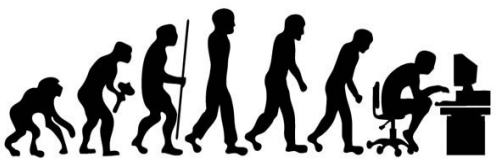
ICT- 151

Intégrer des bases de données dans des applications Web

Introduction à la gestion des bases de données dans le contexte de PHP et de MySQL



Auteur :	Laurent Moine
Date :	27.11.17
Nom du document :	ict151b_v2.0.odt
N° de révision:	2

Table des matières

1 Objectifs opérationnels du module ict151

Selon le plan modulaire de ICT Switzerland[1] , les objectifs à atteindre pour le module ict151 dans sa version 3.0 sont les suivants:

1) Analyser les exigences d'une application Web et de la base de données, respectivement des éléments de données à lier, définir et documenter la technique de liaison.

1.1 Connaître des possibilités pour l'analyse de la structure d'une application Web et l'interactivité des composants logiciels.

1.2 Connaître les caractéristiques et les fonctions d'interfaces vers des systèmes de bases de données et éléments de données.

2) Identifier les informations importantes de protection et de sécurité en tenant compte de la protection des données, et définir les mesures.

2.1 Connaître des critères pour déterminer les informations à protéger, tels que les données personnelles particulièrement sensibles comme aussi des données commerciales.

2.2 Connaître des mesures de sécurité organisationnelles et techniques pour les informations, les données et les applications Web.

3) Réaliser l'intégration de l'application Web avec la base de données, respectivement aux éléments de données, en prêtant attention aux transactions, à la protection et la sécurité des données.

3.1 Connaître des langages de programmation et de scripts pour des applications Web, ainsi que leurs caractéristiques.

3.2 Connaître des techniques courantes pour la mise en œuvre de transactions de bases de données.

3.3 Connaître le déroulement pour garantir l'intégrité des données, la disponibilité, l'authenticité et la confidentialité dans les applications Web.

3.4 Connaître une technique pour verrouiller des éléments de données dans une application.

3.5 Connaître diverses architectures pour l'intégration d'éléments de données, respectivement de bases de données dans une application Web.

4) Mettre en œuvre les souhaits de modifications conformément au déroulement prescrit des modifications.

4.1 Connaître la structure et les caractéristiques d'un concept de modifications et leur importance pour la mise en œuvre intégrale du mandat.

5) Définir et exécuter la procédure de test et de remise, la documenter dans un procès-verbal de tests. Si nécessaire, entreprendre les corrections.

1 https://cf.ict-berufsbildung.ch/modules.php?name=Mbk&a=20101&cmodnr=151&noheader=1

5.1 Connaître des procédures de contrôles et de remise, appropriées pour un examen étendu d'une application Web avec base de données

5.2 Connaître le déroulement pour déterminer les critères de tests et de remise pour une requête de modifications.

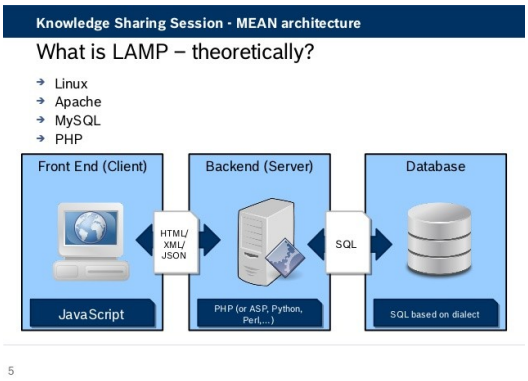
2 Introduction

Dans le contexte du web, la logique applicative située sur le serveur prend en charge l'accès aux bases de données. Elle doit pouvoir ouvrir une session sur un serveur de bases de données. Elle doit aussi être en mesure d'exécuter des requêtes de manipulation des données.

Si des scripts écrits en PHP pilotent la logique applicative, le serveur de bases de données associées sera probablement MySQL, bien que ce ne soit pas une obligation.

2.1 Architecture du système

L'architecture usuelle dans un cas général est la suivante



Le front-end qui s'exécute sur le client (souvent un navigateur) communique via le protocole HTTP, avec les scripts (souvent php, asp, python, ...) implémentés sur le serveur web (souvent Apache, IIS ou ngnix). Ces mêmes scripts accèdent au serveur de base de données (MySql, Oracle, ...) qui est souvent localisé sur le même serveur.

Le but de ce cours est de décortiquer comment fonctionne les interactions entre les scripts côté serveur et la base de données.

2.2 Moteurs de tables de MySQL

MySQL supporte plusieurs moteurs de tables. Ils définissent le stockage de l'information et permettent de gérer différemment les tables selon l'usage que l'on en a. Les principaux sont InnoDB et MyISAM. D'autres existent pour des applications particulières[2].

2 [MySQL]

2.2.1 InnoDB

C'est le moteur de tables par défaut depuis la version 5.5 de MySQL. Il gère les clés étrangères, l'intégrité référentielle et les transactions. En outre, il implémente un système de récupération automatique en cas de défection du serveur. C'est le moteur à privilégier.

2.2.2 MyISAM

C'est le moteur par défaut pour les versions antérieures du serveur. Il ne gère pas les clés étrangères, ni l'intégrité référentielle, ni les transactions. Il est efficace pour effectuer des requêtes SELECT ou les requêtes INSERT.

2.3 Moyen d'accès aux BD proposés par PHP

Les scripts écrits en PHP peuvent communiquer avec une palette large de serveurs de bases de données. Des extensions natives pour MySQL, Oracle, et bien d'autres encore existent. Bien que ces extensions aient de similitudes entre elles, il faudra alors manipuler des fonctions spécifiques différentes selon le SGBD. PHP5 offre une autre solution avec PDO. C'est une couche d'abstraction unifiée, souple et puissante qui peut se connecter à la plupart des serveurs de bases de données actuels.

Le PHP offre trois API différentes pour se connecter à MySQL. La plus ancienne, *mysql_* ne doit plus être utilisée pour de nouveau projet. Elle est vouée à disparaître bientôt. La seconde, *mysqli_* est toujours bien supportée. Elle supporte une approche objet ou une approche procédurale. Dans ce dernier cas, la syntaxe de *mysqli_* ressemble passablement à celle de l'ancienne API *mysql_*. L'utilisation de PDO_MySQL est la dernière alternative.

2.3.1 Approche procédurale

Cette approche utilise l'API *sqli_*[3] en mode procédurale.

```php
<?php
/**
 * Created by PhpStorm.
 * User: Laurent Moine
 * Date: 14.11.15
 * Time: 14:08
 */?>

<?php
$hote='localhost';
$user='root';
$pass='root';
$base='ict151';

//étape 1: connexion
$link= mysqli_connect($hote, $user, $pass);
if(mysqli_connect_error($li)){
    die('ERREUR de connexion: '. mysqli_connect_error());
}
//choix de la base de données
if(!mysqli_select_db($link, $base)){
```

3 [PHP]

```
    die('ERREUR bases: '. mysqli_connect_error());
}

//étape 2: création et exécution de la requête MySQL
$pseudo='Monnot';

//échappement da la variable
$pseudo=mysqli_real_escape_string($link,$pseudo);
$sql="SELECT * FROM tbl_membres WHERE uname='$pseudo'";

$result = mysqli_query($link, $sql);
if(!$result){
    die('ERREUR sql: '. mysqli_sqlstate($link));
}

?>
<!DOCTYPE html >
<html lang="fr">
<head>
    <meta charset="utf-8"/>
    <meta name="DCTERMS.creator" content="Laurent Moine"/>
    <title>demo select sqli</title>
</head>
<body>
<h3>lecture dans une base de données avec l'API sqli_</h3>
<?php
    // étape 3: traitement du résultat
    while($row=mysqli_fetch_assoc($result)){
        echo $row['id'], '</br>';
        echo $row['uname'], '</br>';
        echo $row['upwd'], '</br>';
        echo $row['niveau'], '</br>';
    }
?>
</body>
</html>
<?php
    //étape 4: libération des ressources
    mysqli_free_result($result);
    mysqli_close($link);
?>
```

2.3.2 Approche objet

L'API *mysqli_* et l'abstraction PDO[4] supportent l'approche objet. La suite de ce document se concentre sur PDO. À la différence d'une interface native, PDO est un socle commun pour les connecteurs vers les SGBD. Il s'occupe d'offrir des fonctions de base et d'unifier les interfaces utilisateur.

PDO apporte plusieurs avantages sur les API natives propres à chaque SGDB :

- une conception plus moderne parfois simplifiée,
- des requêtes moins longues et un code plus propre (lisibilité et maintenance à long terme),
- une API unique pour de multiples SGBD.

PDO prend notamment en charge les SGBD suivants[5] :

4 Documentation complète sur : https://secure.php.net/manual/fr/intro.pdo.php
5 [JULP] et https://secure.php.net/manual/fr/pdo.drivers.php

- SQLite, PostgreSQL, ODBC, Oracle, Mysql, Informix, , MS SQL Server ...

