BDW - Programmation web - PHP avancé

Fabien Duchateau

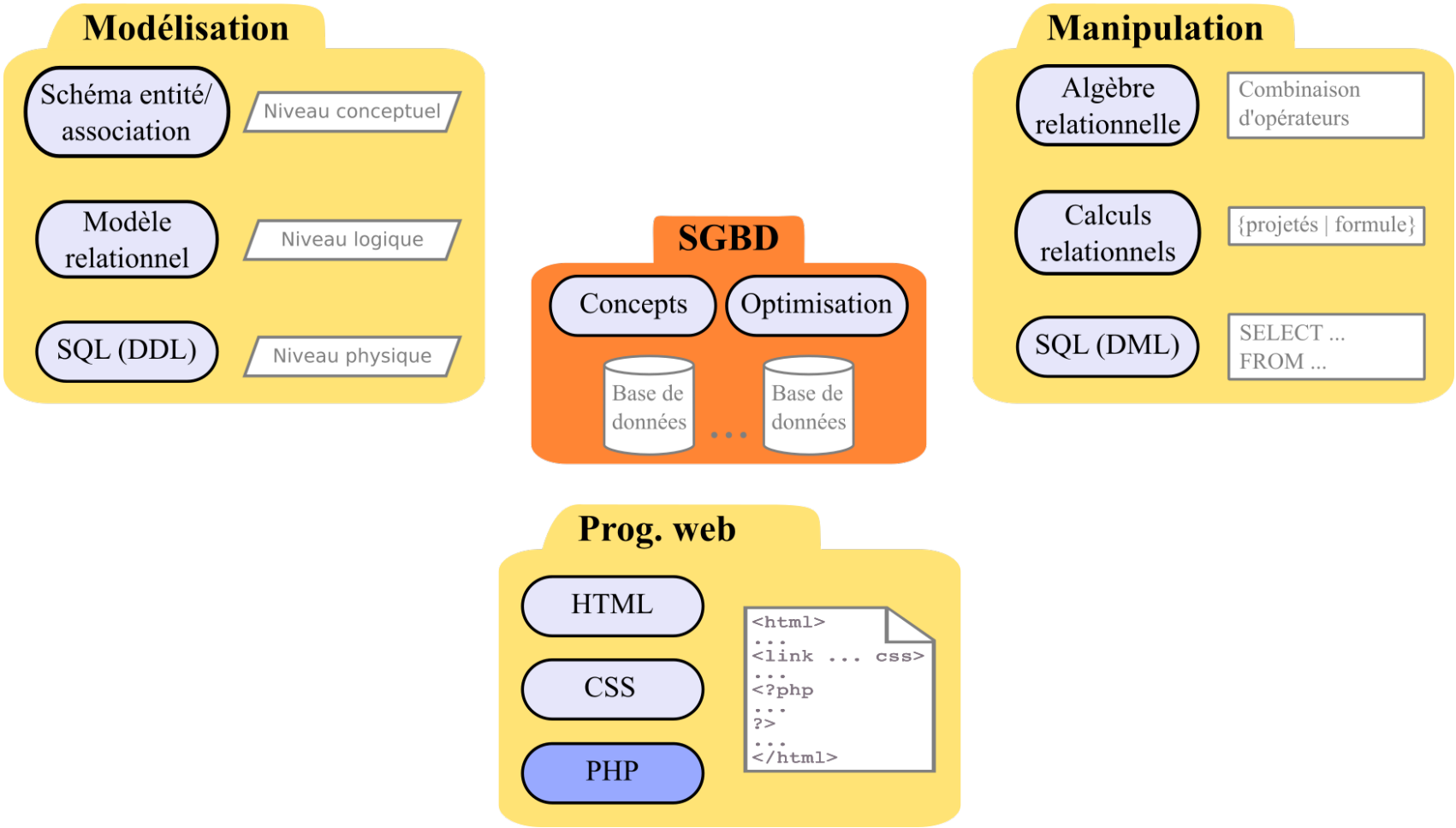
fabien.duchateau [at] univ-lyon1.fr

Université Claude Bernard Lyon 1

2023 - 2024

https://perso.liris.cnrs.fr/fabien.duchateau/BDW/

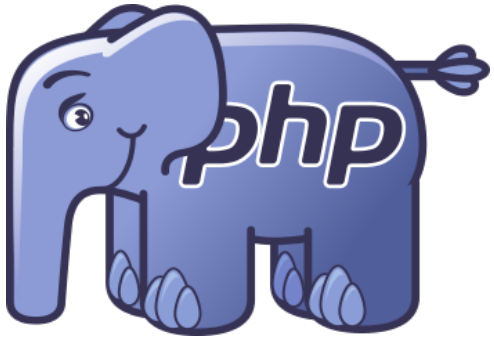
Positionnement dans BDW



Ces diapositives utilisent le genre féminin (e.g., chercheuse, développeuses) plutôt que l'écriture inclusive (moins accessible, moins concise, et pas totalement inclusive)

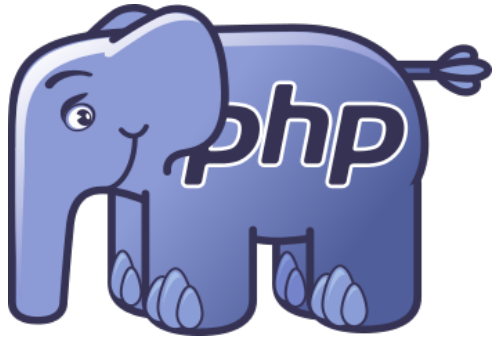
Rappels

- Syntaxe de PHP
- Structures conditionnelles et boucles
- Définition de fonctions



