Introduction Approfondie à PHP pour le Développement Web

Abdelweheb GUEDDES

20 février 2025

Séance 7 : Introduction aux Bases de Données avec PDO et PHP - Approfondissement

Présentation des Bases de Données - En Détail

Concept des Bases de Données - Pourquoi et Comment Une base de données est bien plus qu'un simple endroit pour stocker des données. C'est un système organisé conçu pour :

- **Organiser et Structurer les Données** : Imaginez un tableur géant avec des colonnes et des lignes. Les bases de données relationnelles, comme MySQL, excellent dans cette organisation. Elles permettent de définir des relations entre différentes tables, évitant la redondance et assurant la cohérence des données.
- **Assurer la Persistance des Données** : Contrairement aux variables PHP qui disparaissent à la fin de l'exécution du script, les données stockées dans une base de données sont conservées de manière permanente (jusqu'à suppression explicite). C'est crucial pour les informations importantes comme les comptes utilisateurs, les commandes, etc.
- **Faciliter la Recherche et la Manipulation** : Les bases de données offrent un langage puissant, SQL (Structured Query Language), pour interroger, modifier, ajouter et supprimer des données de manière efficace. Vous pouvez faire des recherches complexes, trier les résultats, et effectuer des opérations en masse.

— **Gérer l'Accès Concurrent et la Sécurité** : Un SGBD (Système de Gestion de Bases de Données) comme MySQL gère les accès multiples à la base de données, assurant que les données restent cohérentes même si plusieurs utilisateurs ou applications y accèdent simultanément. Il gère également la sécurité et les permissions d'accès.

Pour les applications web dynamiques, une base de données est indispensable. Pensez à un site de e-commerce : il a besoin de stocker les produits, les clients, les commandes, les avis... Tout cela est géré efficacement par une base de données. Dans ce cours, nous allons utiliser le SGBD MySQL, un choix très populaire et robuste, avec l'extension PDO de PHP pour interagir avec lui.

Le Rôle de MySQL - Plus Qu'un Simple Logiciel MySQL est un SGBD open source, ce qui signifie que son code source est disponible et modifiable, et qu'il est généralement gratuit. Sa popularité en fait un standard dans le développement web, notamment dans l'écosystème LAMP (Linux, Apache, MySQL, PHP). Voici pourquoi il est si important :

— **Performance et Fiabilité** : MySQL est conçu pour gérer de grandes quantités de données et un trafic important. Il est optimisé pour la vitesse et la stabilité.

— **Facilité d'utilisation et Grande Communauté** : MySQL est relativement simple à apprendre et à utiliser, tout en offrant des fonctionnalités avancées. Sa vaste communauté d'utilisateurs signifie qu'il est facile de trouver de l'aide, des tutoriels et des solutions aux problèmes.

— **Intégration Parfaite avec PHP** : MySQL est historiquement très lié à PHP, et PDO renforce cette intégration en fournissant une interface uniforme et sécurisée.

— **Fonctionnalités Avancées** : Au-delà du stockage basique, MySQL offre des fonctionnalités comme les transactions (pour garantir l'intégrité des opérations), les index (pour accélérer les recherches), les vues (pour simplifier les requêtes complexes), et bien d'autres.

Structure d'une Base de Données - Une Analogie Visuelle Imaginez une bibliothèque :

— **La Base de Données (Bibliothèque)** : C'est le conteneur principal qui regroupe toutes les informations relatives à un projet ou une application (par exemple, "Bibliothèque des étudiants").

— **Les Tables (Rayonnages)** : Chaque rayonnage est spécialisé dans un

type de livre (par exemple, "Rayonnage des étudiants", "Rayonnage des livres empruntés"). Dans une base de données, les tables organisent les données par catégorie (utilisateurs, produits, articles...).

— **Les Colonnes (Étiquettes sur les Rayonnages)** : Chaque étiquette sur un rayonnage indique le type d'information stockée (par exemple, sur le rayonnage "étudiants" : "Nom", "Prénom", "Numéro d'étudiant"). Les colonnes définissent les attributs ou types de données pour chaque enregistrement dans une table.

— **Les Lignes (Livres sur les Rayonnages)** : Chaque livre est un enregistrement unique (par exemple, un livre spécifique sur le rayonnage "étudiants" représente un étudiant particulier). Les lignes contiennent les valeurs concrètes pour chaque colonne, représentant une instance unique de l'entité (un étudiant, un produit, un article...).

Cette structure tabulaire est au cœur des bases de données relationnelles. Comprendre cette organisation est fondamental pour travailler efficacement avec des bases de données.

Création de la Base de Données - Méthodes Détaillées et Recommandations

Choix de la Méthode de Création - Le Bon Outil pour le Bon Travail Vous avez deux approches principales pour créer une base de données MySQL, chacune avec ses avantages et inconvénients :

— **Méthode Graphique (MySQL Server, phpMyAdmin, etc.)** : Utiliser une interface graphique comme phpMyAdmin ou MySQL Workbench.

— **Méthode Programmatique (Requêtes SQL en PHP)** : Écrire du code PHP pour exécuter des commandes SQL de création.

Nous allons explorer les deux méthodes, mais il est important de comprendre quand utiliser chacune.

Méthode 1 : Création avec MySQL Server - Simplicité et Efficacité Visuelle

La création via un outil graphique est généralement la méthode la plus rapide et la plus intuitive pour les débutants et pour la gestion courante des bases de données. phpMyAdmin est un outil web très répandu, souvent inclus dans les distributions comme WAMP, XAMPP ou Laragon. MySQL

Workbench est une application de bureau plus complète, offrant plus de fonctionnalités pour la modélisation et la gestion avancée.

Étapes avec phpMyAdmin (Méthode Recommandée pour l'apprentissage et la gestion simple) :

- **Démarrage du Serveur et Accès à phpMyAdmin** : Assurez-vous que votre serveur (WAMP, XAMPP, etc.) est lancé et que MySQL est démarré. Ouvrez votre navigateur web et accédez à l'adresse `http://localhost/phpmyadmin` (ou l'adresse spécifique de votre installation).
- **Création de la Base de Données "coursphp"** :
  - Dans phpMyAdmin, recherchez l'onglet ou le lien "Bases de données".
  - Dans le champ "Créer une base de données", tapez `coursphp`.
  - Choisissez le "Collationnement" (jeu de caractères) approprié. Pour le français et la plupart des applications web modernes, `utf8mb4_unicode_ci` est un bon choix car il supporte un large éventail de caractères, y compris les emojis. Le collationnement détermine comment les caractères sont comparés et triés.
  - Cliquez sur le bouton "Créer".
- **Création de la Table "etudiants" dans "coursphp"** :
  - Sélectionnez la base de données `coursphp` dans la liste des bases de données (souvent à gauche de l'interface).
  - Recherchez l'onglet ou la section "Créer une table".
  - Dans le champ "Nom de la table", tapez `etudiants`.
  - Indiquez le nombre de colonnes, par exemple 5 (pour id, nom, prenom, email, groupe) et cliquez sur "Exécuter" ou "Go".
  - Définissez les colonnes comme suit :
    - `id` :
      - Nom : `id`
      - Type : `INT` (entier)
      - Longueur/Valeurs : (laissez vide ou mettez une valeur raisonnable, par exemple 11)
      - Index : Choisissez `PRIMARY` (clé primaire). Cochez la case `A_I` (Auto-incrément). La clé primaire identifie de manière unique chaque ligne de la table. L'auto-incrément fait en sorte que MySQL attribue automatiquement un nouvel ID unique à chaque nouvel étudiant ajouté.
    - `nom` :
      - Nom : `nom`

        - Type : `TEXT` (texte long) ou `VARCHAR` (texte de longueur variable, spécifiez une longueur maximale, par exemple 255). `VARCHAR` est souvent préférable pour des raisons de performance si vous connaissez une longueur maximale raisonnable.
    - `prenom` :
        - Nom : `prenom`
        - Type : `TEXT` ou `VARCHAR`(255)
    - `email` :
        - Nom : `email`
        - Type : `TEXT` ou `VARCHAR`(255)
    - `groupe` :
        - Nom : `groupe`
        - Type : `TEXT` ou `VARCHAR`(255)
- Cliquez sur "Enregistrer" ou "Save" pour créer la table.