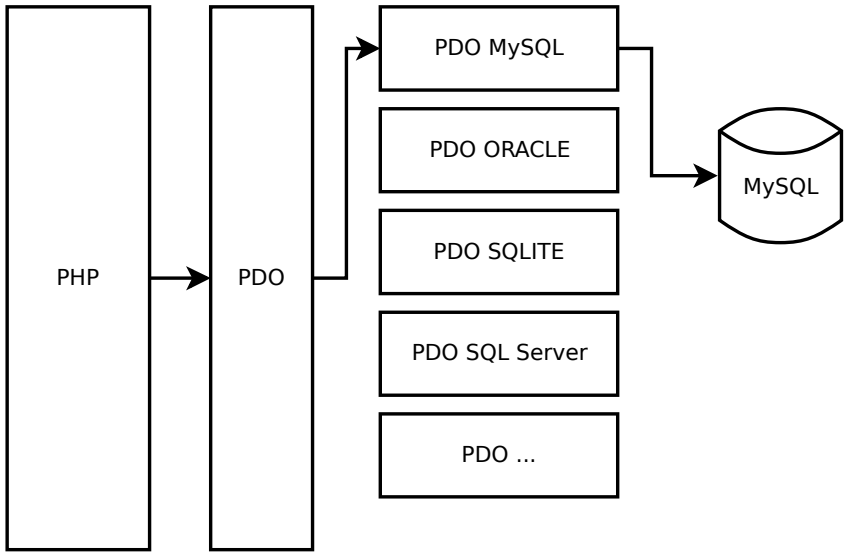


Illustration 1 : PDO, socle unifié d'accès aux SGBD

PDO permet d'exécuter tous les types de requêtes classiques (INSERT, UPDATE, DELETE, SELECT) ainsi que l'exécution de procédures stockées si le SGBD le permet. Les transactions et le mode d'autoévaluation (autocommit) sont bien évidemment possibles. PDO permet d'employer les requêtes paramétrées. Il émule les fonctionnalités que certains SGBD ne supportent pas.

PDO est hautement configurable. Il peut travailler dans différents modes. La gestion des erreurs peut par exemple être traitée par le biais des exceptions[6].

Les pilotes d'accès au serveurs de bases de données s'installent à la demande. La fonction phpinfo() permet de lister la configuration de PHP sur le serveur :

PDO

PDO support	enabled
PDO drivers	mysql, pgsql

Illustration 2: Pilotes PDO installés

L'installation sur un serveur Linux nécessite l'ajout d'un paquet et le redémarrage du serveur :

```
sudo apt-get install php5-mysql
sudo service apache2 restart
```

6 Détail et exemple : https://secure.php.net/manual/fr/pdo.error-handling.php

3 Introduction aux exceptions en PHP

Selon la documentation de référence[7] :

"*Le langage dispose d'une gestion des exceptions similaire à ce qu'offrent les autres langages de programmation. Une exception peut être lancée (`throw`) et attrapée (`catch`) par le code PHP. Le code devra être entouré d'un bloc `try` pour faciliter la saisie d'une exception potentielle. Chaque `try` doit avoir au moins un bloc `catch` ou `finally` correspondant.*

L'objet lancé doit être une instance de la classe Exception ou en hériter. Tenter de lancer un objet qui ne correspond pas à cela résultera en une erreur fatale émise par PHP. "

La classe Exception offre quatre méthodes qui permettent d'accéder aux informations d'erreur en lecture[8] : `getMessage()`, `getCode()`, `getFile()` et `getLine()`.

Try

Les exceptions levées à l'intérieur du bloc qui suit l'instruction *try* sont captées et envoyées au bloc catch le plus adapté.

Catch

Plusieurs blocs catch peuvent être utilisés pour attraper différentes classes d'exceptions. Lorsque qu'aucune exception n'est levée l'exécution normale continue après le dernier bloc catch défini dans la séquence.

Lorsqu'une exception est lancée, le code qui suit le traitement ne sera pas exécuté et PHP tentera d'accéder au premier bloc catch correspondant. Si une exception n'est pas attrapée, une erreur fatale issue de PHP sera envoyée avec un message"Uncaught Exception..."

3.1 Exemple

L'API *mysqli_* peut aussi travailler selon l'approche objet. L'exemple ci-dessous présente les instructions de connexion à une base de données avec *sqli_* en mode objet. Cette démonstration est issue du manuel du PHP.

La fonction *connect* tente de se connecter à la base de données. En cas d'erreur la fonction lève une exception de type `mysqli_sql_exception`.

Cette exception est captée par le bloc try-catch qui entoure l'appel de la fonction `connect(...)`. Une erreur est levée et affichée au cas où la connexion ne se déroule pas correctement.

7 http://php.net/manual/fr/language.exceptions.php
8 http://php.net/manual/fr/language.exceptions.extending.php

```php
<?php
$hote='localhost';
$user='root';
$pass='root';
$base='ict151_';
define("MYSQL_CONN_ERROR", "Unable to connect to database.");
// Ensure reporting is setup correctly
mysqli_report(MYSQLI_REPORT_STRICT);

// fonction de connexion à la base de données
function connect($p_user,$p_pass,$p_base,$p_hote) {
    try {
        $mysqli = new mysqli($p_hote,$p_user,$p_pass,$p_base);
        $connected = true;
    }
    catch (mysqli_sql_exception $e) {
        throw $e;
    }
}

//Les erreurs levées sont attrapées par le bloc try
try {
    connect($user,$pass,$base,$hote);
    echo 'Connected to database';
}
catch (Exception $e) {
    echo "message: ";
    $em=$e->getMessage();
    echo $em;
}
?>
```

4 Connexion à la base de données

Pour créer la connexion avec le SGBD, on instancie un nouvel objet PDO. On fait alors appel à son constructeur[9]. Le constructeur de la classe PDO veut recevoir la chaîne de connexion (dsn) qui définit le nom du pilote à utiliser, l'adresse du serveur et le nom de la base de données cible. Ce constructeur accepte des informations supplémentaires comme les identifiants de l'utilisateur et la base de données cible.

Exemple de connexion à MySQL avec PDO

```php
<?php
   $user='root';
   $pass='root';
   $base='ict151';
   $dsn='mysql:host=localhost;dbname='.$base.';charset=UTF8';
   try
   {
      /*** renvoie une instance (un objet) de la classe PDO ***/
       $dbh = new PDO($dsn, $user, $pass);
      /*** les erreurs sont gérées par des exceptions ***/
      $dbh->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
   }
   catch (PDOException $e)
   {
         echo "erreur ! :". $e->getMessage()."<br/>";
         die();
   }
```

9 http://php.net/manual/fr/pdo.construct.php

```
?>
```

Depuis la version 5.3.6 de PHP, la chaîne DSN accepte un paramètre pour déterminer le jeu de caractères du client.

Le lien vers la connexion est stocké dans la variable $dbh. Cette variable sera utilisée dans toutes les communications vers le serveur de base de données.

Dans la pratique, les instructions de connexions à la base de données sont stockées dans un fichier externe nommé `conn.inc.php`. Ce fichier est inclus dans le script d'accès à la base de données.

Exemple

```
<?php
   require_once("./conn.inc.php"); //initialise $dbh

   //requête de sélection ou autre
   ….
   ….
   //fermeture de la connexion si elle existe
   if($dbh)
      $dbh=null;

?>
```

5 Exécution des requêtes SQL

L'utilisation de la base de données avec PHP s'effectue en 5 étapes comme le montre l'illustration 4

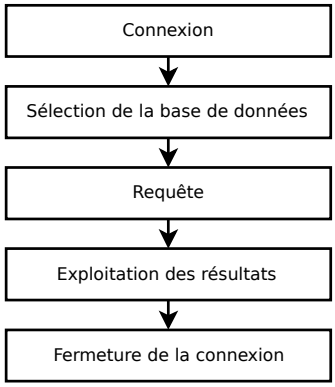


Illustration 3: Cycle de vie d'un accès au SGBD

Une fois la connexion ouverte, le script dispose de la base de données pour lire, modifier ou encore supprimer des données. La communication avec le serveur se fait directement par l'envoi de requêtes SQL. Les API actuelles de PHP peuvent envoyer des requêtes non préparées (statique ou codée en dur) ou des requêtes préparées (dont les données sont passées par des paramètres). Cette deuxième approche qui offre plus de sécurité grâce à une protection accrue aux injections SQL sera traitée au chapitre 5.2.

5.1 Requêtes non préparées

La classe PDO[10] dispose des méthodes `exec()` et `query()` pour envoyer une requête au serveur. Une requête exécutée avec ces méthodes envoie les ordres SQL au serveur. Il faut ensuite traiter le résultat de l'exécution de ces ordres. Deux cas de figure se distinguent :

- Traitement après une requête d'insertion ou de modification.
- Traitement après une requête de sélection qui renvoie un jeu de résultats.