Rappels

- Syntaxe de PHP
- Structures conditionnelles et boucles
- Définition de fonctions



Dans ce cours, fonctionnalités plus spécifiques à l'utilisation de PHP en programmation web via les extensions

Extensions PHP

PHP fournit des extensions, classées en différentes catégories :

- Extensions noyau (e.g., *Arrays*, *Date/Time*, *FileSystem*)
- Extensions intégrées (e.g., *FTP*, *PostgreSQL*, *Zip*)
- Extensions externes (e.g., *MySQLi*, *GeoIP*, *cURL*)

En programmation web, extensions liées aux superglobales, aux sessions, aux bases de données et à la manipulation de fichiers

---

https://www.php.net/manual/en/extensions.membership.php

Connaitre les extensions PHP

- `phpinfo()`, qui fournit de nombreuses informations sur PHP (version, extensions, environnement, etc.)
- `get_loaded_extensions()`, qui retourne un tableau avec les extensions chargées

```
fabien@laptop :-$ php -a
Interactive shell

php > print_r(get_loaded_extensions());
Array
(
        [0] => Core
        [1] => date
        [2] => libxml
        ...
)
php >
```

---

http://php.net/manual/en/function.phpinfo.php
http://php.net/manual/en/function.get-loaded-extensions.php

Plan

Variables superglobales

PHP définit un certain nombre de variables superglobales :

- Accessibles de n'importe où
- Représentées par des tableaux associatifs

Une dizaine de superglobales dont :

- Variables d'environnement, du système d'exploitation (`$_ENV`)
- Variables de session (`$_SESSION`)
- Variables de serveur (`$_SERVER`)
- Variables de paramètres (`$_REQUEST`, `$_GET`, `$_POST`, `$_COOKIE`)
- Fichiers envoyés par formulaire (`$_FILES`)

---

http://www.php.net/manual/en/language.variables.superglobals.php

Superglobale $_GET

La variable $_GET :

- Tableau associatif qui contient les paramètres d'un formulaire soumis
- Ces paramètres ont été transmis via la méthode *get*

Syntaxe de l'utilisation de $_GET :

- Retourne la valeur pour le paramètre `nomParam`

```
$_GET['nomParam']
```

Un exemple de superglobale $_GET

Exemple : vérifier si un formulaire "get" est soumis

```
<form action="#" method="get">


</form>
```

Un exemple de superglobale $_GET

Exemple : vérifier si un formulaire "get" est soumis

```
<form action="#" method="get">
    <input type="text" name="nom">
    <input type="submit" name="bValider" value="Soumettre">
</form>
```

Un exemple de superglobale $_GET

Exemple : vérifier si un formulaire "get" est soumis

```
<?php
if(isset($_GET['bValider'])) // formulaire soumis

else { // formulaire non soumis, affichage du formulaire
 ?>
<form action="#" method="get">
    <input type="text" name="nom">
    <input type="submit" name="bValider" value="Soumettre">
</form>
<?php
}
 ?>
```

Un exemple de superglobale $_GET

Exemple : vérifier si un formulaire "get" est soumis

```
<?php
if(isset($_GET['bValider'])) // formulaire soumis
    echo "Le formulaire a bien été soumis !";
else { // formulaire non soumis, affichage du formulaire
?>
<form action="#" method="get">
    <input type="text" name="nom">
    <input type="submit" name="bValider" value="Soumettre">
</form>
<?php
}
?>
```

Superglobale $_POST

La variable $_POST :

- Tableau associatif qui contient les paramètres d'un formulaire soumis
- Ces paramètres ont été transmis via la méthode *post*

Syntaxe de l'utilisation de $_POST :

- Retourne la valeur pour le paramètre `nomParam`

```
$_POST['nomParam']
```

Un exemple de superglobale $_POST

Exemple : afficher le paramètre *nom* soumis par "*post*"

```
<form action="#" method="post">


</form>
```

Un exemple de superglobale $_POST

Exemple : afficher le paramètre *nom* soumis par "*post*"

```
<form action="#" method="post">
    <input type="text" name="nom">
    <input type="submit" name="bValider" value="Soumettre">
</form>
```

Un exemple de superglobale $_POST

Exemple : afficher le paramètre *nom* soumis par "*post*"

```
<?php
if(isset($_POST['bValider'])) // formulaire soumis




else { // formulaire non soumis, affichage du formulaire
 ?>
<form action="#" method="post">
    <input type="text" name="nom">
    <input type="submit" name="bValider" value="Soumettre">
</form>
<?php
}
 ?>
```

Un exemple de superglobale $_POST

Exemple : afficher le paramètre *nom* soumis par "*post*"

```
<?php
if(isset($_POST['bValider'])) // formulaire soumis
    if(isset($_POST['nom']) && !empty($_POST['nom']))

    else // formulaire soumis mais sans valeur pour le nom

else { // formulaire non soumis, affichage du formulaire
 ?>
<form action="#" method="post">
    <input type="text" name="nom">
    <input type="submit" name="bValider" value="Soumettre">
</form>
<?php
}
 ?>
```