Méthode 2 : Création avec des Requêtes SQL dans PHP - Flexibilité et Automatisation

La création par requêtes SQL en PHP est plus flexible et permet d'automatiser la création de bases de données et de tables directement depuis votre code. C'est utile pour des scripts d'installation, des migrations de bases de données, ou si vous n'avez pas d'accès direct à un outil graphique.

Explication Détaillée du Code PHP :

```php
<?php
    $serveur = "localhost";
    $utilisateur = "root";
    $motDePasse = "";

    try{
    $connexion = new PDO("mysql:host=$serveur", $utilisateur,
    $motDePasse);
      $connexion->setAttribute(PDO::ATTR_ERRMODE, PDO::
   ERRMODE_EXCEPTION);
    }
    catch(PDOException $e)
    {
      die("La connexion au serveur MySQL a échoué : " . $e->
   getMessage()); % Message plus précis
     }

```

```
15    $requeteCreationBase = "CREATE DATABASE IF NOT EXISTS
      coursphp CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci"
      ; % Ajout de IF NOT EXISTS et du charset
16   try{
17     $connexion->exec($requeteCreationBase);
18       }
19   catch(PDOException $e){
20     die("La création de la base de données 'coursphp' a échou
      é : " . $e->getMessage()); % Message plus précis
21   }
22   try {
23    $connexion->exec("USE coursphp");
24     } catch (PDOException $e)
25     {
26         die("Impossible de sélectionner la base de données '
      coursphp': " . $e->getMessage()); % Message plus précis
27     }
28
29     $requeteCreationTable = "CREATE TABLE IF NOT EXISTS
      etudiants ( % Ajout de IF NOT EXISTS
30          id INT AUTO_INCREMENT PRIMARY KEY,
31          nom TEXT,
32          prenom TEXT,
33         email TEXT,
34         groupe TEXT  -- Ajout de la colonne groupe
35       ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_
      unicode_ci"; % Ajout du moteur et du charset pour la table
36     try{
37       $connexion->exec($requeteCreationTable);
38       }
39       catch(PDOException $e){
40           die("La création de la table 'etudiants' a échoué :
      " . $e->getMessage()); % Message plus précis
41       }
42
43     echo "Base de données 'coursphp' et table 'etudiants' cré
      ées avec succès."; % Message plus précis
44    $connexion = null;
45
46 ?>
```

Points Clés du Code :

— **Connexion au Serveur MySQL (sans base de données spécifique)** : On se connecte d'abord au serveur MySQL sans spécifier de base de données dans la chaîne de connexion PDO (‘"mysql :host=$serveur"‘).

C'est nécessaire pour pouvoir créer la base de données.

— **Gestion des Erreurs avec 'try...catch'** : Le code est entouré de blocs 'try...catch' pour gérer les exceptions PDO. Si une erreur se produit (par exemple, problème de connexion, erreur SQL), le code dans le bloc 'catch' est exécuté, affichant un message d'erreur et arrêtant le script ('die()'). C'est essentiel pour le débogage et pour éviter que des erreurs ne passent inaperçues.

— **Création de la Base de Données avec 'CREATE DATABASE IF NOT EXISTS coursphp ...'** :
  — 'CREATE DATABASE coursphp' : C'est la commande SQL pour créer une base de données nommée 'coursphp'.
  — 'IF NOT EXISTS' : Ajouter 'IF NOT EXISTS' est une bonne pratique. Cela permet de vérifier si la base de données existe déjà. Si elle existe, la commande ne fait rien et ne génère pas d'erreur. C'est utile pour exécuter le script plusieurs fois sans provoquer d'erreur si la base de données a déjà été créée.
  — 'CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci' : Spécifie le jeu de caractères et le collationnement pour la base de données. Comme mentionné précédemment, 'utf8mb4_unicode_ci' est un bon choix pour supporter un large éventail de caractères et pour une comparaison correcte des chaînes de caractères en français et dans d'autres langues.

— **Sélection de la Base de Données avec 'USE coursphp'** : Après avoir créé la base de données (ou si elle existait déjà), il faut la sélectionner pour pouvoir créer des tables à l'intérieur. La requête 'USE coursphp' fait cela.

— **Création de la Table avec 'CREATE TABLE IF NOT EXISTS etudiants (...)'** :
  — 'CREATE TABLE etudiants (...)' : C'est la commande SQL pour créer une table nommée 'etudiants'.
  — 'IF NOT EXISTS' : Comme pour la base de données, 'IF NOT EXISTS' évite les erreurs si la table existe déjà.
  — Définition des colonnes : 'id INT AUTO_INCREMENT PRIMARY KEY, nom TEXT, prenom TEXT, email TEXT, groupe TEXT' définit les colonnes avec leurs types de données et contraintes.
    — 'id INT AUTO_INCREMENT PRIMARY KEY' : Colonne 'id' de type entier ('INT'), auto-incrémentée ('AUTO_INCREMENT') et clé primaire ('PRIMARY KEY').

— ‘nom TEXT, prenom TEXT, email TEXT, groupe TEXT‘ : Colonnes de type ‘TEXT‘ pour stocker du texte.
— ‘ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci‘ Spécifie le moteur de stockage (‘InnoDB‘, un moteur transactionnel et robuste, recommandé par défaut pour MySQL) et le jeu de caractères/collationnement pour la table. Il est important de définir le charset au niveau de la table également pour s’assurer de la cohérence.

Quand Choisir Quelle Méthode ?

— **Méthode 1 (phpMyAdmin)** : Idéale pour l’apprentissage, les tâches de gestion courantes, l’exploration de la base de données, la création rapide de tables, l’import/export de données. Plus conviviale pour les opérations manuelles.
— **Méthode 2 (PHP/SQL)** : Essentielle pour l’automatisation, les scripts d’installation, les migrations de bases de données, l’intégration dans des applications, le contrôle précis de la création. Plus puissante pour les tâches complexes et répétitives.

Bien que la méthode graphique (phpMyAdmin) soit plus simple pour commencer, comprendre et savoir utiliser la méthode programmatique (PHP/SQL) est crucial pour devenir un développeur web compétent.

Connexion à une Base de Données avec PDO - Sécurité, Options et Bonnes Pratiques

Introduction à PDO - Pourquoi PDO ? Avantages Clés PDO (PHP Data Objects) est une extension PHP fondamentale pour interagir avec les bases de données de manière moderne et sécurisée. Avant PDO, il existait d’autres extensions MySQL (comme ‘mysql_*‘ et ‘mysqli_*‘), mais PDO offre des avantages significatifs :

— **Abstraction de la Base de Données (Portabilité)** : PDO fournit une interface unifiée pour accéder à différents types de bases de données (MySQL, PostgreSQL, SQLite, Oracle, etc.). Si vous changez de SGBD, vous pouvez souvent modifier seulement la chaîne de connexion et quelques détails de syntaxe SQL, sans réécrire tout votre code d’accès aux données. Cela rend votre code plus portable et adaptable.
— **Sécurité Améliorée (Prévention des Injections SQL)** : PDO

encourage fortement l'utilisation de requêtes préparées avec des paramètres nommés ou anonymes. Les requêtes préparées sont une défense essentielle contre les attaques par injection SQL, l'une des vulnérabilités web les plus courantes et dangereuses. Nous verrons cela en détail.

