Module H - Programmation Web dynamique avec PHP et MySQL

Préparé par Stéphane Brunet
Révisé par Luc Baron
Version du 18 novembre 2020

1 Introduction

1.1 Objectifs

Ce document a pour but de donner un aperçu de l'usage de scripts «côté serveur» pour générer des pages Web dynamiques. Cette introduction ne fait qu'un survol des possibilités.
La question du choix d'un langage de script «côté serveur» alimente en permanence de nombreux forums de discussion sur Internet. Il existe beaucoup d'alternatives telles que Java, PHP, Ruby, Perl, Python, ASP, C# ou même C/C++. Le choix de PHP pour ce cours n'est pas arbitraire pour autant :

- PHP a une communauté d'usagers très importante, d'où la possibilité de trouver très facilement des réponses à ses questions;
- la structure du langage est relativement simple;
- PHP a été créé dans le but de programmer des applications Web;
- la documentation est très bien faite et traduite en plusieurs langues;
- l'interpréteur PHP est gratuit et multiplateforme;
- de nombreuses offres d'hébergement Web commerciales économiques offrent PHP comme langage de script, ce qui n'est pas le cas pour les autres langages;
- la librairie de fonctions disponibles en standard est fournie, par exemple pour la connexion aux bases de données.

Remarque importante : ce document est une introduction ne reflétant pas les tendances actuelles et les règles de l'art en matière de programmation d'applications Web. Beaucoup de thèmes essentiels (e.g. sessions, sécurité, validation des formulaires, gestion des erreurs) ont été volontairement omis de cette introduction. Dans l'ensemble des exemples donnés, le code PHP est complètement intégré au document XHTML, ce qui est contraire à une règle couramment acceptée de séparer la structure du site Web et le programme utilisant cette structure (à mettre en parallèle avec la règle de séparation du contenu et du style).

1.2 Qu'est-ce qu'une page Web générée dynamiquement ?

Dans la partie du cours traitant de XHTML et CSS, toutes les pages étaient entièrement statiques : une fois écrits, les fichiers HTML sont figés, à moins de les éditer à nouveau. Dans la situation où une page statique est demandée, la réponse du serveur consiste tout simplement à transmettre le fichier HTML correspondant à travers le protocole HTTP.
Imaginons maintenant qu'il existe sur le serveur Web, une application générant du code XHTML de la même manière qu'on aurait écrit le fichier dans un éditeur de texte. Maintenant, lorsque le client Web demande une page particulière, le serveur Web est configuré pour exécuter le programme et récupérer le texte écrit par celui-ci, plutôt que de simplement transférer le contenu d'un fichier. C'est le principe d'une application CGI.
L'usage de l'interface CGI permet de choisir n'importe quel langage qui est en mesure d'écrire du texte sur une fenêtre terminal pour écrire des applications Web. Malheureusement, cette méthode n'est pas très performante, car elle nécessite d'exécuter un programme à chaque requête de l'usager. Dans le cas de PHP, il existe un module pour le serveur HTTP Apache permettant de s'affranchir de cette contrainte. Lorsque le serveur est démarré, le module permettant d'interpréter le langage PHP est aussi chargé. Ainsi, lorsqu'un fichier PHP doit être traité, il n'y a pas de programme à démarrer, mais seulement des commandes à exécuter à partir du fichier.
Utiliser des pages Web générées dynamiquement (ou «à la volée») est tout à fait utile lorsqu'il est nécessaire de représenter en (X)HTML une grande quantité d'information changeante ayant des caractéristiques communes. Par exemple, un catalogue de produits avec la possibilité de visualiser la disponibilité de chaque article, un site de nouvelles avec des articles ajoutés régulièrement, etc. Naturellement, un bon moyen de stocker cette quantité d'information changeante est d'utiliser un système de gestion de bases de données. Dans ce cas, le lien entre le site Web et la base de données est l'application Web qui vient puiser l'information dans la base de données, la formater en langage HTML pour ensuite la transmettre au client Web via le serveur HTTP.