Un exemple de superglobale $_POST

Exemple : afficher le paramètre *nom* soumis par "*post*"

```
<?php
if(isset($_POST['bValider'])) // formulaire soumis
    if(isset($_POST['nom']) && !empty($_POST['nom']))
        echo "Merci d'avoir soumis le formulaire, ".$_POST['nom'];
    else // formulaire soumis mais sans valeur pour le nom
        echo "Le formulaire a bien été soumis, mais sans nom !";
else { // formulaire non soumis, affichage du formulaire
?>
<form action="#" method="post">
    <input type="text" name="nom">
    <input type="submit" name="bValider" value="Soumettre">
</form>
<?php
}
?>
```

Un mot sur la sécurité

Les variables superglobales peuvent potentiellement être modifiées par l'utilisatrice (e.g., modifier la valeur d'un paramètre passé en *get* dans l'URL)

```
https://duckduckgo.com/?q=chocolat
https://duckduckgo.com/?q='%3B%20DROP%20TABLE%
20uneTable%3B%20--
```

Votre code doit donc vérifier et valider ces variables :

- Côté client (HTML, Javascript)
- Côté serveur (PHP), avec des fonctions prédéfinies (échappement de caractères spéciaux, encodage d'URL, etc.)

Superglobale $_FILES

Un formulaire peut permettre de sélectionner des fichiers (sur son disque) à envoyer au serveur

- $_FILES est un tableau associatif qui contient des informations sur les fichiers envoyés

Syntaxe de l'utilisation de $_FILES :

- Un formulaire HTML pour sélectionner le(s) fichier(s)
  - Attribut pour le type d'encodage données
  - Champ caché MAX_FILE_SIZE pour définir une limite de taille en octets
  - Champ de type *file*
- Un script PHP pour traiter les fichiers envoyés (dont gestion des erreurs via leur code, e.g., 2 = fichier trop volumineux par rapport à MAX_FILE_SIZE)

---

http://www.php.net/manual/en/features.file-upload
http://www.php.net/manual/en/features.file-upload.errors.php

Superglobale $_FILES

```
1 <form enctype="multipart/form-data" action="#" method="POST">
2   <!-- MAX_FILE_SIZE must precede the file input field -->
3   <input type="hidden" name="MAX_FILE_SIZE" value="30000000" />
4   Sélectionner un fichier : <input name="userfile" type="file"/><br/><br/>
5   <input type="submit" name="bsubmit" value="Envoyer fichier" />
6 </form>
```

Superglobale $_FILES

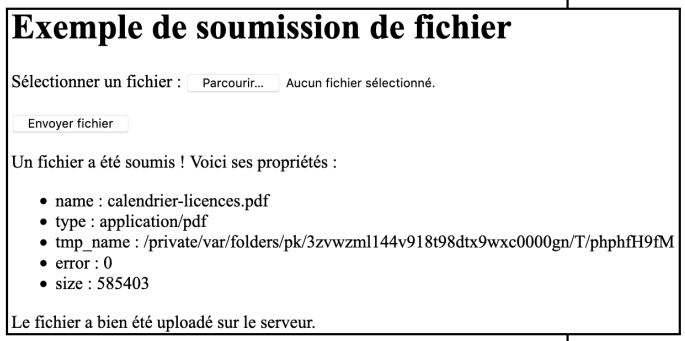
```php
<form enctype="multipart/form-data" action="#" method="POST">
  <!-- MAX_FILE_SIZE must precede the file input field -->
  <input type="hidden" name="MAX_FILE_SIZE" value="30000000" />
  Sélectionner un fichier : <input name="userfile" type="file"/><br/><br/>
  <input type="submit" name="bsubmit" value="Envoyer fichier" />
</form>
```


```php
if(isset($_POST['bsubmit'])) {
  if($_FILES['userfile']['error'] == 0) { // pas d'erreur
    echo 'Un fichier a été soumis ! Voici ses propriétés :<br>';
    //print_r($_FILES);
    echo '<ul>';
    foreach($_FILES['userfile'] as $prop => $val) {
      echo '<li>' . $prop . ' : ' . $val . '</li>';
    }
    echo '</ul>';
    // vérification que fichier sur le serveur
    if(is_uploaded_file($_FILES['userfile']['tmp_name'])) {
      echo 'Le fichier a bien été uploadé sur le serveur.';
    }
  }
  else {
    echo 'Erreur ! Code d\'erreur #' . $_FILES['userfile']['error'];
  }
}
```

Superglobale $_FILES

```php
<form enctype="multipart/form-data" action="#" method="POST">
  <!-- MAX_FILE_SIZE must precede the file input field -->
  <input type="hidden" name="MAX_FILE_SIZE" value="30000000" />
  Sélectionner un fichier : <input name="userfile" type="file"/><br/><br/>
  <input type="submit" name="bsubmit" value="Envoyer fichier" />
</form>
```


```php
if(isset($_POST['bsubmit'])) {
  if($_FILES['userfile']['error'] == 0) { // pas d'erreur
    echo 'Un fichier a été soumis ! Voici ses propriétés :<br>';
    //print_r($_FILES);
    echo '<ul>';
    foreach($_FILES['userfile'] as $prop => $val) {
      echo '<li>' . $prop . ' : ' . $val . '</li>';
    }
    echo '</ul>';
    // vérification que fichier sur le serveur
    if(is_uploaded_file($_FILES['userfile']['tmp_name'])) {
      echo 'Le fichier a bien été uploadé sur le serveur.';
    }
  }
  else {
    echo 'Erreur ! Code d\'erreur #' . $_FILES['userfile']['error'];
  }
}
```



En résumé

- Variables superglobables définies par PHP
- `$_GET` et `$_POST` pour récupérer les paramètres d'un formulaire soumis
- `$_FILES` pour un envoi de fichier(s) sur le serveur
- Effectuer des vérifications lors de la récupération des données

Plan

Généralités

Il peut être utile de conserver des informations d'une page sur l'autre :

- Se souvenir du login de l'utilisatrice pour les sites avec authentification
- Se souvenir des références indiquant à quoi l'utilisatrice s'intéresse
- Se souvenir des dernières pages visitées par l'utilisatrice
- ...