— **Orienté Objet et Moderne** : PDO est une extension orientée objet, ce qui correspond aux pratiques de développement PHP modernes. Il utilise des classes et des objets pour représenter les connexions, les requêtes, les résultats, etc., ce qui rend le code plus organisé et maintenable.

— **Fonctionnalités Avancées** : PDO offre des fonctionnalités avancées comme les transactions, la gestion des erreurs améliorée, différents modes de récupération des résultats, etc.

En résumé, PDO est la méthode recommandée et standard pour interagir avec les bases de données en PHP. Ignorer PDO et utiliser les anciennes extensions est fortement déconseillé, surtout en raison des risques de sécurité.

Le Bloc 'try{ } catch{}' - Gestion Robuste des Erreurs Le bloc 'try{ } catch{}' est un mécanisme essentiel en PHP (et dans de nombreux langages de programmation) pour la gestion des exceptions. Les exceptions sont des erreurs qui se produisent pendant l'exécution du code. Dans le contexte de PDO, les erreurs de connexion à la base de données, les erreurs SQL, etc., sont signalées comme des exceptions de type 'PDOException'.

— **Le Bloc 'try{ }' :** Le code susceptible de générer une exception est placé à l'intérieur du bloc 'try'. C'est la partie "normale" du code que vous voulez exécuter.

— **Le Bloc 'catch(PDOException $e){ }'** : Si une exception de type 'PDOException' se produit à l'intérieur du bloc 'try', l'exécution du bloc 'try' est immédiatement interrompue, et le contrôle est transféré au bloc 'catch'. Le bloc 'catch' reçoit l'objet exception '$e', qui contient des informations sur l'erreur (message, code, etc.). Vous pouvez ensuite traiter l'erreur dans le bloc 'catch' (afficher un message d'erreur, journaliser l'erreur, etc.).

Sans 'try...catch', si une exception PDO n'est pas gérée, PHP affichera un message d'erreur fatal et arrêtera le script. Avec 'try...catch', vous pouvez intercepter l'erreur, la gérer de manière contrôlée, et potentiellement continuer l'exécution du script (si cela est logique dans votre application).

Chaîne de Connexion DSN (Data Source Name) et Pilotes PDO

La chaîne de connexion DSN (Data Source Name) est un élément clé lors de la création d'une instance PDO. Elle spécifie le pilote PDO à utiliser,

ainsi que les informations de connexion spécifiques à la base de données. La structure générale d'un DSN est la suivante :

`pilote:paramètre1=valeur1;paramètre2=valeur2;...`

- **Pilote PDO** : Indique le type de base de données à laquelle vous souhaitez vous connecter. Les pilotes PDO courants incluent :
  - `mysql` : Pour MySQL et MariaDB. (Exemple : `mysql:host=localhost;dbname=mabase`)
  - `pgsql` : Pour PostgreSQL. (Exemple : `pgsql:host=localhost;port=5432;dbname=maba`
  - `sqlite` : Pour SQLite. (Exemple : `sqlite:/chemin/vers/mabase.sqlite`)
  - `oci` : Pour Oracle.
  - `sqlsrv` : Pour Microsoft SQL Server.
  - et d'autres...
- **Paramètres de Connexion** : Varient en fonction du pilote PDO utilisé. Les paramètres courants pour le pilote 'mysql' incluent :
  - `host` : Le nom d'hôte ou l'adresse IP du serveur de données.
  - `dbname` : Le nom de la base de données à laquelle se connecter.
  - `port` : Le numéro de port du serveur de base de données (par défaut pour MySQL est 3306).
  - `unix_socket` : Pour les connexions locales à MySQL via un socket Unix (au lieu d'un port TCP/IP).
  - `charset` : Le jeu de caractères à utiliser pour la connexion (recommandé : `utf8mb4`).

  Consultez la documentation PHP pour la liste complète des paramètres disponibles pour chaque pilote PDO.

Exemple de Connexion PDO Détaillé :

```
<?php
    $serveur = "localhost";
    $utilisateur = "root";
    $motDePasse = "";
    $baseDeDonnees = "coursphp";

    try {
        $connexion = new PDO("mysql:host=$serveur;dbname=$
    baseDeDonnees;charset=utf8mb4", $utilisateur, $motDePasse)
    ; % Ajout de charset dans DSN
         $connexion->setAttribute(PDO::ATTR_ERRMODE, PDO::
    ERRMODE_EXCEPTION);
         $connexion->setAttribute(PDO::ATTR_DEFAULT_FETCH_
    MODE, PDO::FETCH_ASSOC); % Définition du mode de fetch par
     défaut
         echo "<p>Connexion réussie à la base de données '
```

```
    coursphp' avec PDO !</p>";
12      } catch (PDOException $e) {
13          die("La connexion à la base de données a échoué : " .
        $e->getMessage());
14      }
15  ?>
```

Explication du Code de Connexion PDO :

- **Variables de Connexion** : '$serveur', '$utilisateur', '$motDePasse', '$baseDeDonnees' stockent les informations de connexion. En pratique, il est souvent préférable de stocker ces informations dans un fichier de configuration séparé, surtout pour les applications plus importantes, pour des raisons de sécurité et de maintenabilité.
- **Création de l'Objet PDO : 'new PDO(...)'** :
  - '"mysql :host=$serveur ;dbname=$baseDeDonnees ;charset=utf8mb4"' : C'est la chaîne de connexion DSN (Data Source Name) pour MySQL.
    - 'mysql :' : Indique le pilote PDO à utiliser (pour MySQL).
    - 'host=$serveur' : Spécifie le serveur MySQL (souvent 'localhost' pour un développement local).
    - 'dbname=$baseDeDonnees' : Indique le nom de la base de données à laquelle se connecter.
    - ';charset=utf8mb4' : Très important ! Spécifie le jeu de caractères pour la connexion. 'utf8mb4' assure une bonne compatibilité avec les caractères Unicode. Omettre le charset peut entraîner des problèmes d'affichage de caractères spéciaux.
  - '$utilisateur', '$motDePasse' : Les informations d'authentification pour se connecter au serveur MySQL. Utiliser 'root' et un mot de passe vide est acceptable pour un environnement de développement local, mais **jamais** en production. En production, utilisez des utilisateurs MySQL dédiés avec des droits d'accès limités et des mots de passe forts.
- **Définition des Attributs PDO : '$connexion->setAttribute(...)'** :
  - 'PDO : :ATTR_ERRMODE, PDO : :ERRMODE_EXCEPTION' : Configure PDO pour signaler les erreurs SQL comme des exceptions 'PDOException'. C'est la méthode de gestion des erreurs la plus recommandée car elle rend le code plus robuste et permet d'utiliser 'try...catch'. Les autres modes sont 'PDO : :ERRMODE_WARNING' (erreurs signalées comme des warnings PHP) et 'PDO : :ERRMODE_SILENT' (erreurs ignorées, à éviter).

— ‘PDO : :ATTR_DEFAULT_FETCH_MODE, PDO : :FETCH_ASSOC‘ : Définit le mode de récupération des résultats par défaut à ‘PDO : :FETCH_ASSOC‘. ‘PDO : :FETCH_ASSOC‘ signifie que les résultats seront retournés sous forme de tableaux associatifs (indexés par les noms des colonnes). C’est souvent le mode le plus pratique pour manipuler les données en PHP. D’autres modes existent (‘PDO : :FETCH_NUM‘, ‘PDO : :FETCH_OBJ‘, ‘PDO : :FETCH_BOTH‘, etc.).

— **Message de Succès et Gestion de l’Erreur** : Si la connexion réussit, un message de succès est affiché. Si une exception ‘PDOException‘ est levée (erreur de connexion), le bloc ‘catch‘ est exécuté, affichant un message d’erreur plus informatif en utilisant ‘$e->getMessage()‘ (qui contient le message d’erreur détaillé de PDO).

