

Programmation orientée objet

Retour sur PDO
Développement d'un CRUD
en POO avec PDO

PDO

- PDO est une extension au langage PHP pour accéder aux bases de données.
- Avantages :
 - Elle permet entre autres d'accéder à plusieurs types de bases de données (DB2, MySQL, PostgreSQL, ODBC, SQLite et autres) sans changer la programmation.
 - Elle offre de multiples avantages lorsque nous travaillons en programmation orientée objet, comme nous le verrons plus tard

Intro à la gestion des exceptions

- Avant de travailler avec PDO, nous devons parler de la gestion des exceptions, qui est un avantage important de l'utilisation de PDO.
- Lorsqu'un programmeur sait qu'une portion de code pourrait possiblement générer des erreurs (ou exceptions) sous certaines conditions, il doit impérativement se protéger contre cette éventualité.
- Les vieilles méthodes de programmation encourageait la création de fonctions qui retournaient toujours un booléen pour savoir si la fonction avait bel et bien fonctionné ou si elle avait généré une erreur. Ceci donnait lieu à une foule de structures if qui rendait le code lourd à lire et à comprendre.
- Il existe maintenant une structure optimisée pour le traitement de ces erreurs : la structure try catch.

Try et catch

- Tout ce qui pourrait causer une erreur doit se trouver dans un bloc try. Celui-ci représente le code qui sera “essayé”.
- Le traitement de l’erreur se fait dans le bloc catch. Si une erreur survient lors de l’essai, elle sera “attrapée” par le bloc catch, qui s’occupe de gérer cette erreur (par un message à l’usager ou par toute autre méthode). Un bloc catch peut contenir un bloc try (si une erreur survient, essaie ceci) auquel sera associé un autre bloc catch, et ainsi de suite.
- Il est donc possible pour du code de lancer une exception (“throw”) lorsqu’une erreur est détectée, et, à un autre endroit, d’attraper cette exception (“catch”).
- Voir exemple

Les exceptions en PHP

- Les exceptions en PHP possèdent plusieurs attributs et méthodes intéressants.
 - L'attribut message, et la méthode getMessage(), qui fournissent le message d'erreur associé à l'exception générée.
 - Lors du lancement d'une exception, le premier paramètre de l'appel du constructeur d'Exception est le message que vous voulez mettre dans cet attribut.
 - L'attribut ligne et la méthode getLine(), qui fournissent le numéro de la ligne de code ayant soulevé l'exception.
 - L'attribut code et la méthode getCode(), qui fournissent le code de l'erreur.
 - Le second paramètre facultatif du constructeur d'Exception est le code d'erreur que vous voulez mettre dans cet attribut.
 - L'attribut file et la méthode getFile(), qui fournissent le nom du fichier dans lequel l'exception a été générée.
 - L'attribut trace et la méthode getTrace(), qui fournissent la trace d'exécution avant que l'exception ait été soulevée.

Gestion exceptions avec PDO

- En programmation orientée objet, les objets gèrent généralement les erreurs qu'ils peuvent générer en les lançant au code qui appelle leurs méthodes.
- Avec PDO, nous retrouverons plusieurs cas pouvant générer des exceptions. Le premier cas rencontré, par exemple, est celui de l'établissement de la connexion, qui génère des exceptions lorsque celle-ci est ratée.

PDO - Ouverture de connexion

- Comme nous l'avons mentionné, PDO est une solution orientée objet. Il n'est donc pas surprenant de voir que l'ouverture d'une connexion à une base de données avec PDO se fait lors de l'utilisation du constructeur de la classe PDO.
- Ce constructeur prend en paramètres une chaîne de caractères suivant un format spécifique pour définir la connexion, suivi d'un nom d'utilisateur et d'un mot de passe.
 - Le format pour MySQL est le suivant :
mysql:host=adresseHost;dbname=nomBD, où adresseHost est l'adresse du serveur de BD auquel on veut se connecter et nomBD est le nom de la base de données vers laquelle vous voulez effectuer des requêtes.
 - Les autres formats peuvent être trouvés dans la documentation de PDO sur php.net
- Exemple avec mysql :
 - `$dbPDO = new PDO("mysql:host=localhost;dbname=mabase", $username, $password);`
- L'appel au constructeur génère une exception de type PDOException, dont vous devez généralement assurer la gestion. Voir exemple fourni.

