Interaction avec les Bases de Données

Utilisation de PHP-PgSQL

Pour établir une connexion à une base de données , vous avez besoin des informations suivantes:

- Adresse du serveur de base de données
- Nom d'utilisateur pour la connexion à la base de données
- Mot de passe pour la connexion à la base de données
- Nom de la base de données à laquelle vous souhaitez vous connecter

Avec ces informations, vous pouvez utiliser la fonction PHP « **pg_connect()** » pour établir une connexion à une base de données PostgreSQL

```
<?php
    $host = "localhost";
    $port = "5432";
    $database = "your_database_name";
    $username = "your_username";
    $password = "your_password";

    // Create a connection
    $connection = pg_connect("host=$host port=$port dbname=$database
user=$username password=$password");

    if (!$connection) {
        die("Connection failed: " . pg_last_error());
    }

    echo "Connected successfully";
?>
```

Créer (insérer) des données

Commençons par créer (insérer) des données dans un tableau. Supposons que vous ayez une table « utilisateurs » avec les colonnes `id`, `username`et `email`. Voici comment insérer un nouvel utilisateur :

```
<?php
    // SQL query to insert data into the "users" table
    $sql = "INSERT INTO users (username, email) VALUES ('john_doe',
'john.doe@example.com')";
    $insertResult = pg_query($connection, $sql);
    if ($insertResult === false) {
        die("Error: " . pg_last_error());
    }
```

```
    echo "Data inserted successfully";
?>
```

Lire (sélectionner) les données

Vous pouvez récupérer les données de la table « utilisateurs » à l'aide de requêtes SQL. Voici un exemple de sélection de données dans le tableau et de leur affichage :

```
<?php
    // SQL query to retrieve data from the "users" table
    $sql = "SELECT id, username, email FROM users";

    $queryResult = pg_query($connection, $sql);

    if ($queryResult === false) {
        die("Error: " . pg_last_error());
    }

    while ($row = pg_fetch_assoc($queryResult)) {
        echo "ID: " . $row['id'] . ", Username: " . $row['username'] . ",
Email: " . $row['email'] . "<br>";
    }
?>
```

Mettre à jour les données

Voici un exemple de mise à jour de l'adresse e-mail d'un utilisateur :

```
<?php
    // SQL query to update a user's email address
    $sql = "UPDATE users SET email='updated_email@example.com' WHERE
username='john_doe'";

    $updateResult = pg_query($connection, $sql);

    if ($updateResult === false) {
        die("Error: " . pg_last_error());
    }

    echo "Data updated successfully";
?>
```

Supprimer les données

Exemple de suppression d'un utilisateur :

```php
<?php
    // SQL query to delete a user
    $sql = "DELETE FROM users WHERE username='john_doe'";

    $deleteResult = pg_query($connection, $sql);

    if ($deleteResult === false) {
        die("Error: " . pg_last_error());
    }

    echo "User deleted successfully";
?>
```

Gestion des erreurs

Gérez les erreurs avec élégance à l'aide des blocs try-catch :

```php
<?php
    try {
        // Your PostgreSQL operations here
    } catch (Exception $e) {
        echo "Error: " . $e->getMessage();
    }
?>
```

Injection SQL :

L'une des failles les plus critiques. Les attaquants peuvent injecter du code SQL malveillant via les champs du formulaire, compromettant ainsi la base de données.

Demonstration

Utilisation de PDO(PHP Data Objects)

PDO est une autre extension PHP pour interagir avec des bases de données. Il fournit une interface orientée objet pour la connexion et l'interaction avec une base de données et prend en charge plusieurs types de bases de données, tels que MySQL, PostgreSQL et Oracle.

Activer le pilote PDO_PGSQL

La plupart des distributions PHP incluent l'extension PDO_PGSQL par défaut, vous n'avez donc pas besoin d'effectuer de configuration supplémentaire en PHP. Pourtant, si ce n'est pas le cas, vous pouvez activer l'extension en éditant le fichier **php.ini** pour décommenter la ligne suivante:

;extension=php_pdo_pgsql.dll

Il suffit de supprimer le point-virgule (😉 au début de la ligne et redémarrer le serveur Web.

extension=php_pdo_pgsql.dll

```
<?php
    $servername = "localhost";
    $username = "username";
    $password = "password";
    $dbname = "database_name";

    try {
        $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username,
$password);
        // Configurer les erreurs PDO pour être gérées comme des exceptions
        $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
        echo "Connexion établie avec succès";
    } catch(PDOException $e) {
        echo "La connexion à la base de données a échoué: " . $e-
>getMessage();
    }
?>
```

```
<?php
    $dsn = "pgsql:host=your_host;dbname=your_db";
    $username = "your_username";
    $password = "your_password";

    try {
        $pdo = new PDO($dsn, $username, $password);
        echo "Connected to the database";
    } catch (PDOException $e) {
        die("Connection failed: " . $e->getMessage());
    }
?>
```