Exécution de Requêtes SQL avec PDO - Préparation, Exécution, et Récupération en Détail

Préparation de la Requête avec `prepare()` - La Clé de la Sécurité (Injections SQL) La fonction ‘prepare()‘ est **cruciale** pour la sécurité et l’efficacité lorsque vous exécutez des requêtes SQL avec PDO, surtout si ces requêtes contiennent des données provenant de sources externes (formulaires web, paramètres d’URL, etc.). ‘prepare()‘ permet de créer une requête préparée, qui est une requête SQL avec des espaces réservés (placeholders) pour les valeurs variables.

Pourquoi les Requêtes Préparées Protègent Contre les Injections SQL ? Les injections SQL se produisent lorsque des données malveillantes sont injectées dans une requête SQL, modifiant sa signification et permettant à un attaquant de contourner les contrôles de sécurité, d’accéder à des données non autorisées, de modifier ou de supprimer des données, ou même de prendre le contrôle du serveur de données.

Fonctionnement de `prepare()` et des Placeholders :

— **Requête SQL avec Placeholders** : Au lieu d’insérer directement les valeurs variables dans la requête SQL, vous utilisez des placeholders. Il existe deux types de placeholders :

  — **Placeholders Nommés** : Commencent par un deux-points ‘ :‘ suivi d’un nom (par exemple, ‘ :nom‘, ‘ :email‘). Plus lisibles et recommandés pour les requêtes complexes.

  — **Placeholders Anonymes (Points d’interrogation)** : Utilisent

des points d'interrogation '?' à la place des valeurs. Plus courts, mais moins lisibles si vous avez beaucoup de paramètres.

— `prepare()` **Analyse et Compile la Requête** : Lorsque vous appelez '$connexion->prepare($requeteSQL)', PDO envoie la requête SQL au serveur de données. Le serveur analyse (parse) et compile la requête, créant un plan d'exécution. **À ce stade, les placeholders sont traités comme des espaces réservés, pas comme du code SQL exécutable.**

— **Séparation des Données et du Code SQL** : La magie de la sécurité réside dans le fait que les données que vous fournirez plus tard pour remplacer les placeholders seront traitées **uniquement comme des données**, et non comme du code SQL. PDO s'assure que les données sont correctement échappées et encodées pour être insérées en toute sécurité dans la requête, **même si elles contiennent des caractères spéciaux ou du code SQL malveillant**.

— **Objet PDOStatement (Déclaration Préparée)** : **$connexion->prepare($requeteSQL)** retourne un objet de type `PDOStatement`. Cet objet '$statement' représente la requête préparée et contient des méthodes pour exécuter la requête et récupérer les résultats.

Exemple avec Placeholder Nommé :

```
<?php
$requete = "SELECT * FROM etudiants WHERE nom = :nomEtudiant"
    ; % Placeholder nommé :nomEtudiant
$statement = $connexion->prepare($requete);
?>
```

Exécution de la Requête avec `execute()` - Fournir les Valeurs et Exécuter La fonction 'execute()' est utilisée pour exécuter une requête préparée (représentée par un objet `PDOStatement`). C'est à ce moment que vous fournissez les valeurs qui remplaceront les placeholders dans la requête préparée.

Passer les Valeurs à `execute()` : Un Tableau Associatif 'execute()' prend un argument optionnel : un tableau associatif. Les clés de ce tableau correspondent aux noms des placeholders (pour les placeholders nommés) ou à l'ordre des placeholders (pour les placeholders anonymes). Les valeurs du tableau sont les valeurs que vous voulez insérer dans la requête.

Exemple d'Exécution avec `execute()` et Placeholder Nommé :

```
<?php
\$nomRecherche = "Dupont"; % Valeur à rechercher
```

```
3
4 $requete = "SELECT * FROM etudiants WHERE nom = :nomEtudiant"
    ;
5 $statement = \$connexion->prepare(\$requete);
6
7 $statement->execute([':nomEtudiant' => $nomRecherche]); %
    Tableau associatif pour les valeurs
8 ?>
```

Dans cet exemple, '[' :nomEtudiant' => $nomRecherche]' est le tableau associatif passé à 'execute()'. La clé '' :nomEtudiant'' correspond au placeholder nommé ' :nomEtudiant' dans la requête préparée. La valeur '$nomRecherche' ("Dupont") sera utilisée pour remplacer le placeholder.

Récupération des Résultats avec `fetch()` et `fetchAll()` - Explorer les Données Récupérées Après avoir exécuté une requête `SELECT` avec `execute()`, vous devez récupérer les résultats. PDO offre plusieurs méthodes pour cela : `fetch()`, `fetchAll()`, et d'autres (comme `fetchColumn()`, `fetchObject()`, etc.). Nous allons nous concentrer sur `fetch()` et `fetchAll()`, qui sont les plus couramment utilisées.

- `fetch(mode de fetch)` : Récupère **une seule ligne** de résultat à la fois. Chaque appel à `fetch()` avance le curseur au résultat suivant. Retourne 'false' lorsqu'il n'y a plus de lignes à récupérer (fin des résultats). Idéal pour parcourir les résultats ligne par ligne avec une boucle `while`.
- `fetchAll(mode de fetch)` : Récupère **toutes les lignes** de résultat en une seule fois et les retourne sous forme de **tableau**. Plus simple si vous voulez traiter tous les résultats en une seule opération (par exemple, afficher tous les étudiants dans une liste). Peut être moins efficace si vous avez un très grand nombre de résultats, car tout est chargé en mémoire en même temps.

Le Paramètre "Mode de Fetch" : Comment Organiser les Résultats Les fonctions `fetch()` et `fetchAll()` acceptent un paramètre optionnel : le **mode de fetch**. Le mode de fetch détermine comment les données de chaque ligne de résultat seront organisées et retournées en PHP. Les modes de fetch les plus courants sont :

- `PDO::FETCH_ASSOC` : Retourne un tableau **associatif**. Les clés du tableau sont les **noms des colonnes** de la table (par exemple, '['nom' => 'Dupont', 'prenom' => 'Jean', 'email' => 'jean.dupont@example.com']').

Très pratique car vous pouvez accéder aux données en utilisant les noms des colonnes (plus lisible que les index numériques). **C'est le mode de fetch par défaut que nous avons configuré avec '$connexion->setAttribute(PDO::ATTR_DEFAULT_FETCH_MODE, PDO::FETCH_ASSOC);'**

— `PDO::FETCH_NUM` : Retourne un tableau **numérique**. Les clés du tableau sont des **index numériques** (0, 1, 2, ...) correspondant à l'ordre des colonnes dans la requête `SELECT` (par exemple, '['Dupont', 'Jean', 'jean.dupont@example.com']'). Moins lisible que `FETCH_ASSOC` car vous devez vous rappeler l'ordre des colonnes.

— `PDO::FETCH_OBJ` : Retourne un **objet anonyme**. Les attributs de l'objet correspondent aux **noms des colonnes** (par exemple, '(object) {'nom' => 'Dupont', 'prenom' => 'Jean', 'email' => 'jean.dupont@example.com'}'). Utile si vous préférez accéder aux données comme des propriétés d'objet (par exemple, '$row->nom').

— `PDO::FETCH_BOTH` : Retourne un tableau **à la fois associatif et numérique**. Chaque colonne est accessible à la fois par son nom et par son index numérique (par exemple, '[0 => 'Dupont', 'nom' => 'Dupont', 1 => 'Jean', 'prenom' => 'Jean', ...]'). Moins efficace en termes de mémoire et généralement moins pratique que `FETCH_ASSOC` ou `FETCH_NUM`.