Le tableau ci-dessous récapitule la méthode appropriée pour les principales instructions SQL

Requête SQL	Méthode PDO à utiliser
INSERT	exec()
UPDATE	exec()
DELETE	exec()
SELECT	query()

5.1.1 Requêtes d'insertion

La méthode `exec()` est spécialement conçue pour le cas de figure des requêtes INSERT, UPDATE et DELETE.

```
int $objetPDO->exec(string $sql);
```

Cette requête retourne le nombre de lignes retournées ou la valeur FALSE en cas d'erreur.

```php
<?php
header('Content-Type: text/html; charset=utf-8');
//initialise $dbh
require_once("../conn.inc.php");
//requête d'insertion
$sql = "INSERT INTO `ict151`.`tbl_membres`
          (`uname`,`upwd`, `niveau`)
         VALUES
          ('Hauser', 'pass', '2');";
try {
    $insertion = $dbh->exec($sql);
    if ($insertion){
        echo $insertion, "donnée(s) insérée(s)",'<br>';
        echo "Dernier identifiants:",$dbh->lastInsertId();
    }
}
catch (PDOException $e) {
    die(sprintf("%s dans %s à la ligne %d : %s", get_class($e), $e->getFile(), $e-
>getLine(), $e->getMessage()));
}
//fermeture de la connexion si elle existe
if ($dbh)
    $dbh = null;
```

10 http://php.net/manual/fr/class.pdo.php

Connaître la valeur de l'identifiant de la dernière valeur insérée

Lors de l'insertion de données dans une table qui utilise un champ auto-incrémenté ou une séquence, il est utile de pouvoir connaître la valeur de l'identifiant de la dernière ligne insérée. La méthode `lastInsertId()` le permet :

```
string $objetPDO->lastInsertId();
```

```
    $insertion = $dbh->exec($sql);
    echo "Dernier identifiant:",$dbh->lastInsertId();
```

5.1.2 Requêtes de modification

Les requêtes de modification s'appuient aussi sur la méthode `exec()` :

```
string $objetPDO->exec();
```

```
<?php
//definir l'encodage
header('Content-Type: text/html; charset=utf-8');
//initialise $dbh
require_once("../conn.inc.php");
//requête de modification
$sql = "UPDATE `ict151`.`tbl_membres` SET `niveau`= 1;";
try {
    $res = $dbh->exec($sql);
    if ($res)
        echo $res,' donnée(s) modifiée(s)';
    else
        echo 'Pas de changements';
}
catch (PDOException $e) {
    die(sprintf("%s dans %s à la ligne %d : %s", get_class($e), $e->getFile(),
                $e->getLine(), $e->getMessage()
            )
        );
}
//fermeture de la connexion si elle existe
if ($dbh)
    $dbh = null;
```

5.1.3 Requêtes de suppression

Les requêtes de suppression s'appuient aussi sur la méthode `exec()` :

```
<?php
//definir l'encodage
header('Content-Type: text/html; charset=utf-8');
//initialise $dbh
require_once("../conn.inc.php");
//requete de modification
$sql = "DELETE FROM `ict151`.`tbl_membres` WHERE `id`= 24;";
try {
    $res = $dbh->exec($sql);
    if ($res)
```

```
        echo $res,' donnée(s) effacée(s)';
    else
        echo 'Pas de changements';
}
catch (PDOException $e) {
    die(sprintf("%s dans %s à la ligne %d :</br> %s", get_class($e), $e->getFile(), $e-
>getLine(), $e->getMessage()));
}
//fermeture de la connexion si elle existe
if ($dbh)
    $dbh = null;
```

5.1.4 Requêtes de sélection

Les requêtes de sélection renvoient un jeu de valeur. Il ne faut utiliser la méthode `query()`

```
PDOStatement $objetPDO->query(string $sql);
```

Après l'exécution de la requête de sélection, les données ne sont pas directement affichées, elles sont simplement mises en mémoire. Il faut aller les chercher et les afficher. La méthode `query()` renvoie une instance de `PDOStatement`[11]. Cette classe dispose de deux méthodes qui permettent de manipuler les données renvoyées :

1. La méthode `fetchAll()` qui renvoie l'ensemble des données sous forme d'un tableau PHP. Elle peut recevoir des paramètres facultatifs

```
array $objetPDO->fetchAll([…]);
```

Cette manière de faire à l'avantage de permettre d'accéder facilement à toutes les données et d'exécuter des requêtes tierces en cours de traitement. Elle n'est à utiliser que sur des jeux de résultats restreints, puisqu'elle charge la totalité des données en mémoire.

```
<?php
//definir l'encodage
header('Content-Type: text/html; charset=utf-8');
//initialise $dbh
require_once("../conn.inc.php");

//requête de modification
$sql = "SELECT * FROM `ict151`.`tbl_membres;";
try {
   //exécution de la requête
   $res = $dbh->query($sql);
   //récuperation du nombre de lignes affectées
   if ($res->rowCount())
      echo $res->rowCount(),' lignes(s) lue(s)';
   else
      echo 'Pas de valeurs';
   //récupère toute les données dans un tableau PHP
   $tabLignes=$res->fetchAll();
   //affichage des enregistrements
   foreach ($tabLignes as $row) {
      var_dump($row);
   }
}
catch (PDOException $e) {
   die( $e->getMessage());
```

11 http://php.net/manual/fr/class.pdostatement.php

```
}
//fermeture de la connexion si elle existe
if ($dbh)
    $dbh = null;
```

2. La méthode `fetch()` permet une lecture séquentielle du résultat. Elle renvoie `FAUX` lorsqu'il n'y plus de résultat disponible. L'exécution de requêtes tierces sur la même connexion n'est pas possible avant la fin du parcours de toutes les lignes. Cette méthode est très utile pour le traitement de gros résultat. Le paramètre `fetch_style` permet de définir sous quelle forme les résultats sont récupérés. Le chapitre 5.1.5 explique plus en détail l'utilisation de ce paramètre.

```
mixed $objetPDO->fetch( int$fetch_style);
```

```
//exécution de la requête
$res = $dbh->query($sql);
//Récupération séquentielle des résultats
while($row=$res->fetch()){
    var_dump($row);
}
```

PDOStatement implémente l'interface `Traversable`[12] qui rend le jeu d'enregistrements itérable avec une boucle `foreach`. Cette solution est élégante lorsqu'il s'agit d'afficher séquentiellement tous les résultats d'un jeu d'enregistrement.

```
//requête de sélection
$sql = "SELECT * FROM `ict151`.`tbl_membres;";
try {
    //exécution de la requête
    $res = $dbh->query($sql);
    //lecture séquentielle et affichage des enregistrements
    foreach ($res as $row) {
        var_dump($row);
    }
}
catch (PDOException $e) {
    die( $e->getMessage());
}
//fermeture de la connexion si elle existe
if ($dbh)
    $dbh = null;
```

5.1.5 Lecture des résultats

Le paramètre `fetch_style`[13] des méthodes `fetch()` et `fetchAll()` détermine la façon dont PDO retourne les résultats. Il permet de définir de quel type sera le retour : tableau associatif, tableau indicé ou objet. Le tableau ci-dessous récapitule les principaux modes applicables. C'est la valeur PDO::FETCH_BOTH qui est utilisée par défaut.

12 http://php.net/manual/fr/class.traversable.php
13 http://php.net/manual/fr/pdostatement.fetch.php

Valeur de la constante	Action
PDO::FETCH_ASSOC	Renvoie un tableau associatif où les noms de colonne correspondent au nom de colonne de la requête
PDO::FETCH_NUM	Renvoie un tableau. indicé où l'indice correspond à la position de la colonne dans la requête. La première colonne prend l'indice [0], la suivante prend l'indice [1]...
PDO::FETCH_BOTH	C'est le mode par défaut. Les enregistrements sont retournés dans un tableau indexé par les noms de colonnes et aussi indexé par les numéros de colonnes.
PDO::FETCH_OBJ	Renvoie un objet anonyme avec les noms d'attributs qui correspondent aux noms des colonnes retournés dans le jeu de résultat.

Le paramètre `fetch_style` est envoyé lors de l'utilisation des méthodes `fetch()` et `fetchAll()`. Lors de la récupération des résultats grâce aux itérateurs et aussi d'une manière générale, il est aussi possible de définir le mode de retour avec la méthode `setFetchMode`

```
boolean $objetPDOStatement->setFetchMode(int $mode);
```

```php
$res = $dbh->query($sql);
//changement de mode de fetch
$res->setFetchMode(PDO::FETCH_ASSOC);
foreach ($res as $row) {
   print_r($row);
}
```

```
PDO::FETCH_ASSOC

Array
(
    [id] => 5
    [uname] => Monnot
    [upwd] => pass
    [niveau] => 1
)

PDO::FETCH_NUM

Array
(
    [0] => 5
    [1] => Monnot
    [2] => pass
    [3] => 1
)
```

```
PDO::FETCH_BOTH

Array
(
    [id] => 5
    [0] => 5
    [uname] => Monnot
    [1] => Monnot
    [upwd] => pass
    [2] => pass
    [niveau] => 1
    [3] => 1
)

PDO::FETCH_OBJ

stdClass Object
(
    [id] => 5
    [uname] => Monnot
    [upwd] => pass
    [niveau] => 1
)
```

illustration 4: différents styles de présentation des résultats

5.1.6 Récupération d'un résultat unique

Certaines requêtes ne renvoient obligatoirement qu'une ligne. C'est le cas pour les requêtes qui utilise des fonctions d'agrégations comme `COUNT()`, `MIN()`, `SUM`, …

La récupération du résultat de telles requêtes peut se faire sans parcourir le jeu d'enregistrement avec une boucle. Il suffit de vérifier qu'il n'y ait bien qu'un seul résultat retourné, puis de le récupérer dans un tableau ou un objet.

```php
//requête de séelction agrégée
$sql = "SELECT COUNT(*) as nb FROM `ict151`.`tbl_membres;";
try {
   //exécution de la requête
   $res = $dbh->query($sql);

   //récuperation du nombre de lignes afectées
   $nbLigne=$res->rowCount();
   if ($nbLigne===1){
      $tabLignes=$res->fetchAll(PDO::FETCH_ASSOC);
      echo 'nombre d\'enregistrement : ',$tabLignes[0]['nb'];
   }
   else
      echo 'Plus d\'une ligne';
}
catch (PDOException $e) {
   die( $e->getMessage());
}
```

5.2 Protections des données

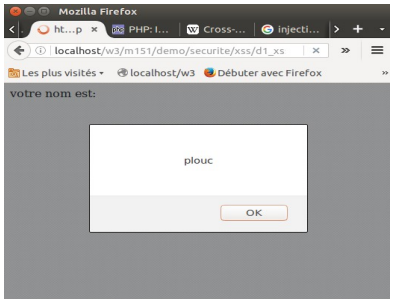
Chaque fois que l'internaute est susceptible d'envoyer des données au serveur via la barre d'adresse (url) ou via un formulaire, il faut se souvenir que ces données peuvent être corrompues et affecter le bon fonctionnement de l'application web. Ceci est encore plus vrai lorsque cette application est connectée à une base de données.