Généralités

Il peut être utile de conserver des informations d'une page sur l'autre :

- Se souvenir du login de l'utilisatrice pour les sites avec authentification
- Se souvenir des références indiquant à quoi l'utilisatrice s'intéresse
- Se souvenir des dernières pages visitées par l'utilisatrice
- ...

Jusqu'ici, un seul moyen : utiliser des paramètres et penser à les remettre sur chaque page/formulaire
⇒ **Programmation fastidieuse et source de problèmes**

Sessions en PHP

Une session peut être vue comme un ensemble d'informations concernant une utilisatrice d'un site :

- Par utilisatrice, on entend un navigateur sur une machine
- Les informations sont conservées entre deux pages
- Une page PHP peut ajouter ou modifier les informations de la session

En PHP, la session est vue comme une variable superglobale appelée `$_SESSION` :

- C'est donc un ... ?

---

http://www.php.net/manual/fr/book.session.php

Sessions en PHP

Une session peut être vue comme un ensemble d'informations concernant une utilisatrice d'un site :

- Par utilisatrice, on entend un navigateur sur une machine
- Les informations sont conservées entre deux pages
- Une page PHP peut ajouter ou modifier les informations de la session

En PHP, la session est vue comme une variable superglobale appelée `$_SESSION` :

- C'est donc un tableau associatif

---

http://www.php.net/manual/fr/book.session.php

Utilisation des sessions

Une page PHP utilisant une session doit **obligatoirement, avant même d'afficher quoi que ce soit,** commencer par l'instruction :

```
session_start() ; // pas d'espace ou ligne vide avant !
```

Cette instruction crée la variable `$_SESSION` et la remplit avec les valeurs qu'elle avait dans la page PHP précédente

- La variable `$_SESSION` se manipule ensuite comme un tableau associatif classique

Exemple de session

```php
<?php // session
session_start();
$_SESSION["dateConnex"] = date("r");
echo "Identifiant de session : " . session_id();
echo "Date de dernière connexion : " .
    $_SESSION['dateConnex'];
?>
```

Identifiant de session : 1d98975cf0e5695e2275c6bbc054078
Date de dernière connexion : 14 Oct 2023 10 :50 :58 +0000

Déconnexion

Lorsque l'utilisatrice se déconnecte, il est important de détruire sa session (e.g., éviter qu'une seconde personne utilisant le même ordinateur ne se fasse passer pour la première personne)

Deux étapes : réinitialiser *$_SESSION* pour effacer toutes les variables de la session courante, et détruire la session

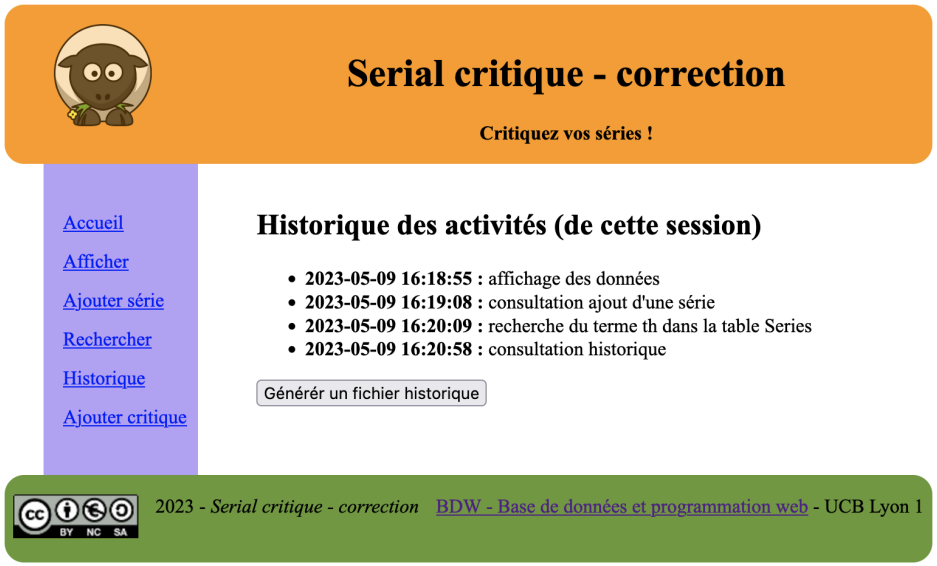
- Destruction du cookie de session pour plus de sécurité

```
$_SESSION = array(); // réinitialisation
session_destroy(); // destruction
```

http://www.php.net/manual/fr/function.session-destroy.php

En résumé

- Une session PHP est vue comme une variable superglobale, et donc gérée comme un tableau associatif
- Instruction *session_start()* **avant tout affichage** !
- Déconnexion en deux étapes (initialisation et destruction de la session)



Page historique utilisant la session

Plan

Généralités

Pour manipuler les bases de données, PHP possède plus d'une vingtaine d'extensions et des couches d'abstraction

Pour les SGBD MySQL / MariaDB, trois API différentes :

- **mysql** : API d'origine, aujourd'hui **obsolète**
- **mysqli** : *mysql improved*, avec de nouvelles fonctionnalités
- **PDO_MySQL** : couche d'abstraction d'accès aux données à travers des PHP Data Objects