Exemple avec `fetch()` et `PDO::FETCH_ASSOC` :

```php
<?php
    // ... (connexion PDO, préparation et exécution de la requête SELECT ...)

    echo "<ul>";
    while ($row = $statement->fetch(PDO::FETCH_ASSOC)) { % Récupération ligne par ligne en mode associatif
        echo "<li>Nom: " . $row["nom"] . ", Prenom: " . $row["prenom"] . ", Email: " . $row["email"] . "</li>";
    }
    echo "</ul>";
?>
```

Exemple avec `fetchAll()` et `PDO::FETCH_ASSOC` :

```php
<?php
    // ... (connexion PDO, préparation et exécution de la requête SELECT ...)

    $resultats = $statement->fetchAll(PDO::FETCH_ASSOC); % Récupération de toutes les lignes en mode associatif
```

```php

    echo "<ul>";
    foreach ($resultats as $row) { % Parcours du tableau de r
    ésultats
        echo "<li>Nom: " . $row["nom"] . ", Prenom: " . $row[
    "prenom"] . ", Email: " . $row["email"] . "</li>";
    }
    echo "</ul>";
?>
```

Exercice Pratique Guidé (TP7) - CRUD Complet avec PDO

Appliquez les connaissances acquises pour créer une application CRUD complète (Créer, Lire, Mettre à jour, Supprimer) pour la gestion des étudiants en utilisant PDO.

Objectif du TP - CRUD Complet pour la Table Étudiants

Créer une page PHP nommée `tp7.php` et des fichiers associés qui permettent de :

1. **Créer** de nouveaux étudiants dans la table "etudiants".
2. **Lire** et afficher la liste des étudiants présents dans la table.
3. **Mettre à jour** les informations d'un étudiant existant.
4. **Supprimer** un étudiant de la table.

Structure des Fichiers pour le TP7

Pour organiser votre code, créez les fichiers PHP suivants dans un dossier 'tp7' :

- `connexion.php` : Contiendra la fonction de connexion à la base de données.
- `ajouter_etudiant.php` : Formulaire et code pour ajouter un étudiant.
- `liste_etudiants.php` : Code pour afficher la liste des étudiants (avec liens Modifier/Supprimer).
- `modifier_etudiant.php` : Formulaire et code pour modifier un étudiant.

— `supprimer_etudiant.php` : Code pour supprimer un étudiant (peut être géré via un lien et redirection).
— `tp7.php` : Page principale qui inclut les autres fichiers et structure la page web.

Fichier `connexion.php` (à Créer) Créez un fichier nommé `connexion.php` dans votre dossier `tp7` et copiez-y le code suivant. Ce fichier contiendra la fonction ‘seConnecter()‘ que vous réutiliserez dans les autres fichiers PHP.

```php
<?php
    $serveur = "localhost";
    $utilisateur = "root";
    $motDePasse = "";
    $baseDeDonnees = "coursphp";
    $connexion = null; // Initialisation de la connexion à null pour la portée globale

function seConnecter() {
    global $serveur, $utilisateur, $motDePasse, $baseDeDonnees, $connexion;
    try {
        $connexion = new PDO("mysql:host=$serveur;dbname=$baseDeDonnees;charset=utf8mb4", $utilisateur, $motDePasse);
        $connexion->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
        $connexion->setAttribute(PDO::ATTR_DEFAULT_FETCH_MODE, PDO::FETCH_ASSOC);
        return true; // Connexion réussie
    } catch (PDOException $e) {
        echo "<div style='color:red;'>Erreur de connexion à la base de données : " . htmlspecialchars($e->getMessage()) . "</div>";
        return false; // Connexion échouée
    }
}
?>
```

Fichier `ajouter_etudiant.php` (à Créer) Créez un fichier nommé `ajouter_etudiant.php` dans votre dossier `tp7` et ajoutez-y le formulaire HTML et le code PHP pour traiter l’ajout d’un étudiant.

Formulaire HTML (dans `ajouter_etudiant.php`)

```
<h2>Ajouter un Etudiant</h2>
<form method="POST" action="">
    <input type="hidden" name="action" value="ajouter"> <%-- Champ caché pour identifier l'action --%>
    <label for="nom">Nom:</label>
    <input type="text" id="nom" name="nom" required><br>

    <label for="prenom">Prénom:</label>
    <input type="text" id="prenom" name="prenom" required><br>

    <label for="email">Email:</label>
    <input type="email" id="email" name="email" required><br>

    <label for="groupe">Groupe:</label>
    <input type="text" id="groupe" name="groupe"><br>

    <input type="submit" value="Ajouter Etudiant">
</form>
```

Code PHP pour Traiter l'Ajout (dans `ajouter_etudiant.php` - à placer avant le formulaire HTML) :

```
<?php
include 'connexion.php'; // Inclure connexion.php au début
if ($_POST["action"] === 'ajouter') {
    if (seConnecter()) {
        $nom = $_POST["nom"] ?? '';
        $prenom = $_POST["prenom"] ?? '';
        $email = $_POST["email"] ?? '';
        $groupe = $_POST["groupe"] ?? '';

        try {
            $requeteInsertion = "INSERT INTO etudiants (nom, prenom, email, groupe) VALUES (:nom, :prenom, :email, :groupe)";
            $statementInsertion = $connexion->prepare($requeteInsertion);
            statementInsertion->execute([
                ':nom' => $nom,
                ':prenom' => $prenom,
                ':email' => $email,
                ':groupe' => $groupe
            ]);
            echo "<div style='color:green;'>Etudiant ajouté avec succès.</div>";
```

```
        } catch (PDOException $e) {
            echo "<div style='color:red;'>Erreur lors de l'
    ajout de l'étudiant : " . htmlspecialchars($e->getMessage
    ()) . "</div>";
        } finally {
            $connexion = null; // Déconnexion après l'opé
    ration
        }
    }
}
?>
```

Fichier `liste_etudiants.php` (à Créer) Créez un fichier nommé `liste_etudiants.php` dans votre dossier `tp7` et ajoutez-y le code PHP pour afficher la liste des étudiants.

Code PHP pour Afficher la Liste (dans `liste_etudiants.php`) :

```
<h2>Liste des Etudiants</h2>
<?php
include 'connexion.php'; // Inclure connexion.php au début
if (seConnecter()) {
    try {
        $requeteSelection = "SELECT * FROM etudiants ORDER BY
     nom, prenom";
        $statementSelection = $connexion->prepare($
    requeteSelection);
        $statementSelection->execute();
        $etudiants = $statementSelection->fetchAll();

        if (count($etudiants) > 0) {
            echo "<ul>";
            foreach ($etudiants as $etudiant) {
                echo "<li>";
                echo htmlspecialchars($etudiant["nom"]) . " "
     . htmlspecialchars($etudiant["prenom"]) . " - " .
    htmlspecialchars($etudiant["email"]) . " (Groupe: " .
    htmlspecialchars($etudiant["groupe"]) . ") ";
                echo "<a href='tp7.php?action=modifier&id=" .
     $etudiant["id"] . "'>Modifier</a> | ";
                echo "<a href='tp7.php?action=supprimer&id="
    . $etudiant["id"] . "' onclick='return confirm("Etes-vous
    sûr de vouloir supprimer cet étudiant ?")'>Supprimer</a>";
                echo "</li>";
            }
```

```
            echo "</ul>";
        } else {
            echo "<p>Aucun étudiant trouvé.</p>";
        }
    } catch (PDOException \$e) {
        echo "<div style='color:red;'>Erreur lors de la récup
    ération des étudiants : " . htmlspecialchars(\$e->
    getMessage()) . "</div>";
    } finally {
        \$connexion = null; // Déconnexion après l'opération
    }
}
?>
```

Fichier `modifier_etudiant.php` (à Créer) Créez un fichier nommé `modifier_etudiant.php` dans votre dossier `tp7` et ajoutez-y le formulaire HTML et le code PHP pour traiter la modification d'un étudiant.