Les données corrompues peuvent être injectés dans l'application de différentes manières, notamment par des injections de scripts Javascript dans l'interface HTML ou par l'injection de commande SQL parasites aux scripts de traitement des formulaires.

5.2.1 Injection XSS permanente

Les injections XSS[14] consistent à insérer une ligne de script JavaScript dans un champ de formulaire ou dans L'URL de la page. Lorsque le script PHP retourne ce contenu dans le flux HTML, le navigateur l'interprète comme du code exécutable du côté client.

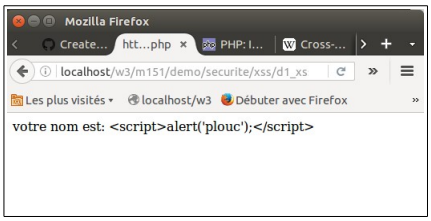
```php
<?php
//$nom="Perret";
$nom="<script>alert('plouc');</script>"
?>
<p>votre nom est: <?php echo $nom;?></p>
```



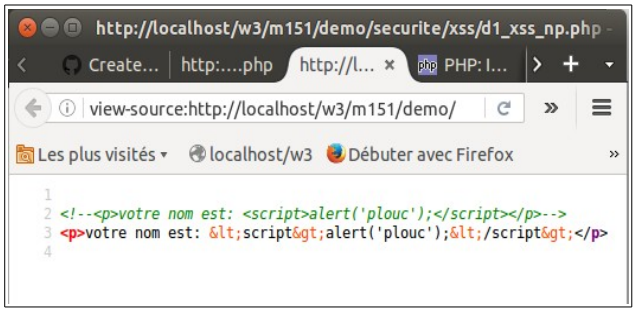
14 https://fr.wikipedia.org/wiki/Cross-site_scripting

La solution pour se protéger consiste à transformer les caractères spéciaux de la chaîne à afficher en entités HTML :

```php
<?php
//$nom="Perret";
$nom="<script>alert('plouc');</script>"
?>
<p>votre nom est: <?php echo htmlentities($nom);?></p>
```



La source de la page permet de visualiser les entités en question.



Lorsque les données potentiellement infectées du formulaires sont stockées dans la base de données, l'injection devient permanente et peut mettre l'entier de l'application web à genoux.

Pour s'en protéger, les données issues du formulaire sont stockées telles quelles dans la base de données. Au moment de les injecter dans le flux HTML, il faut impérativement appliquer la séquence d'échappement des données en les transformant en entités à l'aide des fonctions `htmlentities()`[15], `htmlspecialchars()`[16] ou encore à l'aide des fonction de filtres[17] tels que `filter_var()` ou `filter_input()` associées au filtres de nettoyages[18] `FILTER_SANITIZE_SPECIAL_CHARS` ou `FILTER_SANITIZE_STRING`

L'illustration ci-dessous montre un exemple de code JavaScript injecté dans la base de donnée, qui risque de rediriger l'internaute vers un site indésiré.

7	7	Jonas
8	23	<script>alert("hello");</script>
9	25	<script>window.location.assign("http://esig.moinau.ch");</script>

illustration 5: code JS injecté dans la base de donnée

15 http://php.net/manual/fr/function.htmlentities.php
16 http://php.net/manual/fr/function.htmlspecialchars.php
17 http://php.net/manual/fr/ref.filter.php
18 http://php.net/manual/fr/filter.filters.sanitize.php

5.2.2 Injection SQL

Ce type d'injection consiste à tenter de modifier les requêtes SQL que les scripts PHP envoient au serveur de base de données dans l'optique d'obtenir en résultat un jeu de données non prévus par le concepteur de l'application. Cette technique peut aussi être utiliser pour tenter de modifier les droits d'accès à l'application.

Les deux exemples ci-dessous[19] se base sur la modification de la requête suivante :

```
// Requête SQL
$login = 'Jojo'; //simmule une entrées de formulaire
$password = 'pass'; //simmule une entrées de formulaire
$sql = "SELECT * FROM tbl_secu WHERE uname='$login' AND upwd='$password'";
$res = $bdd->query($sql);
```

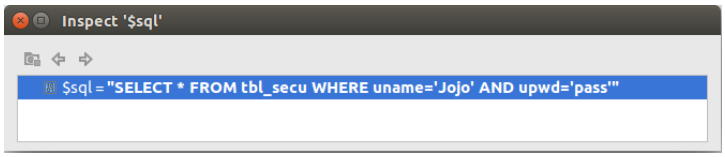


illustration 6: requête avant l'injection

Exemple 1 : Tronquer une requête SQL

La variable `$login` est modifié dans le but de tronquer le reste de la requête :

```
// Requête SQL
$login = 'Jojo\' #'; //injection Le ' ferme le SQL,le # met ce qui suit en commentaire
$password = 'pass'; //simmule une entrées de formulaire
$sql = "SELECT * FROM tbl_secu WHERE uname='$login' AND upwd='$password'";
$res = $bdd->query($sql);
```

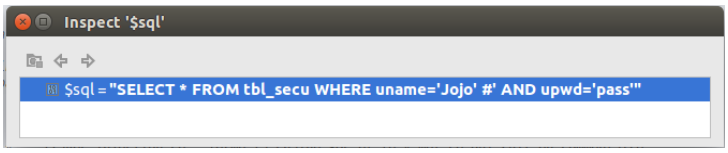


illustration 7: requête avec l'injection

L'exécution de cette requête renvoie les utilisateurs dont le nom est *Jonas* quelque soit le mot de passe. Il suffit à celui qui manipule ainsi la requête, de posséder un nom d'utilisateur sans en connaître le mot de passe associé. Le problème vient du fait qu'il est possible d'injecter une partie structurelle de la requête (#) en passant par les données de la requête.

Exemple 2 : Forcer la condition de sortie de l'opérateur AND de la requête SQL

La variable `$password` est modifié afin de ne pas tenir compte de la valeur proposée comme mot de passe :

```
// Requête SQL
$login = 'Jojo';
$password = "pass' OR '1'='1 "; //injecte une condition OU qui renvoie toujours vrai.
$sql = "SELECT * FROM tbl_secu WHERE uname='$login' AND upwd='$password'";
$res = $bdd->query($sql);
```

19 https://openclassrooms.com/courses/protegez-vous-efficacement-contre-les-failles-web/l-injection-sql-1

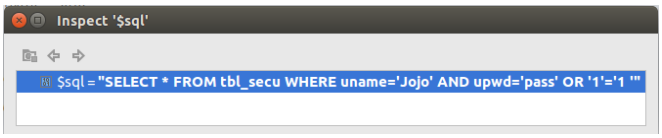


illustration 8: requête avec l'injection

L'exécution de cette requête renvoie les tous utilisateurs. La condition `where` de la requête est complètement court-circuitée !

Pour pallier à tous ces dangers d'injections, il suffira de séparer les données de la requête de sa structure, tout en prêtant toujours attention à l'échappement de caractères spéciaux. PDO offre un mécanisme de requêtes préparés qui facilitent grandement ces mises en œuvre.

5.3 Requêtes préparées

Le principe des requêtes préparées[20] consiste à séparer le code SQL de la requête de ses données. Sont considérées comme données, toutes les valeurs prises par les différents champs lors de l'écriture de requêtes UPDATE ou INSERT, ainsi que toutes les valeurs injectées dans les conditions de requêtes SELECT, UPDATE ou DELETE au niveau des clauses WHERE.

Le but des requêtes préparées est double :

- Gain de performance et réduction des échanges entre le client (PHP dans ce contexte) et le serveur de bases de données. En effet le code de la requête n'est analysé qu'une seule fois sur le serveur. Cet avantage n'est valable que si la requête est exécutée plusieurs fois.
- Gain de sécurité grâce à l'isolation de la requête SQL et des données qui la compose, il n'y a plus de sensibilité aux injections. C'est le serveur qui se charge des échappements. "*Si votre application utilise exclusivement les requêtes préparées, vous pouvez être sûr qu'aucune injection SQL n'est possible*"

D'un point de vue pratique, la requête s'exécute comme illustration 9 :

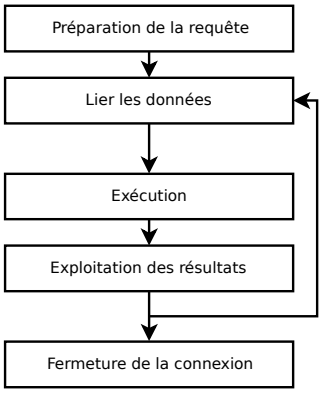


illustration 9: Principe des requêtes préparées

20 https://secure.php.net/manual/fr/pdo.prepared-statements.php

5.3.1 Construction de la requête

Pour construire une requête préparée, il suffit de remplacer chaque paramètre par un point d'interrogation ou par un paramètre nommé. Les valeurs à substituer aux paramètres seront fournies lors de l'exécution de la requête.

```
//requete normale, non préparée
$sql1 = "INSERT INTO `ict151`.`tbl_membres` (`uname`,`upwd`, `niveau`)
         VALUES ('Hauser', 'pass', '2');";

//modèle de requête préparée avec paramètres nommés
$sql2 = "INSERT INTO `ict151`.`tbl_membres` (`uname`,`upwd`, `niveau`)
         VALUES (:nom, :mdp, :niveau);";

//modèle de requête préparée avec points d'interrogation
$sql3 = "INSERT INTO `ict151`.`tbl_membres` (`uname`,`upwd`, `niveau`)
         VALUES (?,?,?);";
```

Il n'est pas possible d'utiliser à la fois des noms de paramètre et des points d'interrogation.