---

http://php.net/manual/fr/refs.database.php

Généralités

Pour manipuler les bases de données, PHP possède plus d'une vingtaine d'extensions et des couches d'abstraction

Pour les SGBD MySQL / MariaDB, trois API différentes :

- 

mysql : API d'origine, aujourd'hui **obsolète**
- **mysqli** : *mysql improved*, avec de nouvelles fonctionnalités
- **PDO_MySQL** : couche d'abstraction d'accès aux données à travers des PHP Data Objects

---

http://php.net/manual/fr/refs.database.php

Généralités (2)

Utilisation de l'API **mysqli** :

- Support des paradigmes de programmation procédurale et orientée objet
- Support des requêtes multiples et des requêtes préparées
- Support des transactions
- ...

http://www.php.net/manual/fr/book.mysqli.php

Manipulation d'une base de données

Quatre étapes pour manipuler une base de données :

1. Connexion au SGBD et sélection d'une base
2. Exécution d'une requête (préparée ou non)
3. Récupération et utilisation du résultat
4. Fermeture de la connexion

On peut itérer les étapes 2 à 3 autant de fois que l'on veut avant de fermer la connexion à l'étape 4

Étape de connexion

Tentative de connexion avec *mysqli_connect* :

- Paramètres : *machine*, *utilisatrice*, *mot de passe* et *base de données*
- Retourne une ressource *$connexion*, qui représente le lien de connexion vers la base de données sélectionnée

```
$server = "127.0.0.1"; // serveur sur lequel tourne le SGBD
$user = "un_nom"; // utilisatrice du SGBD
$mdp = "un_mdp"; // mot de passe de l'utilisatrice
$bd = "une_bd"; // base de données à laquelle se connecter
$connexion = mysqli_connect($server, $user, $mdp, $bd);
```

Étape de connexion - statut

mysqli_connect_errno() // erreur si > 0

- `mysqli_connect_errno()` retourne le statut de la dernière tentative de connexion, soit un code d'erreur, soit 0 en cas de succès

```php
<?php // connexion à MariaDB
function connectBD() {
  $connexion = mysqli_connect($serveur, $user, $mdp, $bd);
  if (mysqli_connect_errno()) {
    printf("Échec de la connexion : %s\n", mysqli_connect_error());
    exit();
  }
  return $connexion;
}
$connexion = connectBD();
?>
```

Étape d'exécution d'une requête non préparée

```
// on possède un lien de connexion $connexion
$req = "une_requete_sql";
$resultat = mysqli_query($connexion, $req);
```

- `mysqli_query` exécute une requête non préparée
- Un lien de connexion est requis en paramètre
- La requête *$req* (SELECT, INSERT, etc.) est un paramètre :
  - Si la requête est construite avec les données d'un formulaire, il faut échapper ces données avec la fonction `mysqli_real_escape_string`
- La fonction retourne un booléen ou un objet

---

http://www.php.net/manual/fr/mysqli.real-escape-string.php
http://www.php.net/manual/fr/class.mysqli-result.php
http://kunststube.net/escapism/

Étape d'exécution d'une requête non préparée - remarques

- Précisions sur la valeur de retour de `mysqli_query` :
  - `false` en cas d'erreur
  - `true` pour des requêtes INSERT, UPDATE, et DELETE
  - un objet *$resultat* de type *mysqli_result*, qui contient les n-uplets du résultat pour des requêtes SELECT
- Pendant le développement, il peut être utile d'afficher la requête (*$req*) avant son exécution
- Une seule requête avec la fonction `mysqli_query` (voir `mysqli_multi_query` pour exécuter plusieurs requêtes)
- Nombreuses autres fonctions dans l'API (e.g., le nombre de n-uplets dans le résultat)

---

http://php.net/manual/fr/mysqli.multi-query.php
http://php.net/manual/fr/book.mysqli.php

Étape d'exécution d'une requête non préparée - exemple

```php
<?php // exécution requête
$nomSerie = "The wire";
$requete = 'INSERT INTO Series
  ↪  VALUES(\''.$nomSerie.'\');';
echo $requete;
$resultat = mysqli_query($connexion, $requete);
if($resultat == FALSE) {
  echo "<p>Erreur lors de l'insertion de la série !</p>";
  exit();
}
echo "<p>La série (".$nomSerie.") a été ajoutée avec
  ↪  succès !</p>";
?>
```

Étape de récupération et utilisation d'un tuple

```
// on possède un objet résultat $resultat
while ($ligne = mysqli_fetch_assoc($resultat)) {
  // utilisation des valeurs $ligne['champ1'], ...
}
```

- Récupérer et stocker un seul n-uplet (ligne) du résultat dans un tableau numérique (`mysqli_fetch_row`) ou associatif (`mysqli_fetch_assoc`)
- Besoin d'une boucle pour récupérer tous les n-uplets
- Accès aux valeurs par $ligne[0] ou $ligne['champ']
- Retourne `null` quand il n'y a plus de n-uplet
- `mysqli_fetch_array` retourne un tableau à la fois associatif et numérique (si possible)

---

http://php.net/manual/fr/mysqli-result.fetch-row.php
http://php.net/manual/fr/mysqli-result.fetch-assoc.php