Formulaire HTML (dans `modifier_etudiant.php` - à placer après le code PHP de traitement de la modification) :

```
<?php
include 'connexion.php'; // Inclure connexion.php au début
if (isset($_GET["id"])) {
    $idEtudiantModifier = $_GET["id"];
    if (seConnecter()) {
        try {
            $requeteSelectionEtudiant = "SELECT * FROM
    etudiants WHERE id = :id";
            $statementSelectionEtudiant = $connexion->prepare
    ($requeteSelectionEtudiant);
            $statementSelectionEtudiant->execute([':id' => $
    idEtudiantModifier]);
            $etudiant = $statementSelectionEtudiant->fetch();

            if ($etudiant) {
                ?>
                <h2>Modifier un Etudiant</h2>
                <form method="POST" action="tp7.php"> <%--
    Action redirige vers tp7.php pour le traitement --%>
                    <input type="hidden" name="action" value=
    "modifier"> <%-- Champ caché pour identifier l'action --%>
                    <input type="hidden" name="id" value="<?
    php echo htmlspecialchars($etudiant['id']); ?>"> <%--
    Champ caché pour l'ID de l'étudiant --%>
```

```
                    <label for="nom">Nom:</label>
                    <input type="text" id="nom" name="nom"
    value="<?php echo htmlspecialchars($etudiant['nom']); ?>"
    required><br>

                    <label for="prenom">Prénom:</label>
                    <input type="text" id="prenom" name="
    prenom" value="<?php echo htmlspecialchars($etudiant['
    prenom']); ?>" required><br>

                    <label for="email">Email:</label>
                    <input type="email" id="email" name="
    email" value="<?php echo htmlspecialchars($etudiant['email
    ']); ?>" required><br>

                    <label for="groupe">Groupe:</label>
                    <input type="text" id="groupe" name="
    groupe" value="<?php echo htmlspecialchars($etudiant['
    groupe']); ?>"><br>

                    <input type="submit" value="Mettre à jour
     Etudiant">
                </form>
                <?php
            } else {
                echo "<div style='color:red;'>Etudiant non
    trouvé.</div>";
            }
        } catch (PDOException $e) {
            echo "<div style='color:red;'>Erreur lors de la r
    écupération de l'étudiant pour modification : " .
    htmlspecialchars($e->getMessage()) . "</div>";
        } finally {
            $connexion = null; // Déconnexion après l'opé
    ration
        }
    }
}
?>
```

Code PHP pour Traiter la Mise à Jour (dans `modifier_etudiant.php` - à placer au début du fichier) :

```
<?php
if ($_POST["action"] === 'modifier') {
    if (seConnecter()) {
```

```
        $id = $_POST["id"] ?? '';
        $nom = $_POST["nom"] ?? '';
        $prenom = $_POST["prenom"] ?? '';
        $email = $_POST["email"] ?? '';
        $groupe = $_POST["groupe"] ?? '';

        try {
            $requeteMiseAJour = "UPDATE etudiants SET nom = :nom, prenom = :prenom, email = :email, groupe = :groupe WHERE id = :id";
            $statementMiseAJour = $connexion->prepare($requeteMiseAJour);
            $statementMiseAJour->execute([
                ':id' => $id,
                ':nom' => $nom,
                ':prenom' => $prenom,
                ':email' => $email,
                ':groupe' => $groupe
            ]);
            echo "<div style='color:green;'>Etudiant mis à jour avec succès.</div>";
        } catch (PDOException $e) {
            echo "<div style='color:red;'>Erreur lors de la mise à jour de l'étudiant : " . htmlspecialchars($e->getMessage()) . "</div>";
        } finally {
             $connexion = null; // Déconnexion après l'opération
        }
    }
}
?>
```

Fichier `supprimer_etudiant.php` (à Créer) Créez un fichier nommé `supprimer_etudiant.php` dans votre dossier `tp7` et ajoutez-y le code PHP pour traiter la suppression d'un étudiant. Ce fichier sera appelé lorsque l'utilisateur clique sur le lien "Supprimer" dans la liste des étudiants. Après la suppression, vous pouvez rediriger l'utilisateur vers la page principale `tp7.php`.

Code PHP pour Traiter la Suppression (dans `supprimer_etudiant.php`) :

```
<?php
include 'connexion.php'; // Inclure le fichier de connexion

if (isset($_GET["id"])) {
```

```
    $idEtudiantSupprimer = $_GET["id"];
    if (seConnecter()) {
        try {
            $requeteSuppression = "DELETE FROM etudiants WHERE id = :id";
            $statementSuppression = \$connexion->prepare($requeteSuppression);
            $statementSuppression->execute([':id' => $idEtudiantSupprimer]);
            echo "<div style='color:green;'>Etudiant supprimé avec succès.</div>";
        } catch (PDOException $e) {
            echo "<div style='color:red;'>Erreur lors de la suppression de l'étudiant : " . htmlspecialchars($e->getMessage()) . "</div>";
        } finally {
            $connexion = null; // Déconnexion après l'opération
            header("Location: tp7.php"); // Rediriger vers la page principale après la suppression
            exit(); // Terminer le script après la redirection
        }
    }
} else {
    echo "<div style='color:red;'>ID de l'étudiant à supprimer non spécifié.</div>";
}
?>
```

Fichier `tp7.php` (Page Principale - à Créer) Créez un fichier nommé `tp7.php` dans votre dossier `tp7`. Ce fichier sera la page principale qui inclura les autres fichiers PHP pour composer l'application CRUD.

Code HTML et Inclusion des Fichiers (dans `tp7.php`) :

```
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <title>TP7 - Gestion des Etudiants</title>
</head>
<body>

<h1>Gestion des Etudiants</h1>
```

```php

<?php
    include 'connexion.php'; // Inclure le fichier de connexion

    // Traitement de la modification (s'il y a soumission du formulaire de modification)
    include 'modifier_etudiant.php'; // Inclut aussi le formulaire de modification, mais ne l'affiche que si action=modifier est dans l'URL

    include 'ajouter_etudiant.php'; // Formulaire et traitement pour ajouter un étudiant
    echo "<hr>"; // Séparateur visuel
    include 'liste_etudiants.php';   // Liste des étudiants et liens Modifier/Supprimer

?>

</body>
</html>
```

À Compléter par l'Étudiant - Exercices d'Approfondissement CRUD

Modifiez et enrichissez le code CRUD précédent pour améliorer l'application de gestion des étudiants. Ces exercices visent à renforcer vos compétences en développement web avec PHP et PDO.

TP8 : Amélioration de la Validation des Données et Gestion des Erreurs Utilisateur Objectif du TP8 : Rendre l'application plus robuste et conviviale en ajoutant des validations de données côté serveur et en améliorant la gestion des erreurs utilisateur.

1. **Validation Côté Serveur Plus Robuste** :
   - **Dans les fichiers `ajouter_etudiant.php` et `modifier_etudiant.php`,** modifiez le code PHP pour ajouter des validations plus complètes côté serveur pour tous les champs du formulaire avant d'insérer ou de mettre à jour les données.
     - **Nom et Prénom :** Utilisez des expressions régulières (fonction 'preg_match' en PHP) pour vérifier que le nom et le prénom ne contiennent que des lettres et des espaces. Par exemple,

‘preg_match("/[a-zA-Z
s]+$/", $nom)‘.
- **Email :** Utilisez la fonction ‘filter_var‘ avec le filtre ‘FILTER_VALIDATE_EMAIL‘ pour valider le format de l’adresse email. Par exemple, ‘filter_var($email, FILTER_VALIDATE_EMAIL)‘.
- **Groupe :** Vérifiez que le champ "groupe" ne dépasse pas une certaine longueur maximale (par exemple, 255 caractères) en utilisant ‘strlen($groupe)‘.
- **Champs Obligatoires :** Vérifiez que les champs obligatoires (nom, prénom, email) ne sont pas vides en utilisant ‘empty($nom)‘, ‘empty($prenom)‘, ‘empty($email)‘.