5.3.2 Préparation de la requête

Une fois le modèle de requête construit, il faut l'envoyer au serveur[21] pour qu'il l'analyse et la stocke. Le serveur renvoie un objet PDOStatement qui permettra d'exploiter la requête. La valeur FAUX est renvoyée en cas d'erreur lors de la préparation.

```
PDOStatement $objetPDO->prepare(string $sql);
```

```
$sql2 = "INSERT INTO `ict151`.`tbl_membres` (`uname`,`upwd`, `niveau`)
         VALUES (:nom, :mdp, :niveau);";
$stmt=$dbh->prepare($sql2);
```

5.3.3 Liaison des données et exécution de la requête

Dès que la requête est préparée sur le serveur, le script peut l'exécuter en invoquant la méthode `execute()`[22] de la classe PDOStatement. Son paramètre prend les différentes valeurs passées sous forme de tableau

```
boolean $objetPDOStatement->execute(array $parametres = array());
```

La manière de procéder sera différente selon la méthode utilisée lors de la préparation de la requête. L'exemple ci-dessous retient le mode de transmission par des paramètres nommés. Cette approche offre le plus de lisibilité bien qu'elle soit plus longue à écrire.

```
1. try {
2.     $sql2 = "INSERT INTO `ict151`.`tbl_membres` (`uname`,`upwd`, `niveau`)
3.          VALUES (:nom, :mdp, :niveau);";
4.     $stmt=$dbh->prepare($sql2);
5.
6.     //lier les données
7.     $nom="Jojo";
8.     $mdp="passpass";
```

21 https://secure.php.net/manual/fr/pdo.prepare.php
22 https://secure.php.net/manual/fr/pdostatement.execute.php

```
    $niveau=2;

    //exécution de la requête
    $stmt->execute(array(':nom'=> $nom, ':mdp'=> $mdp, ':niveau'=> $niveau));

    //lier d'autres données
    $nom="Jonas";
    $mdp="passPartout";
    $niveau=1;

    //exécution de la requête
    $stmt->execute(
       array(
          ':nom'=> $nom,
          ':mdp'=> $mdp,
          ':niveau'=> $niveau
       )
    );
    //fermeture de la requête préparée
    $stmt=null;
}
catch (PDOException $e) {
    die($e->getMessage());
}
```

Dans exemple ci-dessus, la requête est tout d'abord préparée. Elle est ensuite appelée à deux reprise consécutives avec des données différentes.

L'association paramètres / valeurs

Le passage des paramètres à la requête préparée par le la méthode `execute()` risque de poser problème. :

```
$sql2 = "SELECT * FROM `ict151`.`tbl_membres` LIMIT :offset,:limit;";
$stmt=$dbh->prepare($sql2);
//lier les données
$offset=0;
$limit=3;
//exécution de la requête
$stmt->execute(array(':offset'=> $offset, ':limit'=> $limit));
```

Lors de l'exécution, cette requête de sélection dont la portée est limitée par des valeurs passées en paramètre ne fonctionne pas correctement. Effectivement, le serveur considère les paramètres comme des chaînes de caractères. L'exception levée par l'exécution de cette requête indique que les valeurs de la limite sont incorrectes. Voilà à quoi ressemble la requête interprétée par le serveur :

```
SELECT * FROM `ict151`.`tbl_membres` LIMIT '0','3';
```

Il est nécessaire de transmettre le type du paramètre au serveur. La classe PDOStatement implément les méthodes de `bindValue()`[23] et `bindParam()`[24] qui permettent de lier directement un marqueur de paramètre avec une valeur ou avec une référence sur une variable.

23 https://secure.php.net/manual/fr/pdostatement.bindparam.php
24 https://secure.php.net/manual/fr/pdostatement.bindvalue.php

```
boolean $objetPDOStatement->bindValue(mixed $parametre, mixed $value,
int $data_type = PDO::PARAM_xx);
```

```
boolean $objetPDOStatement->bindParam(mixed $parametre, mixed
&$variable, int $data_type = PDO::PARAM_xx);
```

Dans le cas de `bindParam()` le paramètre est passé par référence. Il est directement lié à une variable. Ainsi, entre deux exécutions, il ne sera nécessaire que de changer la valeur de la variable.

Le type de la variable est défini dans le 3e paramètre des méthodes. Il peut prendre la valeur du type SQL réel désiré parmi la liste non exhaustive ci-dessous:

Valeur de la constante	Action
PDO::PARAM_BOOL	un booléen (si ce type n'existe pas réellement sur votre SGBD, PDO effectuera les conversions nécessaires)
PDO::PARAM_NULL	la valeur NULL (en SQL)
PDO::PARAM_INT	un entier
PDO::PARAM_STR	une chaîne SQL (inclut les dates, les flottants, ...)

```php
try {
    $sql2 = "SELECT * FROM `ict151`.`tbl_membres` LIMIT :offset,:limit;";
    $stmt=$dbh->prepare($sql2);

    //association du marqueur à une variable (E/S)
    $stmt->bindParam(':offset',$offset,PDO::PARAM_INT);
    $stmt->bindParam(':limit',$limit,PDO::PARAM_INT);

    //Première exécution de la requête
    //charger les données
    $offset=0;
    $limit=3;
    $stmt->execute();

      //traitement des résultats

    //deuxième exécution de la requête
    //modification de la valeur et exécution de la requête
    $limit=2;
    $stmt->execute();

      //traitement des résultats

    $stmt=null; //fermeture de la requête préparée
}
```

5.3.4 Exploitation d'une requête préparée de sélection

Le fonctionnement est assez similaire au mode de fonctionnement des requêtes classiques présenté au chapitre 5.1.4 à la page 15. Après l'exécution de la requête, les données sont en

mémoire. Il faut aller les chercher et les récupérer selon un style défini (tableau associatif, tableau indicé, objets). Le parcours des enregistrements peut alors se faire avec les méthodes classiques comme `fetch()`, `fetchAll()` ou l'accès par l'itérateur et la boucle `foreach`.

```php
try {
    //préparation de la requête
    $sql = "SELECT * FROM ict151.tbl_membres WHERE upwd=
:mdp;";
    $stmt=$dbh->prepare($sql);
    //association du marqueur à une variable (E/S)
    $stmt->bindParam(':mdp',$mdp,PDO::PARAM_STR);
    //lier les données
    $mdp='pass';

    //exécution de la requête
    $stmt->execute();
    //choix du mode de récuperation
    $stmt->setFetchMode(PDO::FETCH_ASSOC);


    //affichage des résultats
    foreach($stmt as $row){
        echo '<p>',$row['id'], ' - ',$row['uname'],
             ' - ',$row['upwd'], ' - ',$row['niveau'],'</p>';
    }
    echo '<hr>';

    //deuxième exécution
    //exécution de la requête
    $mdp='passpass';
    $stmt->execute();
    $stmt->setFetchMode(PDO::FETCH_ASSOC);


    //affichage des résultat
    foreach($stmt as $row){
        echo '<p>',$row['id'], ' - ',$row['uname'],
             ' - ',$row['upwd'], ' - ',$row['niveau'],'</p>';
    }
    echo '<hr>';


    //fermeture de la requête préparée
    $stmt=null;
}
catch (PDOException $e) {
    die($e->getMessage());
}
```

3 - Moine - pass - 1
4 - Aùbré - pass - 1
5 - Monnot - pass - 1
15 - Hauser - pass - 2
16 - Hauser - pass - 2
17 - Hauser - pass - 2
18 - Hauser - pass - 2

26 - Jojo - passpass - 2

5.3.5 Exploitation d'une requête préparée d'ajout

La méthode proposé ci-dessous s'appuie sur la liaison des données à la requête au travers de l'invocation de la méthode `bindParam()` de la classe `PDOStatement`. Dans un premier temps il faut écrire la requête d'insertion en déterminant le nom des marqueurs à utiliser pour réaliser la liaison entre les variable PHP et la requête SQL. Dans un 2ème temps, l'invocation de la méthode `bindParam()`, permet de relier effectivement le marqueur et la variable tout en exprimant le mode de conversion explicite demandé.