1.3 Transférer des paramètres du client vers l'application Web

Afin de pouvoir interagir avec l'application Web, le client Web doit être en mesure de transférer des paramètres et leur valeur associée via le protocole HTTP utilisé pour le transfert des données. Pour ce faire, le protocole HTTP permet d'utiliser deux méthodes, GET et POST.

La méthode GET

Avec la méthode GET, les paramètres et leur valeur associée sont ajoutés à la suite de l'URL de la page dynamique faisant appel au script. Par exemple :

```
http://monsite.com/page.php?option1=45&forme=rond
```

Cet URL fait appel au script `page.php` se trouvant sur le serveur Web `monsite.com`. Les paramètres transférés et leur valeur associée sont les suivantes :

- le paramètre «option1» ayant pour valeur 45,
- le paramètre «forme» ayant pour valeur «rond».

La méthode POST

Avec la méthode POST, les paramètres et leur valeur associée sont intégrés à la requête HTTP mais séparés de l'URI de la page dynamique faisant appel au script. Dans le cadre de ce cours, nous n'avons pas parlé de la structure du requête HTTP donc la manière dont les paramètres sont intégrés à la requête dépasse les objectifs du cours.
Comme les paramètres ne sont pas dans l'URI, il faut nécessairement trouver un moyen d'ajouter les paramètres à la requête lorsque l'usager veut soumettre les données à l'application Web. C'est le client Web qui se charge de cette tâche.
Utiliser la méthode POST a deux avantages majeurs par rapport à la méthode GET :

- l'URL n'est pas encombré de paramètres, car ces derniers sont séparés;
- intégrer les paramètres à la requête HTTP offre moins de restriction quant au format et à la taille des paramètres envoyés au serveur.

Par contre, vu qu'il est nécessaire pour le client Web de connaître les noms des paramètres et leur valeur associée avant de consulter la page dynamique, il n'est pas possible de simplement entrer l'URI de la page dans le navigateur.

2 Les formulaires en XHTML

2.1 Introduction

Le but des formulaires est de permettre à l'utilisateur d'entrer des informations qui seront ensuite envoyées au serveur puis traitées par une application. Le standard (X)HTML propose une série de balises pour les différents objets de contrôles couramment employés : zone de texte, boutons, cases à cocher, boutons radio, listes de sélection, listes déroulantes.
L'usage des formulaires est décrit dans la section 17 du standard HTML 4.01. L'information est toujours valide pour le XHTML 1.0, sans oublier les petites différences qui existent entre les deux standards ...
Voici l'exemple, traduit et adapté pour XHTML, de la section 17.4.2 du standard HTML 4.01 :

```
<form action="URI_du_script" method="post">
  <p>
  prénom: <input type="text" name="firstname" /> <br />
  nom: <input type="text" name="lastname" /> <br />
  courriel: <input type="text" name="courriel" /> <br />
  <input type="radio" name="sexe" value="masculin" /> masculin <br />
  <input type="radio" name="sexe" value="féminin" /> féminin <br />
  <input type="submit" value="Envoyer" />
  <input type="reset" value="Remise à zéro" />
  </p>
</form>
```

Et le résultat :

prénom:
nom:
courriel:
masculin
féminin
Envoyer Remise à zéro

Quelques explications et remarques sur cet exemple :

- tout ce qui se trouve entre les éléments `<form>` et `</form>` fait partie du formulaire. L'attribut `action` donne l'URL du script qui va traiter les données. L'attribut `method` permet de choisir entre GET ou POST pour le transfert des paramètres;
- le contenu du formulaire doit se trouver à l'intérieur d'un élément de type `<p>` ou `<div>` pour être conforme au standard XHTML;
- les objets de contrôles (zone de texte, boutons radio et boutons) sont des éléments `<input />`. Les principaux attributs sont les suivants :
  - `type` : le type d'objet de contrôle,
  - `name` : le nom du paramètre dans lequel va être stocké l'information,
  - `value` : la valeur prise par le paramètre (a une signification différente selon l'objet de contrôle utilisé).
- les objets de type `submit` et `reset` sont des boutons particuliers permettant respectivement de soumettre le formulaire au script de traitement des données ou de remettre à zéro le formulaire avec les valeurs par défaut.

