  Structure des tables et types de données courants en MySQL.
-  Exécution de requêtes SQL avec **MySQLi** et **PDO**.

CRUD (CREATE, READ, UPDATE, DELETE) AVEC PHP ET MYSQL

Qu'est-ce que le CRUD ?

CRUD (**Create**, **Read**, **Update**, **Delete**) est un ensemble d'opérations fondamentales permettant d'interagir avec une base de données. Ces opérations permettent d'ajouter, de récupérer, de modifier et de supprimer des données.

Principe du CRUD et son importance

-  **Create** : insérer de nouveaux enregistrements.
-  **Read** : lire et afficher les données stockées.
-  **Update** : modifier les informations existantes.
-  **Delete** : supprimer des enregistrements obsolètes.

Pourquoi est-ce important ?

Le CRUD est la base de toute application interactive utilisant une base de données. Il permet aux utilisateurs d'ajouter des informations, de les consulter, de les modifier et de les supprimer selon leurs besoins.

Création d'enregistrements (CREATE)

Insérer un utilisateur avec PDO

```
<?php
try {
    $pdo = new PDO('mysql:host=localhost;dbname=my_app', 'webuser', '12345');
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    $sql = "INSERT INTO utilisateurs (nom, email, mot_de_passe)
            VALUES (:nom, :email, :mot_de_passe)";
    $stmt = $pdo->prepare($sql);

    $stmt->execute([
        ':nom' => 'Dupont',
        ':email' => 'dupont@example.com',
        ':mot_de_passe' => password_hash('monmotdepasse', PASSWORD_BCRYPT)
    ]);

    echo "Utilisateur inséré avec succès !";
} catch (PDOException $e) {
    echo "Erreur : " . $e->getMessage();
}
?>
```

Lecture et affichage des données (READ)

Récupérer tous les utilisateurs avec PDO

```
<?php
try {
    $pdo = new PDO('mysql:host=localhost;dbname=my_app', 'webuser', '12345');
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    $sql = "SELECT id, nom, email FROM utilisateurs";
    $stmt = $pdo->query($sql);

    foreach ($stmt as $row) {
        echo "ID: " . $row['id'] . " - Nom: " . $row['nom'] .
            " - Email: " . $row['email'] . "<br>";
    }
} catch (PDOException $e) {
    echo "Erreur : " . $e->getMessage();
}
?>
```

Mise à jour des données (UPDATE)

Mettre à jour un utilisateur avec PDO

```
<?php
try {
    $pdo = new PDO('mysql:host=localhost;dbname=my_app', 'webuser', '12345');
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    $sql = "UPDATE utilisateurs SET nom=:nom WHERE id=:id";
    $stmt = $pdo->prepare($sql);

    $stmt->execute([
        ':nom' => 'Martin',
        ':id' => 1
    ]);

    echo "Enregistrement mis à jour avec succès !";
} catch (PDOException $e) {
    echo "Erreur : " . $e->getMessage();
}
?>
```

Suppression des données (DELETE)

Supprimer un utilisateur avec PDO

```
<?php
try {
    $pdo = new PDO('mysql:host=localhost;dbname=my_app', 'webuser', '12345');
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    $sql = "DELETE FROM utilisateurs WHERE id=:id";
    $stmt = $pdo->prepare($sql);

    $stmt->execute([':id' => 1]);

    echo "Enregistrement supprimé avec succès !";
} catch (PDOException $e) {
    echo "Erreur : " . $e->getMessage();
}
?>
```

Utilisation des requêtes préparées pour renforcer la sécurité





Pourquoi utiliser des requêtes préparées ?

Les requêtes préparées empêchent les attaques par injection SQL en séparant la structure de la requête des données insérées. Cela améliore la sécurité et les performances.