Une autre manière de transmettre les données à la requête, via les paramètres de la méthode `execute()` est possible (cf. exemple du chapitre 5.3.3 à la page 22). Cette méthode ne permet toutefois pas de forcer la conversion vers un type défini.

```
1.  try {
2.
3.      $sql2 = "INSERT INTO ict151.tbl_membres (uname, upwd, niveau)
4.           VALUES (:nom, :mdp, :niveau);";
5.
6.      //préparation de la requête sur le serveur
7.      $stmt=$dbh->prepare($sql2);
8.
9.      //Liaison des données
10.     $stmt->bindParam(':nom',$nom,PDO::PARAM_STR);
11.     $stmt->bindParam(':mdp',$mdp,PDO::PARAM_STR);
12.     $stmt->bindParam(':niveau',$niveau,PDO::PARAM_INT);
13.     $nom="Kolwitz";
14.     $mdp="Platz";
15.     $niveau=55;
16.
17.
18.     // PREMIERE exécution de la requête.
19.     // Les nouvelles données sont dans les variables liées
20.     $stmt->execute();
21.     //lier d'autres données
22.     $nom="Jonas";
23.     $mdp="passPartout";
24.     $niveau="1";
25.
26.     // DEUXIEME exécution de la requête.
27.     // Les nouvelles données sont dans les variables liées
28.     $stmt->execute();
29.     //fermeture de la requête préparée
30.     $stmt=null;
31. }
32. catch (PDOException $e) {
33.     die($e->getMessage());
34. }
35. //fermeture de la connexion si elle existe
36. if ($dbh)
37.     $dbh = null;
```

5.3.6 Exploitation d'une requête préparée de modification

La méthode proposée ci-dessous est très proche de celle utilisée pour l'ajout de données présentée au chapitre 5.3.5 à la page 25. La requête de modification est écrite avec ses marqueurs. Les variables PHP sont liées aux marqueurs par la méthode `bindParam()` tout en spécifiant le type requis dans la base de données afin de forcer les conversions explicites de type.

Comme dans le cas de l'ajout, il reste possible d'envoyer des données à la requête via les paramètres de la méthode `execute()`.

```
//exécution de la requête
$stmt->execute(array( ':oldNiv'=> $ancienNiveau,':newNiv'=> $nouveauNiveau));
```

Cette technique ne requière plus l'association des variables aux marqueurs par `bindParam()`, ne permet toutefois plus de contrôler les conversions explicites de type.

```
1. //modèle de requête avec paramètre nommés
2. $sql = "UPDATE `ict151`.`tbl_membres` SET `niveau`= :newNiv WHERE
   `niveau`=:oldNiv;";
3.
4. try {
5.     $stmt=$dbh->prepare($sql);
6.     //association du marqueur à une variable (E/S)
7.     $stmt->bindParam(':newNiv',$nouveauNiveau,PDO::PARAM_INT);
8.     $stmt->bindParam(':oldNiv',$ancienNiveau,PDO::PARAM_INT);
9.
10.    //affectation de nouvelles valeurs
11.    $ancienNiveau=3;
12.    $nouveauNiveau=2;
13.
14.    //exécution de la requête
15.    //Les données sont dans les variables liées
16.    $stmt->execute();
17.
18.    //récupère le nombre de ligne affectée par l'exécution de la requête
19.    $nbModification=$stmt->rowCount();
20.    if($nbModification){
21.        echo $nbModification, ' ligne(s) modifiée(s)';
22.    }
23.    else{
24.        echo 'aucune ligne modifiée';
25.    }
26.
27.    //fermeture de la requête préparée
28.    $stmt=null;
29. }
30. catch (PDOException $e) {
31.    die($e->getMessage());
32. }
33. //fermeture de la connexion si elle existe
34. if ($dbh)
35. $dbh = null;
```

5.3.7 Exploitation d'une requête préparée de suppression

Toujours selon le même principe que pour l'ajout ou la modification, la suppression d'enregistrements se base sur une requête préparée, dotées de marqueurs liés à des variables PHP.

L'exemple ci-dessous lève deux exceptions différentes. La gestion du paramètre d'URL `id` peut provoquer la levé d'une exception ainsi que toutes erreurs lors de l'accès à la base de données.

```
try {
```

```php
    $id=filter_var($_GET['id'],FILTER_VALIDATE_INT);
    if(!isset($_GET['id']) || $id===false )
    {
        $excep= new Exception(" Ajoutez un paramère id= à l'url...");
        throw $excep;
    }

    $stmt=$dbh->prepare($sql);
    //association du marqueur à une variable (E/S)
    $stmt->bindParam(':id',$id,PDO::PARAM_INT);
    //exécution de la requête
    $stmt->execute();

    $nbModification=$stmt->rowCount();
    if($nbModification){
        echo $nbModification, ' ligne(s) supprimé(s)';
    }
    else{
        echo 'aucune ligne supprimé';
    }
    //fermeture de la requête préparée
    $stmt=null;
}

   //gère les exceptions levées par la BD
catch (PDOException $e) {
    die($e->getMessage());
}

   //gère les exceptions levées par le contrôle du paramètre d'URL
catch (Exception $e){
    die($e->getMessage());
}
//fermeture de la connexion si elle existe
if ($dbh)
    $dbh = null;
```

6 Gestion de l'authentification

Dans le contexte du web, les utilisateurs désirent fréquemment proposer des pages à accès limité. Cette approche répond aux concepts d'internet, d'intranet ou extranet.

Pour résumer simplement, l'internet est un espace public ouvert à tout le monde. Son accès ne requiert aucune authentification particulière. L'intranet est un espace semi-privé qui requiert des droits particuliers pour s'y rendre. Dans le cas des organisations, l'intranet est souvent retenu pour proposer un accès à des ressources internes sous forme de site web. L'extranet est un moyen d'accéder à des ressources internes comme celles de l'internet depuis l'extérieur de l'organisation.

Exemple d'accès à des ressources publiques et privées

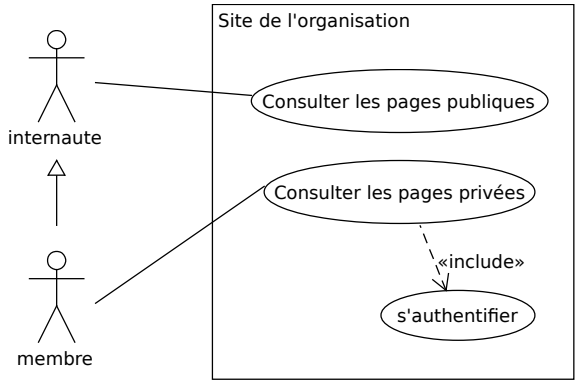


fig. 1: différents niveaux d'accès

Le membre est un internaute particulier. Il peut agir comme un internaute et il peut aussi accéder aux pages privées moyennant une authentification.

La réalisation de tels systèmes s'articule autour de deux objectifs distincts :

a) La gestion de l'authentification (cf. chapitre 6.1).

b) La gestion de la limitation de l'accès aux ressources du serveur (cf. chapitre7).

6.1 Authentification propre à l'application

Une authentification basique peut être construite sur les possibilités du protocole HTTP décrite dans la RFC 2617[25] permet de s'identifier sur un serveur web comme Apache ou ISS en lui montrant qu'un nom d'utilisateur et un mot de passe valable sont connus.

Il également possible de s'authentifier de manière décentralisée. Selon le fournisseur choisi, cette approche permet de profiter d'une authentification unique, sur une palette de sites différents. Le risque réside dans la perte ou la divulgation de cet identifiant unique. Ce type

25 http://www.faqs.org/rfcs/rfc2617.html

d'authentification peut se réaliser auprès de quantité de fournisseurs comme OpenID, SuisseID ou bien d'autres[26]

Ici nous nous intéresserons à l'authentification entièrement réalisée dans l'application et dont les données seront stockée en base de données. Le processus doit être entièrement géré en interne, entre l'application PHP, la base de données et la logique applicative du site. Le processus de gestion global d'une telle authentification est le suivant :

1. Présenter un formulaire d'authentification
2. Récupérer les données saisies par l'utilisateur
3. Valider les données saisies par l'utilisateur en regard d'une liste d'utilisateurs valides souvent stockée dans une base de données
4. Inscrire l'utilisateur dans l'application (souvent au moyen de variables de sessions)
5. Rediriger l'utilisateur au bon endroit.

Cette démarche ressemble à celle mise en œuvre pour l'authentification via un formulaire d'authentification HTML et la gestion des sessions.

6.1.1 Présenter un formulaire d'authentification

Le principe retenu est le même que pour des formulaires quelconques. Le formulaire peut être plus ou moins élaboré. La gestion des règles contraintes sur les champs ou la gestion de la rémanence sont de la responsabilité du programmeur.

Exemple de code du formulaire d'authentification

```
1. <form action="" method="post">
2.     <p>
3.         <label>Nom *: </label>
4.         <input type="text" name="v_nom"
5.                value="<?php
6.                if (!empty ($_POST['v_nom']))
7.                    echo filter_input(INPUT_POST,'v_nom',FILTER_SANITIZE_STRING);
8.                ?>"
9.     </p>
10.    <p>
11.        <label>Mot de passe *: </label>
12.        <input type="text" name="v_mdp"
13.               value="<?php
14.               if (!empty ($_POST['v_mdp']))
15.                   echo filter_input(INPUT_POST,'v_mdp',FILTER_SANITIZE_STRING);
16.               ?>"
17.    </p>
18.    <p>
19.        <button type="submit">Envoyer</button>
20.        <input type="hidden" name="v_frm_identite" value="envoye"
21.    </p>
22. </form>
```

Les 2 champs à saisir sont obligatoires. Le formulaire offre une rémanence utilisée en cas d'erreur de saisie.