Les requêtes avec PDO

- Les requêtes qui ne retournent pas de données provenant de la base s'effectuent avec la méthode `exec()` de l'objet PDO.
 - `$count = $dbPDO->exec("INSERT INTO users(prenom, nom) VALUES ('Guillaume', 'Harvey')");`
 - `exec()` retourne toujours le nombre de rangées affectées par la requête envoyée en paramètres, et `FALSE` dans le cas où une erreur est survenue.
 - Dans le cas d'une insertion, comme dans l'exemple, la requête retournera 1 si l'insertion a été réussie, et `FALSE` sinon, ce qui est très important.
 - On utilise aussi `exec()` avec les requêtes `UPDATE` et `DELETE`.

Requêtes avec PDO (suite)

- Les requêtes qui retournent des données se font avec la méthode `query()`, qui retourne un « result set », ou un ensemble de résultats.
- On peut ensuite boucler dans les rangées retournées, dont on peut accéder chacune des colonnes en utilisant le nom de celles-ci comme indice dans les `[]` de la rangée.
- Ex :

```
$sql = "SELECT * FROM users";  
foreach ($dbPDO->query($sql) as $row)  
{  
    print $row['prenom'] . ' - ' . $row['nom'] . '<br />';  
}
```

Traitement des erreurs avec PDO

- Si l'on veut que les erreurs générées par les requêtes SQL génèrent des exceptions (plutôt que de soulever des warnings, ou de retourner FALSE), il est possible de spécifier à PDO de générer des exceptions.
- On le fait en utilisant cette instruction :
 - \$dbPDO->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION); //génère une exception
 - \$dbPDO->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_WARNING); //génère un warning
 - \$dbPDO->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_SILENT); //reste silencieux

Les prepared statements de PDO

- Un des très grands avantages d'utiliser PDO est celui des Prepared Statements, qui permettent à un programmeur de paramétrer des requêtes SQL.
- On spécifie à PDO l'emplacement des paramètres dans une requête SQL avec l'opérateur : situé avant le nom du paramètre.
 - Il est aussi possible de ne pas nommer les paramètres et de spécifier seulement leurs emplacements avec des ?.
- On appelle ensuite la fonction `prepare()` de l'objet PDO en lui passant en paramètres la requête SQL paramétrée à préparer. Cet appel nous retourne un objet de type `PDOStatement`, que l'on utilisera par la suite.
 - `$stmt = $dbh->prepare("SELECT * FROM users WHERE prenom = :prenomUser");`
- On poursuit en utilisant la fonction `bindParam()` de l'objet `PDOStatement`, qui lie une VARIABLE à un paramètre dans cette requête. Notez ici que l'on ne parle pas d'attribuer une valeur au paramètre, mais bien de le lier à une variable.
 - `$stmt->bindParam(':prenomUser', $prenomVariable, PDO::PARAM_STR, 12);`
 - Où 12 est la longueur du champ prenom...
 - Si l'on utilise plutôt la méthode `bindValue`, ce sera seulement la valeur qui sera attribuée au paramètre.
- On appelle ensuite la méthode `exec()` de notre objet `PDOStatement`. On peut lui passer en paramètres un tableau de valeurs.
- Voir exemples

Pourquoi les prepared statements?

- PHP.net nous dit :
 - Appeler **PDO::prepare()** et [PDOStatement::execute\(\)](#) pour les requêtes qui doivent être exécutées plusieurs fois avec différentes valeurs de paramètres optimisent les performances de votre application en autorisant le pilote à négocier coté client et/ou serveur avec le cache des requêtes et les metainformations, et aident à prévenir les attaques par injection SQL en éliminant le besoin de protéger les paramètres manuellement.
- La seconde partie est cruciale, puisque les attaques par injection SQL sont les attaques les plus faciles à réaliser.