Il existe plusieurs méthodes pour mettre en page un formulaire. La technique la plus simple est celle utilisée ci-dessus, mais ne donne pas nécessairement un résultat très esthétique. Une autre technique couramment utilisée est celle utilisant un tableau pour ordonner et aligner proprement les différents champs. Enfin, il est possible d'utiliser des feuilles de styles pour obtenir le même effet.

2.2 Usage des différents objets de contrôle

Les prochains paragraphes présentent les différents types d'objets de contrôle avec les caractéristiques particulières qui leur sont associées.

2.2.1 Ligne de texte et zone de mot de passe

Une ligne de texte est définie avec l'élément `<input />` en utilisant les attributs décrits ci-dessous :

- `type` : égal à `text` pour obtenir une zone de texte d'une seule ligne ou bien égal à `password` pour une zone de mot de passe où les caractères entrés seront masqués;
- `name` : nom du paramètre, tel qu'envoyé au script de traitement;
- `value` : valeur initiale/par défaut;
- `maxlength` : nombre maximum de caractères;
- `size` : largeur à l'écran (en nombre de caractères) de la zone de texte.

Remarque : en utilisant une zone de mot de passe, le contenu du champ n'est pas visible à l'écran. Par contre, lors de l'envoi, celui-ci est transmis sans encryptage et donc lisible par n'importe quelle personne capable d'intercepter la requête HTTP, à moins d'utiliser une version sécurisée du protocole (HTTPS). Dans tous les cas, il n'est pas recommandé d'utiliser la méthode GET pour transférer un mot de passe!

2.2.2 Zone de texte

Une zone de texte utilise l'élément `<textarea>`, par exemple de la manière suivante :

```
<textarea name="zoneDeTexte" cols="25" rows="10">
Ceci est le texte par défaut.
Dans la zone de texte.
Il peut prendre plusieurs lignes.
</textarea>
```

dont le résultat donne :

Ceci est le texte par défaut.
Dans la zone de texte.
Il peut prendre plusieurs lignes.

La valeur initiale est le texte compris entre les balises de début et de fin. Les attributs utilisés sont décrits ci-dessous :

- `name` : nom du paramètre;
- `cols` : largeur à l'écran (en nombre de caractères) de la zone de texte;
- `rows` : hauteur à l'écran (en nombre de caractères) de la zone de texte.

2.2.3 Cases à cocher

Les cases à cocher sont des choix d'options qui peuvent être individuellement cochées. Voici un exemple d'utilisation :

```
<input type="checkbox" name="opt1" checked="checked" /> Option 1 - cochée par défaut <br />
<input type="checkbox" name="opt2" /> Option 2 <br />
<input type="checkbox" name="opt3" /> Option 3 <br />
<input type="checkbox" name="opt4" /> Option 4 <br />
```

dont le résulat donne :

- [x] Option 1 - cochée par défaut
- [ ] Option 2
- [ ] Option 3
- [ ] Option 4

L'élément utilisé est `<input />` avec les attributs décrits ci-dessous :

- `type` : égal à `checkbox` pour obtenir une case à cocher;
- `name` : nom du paramètre;
- `checked` : si on veut que la case soit initialement cochée, il faut que cet attribut soit présent. Dans ce cas, il faut lui donner la valeur `checked`;
- `value` : lorsqu'une case est cochée, le formulaire enverra le paramètre avec la valeur "on" à moins qu'on ait spécifié une valeur particulière avec cet attribut.

Remarque : seules les cases cochées seront prises en compte lors du transfert des paramètres et de leur valeur associés au script de traitement des données.