26 Pour aller plus loin : https://fr.wikipedia.org/wiki/Authentification_unique, http://oauth.net/

6.1.2 Récupérer et valider un mot de passe en base de données

Le formulaire et son traitement sont stockés dans le même fichier. Il s'agit d'un autoformulaire. Le traitement du formulaire se charge des étapes 2-5 du processus présenté plus haut :

Exemple de code de traitement du formulaire d'authentification

```
1.  //force l'affichage du formulaire d'authentification
2.  $estAutherntifie=false;
3.  //si le formulaire à été envoyé
4.  if(isset($_POST['v_frm_identite']))
5.  {
6.      //récupère et nettoie les entrées du formulaire
7.      $nom=filter_input(INPUT_POST,'v_nom',FILTER_SANITIZE_STRING);
8.      $mdp=filter_input(INPUT_POST,'v_mdp',FILTER_SANITIZE_STRING);
9.      //le nom est obligatoire
10.     if(empty($nom))
11.     {?>
12.         <ul><li>ATTENTION le champ nom est obligatoire</li></ul>
13.         <?php
14.     }
15.     //le mot de passe est obligatoire
16.     if(empty($mdp))
17.     {?>
18.         <ul><li>ATTENTION le champ mot de passe est obligatoire</li></ul>
19.         <?php
20.     }
21.     //si les champs son remplis
22.     if (!empty ($nom) && !empty ($mdp)) {
23.         //recherche dans la base de données
24.         //initialise $dbh
25.         require_once("./conn.inc.php");
26.         //préparation de la requête
27.         try {
28.             $sql= "SELECT * FROM tbl_membres WHERE uname=:uname AND upwd=:umdp;";
29.             $stmt=$dbh->prepare($sql);
30.             //association du marqueur à une variable (E/S)
31.             $stmt->bindParam(':uname',$nom,PDO::PARAM_STR);
32.             $stmt->bindParam(':umdp',$mdp,PDO::PARAM_STR);
33.             $stmt->execute();
34.             //choix du mode de récuperation
35.             $stmt->setFetchMode(PDO::FETCH_ASSOC);
36.             $nbEnreg=$stmt->rowCount();
37.             //si la réponse ne contient pas UN SEUL enregistrement correspondant
38.             if($nbEnreg!=1) {
39.                 ?>
40.                 <ul>
41.                     <li>ATTENTION le nom ou le mot de passe est faux</li>
42.                 </ul>
43.                 <?php
44.             }
45.             else{
46.                 $estAutherntifie=true;
47.                 //incription des données de l'utilisateur dans l'application
48.                 $row=$stmt->fetchAll();
49.                 $_SESSION['nom']=$row[0]['uname'];
50.                 $_SESSION['prenom']=$row[0]['prenom'];
51.                 $_SESSION['niveau']=$row[0]['niveau'];
52.             }
53.         }
54.         catch (PDOException $e) {
55.             die($e->getMessage());
56.         }
57.     }
```

```
58. }
59. ?>
```

La section allant de la ligne 4 à la ligne 20 permet de gérer les valeurs envoyées par le formulaire :

- Lignes 7-8 : Récupération et nettoyage des valeurs saisies dans le formulaire.
- Lignes 10-20 : Vérification des données reçues. L'exemple affiche des messages d'erreurs. Il serait aussi possible d'uniquement les préparer dans des variables et de les afficher dans le corps de la page.

La section allant de la ligne 22 à la ligne 58 permet de vérifier l'authentification :

- Lignes 25-33 : Recherche du couple *nom/mot de passe* dans la base de données. Dans la table, le champ nom doit être unique pour éviter les doublons..
- Lignes 35-44 : L'authentification est considérée comme valable si la requête renvoie un et un seul enregistrement. Si le couple *nom/mot de passe* n'existe pas, l'exemple affiche une erreur.
- Lignes 46-52 : Le couple *nom/mot de passe* existe dans la base de données. L'enregistrement est lu. Les informations jugées importantes sont inscrites dans la session de l'utilisateur en cours. L'exemple retient une valeur de *niveau* propre à chaque utilisateur. Cette valeur est utilisée plus tard pour définir les droits d'accès aux différentes pages du site.

Cet exemple, au même titre que celui de l'authentification HTTP transfère le mot de passe en clair lors de l'envoi du formulaire. Pour le rendre utilisable, il faut s'assurer que le protocole utilisé soit sécurisé (HTTPS).

6.2 Stockage du mot de passe

Pour éviter que des listes de mots de passe puissent être involontairement ou volontairement divulguées, il faut prendre l'habitude de ne pas stocker les mots de passe en clair dans la base de données.

Le hachage d'un mot de passe est une opération à sens unique qui permet d'en obtenir une chaîne de caractères non prédictible et qui semble aléatoire. Comme l'opération est à sens unique, personne ne pourra décoder la chaîne résultante et revenir à la donnée d'origine. Pas même le propriétaire du mot de passe.

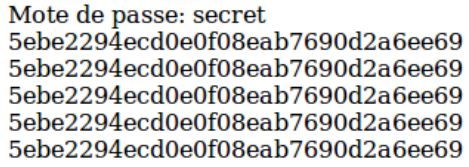


Illustration 10: hachages successifs d'une même chaîne avec MD5

La fonction *crypt()* de PHP[27] permet de réaliser de telles empreintes. Le mot de passe fourni à la fonction peut être codé avec différents algorithmes selon la version de PHP et sa configuration. Les algorithmes fréquents classés par ordre de robustesse sont DES, MD5 ou BLOWFISH. La fonction c*rypt()* ajoute un sel[28] à la chaîne à crypter. Ce sel permet de renforcer la chaîne à crypter. Il peut s'agir d'une valeur constante ou d'une valeur dynamique. Cette valeur dynamique consiste souvent en un chaîne aléatoire qui rend l'empreinte unique.

```
CRYPT_BLOWFISH is enabled!
crypt md5: $1$SHm0SQK7$YYPadTa/7CAL7TKxQGXbe/
crypt md5: $1$VszKjtHK$5114zfcm02IdyB4anWs0k1

crypt bfish: $2y$10$w65YIyXIUZAD7/qAGFNjxutZcKlQEGDf/fd1E30XpxAF0oUStKhqG
crypt bfish: $2y$10$s27QPogUJwU9a6Od.TjhOOXuhljBj0XfQ2fDOuFVT6RARbtENZ.Cy
```

Illustration 11: empreinte md5 et blowfish

6.2.1 Chiffrer un mot de passe

Depuis la version 5.5 de PHP, la fonction *password_hash()*, compatible avec *crypt()*, permet de crypter les données en spécifiant l'algorithme. L'algorithme par défaut risque d'évoluer avec le temps[29].

```php
$password="pass"; //ou valeur reçue via le formulaire
//(depuis PHP 5.5)
$password_hash = password_hash($password,PASSWORD_DEFAULT);

// sauvegarde de l'empreinte dans la base de données
...
```

La taille des empreintes est différente selon les algorithmes utilisés. Il est préférable de définir un champ suffisamment grand dans la base de données pour stocker ces informations. Pour assurer la compatibilité avec les algorithmes à venir, la documentation de PHP préconise l'utilisation d'un champs de **255 caractères** lors de l'utilisation du chiffrement par défaut de la fonction *password_hash()*.

6.2.2 Valider un mot de passe

Pour valider un mot de passe alors qu'on ne connaît que son empreinte (résultat du hachage), il suffit de comparer cette empreinte avec le résultat de l'opération de hachage sur le mot de passe fourni par l'utilisateur.

27 http://php.net/manual/fr/function.crypt.php
28 https://fr.wikipedia.org/wiki/Salage_(cryptographie) , https://chiffrer.info/
29 http://php.net/manual/fr/function.password-hash.php

Depuis la version 5.5 de PHP la fonction *password_verify()* permet de comparer deux empreintes[30] :

```php
$password_entered="xxxx"; //ou valeur reçue via le formulaire de login
$password_hash = … //valeur lue dans la base de données

if(password_verify($password_entered, $password_hash))
    echo "auth OK","<br>";
else
    echo "auth PAS BON","<br>";
```

Exemple de code de traitement du formulaire d'authentification (mot de passe crypté)

```
1. if (!empty ($nom) && !empty ($mdp)) {
2.     //recherche dans la base de données
3.     //initialise $dbh
4.     require_once("./conn.inc.php");
5.     //préparation de la requête
6.     try{
7.         $sql = "SELECT * FROM tbl_membres_crypt WHERE uname= :uname;";
8.         $stmt=$dbh->prepare($sql);
9.
10.        //association du marqueur à une variable (E/S)
11.        $stmt->bindParam(':uname',$nom,PDO::PARAM_STR);
12.        $stmt->execute();
13.        //choix du mode de récuperation
14.        $stmt->setFetchMode(PDO::FETCH_ASSOC);
15.
16.        $row=$stmt->fetchAll();
17.        $mdp_hash_db=$row[0]['upwd'];
18.
19.        //compare l'empreinte avec le mot de passe
20.        if(!password_verify($mdp, $mdp_hash_db)) {
21.          ?>
22.            <ul>
23.               <li>ATTENTION le nom ou le mot de passe est faux</li>
24.            </ul>
25.          <?php
26.        }
27.        else{
28.            $estAutherntifie=true;
29.            //incription des données de l'utilisateur dans l'application
30.            $_SESSION['nom']=$row[0]['uname'];
31.            $_SESSION['prenom']=$row[0]['prenom'];
32.            $_SESSION['niveau']=$row[0]['niveau'];
33.            //redirection possible..
34.        }
35.    }
36.    catch (PDOException $e) {
37.        die($e->getMessage());
38.    }
39.}
```

La section allant de la ligne 1 à la ligne 39 permet de vérifier l'authentification en comparant le mot de passe saisi avec l'empreinte stockée en base de données :

30 http://php.net/manual/fr/function.password-verify.php

- Lignes 7-17 : Recherche de l'enregistrement correspondant au nom d'utilisateur saisi dans le formulaire. L'empreinte retournée est stockée dans une variable.
- Ligne 20 : Le mot de passe saisi est comparé à l'empreinte stockée en base de données. On ne peut pas comparer l'équivalence des empreintes. Elles ne sont jamais identiques (c.f. Illustration 11)
- Lignes 21-24 : Affichage d'une erreur si la comparaison échoue
- Lignes 28-33 : le mot de passe saisi correspond à l'empreinte dans la base de données. Les informations jugées importantes sont inscrites dans la session de l'utilisateur en cours. L'exemple retient une valeur de *niveau* propre à chaque utilisateur. Cette valeur est utilisée plus tard pour définir les droits d'accès aux différentes pages du site.