— **Gestion des Erreurs de Validation :** Si des erreurs de validation sont détectées, n’exécutez pas les requêtes ‘INSERT‘ ou ‘UPDATE‘. Stockez les messages d’erreur dans un tableau (par exemple, ‘$erreurs = []‘). Affichez ces messages d’erreur à l’utilisateur au-dessus du formulaire, indiquant quels champs sont incorrects et pourquoi. Pour faciliter la correction, réaffichez les valeurs saisies par l’utilisateur dans les champs du formulaire (en utilisant l’attribut ‘value‘ des champs ‘input‘).

2. **Gestion des Erreurs Utilisateur Améliorée** :
   — **Messages Flash (Sessions) :** Au lieu d’afficher les messages d’erreur et de succès directement dans la page, utilisez des sessions PHP pour implémenter un système de messages flash.
   - **Au début de votre fichier `tp7.php` (et potentiellement dans `supprimer_etudiant.php`), démarrez une session avec ‘session_start() ;‘**.
   - **Dans les fichiers de traitement (ajouter, modifier, supprimer),** en cas de succès, stockez un message de succès dans la session (par exemple, ‘$_SESSION[’message_succes’] = "Etudiant ajouté avec succès." ;‘). En cas d’erreur (validation ou erreur PDO), stockez un message d’erreur (par exemple, ‘$_SESSION[’message_erreur’] = "Erreur lors de l’ajout de l’étudiant." ;‘).
   - **Dans `tp7.php`,** avant d’inclure les autres fichiers, ajoutez un code PHP pour vérifier s’il y a des messages flash dans la session. Si oui, affichez-les (dans des divs stylisés par exemple) et **supprimez-les de la session immédiatement après l’affichage** (pour qu’ils ne s’affichent qu’une seule fois). Utilisez ‘is-

set($_SESSION['message_succes'])‘, ‘echo $_SESSION['message_succes']‘, puis ‘unset($_SESSION['message_succes'])‘. Faites de même pour les messages d'erreur.

— **Amélioration de l'Interface Utilisateur (CSS) :** Créez un fichier CSS simple (par exemple, `style.css` dans le dossier `tp7`) ou utilisez un framework CSS (Bootstrap, Tailwind CSS). Lie

TP9 : Pagination de la Liste des Étudiants et Fonctionnalité de Recherche Objectif du TP9 : Améliorer la performance et la convivialité de l'affichage de la liste des étudiants en ajoutant la pagination et une fonctionnalité de recherche.

(a) **Pagination de la Liste des Étudiants** :
— **Modifier `liste_etudiants.php` :**
— **Définir le nombre d'étudiants par page :** Choisissez un nombre raisonnable, par exemple 5 ou 10, et stockez-le dans une variable PHP (par exemple, ‘$etudiantsParPage = 10 ;‘).
— **Calculer la page courante** : Récupérez le numéro de page courant à partir des paramètres de l'URL (par exemple, via ‘$_GET['page']‘). Si le paramètre 'page' n'est pas défini ou n'est pas un nombre valide, la page courante est 1.
— **Calculer l'offset SQL :** Calculez la clause ‘OFFSET‘ pour la requête SQL en fonction de la page courante et du nombre d'étudiants par page. La formule est : ‘$offset = ($pageCourante - 1) * $etudiantsParPage ;‘.
— **Modifier la requête SQL :** Modifiez la requête ‘SELECT‘ dans `liste_etudiants.php` pour utiliser les clauses ‘LIMIT‘ et ‘OFFSET‘. Par exemple :

```
1                          SELECT * FROM etudiants
    ORDER BY nom, prenom LIMIT :limit OFFSET :
   offset
2
```

Utilisez des placeholders nommés ‘ :limit‘ et ‘ :offset‘ et liez-les avec ‘bindValue()‘ en utilisant les valeurs de ‘$etudiantsParPage‘ (pour ‘LIMIT‘) et ‘$offset‘ (pour ‘OFFSET‘), en veillant à convertir ces valeurs en entiers avec ‘PDO : :PARAM_INT‘.

- **Récupérer le nombre total d'étudiants :** Avant de récupérer les étudiants paginés, exécutez une requête 'SELECT COUNT(*) FROM etudiants' pour obtenir le nombre total d'étudiants dans la base de données.
- **Calculer le nombre total de pages :** Calculez le nombre total de pages en divisant le nombre total d'étudiants par le nombre d'étudiants par page et en arrondissant à l'entier supérieur (fonction 'ceil()' en PHP).
- **Afficher les liens de pagination :** Affichez des liens "Page précédente", "Page suivante" et/ou une numérotation des pages en dessous de la liste des étudiants. Ces liens doivent inclure le paramètre 'page' dans l'URL (par exemple, 'tp7.php ?page=2'). Gérez les cas limites (page 1, dernière page) pour désactiver les liens "Page précédente" et "Page suivante" si nécessaire.

— **Fonctionnalité de Recherche par Nom ou Email** :

- **Ajouter un formulaire de recherche dans `tp7.php` :** Ajoutez un formulaire HTML au-dessus de la liste des étudiants dans `tp7.php`. Ce formulaire contiendra un champ de type 'text' nommé 'recherche' et un bouton "Rechercher".
- **Modifier `liste_etudiants.php` :**
  - **Récupérer le mot-clé de recherche :** Dans `liste_etudiants.php`, récupérez le mot-clé de recherche saisi par l'utilisateur à partir des paramètres de l'URL (par exemple, via '$_GET['recherche']'). Utilisez 'isset()' et 'trim()' pour vérifier si un mot-clé a été saisi et pour supprimer les espaces inutiles.
  - **Modifier la requête SQL conditionnellement :**
    - **Si un mot-clé de recherche est présent :** Modifiez la requête 'SELECT' pour inclure une clause 'WHERE' avec l'opérateur 'LIKE' pour rechercher les étudiants dont le nom ou l'email contient le mot-clé saisi. Par exemple :

```
1                                   SELECT * FROM
  etudiants WHERE nom LIKE :recherche OR
  email LIKE :recherche ORDER BY nom,
  prenom LIMIT :limit OFFSET :offset
2
```

Utilisez un placeholder nommé ' :recherche' et liez-le avec 'bindValue()' en ajoutant des jokers '%' avant

et après le mot-clé pour la recherche ‘LIKE‘ (par exemple, ‘bindValue(’ :recherche’, ’

— **Si aucun mot-clé de recherche n’est présent :** Exécutez la requête ‘SELECT‘ originale (sans clause ‘WHERE‘), mais en conservant les clauses ‘LIMIT‘ et ‘OFFSET‘ pour la pagination.

— **Adapter l’affichage :** Affichez uniquement les étudiants correspondant à la recherche. Si aucun étudiant ne correspond à la recherche, affichez un message indiquant "Aucun étudiant trouvé pour cette recherche." Si aucun mot-clé n’est saisi, affichez la liste paginée complète (ou la page courante).

— **Conserver le mot-clé dans le formulaire :** Réaffichez le mot-clé de recherche dans le champ de recherche après la soumission du formulaire (en utilisant l’attribut ‘value‘ du champ ‘input‘) pour que l’utilisateur puisse facilement modifier sa recherche.

TP10 : Tri Dynamique de la Liste des Étudiants et Sécurité Renforcée **Objectif du TP10 :** Ajouter le tri dynamique de la liste des étudiants et renforcer la sécurité de l’application.