Exemple de requête sécurisée

```
$sql = "SELECT * FROM utilisateurs WHERE email = :email";
$stmt = $pdo->prepare($sql);
$stmt->execute([':email' => 'test@example.com']);
```

Résumé de la section

-  **Create** : insérer un nouvel utilisateur avec une requête sécurisée.
-  **Read** : récupérer et afficher les données de la base.
-  **Update** : modifier les informations d'un enregistrement.
-  **Delete** : supprimer un enregistrement en toute sécurité.
- **Sécurité** : utilisation des requêtes préparées pour éviter les attaques SQL.



GESTION AVANCÉE DES DONNÉES



Pourquoi une gestion avancée des données ?

Une base de données performante et bien structurée améliore la rapidité des requêtes, assure l'intégrité des données et permet une manipulation efficace. Dans cette section, nous allons explorer les **jointures**, les **transactions SQL** et les **optimisations** pour améliorer nos performances.

Utilisation des jointures SQL (INNER JOIN, LEFT JOIN, RIGHT JOIN, FULL JOIN)

Les jointures SQL

Les jointures permettent de récupérer des données en combinant plusieurs tables grâce à une relation clé étrangère. Elles sont essentielles pour éviter la redondance et structurer les données de manière efficace.

- **INNER JOIN** : Retourne uniquement les correspondances entre les deux tables.
-  **LEFT JOIN** : Retourne tous les enregistrements de la table de gauche, même sans correspondance.
-  **RIGHT JOIN** : Retourne tous les enregistrements de la table de droite, même sans correspondance.
- **FULL JOIN** : Combine les données de gauche et de droite (non disponible nativement en MySQL sans UNION).

Exemple : Récupérer les commandes des utilisateurs

```
SELECT utilisateurs.nom, commandes.ref_commande
FROM utilisateurs
INNER JOIN commandes
ON utilisateurs.id = commandes.user_id;
```

Bonnes pratiques

- Toujours utiliser des index sur les colonnes jointes (**FOREIGN KEY** et **INDEX**).
- Privilégier **INNER JOIN** pour éviter des résultats inutiles et optimiser les performances.
- Limiter les résultats avec **LIMIT** pour éviter des jointures coûteuses.

Transactions SQL et contrôle d'intégrité (COMMIT, ROLLBACK)

Pourquoi utiliser des transactions SQL ?

Une transaction regroupe plusieurs requêtes SQL sous une même exécution. Si une erreur survient, la transaction peut être annulée (**ROLLBACK**), garantissant ainsi l'intégrité des données.

Exemple : Transaction avec COMMIT et ROLLBACK

```
<?php
try {
    $pdo = new PDO('mysql:host=localhost;dbname=my_app', 'webuser', '12345');
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    $pdo->beginTransaction(); // Démarrer la transaction

    // Insertion de l'utilisateur
    $stmt1 = $pdo->prepare("INSERT INTO utilisateurs (nom, email) VALUES (?, ?)");
    $stmt1->execute(['Test1', 'test1@example.com']);

    // Erreur volontaire (email trop long)
    $stmt2 = $pdo->prepare("INSERT INTO utilisateurs (nom, email) VALUES (?, ?)");
    $stmt2->execute(['Test2', str_repeat('a', 300).'@example.com']);

    $pdo->commit(); // Valider la transaction
    echo "Transaction réussie !";

} catch (PDOException $e) {
    $pdo->rollBack(); // Annuler en cas d'erreur
    echo "Erreur, transaction annulée : " . $e->getMessage();
}
?>
```

Quand utiliser des transactions ?

- Lors de mises à jour critiques (UPDATE, DELETE).
- Pour des opérations en cascade impliquant plusieurs tables.
- Lorsque l'intégrité des données est essentielle (paiements, stock...).

Optimisation des requêtes (indexation, analyse EXPLAIN)

Pourquoi optimiser ses requêtes SQL ?

Des requêtes non optimisées peuvent ralentir votre application. Les index et l'analyse EXPLAIN permettent d'améliorer la vitesse d'exécution.