Opérations CRUD :

SELECT (Read) :

```
<?php
$sql = "SELECT * FROM your_table";
$stmt = $pdo->query($sql);

while ($row = $stmt->fetch(PDO::FETCH_ASSOC)) {
    echo "ID: {$row['id']} - Name: {$row['name']} - Email: {$row['email']}
<br>";
}
?>
```

INSERT (Create) :

```php
<?php
$name = "John Doe";
$email = "john@example.com";

$sql = "INSERT INTO your_table (name, email) VALUES (:name, :email)";
$stmt = $pdo->prepare($sql);

$stmt->bindParam(':name', $name);
$stmt->bindParam(':email', $email);

if ($stmt->execute()) {
    echo "Record inserted successfully";
} else {
    echo "Error inserting record: " . $stmt->errorInfo()[2];
}
?>
```

UPDATE :

```php
<?php
$newName = "John Updated";
$idToUpdate = 1;

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

$stmt->bindParam(':name', $newName);
$stmt->bindParam(':id', $idToUpdate);

if ($stmt->execute()) {
    echo "Record updated successfully";
} else {
    echo "Error updating record: " . $stmt->errorInfo()[2];
}
?>
```

DELETE :

```php
<?php
$idToDelete = 1;

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

```
$stmt->bindParam(':id', $idToDelete);

if ($stmt->execute()) {
    echo "Record deleted successfully";
} else {
    echo "Error deleting record: " . $stmt->errorInfo()[2];
}
?>
```

Fermer la connexion :

N'oubliez pas de fermer la connexion après avoir terminé les opérations.

```
<?php
$pdo = null; // Fermer la connexion
?>
```

Sécurité des requêtes SQL

La sécurisation des requêtes SQL est une étape cruciale pour prévenir les attaques par injection SQL. Voici quelques bonnes pratiques pour sécuriser vos requêtes SQL lors de l'utilisation de PDO avec PHP :

1. **Utilisez des requêtes préparées :**

   Utilisez toujours des requêtes préparées avec des paramètres liés plutôt que d'incorporer directement des valeurs dans vos requêtes SQL. Cela aide à prévenir les attaques par injection SQL.

   phpCopy code

   ```
   $stmt = $pdo->prepare("SELECT * FROM users WHERE username =
   :username");
   $stmt->bindParam(':username', $username);
   $stmt->execute();
   ```

2. **Échappez les valeurs si vous ne pouvez pas utiliser les requêtes préparées :**

   Si, pour une raison quelconque, vous ne pouvez pas utiliser de requêtes préparées, assurez-vous d'échapper correctement les valeurs à l'aide de la fonction `quote`.

   phpCopy code

   ```
   $username = $pdo->quote($unsafe_username);
   $sql = "SELECT * FROM users WHERE username = $username";
   $stmt = $pdo->query($sql);
   ```

3. **Limitez les privilèges de l'utilisateur de la base de données :**

Accordez à votre application uniquement les permissions nécessaires pour effectuer des opérations. Évitez d'utiliser un utilisateur avec des privilèges excessifs.

4. **Évitez de divulguer des informations sensibles dans les erreurs :**

   Configurez PDO pour ne pas afficher les erreurs détaillées dans l'environnement de production. Vous pouvez le faire en définissant l'attribut `PDO::ATTR_ERRMODE` sur `PDO::ERRMODE_SILENT` ou `PDO::ERRMODE_EXCEPTION` en fonction de vos besoins.

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

5. **Validation côté serveur :**

   Avant d'exécuter des requêtes SQL, effectuez une validation côté serveur des données d'entrée pour vous assurer qu'elles sont dans un format attendu.

   ```
   if (filter_var($user_input, FILTER_VALIDATE_INT)) {
       // Utiliser la valeur dans la requête
   } else {
       // Gérer l'erreur ou rejeter la requête
   }
   ```

6. **Utilisez des rôles de base de données :**

   Si possible, utilisez des rôles de base de données pour définir des permissions plus granulaires au niveau de la base de données.

Assurer la sécurité des requêtes SQL est essentiel pour prévenir les vulnérabilités. En mettant en œuvre ces pratiques, vous réduisez considérablement le risque d'attaques par injection SQL.