Étape de récupération et utilisation d'un tuple - exemple

```php
<?php // traitement résultat requête
$resultat = mysqli_query($connexion, 'SELECT DISTINCT nomSerie FROM Series;');

if($resultat == FALSE) {
  printf("<p>Un problème est survenu lors de la récupération des séries.</p>");
}
else {
  echo '<h2>Liste des séries</h2><p><ul>';
  while ($row = mysqli_fetch_assoc($resultat)) {
    echo '<li>' . $row['nomSerie'] . '</li>';
  }
  echo '</ul></p>';
}
?>
```

Étape de récupération de tous les tuples

```
// on possède un objet résultat $resultat
$lignes = mysqli_fetch_all($resultat, MYSQLI_ASSOC))
foreach ($lignes as $ligne) {
  // utilisation des valeurs $ligne['champ1'], ...
}
```

- Récupérer et stocker tous les n-uplets (lignes) du résultat dans un tableau PHP de tableaux numériques (`MYSQLI_NUM`), associatifs (`MYSQLI_ASSOC`) ou les deux (`MYSQLI_BOTH`)
- Besoin d'une boucle pour utiliser chaque n-uplet
- Accès aux valeurs par $ligne[0] ou $ligne['champ']
- Utile avec architecture MVC pour l'indépendance entre le modèle et les autres couches

---

https://www.php.net/manual/fr/mysqli-result.fetch-all.php

Étape de déconnexion

```php
// on possède un lien de connexion $connexion
mysqli_close($connexion);
```

- `mysqli_close` ferme une connexion à MariaDB / MySQL
- Utilisation au préalable de *mysqli_kill* pour détruire le thread

---

http://php.net/manual/fr/mysqli.close.php
http://php.net/manual/fr/mysqli.kill.php

Requêtes préparées

Une requête préparée (ou requête paramétrable) est une requête récurrente, que l'on compile avec des variables, et donc réutilisable en fournissant les valeurs manquantes (visibilité limitée à la session du SGBD)

Avantages d'une requête préparée :

- Performances (la requête est déjà compilée, cf optimisation)
- Éviter les risques d'injection SQL (paramètres transmis sous forme binaire)
- Économiser de la bande passante

---

http://php.net/manual/fr/mysqli.quickstart.prepared-statements.php

Requêtes préparées - étapes

Quatre étapes pour exécuter une requête préparée :

1. Préparation de la requête
2. Lien entre variables et paramètres de la requête préparée
3. Exécution de la requête préparée
4. Traitement du résultat de la requête

On peut itérer les étapes 2 à 4 autant de fois que l'on veut (après modification des valeurs des variables)

```
SELECT *
FROM Serie
WHERE nomSerie = 'TBBT';
```

...

```
SELECT *
FROM Serie
WHERE nomSerie = 'GoT';
```

Préparation d'une requête préparée

```
// on possède un lien de connexion $connexion
$req = "SELECT attribut_k FROM table WHERE attribut_n = ?";
$stmt = mysqli_prepare($connexion, $req);
```

- `mysqli_prepare` prépare une requête
- Un lien de connexion est requis en paramètre
- Les variables de la requête sont remplacées par le caractère **?**
- Pas de point virgule à la fin de la requête !

---

http://php.net/manual/fr/mysqli-stmt.prepare.php

Lien variables / paramètres

```
// on a préparé une requête, représentée par $stmt
$var1 = "valeur1" ; ... ; $varn = "valeurn" ;
mysqli_stmt_bind_param($stmt, $types, $var1, ..., $varn) ;
```

- `mysqli_stmt_bind_param` permet de lier des variables aux paramètres de la requête préparée
- Le premier paramètre est la requête préparée
- Le second paramètre représente le type de chaque variable
  - "i" = entier, "s" = string, "d" = décimal, "b" = blob
  - *"sid"* correspond à une première variable de type string, une seconde de type entier, et une troisième de type décimal
- Paramètre(s) suivant(s) : un paramètre pour chaque variable de la requête préparée
- Retourne `true` (succès) or `false` (échec)

---

http://php.net/manual/fr/mysqli-stmt.bind-param.php

Exécution d'une requête préparée

```
// on a préparé une requête, représentée par $stmt
mysqli_stmt_execute($stmt);
```

- `mysqli_stmt_execute` exécute une requête préparée
- Retourne TRUE (succès) or FALSE (échec)

```php
<?php // requête préparée
$dec = 0.5;
$str = "abc";
mysqli_stmt_bind_param($stmt1, "s", $str);
mysqli_stmt_execute($stmt1);
mysqli_stmt_bind_param($stmt2, "ds", $dec, $str);
mysqli_stmt_execute($stmt2);
?>
```

Traitement d'une requête préparée

Pour récupérer le nombre de n-uplets affectés par une requête préparée *$stmt* :

- De type SELECT (obligation de stocker le résultat au préalable)

```
mysqli_stmt_store_result($stmt);
$nb = mysqli_stmt_num_rows($stmt);
```

- De type INSERT, UPDATE, DELETE

```
$nb = mysqli_stmt_affected_rows($stmt);
```

---

http://www.php.net/manual/fr/mysqli-stmt.store-result.php

Traitement d'une requête préparée (2)

Pour récupérer les n-uplets résultat d'une requête SELECT, il faut lier le résultat à des variables :

- Autant de variables que de colonnes !

```
// soit une requête $stmt qui retourne n colonnes
mysqli_stmt_bind_result($stmt, $var_1, ..., $var_n);
```

Récupération d'une ligne résultat avec *mysqli_stmt_fetch* :

- Retourne TRUE (succès) or FALSE (échec), ou NULL (plus de ligne à lire)
- Les variables $var_1, ..., $var_n reçoivent les valeurs de la première ligne résultat (boucle pour les lignes suivantes)

```
mysqli_stmt_fetch($stmt);
```