 **Indexation** : Créer un index sur une colonne pour accélérer les recherches.

Exemple : Créer un index sur la colonne nom

```
CREATE INDEX idx_nom ON utilisateurs (nom);
```

 **Analyse de requête avec EXPLAIN** :



Exemple : Analyser une requête SQL

```
EXPLAIN SELECT * FROM utilisateurs WHERE email='dupont@example.com';
```

Bonnes pratiques pour optimiser les performances

- Utiliser des **index** sur les colonnes fréquemment utilisées dans les **WHERE**.
- Éviter **SELECT *** et ne sélectionner que les colonnes nécessaires.
- Préférer les **jointures indexées** au lieu des requêtes multiples.
- Utiliser **LIMIT** pour réduire la charge des grosses requêtes.

Résumé de la section

- **Jointures SQL** : Permettent de relier plusieurs tables pour obtenir des données structurées.
-  **Transactions SQL** : Assurent la cohérence des données en regroupant plusieurs opérations.
-  **Optimisation des requêtes** : Utilisation des index et analyse avec **EXPLAIN**.

SÉCURITÉ ET BONNES PRATIQUES EN PHP/MYSQL

Pourquoi la sécurité est-elle essentielle ?

Une mauvaise gestion de la sécurité en PHP/MySQL peut entraîner des vulnérabilités critiques comme les injections SQL, le vol de session, et les fuites de données. Adopter les bonnes pratiques réduit les risques et renforce la protection des utilisateurs.

Protection contre les injections SQL

Qu'est-ce qu'une injection SQL ?

Une injection SQL se produit lorsqu'un attaquant manipule une requête SQL en insérant du code malveillant via un champ de saisie non sécurisé.

Bonnes pratiques pour se protéger :

- Utiliser les requêtes préparées avec **PDO** ou **MySQLi**.
- Éviter la concaténation de variables dans les requêtes SQL.
- Valider et filtrer toutes les données entrantes.
- Échapper les entrées utilisateur avec `mysqli_real_escape_string()` si nécessaire.

Exemple sécurisé avec PDO

```
$stmt = $pdo->prepare("SELECT * FROM utilisateurs WHERE email = :email");
$stmt->execute([':email' => $email_saisi]);
```

Gestion des sessions et cookies sécurisés

Bonnes pratiques pour sécuriser les sessions :

- Toujours démarrer la session via `session_start()` en haut des pages.
- Régénérer l'ID de session après connexion (`session_regenerate_id()`) pour éviter la fixation de session.
- Détruire les sessions après déconnexion (`session_destroy()`).

Sécurisation des cookies :

- Définir `HttpOnly` pour empêcher l'accès JavaScript (`setcookie(..., ..., ..., true)`).
- Utiliser `Secure` pour les cookies si le site est en HTTPS.
- Restreindre la durée de vie des cookies pour limiter les risques d'attaques XSS.

Exemple : Configuration sécurisée d'un cookie

```
setcookie("session", $sessionID, [
    'expires' => time() + 3600,
    'httponly' => true,
    'secure' => true,
    'samesite' => 'Strict'
]);
```

Authentification des utilisateurs et gestion des rôles

Bonnes pratiques en authentification

- Hasher les mots de passe avec `password_hash()` (BCRYPT ou ARGON2).
- Vérifier les mots de passe avec `password_verify()`.
- Mettre en place un système de rôles (admin, éditeur, utilisateur).
- Restreindre l'accès aux pages selon les permissions.

Exemple : Hashage et vérification d'un mot de passe

```
$hash = password_hash($motDePasse, PASSWORD_BCRYPT);
if (password_verify($motDePasse, $hash)) {
    echo "Authentification réussie !";
}
```

Stockage sécurisé des mots de passe avec bcrypt

Pourquoi bcrypt ?

Le hachage avec `bcrypt` (via `password_hash()`) ajoute un **sel** et rend les mots de passe stockés plus difficiles à attaquer.