Le stockage de l'empreinte du mot de passe assure contre la divulgation des mots de passe. Par contre il n'améliore en rien le transfert du mot de passe entre le client et le serveur. Le seul moyen disponible pour sécuriser la requête HTTP (par exemple en passant sur le protocole HTTPS).

7 La gestion de l'accès aux ressources du serveur

Les pages d'une application ne sont pas accessibles à tous les utilisateurs. Les utilisateurs possèdent des rôles dans le système qui leur donnent des droits d'accès à certaines pages de l'application. Les accès aux pages devront être réglementés par une logique applicative spécifique.

7.1 Rôle des utilisateurs

Dans le cas du site d'une organisation, plusieurs niveaux de droits d'accès peuvent cohabiter. Il est certaines fois possible de hiérarchiser ces accès en définissant le niveau d'accès le élevé accessible à un utilisateur ainsi que le niveau de droit minimum requis pour chaque page du site:

Utilisateurs	Niveau d'accès maximum autorisé
userA	1
userB	4
userC	2
userD	3

Page du site	Niveau d'utilisateur maximum autorisé
page1.php	tous
page2.php	1
page3.php	2
page4.php	4

Les tableaux ci-dessus permettent de lire que l'utilisateur *userC* de niveau **2** peut accéder aux pages suivantes :

- **page1.php** qui est en accès libre
- **page2.php** et **page3.php**, accessible à toutes personnes de niveau supérieur à 2

Dans ce système l'administrateur a le niveau le plus élevé. Ce serait l'utilisateur *userB*

7.2 Mémorisation de l'utilisateur actuel dans l'application

La mémorisation dans l'application de l'utilisateur authentifié, se résume à l'inscrire dans la session ($_SESSION) juste après la validation de son mot de passe.

```
1.  //inscription des données de l'utilisateur dans l'application
2.  $_SESSION['nom']=$row[0]['uname'];
3.  $_SESSION['prenom']=$row[0]['prenom'];
4.  $_SESSION['niveau']=$row[0]['niveau'];
5.  //redirection possible..
```

Pour bien fonctionner, il faut nécessairement inscrire le niveau de l'utilisateur.

La déconnexion d'un utilisateur se fait en supprimant ses données de la session ou toute sa session :

```
if(isset($_SESSION))
    session_destroy();
```

7.3 Limitation d'accès aux pages

La limitation de l'accès à une page doit répondre aux deux règles suivantes :

1. Interdire l'accès direct à une page qui n'est pas publique :
2. Limiter l'accès des pages qui ne sont pas publiques en fonction des droits effectifs de l'utilisateur authentifié

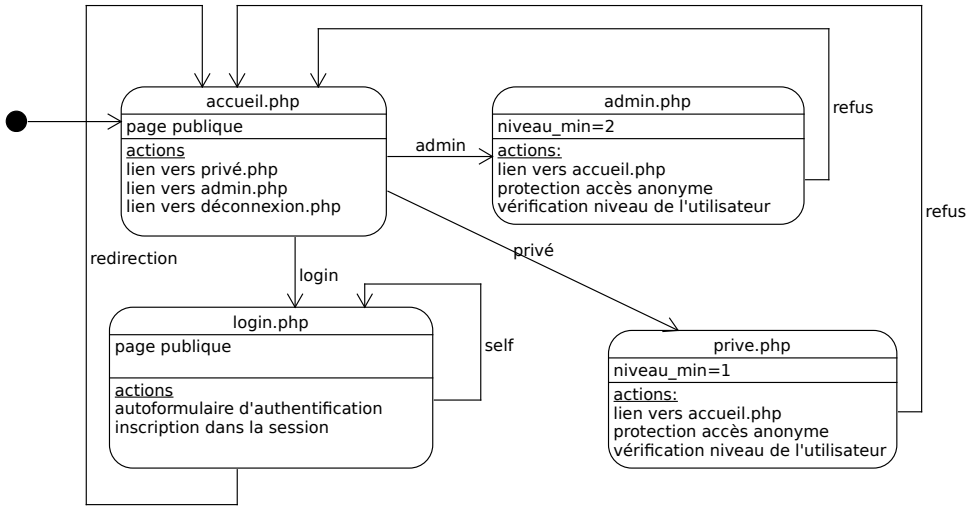


Illustration 12: logique de l'accès protégée aux pages

La figure ci-dessus décrit le principe général du contrôle d'accès aux page de l'application web.

- Les pages publiques (*accueil.php, login.php*)sont accessibles à tout le monde.
- Les pages privées (*admin.php, privé.php*) sont inaccessibles aux internautes anonymes. Il sont refusés et redirigés vers la page d'accueil.
- Les pages privés (*admin.php, prive.php*) sont d'accès limité aux membres (les internaute authentifiés). Lorsque leur niveau d'accès est insuffisant, ils sont refusés et redirigés vers la page d'accueil.

Les limitations et protections décrites dans cette liste sont mis en œuvre par les principes expliqué ci-dessous.

7.3.1 Interdire l'accès direct anonyme à une page privé

Un utilisateur non authentifié ne possède pas de données le caractérisant dans la session. Il suffit de vérifier la présence de ces données (par exemple sont *niveau*) pour décider si la page doit s'afficher ou non. Généralement, en cas d'échec l'internaute débouté est renvoyé sur la page d'accueil ou sur la page d'authentification.

Exemple de code de limitation de l'accès direct

```
1. session_start();
2. //definir l'encodage
3. header('Content-Type: text/html; charset=utf-8');
4. if(!isset($_SESSION['niveau'])){
5.     //header("location:accueil.php");
6.     header("refresh:3;url=accueil.php");
7.     echo "pas authentifié !";
8.     exit;
9. }
```

Dans l'exemple, l'utilisateur reçoit un message d'erreur avant d'être redirigé.

7.3.2 Limiter l'accès en fonction du niveau de l'utilisateur

Les utilisateurs authentifiés possèdent dans la session, des données relatives à leur droit. Pour établir si l'utilisateur peut visualiser la page, il suffit de comparer le niveau maximum acquis par l'utilisateur avec le niveau minimum requis par la page. Généralement, en cas d'échec l'internaute débouté est renvoyé sur la page d'accueil ou sur la page d'authentification.

Exemple de code de limitation en fonction du niveau de droit de l'utilisateur

```
1. //définir le niveau minimum requis pour visualiser la page
2. $niveau_min_page=2;
3. if($_SESSION['niveau']<$niveau_min_page){
4.     //header("location:accueil.php");
5.     header("refresh:3;url=accueil.php");
6.     echo "pas le niveau !";
7.     exit;
8. }
```

Dans l'exemple, l'utilisateur authentifié qui possède un niveau de 2 ou supérieur à 2 peut visualiser la page.

Exemple complet de gestion de l'accès à la page

```
1. <?php
2. session_start();
3.
4. //Protection contre l'accès directe anonyme
5. if(!isset($_SESSION['niveau'])){
6.     //header("location:accueil.php");
7.     header("refresh:3;url=accueil.php");
8.     echo "pas authentifié !";
9.     exit;
10.}
11.
12.//définir le niveau minimum requis pour visualiser la page
13.$niveau_min_page=2;
14.if($_SESSION['niveau']<$niveau_min_page){
15.     //header("location:accueil.php");
16.     header("refresh:3;url=accueil.php");
17.     echo "pas le niveau !";
18.     exit;
19.}
20.
21.//inscription des données de l'utilisateur dans l'application
22.$id=$_SESSION['id'];
23.$nom=$_SESSION['nom'];
24.$niveau=$_SESSION['niveau'];

25. //Corps de la page privée ….
```

8 Références et ressources

Ouvrages et revues

[ICT151] Collectif, "module ICT 151", CPLN, Neuchâtel, 2015
[ED-06] Eric Daspet, "PHP5 avancé" (3e édition), Eyrolle, 2006, Paris

Ressources en ligne

[PHP] Manuel de PHP [en ligne]
http://php.net/manual/fr/langref.php
http://www.php.net/manual/fr/mysqli.summary.php
http://php.net/manual/fr/language.exceptions.php

Manuel de PHP [en ligne]
la plupart de liens sur le manuel sont directement insérer en bas de pages

Manuel de MySQL [en ligne]
http://dev.MySQL.com/doc/refman/5.7/en/storage-engines.html

[JULP] Mode d'emploi PDO [en ligne et en EPUB]
http://www.julp.fr/articles/2-pdo-mode-d-emploi.html