1. **Tri Dynamique de la Liste des Étudiants** :
   — **Modifier `liste_etudiants.php` :**
   — **Liens de tri dans l’en-tête du tableau (si vous utilisez un tableau pour afficher la liste, sinon, adaptez) :** Modifiez l’affichage de la liste des étudiants pour afficher un en-tête (par exemple, au-dessus de la liste ‘<ul>‘ ou dans un ‘<thead>‘ d’un tableau). Dans cet en-tête, ajoutez des liens cliquables pour chaque colonne que vous souhaitez trier (par exemple, "Nom", "Prénom", "Email", "Groupe").
   — **Paramètres de tri dans l’URL :** Lorsque l’utilisateur clique sur un lien de tri, passez le nom de la colonne de tri et l’ordre de tri (ascendant ou descendant) dans l’URL. Par exemple, pour trier par nom en ordre ascendant, le lien pourrait être :

   ‘tp7.php ?tri=nom&ordre=asc‘. Pour l’ordre descendant,

   ‘tp7.php ?tri=nom&ordre=desc‘. Utilisez des icônes (par exemple, des flèches) pour indiquer visuellement l’ordre de tri courant.

— **Récupérer les paramètres de tri :** Dans `liste_etudiants.php`, récupérez le nom de la colonne de tri (‘$_GET[’tri’]‘) et l’ordre de tri (‘$_GET[’ordre’]‘) à partir des paramètres de l’URL. Validez ces paramètres pour vous assurer qu’ils sont valides (par exemple, que la colonne de tri est bien une colonne existante de la table ‘etudiants‘ et que l’ordre de tri est ‘asc‘ ou ‘desc‘). Si les paramètres ne sont pas valides, utilisez un tri par défaut (par exemple, par nom ascendant).

— **Modifier la requête SQL avec `ORDER BY` :** Modifiez la requête ‘SELECT‘ dans `liste_etudiants.php` pour utiliser la clause ‘ORDER BY‘ en fonction des paramètres de tri récupérés. Par exemple :

```
1                          SELECT * FROM etudiants
   ORDER BY :colonneTri :ordreTri LIMIT :limit
   OFFSET :offset
2
```

Utilisez des placeholders nommés ‘ :colonneTri‘ et ‘ :ordreTri‘ pour la colonne de tri et l’ordre de tri. **Attention :** Vous ne pouvez pas directement lier les noms de colonnes et les ordres de tri comme des valeurs avec ‘bindValue()‘ car ce sont des identifiants SQL, pas des valeurs de données. Vous devrez faire une **liste blanche** des colonnes de tri autorisées et des ordres de tri autorisés et les insérer directement dans la requête après validation (sans ‘bindValue()‘ pour ces parties spécifiques, mais en utilisant toujours ‘bindValue()‘ pour les autres paramètres comme ‘LIMIT‘ et ‘OFFSET‘ et le mot-clé de recherche si applicable). Par exemple, construisez la clause ‘ORDER BY‘ dynamiquement en PHP après avoir validé ‘$_GET[’tri’]‘ et ‘$_GET[’ordre’]‘.

— **Mémoriser le tri courant :** Conservez les paramètres de tri courants (colonne et ordre) dans l’URL lors de la pagination ou de la recherche, afin de maintenir l’état de tri lorsque l’utilisateur navigue dans les pages ou effectue une recherche. Ajoutez les paramètres de tri aux liens de pagination et au formulaire de recherche.

— **Sécurité Renforcée - Prévention des Injections SQL Avancées et XSS** :

— **Revue du Code et Requêtes Préparées :** Revérifiez attentivement **tout** le code PHP de votre application CRUD (fichiers `ajouter_etudiant.php`, `liste_etudiants.php`, `modifier_etudiant.php`, `supprimer_etudiant.php`, `tp7.php` et `connexion.php`). Assurez-vous que **toutes** les requêtes SQL, sans exception, utilisent des requêtes préparées avec des placeholders pour prévenir les injections SQL. Soyez particulièrement vigilant dans les parties où vous construisez dynamiquement des requêtes SQL (comme pour le tri dynamique), et assurez-vous de valider et de nettoyer correctement toutes les données provenant de l'utilisateur (‘$_POST‘, ‘$_GET‘, ‘$_SESSION‘, ‘$_COOKIE‘).
— **Échappement XSS Systématique :** Assurez-vous d'utiliser **systématiquement** la fonction ‘htmlspecialchars()‘ pour **échapper toutes les données affichées** dans la page HTML, afin de prévenir les attaques XSS (Cross-Site Scripting). Cela inclut :
  — Les données récupérées de la base de données et affichées dans la liste des étudiants, les formulaires de modification, etc.
  — Les messages de succès et d'erreur affichés à l'utilisateur.
  — Les valeurs réaffichées dans les champs de formulaire après une validation (pour éviter que du code malveillant ne soit stocké dans les champs et exécuté lors de l'affichage).
— **Configuration de Sécurité du Serveur Web (Optionnel, mais important en production) :** Si vous avez accès à la configuration de votre serveur web (Apache, Nginx), renseignez-vous sur les bonnes pratiques de sécurité pour PHP et MySQL (par exemple, désactiver l'affichage des erreurs PHP en production, configurer correctement les permissions des fichiers, sécuriser l'accès à phpMyAdmin, etc.). Bien que cela dépasse le cadre strict de ce TP d'introduction à PDO, c'est un aspect crucial de la sécurité web à connaître pour de futurs projets.

Conclusion de la Séance 7 et Perspectives

Félicitations ! Vous avez exploré en profondeur l'interaction entre PHP et les bases de données avec PDO. Nous avons couvert :

* L'importance cruciale des bases de données pour les applications web dynamiques. * Le rôle de MySQL comme SGBD open source populaire et performant. * L'introduction à PDO (PHP Data Objects) comme extension PHP moderne et sécurisée pour accéder aux bases de données. * La connexion à une base de données MySQL avec PDO, en mettant l'accent sur la sécurité et les bonnes pratiques. * L'exécution de requêtes SQL avec PDO, en détaillant la préparation des requêtes, l'utilisation de placeholders pour prévenir les injections SQL, l'exécution des requêtes et la récupération des résultats avec 'fetch()' et 'fetchAll()'. * La mise en œuvre d'un exemple concret de CRUD (Créer, Lire, Mettre à jour, Supprimer) complet pour la gestion des étudiants, en utilisant PDO. * Des exercices d'approfondissement (TP8, TP9, TP10) pour améliorer la validation des données, la gestion des erreurs, la pagination, la recherche, le tri dynamique et la sécurité de l'application.

Vous avez maintenant les bases solides pour développer des applications web dynamiques et interactives qui stockent et manipulent des données de manière efficace et sécurisée.

Perspectives pour les prochaines séances :

* **Séance 8 et suivantes :** Nous explorerons des concepts plus avancés des bases de données et de PHP, tels que : * Les relations entre les tables (clés étrangères, jointures SQL - 'JOIN'). * La conception de bases de données plus complexes et normalisées. * L'authentification et la gestion des utilisateurs (sessions, cookies, mots de passe sécurisés). * Les transactions pour garantir l'intégrité des données lors d'opérations complexes. * L'utilisation de frameworks PHP pour simplifier et accélérer le développement d'applications web robustes et maintenables. * Les bonnes pratiques de sécurité web avancées.

N'oubliez pas de **réaliser activement les exercices "À compléter par l'étudiant" (TP8, TP9, TP10)**. C'est en pratiquant et en expérimentant que vous consoliderez vos connaissances et développerez vos compétences de développeur web. Explorez la documentation PHP sur PDO, MySQL et SQL pour approfondir votre compréhension et découvrir de nouvelles fonctionnalités.

Le développement web est un domaine en constante évolution. Continuez à apprendre, à pratiquer et à explorer les nouvelles technologies et les meilleures pratiques pour devenir un développeur web compétent et recher-

ché.