Attaque par injection SQL

- Prenons en exemple un formulaire de login typique, suivi d'une procédure d'authentification par SQL usuelle.
- Il est possible pour un hacker d'envoyer dans le champs texte « username » quelque chose qui contournera votre requête SQL.
- Par exemple :
 - ```
$res = mysql_query("SELECT * FROM users WHERE username = '$_POST['username']' AND password = '$_POST['password']'");
if(mysql_num_rows($res) > 0)
{
 // usager est authentifié..
}
```
  - Qu'est-ce qui se passe lorsque `$_POST['password'] = « youarehacked' OR 1=1 »`.
  - On peut se retrouver avec la requête `SELECT * FROM users WHERE username = '$_POST['username']' AND password = 'youarehacked' OR 1=1'`, ce qui retournerait toutes les entrées de la base...
  - L'utilisateur pourrait aussi ajouter une requête DROP de la même façon, détruisant vos données...
- On empêche ce problème lorsque l'on utilise les prepared statements de PDO, ou encore en utilisant `mysql_real_escape_string()` avec chacun des paramètres à entrer...

# Qu'est-ce qu'un CRUD?

- CRUD est une abréviation signifiant :
  - Create
  - Read
  - Update
  - Delete
- Dans la pratique, un CRUD est un système de manipulation de données que l'on implémente de façon très courante en programmation, généralement répliqué pour chacune des tables que l'on veut manipuler dans une base de données!
- Évidemment, certaines de ces opérations sont extrêmement similaires d'une table à l'autre.
  - La suppression d'une rangée : `DELETE from NomTable WHERE CléPrimaire = valeur.`
  - La lecture d'une rangée : `SELECT * from NomTable WHERE CléPrimaire = valeur.`
  - La lecture de toutes les rangées : `SELECT * from NomTable.`
  - etc.
- D'autres sont très différentes d'une table à l'autre.
  - La mise à jour, l'insertion... où il faut spécifier les noms de colonne!

# Une classe d'accès à la BD générique

- Une des possibilités en programmation orientée objet (un *patron de conception* communément appelé Table Data Gateway... on y reviendra) est de développer une classe pour chaque table (un modèle) permettant donc de regrouper les 4 opérations de base à un seul endroit.
  - Ceci amènera toutefois de la répétition de code.
  - Voir exemple.
- Nous pourrions éviter cette répétition en utilisant l'héritage et les classes abstraites, ce qui nous permettra de spécifier les comportements génériques dans une classe de base et de traiter ceux plus spécifiques dans une classe qui en hérite!
  - Voir exemple.

# Retour sur les requêtes préparées en PDO

- Comme nous l'avons vu, les requêtes préparées avec `prepare()` peuvent recevoir des paramètres (la syntaxe est `:nomParam`).
  - Lors de l'exécution, toutefois, nous n'avons pas à spécifier nécessairement les paramètres un à la fois. Ils peuvent en effet être spécifiés sous forme de tableau.
  - Voir exemple.
- De cette façon, nous pouvons construire une classe de base qui contiendra une méthode générique qui permet d'exécuter n'importe quelle requête avec n'importe quels paramètres, en autant qu'on lui envoie un tableau dument rempli. En plus, il est possible de transformer un objet en tableau très facilement en PHP.
  - Voir exemple.



# PDO et les objets : fetchObject, PDO::FETCH\_CLASS

- En plus de tout ce que nous avons vu, il est aussi possible de remplir directement un objet avec une rangée d'une table dans une base de données, si la classe de cet objet a les mêmes attributs que les noms des colonnes dans la table.
  - Voir exemple.
- Il est même possible de faire cette opération pour plusieurs rangées en même temps avec `fetchAll()`.
- Par contre, il est préférable dans ce cas d'éviter d'avoir un constructeur dans la classe qui reçoit les informations. De toute façon, PDO peut accéder directement aux attributs de votre classe, même s'ils sont privés.
- Si une validation est vraiment nécessaire (les données devraient être bonne dans la table, mais ça peut arriver...), alors il est possible d'appeler un constructeur lors du remplissage, mais c'est relativement complexe et cela dépasse le cadre du cours.