Exemple de requête préparée

```php
<?php // affichage des épisodes avec requête préparée
$reqp = "SELECT titre FROM Episodes WHERE numero = ?";
if(!($stmt = mysqli_prepare($connexion, $reqp)))
  echo "Erreur de préparation (".mysqli_errno($connexion).") :
  ↪ ".mysqli_error($connexion);
else {
  $var = 1; // valeur 1 pour les épisodes numérotés 1
  mysqli_stmt_bind_param($stmt, "i", $var); // lier la variable $var
  ↪ (type entier) au paramètre de la requête
  mysqli_stmt_execute($stmt); // exécution de la requête
  mysqli_stmt_bind_result($stmt, $episodes); // récupération des tuples
  ↪ résultats
  echo "<h2>Episodes numérotés $var :</h2><p><ul>";
  while (mysqli_stmt_fetch($stmt)) {
    echo "<li>$episodes</li>";
  }
  echo "</ul>";
}
?>
```

Exemple de requête préparée

```php
<?php // affichage des épisodes avec requête préparée
$reqp = "SELECT titre FROM Episodes WHERE numero = ?";
if(!($stmt = mysqli_prepare($connexion, $reqp)))
  echo "Erreur de préparation (".mysqli_errno($connexion).") :
  ↪ ".mysqli_error($connexion);
else {
  $var = 1; // valeur 1 pour les épisodes numérotés 1
  mysqli_stmt_bind_param($stmt, "i", $var); // lier la variable $var
  ↪ (type entier) au paramètre de la requête
  mysqli_stmt_execute($stmt); // exécution de la requête
  mysqli_stmt_bind_result($stmt, $episodes); // récupération des tuples
  ↪ résultats
  echo "<h2>Episodes numérotés $var :</h2><p><ul>";
  while (mysqli_stmt_fetch($stmt)) {
    echo "<li>$episodes</li>";
  }
  echo "</ul>";
}
?>
```

Episodes numérotés 1 :

- The Skank Reflex Analysis
- The Date Night Variable
- Winter is coming

Exemple de requête préparée (2)

Pour réutiliser une requête préparée :

- Modification de la valeur de la variable (déjà) liée
- Exécution de la requête

```php
<?php // réutilisation requête préparée
  $var = 2; // valeur 2 pour les épisodes numérotés 2
  mysqli_stmt_execute($stmt); // pas besoin de lier, exécution directe de la requête
  echo "<h2>Episodes numérotés $var :</h2><p><ul>";
  while (mysqli_stmt_fetch($stmt)) {
    echo "<li>$episodes</li>";
  }
  echo "</ul><br>";
?>
```

Exemple de requête préparée (2)

Pour réutiliser une requête préparée :

- Modification de la valeur de la variable (déjà) liée
- Exécution de la requête

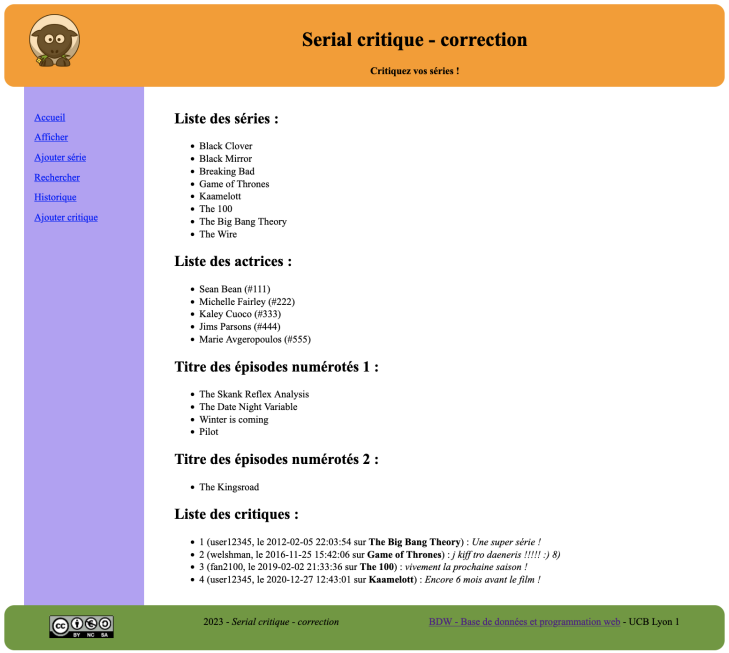
```php
<?php // réutilisation requête préparée
  $var = 2; // valeur 2 pour les épisodes numérotés 2
  mysqli_stmt_execute($stmt); // pas besoin de lier, exécution directe de la requête
  echo "<h2>Episodes numérotés $var :</h2><p><ul>";
  while (mysqli_stmt_fetch($stmt)) {
    echo "<li>$episodes</li>";
  }
  echo "</ul><br>";
?>
```

Episodes numérotés 2 :

- The Kingsroad

En résumé

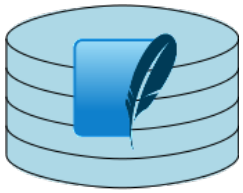
- PHP inclut trois APIs pour MariaDB / MySQL, dont **mysqli**
- Quatre étapes pour se connecter et interroger MySQL
- Factorisation du code et sécurité avec les requêtes préparées