Exemple : Stockage sécurisé d'un mot de passe




```
$hash = password_hash($motDePasse, PASSWORD_BCRYPT);
```

À éviter absolument !




- Stocker des mots de passe en clair dans la base de données.
- Utiliser `md5()` ou `sha1()` pour hacher les mots de passe.
- Ne pas utiliser de **sel** pour renforcer le hachage.

Bonnes pratiques pour éviter les failles de sécurité courantes

Checklist de sécurité en PHP/MySQL

- **Mettre à jour** PHP, MySQL et toutes les dépendances.
-  **Désactiver l'affichage des erreurs** en production.
-  **Limiter les privilèges MySQL** pour chaque utilisateur.
-  **Restreindre l'accès aux fichiers sensibles** (`.env`, `config.php`).
- **Utiliser HTTPS** pour chiffrer les communications.

Résumé de la section





-  **Évitez les injections SQL** avec des requêtes préparées.
-  **Sécurisez les sessions et cookies** pour éviter le vol de données.
-  **Authentifiez correctement les utilisateurs** avec des mots de passe hachés.
- **Appliquez les meilleures pratiques** pour une application sécurisée.



CONCLUSION ET RESSOURCES COMPLÉMENTAIRES






✓ Objectif atteint !

Vous avez désormais acquis une solide compréhension du développement web dynamique avec PHP et MySQL. Ce cours vous a permis de maîtriser :

-  La gestion et la configuration de MySQL.
-  La connexion de PHP à MySQL via MySQLi ou PDO.
-  La création, manipulation et optimisation des bases de données.
-  Les opérations CRUD essentielles.
- Les jointures, transactions et indexation avancées.
- Les bonnes pratiques de sécurité.

Avec ces compétences, vous êtes prêt à développer des applications web robustes et sécurisées !






</> 💡 Récapitulatif des notions clés

-  Compréhension du fonctionnement conjoint de PHP et MySQL.
-  Installation et configuration de MySQL (local et distant).
-  Connexion sécurisée via MySQLi ou PDO.
-  Création, manipulation et structuration des tables.
-  Opérations CRUD : insérer, lire, mettre à jour, supprimer des données.
- Gestion avancée : jointures, transactions, indexation.
- Sécurité : injections SQL, sessions, mots de passe hachés, etc.

</> 📖 Ressources pour approfondir

🌐 Apprendre et s'améliorer





Voici des ressources fiables pour approfondir vos connaissances :

-  [Documentation officielle PHP](#).
-  [Documentation officielle MySQL](#).
-  [OpenClassrooms](#) : tutoriels et cours en ligne.
-  [Stack Overflow](#) : communauté pour poser des questions et trouver des solutions.
-  [W3Schools PHP/MySQL](#) : tutoriels pratiques.

Prochaines étapes pour aller plus loin





Vers le niveau supérieur

Pour continuer votre progression, voici quelques pistes à explorer :

-  Approfondir l'architecture MVC en PHP.
-  Découvrir des frameworks modernes (**Laravel**, **Symfony**).
- Apprendre à gérer des **APIs REST** et échanger des données en **JSON/XML**.
-  Explorer les bases du **front-end moderne** (**HTML5**, **CSS3**, **JavaScript**).
-  Expérimenter l'hébergement et le déploiement d'applications PHP/MySQL en ligne.

Félicitations !

En associant PHP et MySQL, vous disposez d'un environnement complet pour créer des sites et applications web dynamiques. Vous avez abordé :

-  L'installation et la configuration de MySQL (localement ou à distance).
-  Les différentes méthodes de connexion en PHP (**MySQLi**, **PDO**).
-  La création, l'utilisation et la gestion sécurisée des bases de données.
-  Le **CRUD**, la gestion avancée (**jointures**, **transactions**) et l'optimisation.
- Les **bonnes pratiques de sécurité** indispensables pour protéger vos applications.