Exemple de page utilisant l'extension mysqli

Plan

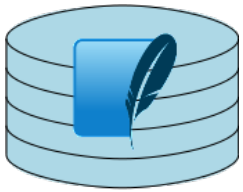
Introduction au DAO




```php
$db = sqlite_open('bd-series.db', 0666,
    $error);
if(!empty($error))
  die('Erreur SQLite : ' . $error);
$result = sqlite_array_query($db, 'SELECT
    * FROM Actrices', SQLITE_ASSOC);
foreach($result as $key => $row) {
  print_r($row);
}
```

Code d'affichage des actrices pour SQLite 2

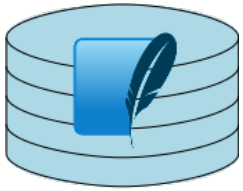
Introduction au DAO



SQLite 2

```php
$db = sqlite_open('bd-series.db', 0666, $error);
if(!empty($error))
  die('Erreur SQLite : ' . $error);
$result = sqlite_array_query($db, 'SELECT * FROM Actrices', SQLITE_ASSOC);
foreach($result as $key => $row) {
  print_r($row);
}
```

Code d'affichage des actrices pour SQLite 2



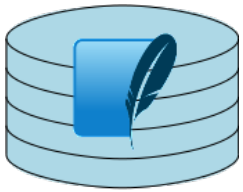
SQLite 3

```php
try {
  $db = new SQLite3('bd-series.db');
} catch (Exception $exception) {
  die("Erreur sqlite3 : " . $exception->getMessage());
}
$result = $db->query('SELECT * FROM Actrices');
while($row = ($result->fetchArray(SQLITE3_ASSOC))) {
  print_r($row);
}
```

Code d'affichage des actrices pour SQLite 3

En cas de montée de version du SGBD, besoin de réécrire une partie du code !

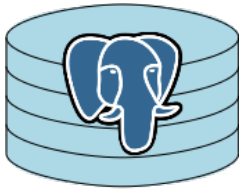
Introduction au DAO



SQLite 2

```php
$db = sqlite_open('bd-series.db', 0666, $error);
if(!empty($error))
  die('Erreur SQLite : ' . $error);
$result = sqlite_array_query($db, 'SELECT * FROM Actrices', SQLITE_ASSOC);
foreach($result as $key => $row) {
  print_r($row);
}
```

Code d'affichage des actrices pour SQLite 2



PostgreSQL

```php
$db = pg_pconnect("dbname=bd-series");
if(!$db)
  die("Erreur de connexion à Postgres");
$result = pg_query($db, "SELECT * FROM Actrices");
while ($row = pg_fetch_assoc($result)) {
  print_r($row);
}
```

Code d'affichage des actrices pour PostgreSQL

Si deux client.e.s ont un projet fortement similaire, mais des SGBD différents, besoin d'écrire deux fois le code d'accès aux données !

Data Access Object (DAO)

Un DAO (objet d'accès aux données) est un "design pattern" qui fournit une interface abstraite d'accès à plusieurs SGBD :

- Implémente les fonctions de base de données (SQL)
- Retourne des "objets métier", e.g. des (instances de) séries, actrices
- Séparation entre le métier (modèle) et l'accès aux données
- Permet de changer de SGBD en modifiant le DAO
- Améliore les performances (en cas de lecture tuple par tuple)

PHP propose une sorte de DAO générique appelé PHP Data Objects (PDO)

---

http://en.wikipedia.org/wiki/Data_access_object
http://en.wikipedia.org/wiki/Design_pattern

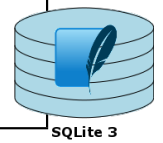
Exemple avec PHP Data Objects (PDO)

```php
function connexion($db) {
  try {
    $dbh = new PDO($db);
    return $dbh;
  } catch (PDOException $e) {
      print "Erreur de connexion ! " .
      ↪ $e->getMessage();
    return null;
  }
}

function get_actrices($dbh) {
  $result = $dbh->query('SELECT * FROM
  ↪ Actrices');
  return $result->fetchAll();
}
```

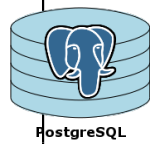
Fonctions PDO génériques de connexion et de récupération des séries

```php
$db_sqlite = 'sqlite:bd-series.db';
$dbh = connexion($db_sqlite);
if($dbh)
  print_r(get_actrices($dbh));
```



Code d'affichage des séries pour SQLite

```php
$db_postgres =
↪ 'pgsql:host=localhost;port=5432;
↪ dbname=bd-series;user=calvin;
↪ password=hobbes';
$dbh = connexion($db_postgres);
if($dbh)
  print_r(get_actrices($dbh));
```



Code d'affichage des séries pour Postgres

http://www.php.net/manual/fr/book.pdo.php

Plan

Généralités

Avec PHP, il est possible de manipuler le système de fichiers du serveur pour :

- Lire ou écrire dans un fichier
- Obtenir des informations sur un fichier
- Gérer des fichiers uploadés par l'utilisatrice
- Lister les fichiers d'un répertoire
- Créer, supprimer des fichiers ou répertoires
- ...

Extension **filesystem** incluse dans le noyau de PHP

---

http://www.php.net/manual/fr/book.filesystem.php

Bilan

Quelques extensions de PHP :

- Variables **superglobales** (dont celle de **session**)
- Manipulation des **bases de données** (ici MariaDB / MySQL)
- Manipulation du **système de fichiers** du serveur

Bilan

Quelques extensions de PHP :

- Variables **superglobales** (dont celle de **session**)
- Manipulation des **bases de données** (ici MariaDB / MySQL)
- Manipulation du **système de fichiers** du serveur

Prochain cours :
structuration d'un site web et architecture MVC