2.2.4 Boutons radio

Les boutons radio diffèrent des cases à cocher dans la mesure où un seul bouton à la fois d'un même groupe peut être activé. Voici un exemple d'utilisation :

```
<input type="radio" name="grpOpt" value="1" checked="checked" /> Option 1 -
  présélectionnée <br />
<input type="radio" name="grpOpt" value="2" /> Option 2 <br />
<input type="radio" name="grpOpt" value="3" /> Option 3 <br />
<input type="radio" name="grpOpt" value="4" /> Option 4 <br />
```

dont le résulat donne :

L'élément utilisé est **`<input />`** avec les attributs décrits ci-dessous :

- **`type`** : égal à **`radio`** pour obtenir un bouton radio;
- **`name`** : nom du paramètre (notez que celui-ci doit être commun aux boutons radio d'un même groupe);
- **`checked`** : si on veut que le bouton soit initialement activé, il faut que cet attribut soit présent. Dans ce cas, il faut lui donner la valeur **`checked`**. Il est recommandé de donner une valeur par défaut pour chaque groupe de boutons radio;
- **`value`** : valeur associée au bouton radio. Lorsque le contenu du formulaire est transmis, le paramètre inclus dans l'attribut **`name`** va prendre la valeur de l'attribut **`value`** du bouton radio actif;

2.2.5 Paramètre caché

Un paramètre caché est un élément **`<input />`** avec les attributs décrits ci-dessous :

- **`type`** : égal à **`hidden`** pour obtenir un paramètre caché;
- **`name`** : nom du paramètre;
- **`value`** : valeur associée au paramètre.

Bien que ce paramètre ne s'affichera pas à l'écran, il suffit d'afficher le code source de la page pour en découvrir sa valeur. Le but n'est donc pas de véhiculer des informations confidentielles, mais plutôt des informations nécessaires au traitement du formulaire, mais que l'usager de la page Web n'a pas à modifier ni connaître.

2.2.6 Listes d'options

Les listes d'options peuvent être du type "déroulantes" ou non, à choix simple ou multiple selon les propriétés qu'on lui donne. Voici un exemple d'utilisation :

```
<select size="1" name="options">
  <option value="1">Option 1</option>
  <option value="2">Option 2</option>
  <option value="3">Option 3</option>
  <option value="4">Option 4</option>
  <option value="5">Option 5</option>
  <option value="6">Option 6</option>
  <option value="7">Option 7</option>
</select>
```

dont le résulat donne :

L'élément définissant la liste d'option est **`<select>`** avec les attributs décrits ci-dessous :

- **`size`** : nombre de ligne pour la liste d'option. Si cet attribut a la valeur 1, c'est une liste déroulante, sinon c'est une liste avec une barre de défilement;
- **`name`** : nom du paramètre;
- **`multiple`** : si on veut que l'utilisateur puisse sélectionner plusieurs valeurs dans la liste, il faut que cet attribut soit présent. Dans ce cas, il faut lui donner la valeur **`multiple`**.

Chaque option de la liste d'option est un élément **`<option>`** avec les attributs décrits ci-dessous :

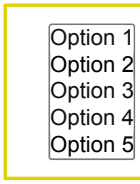
- **`value`** : valeur associée à l'option. Si cet attribut est omis, c'est le contenu de la balise qui est utilisée;
- **`name`** : nom du paramètre;
- **`selected`** : si on veut que cette option soit présélectionnée, il faut que cet attribut soit présent. Dans ce cas, il faut lui donner la valeur **`selected`**.

Voici un autre exemple avec une liste non déroulante et sélection multiple. Sous PC, il faut garder le bouton "Ctrl" enfoncé pour sélectionner/désélectionner plusieurs options :

```
<select multiple="multiple" size="5" name="options">
  <option selected="selected" value="1">Option 1</option>
  <option value="2">Option 2</option>
  <option selected="selected" value="3">Option 3</option>
  <option selected="selected" value="4">Option 4</option>
  <option value="5">Option 5</option>
```

```
  <option value="6">Option 6</option>
  <option value="7">Option 7</option>
</select>
```

dont le résulat donne :



2.2.7 Boutons d'envoi et de remise à zéro

Les deux boutons les plus couramment utilisés, pour l'envoi et la remise à zéro du formulaire, sont définis avec l'élément `<input />` en utilisant les attributs décrits ci-dessous :

- `type` : égal à `submit` pour le bouton d'envoi du formulaire au script de traitement ou bien égal à `reset` une remise à zéro (valeurs initiales) des paramètres du formulaire;
- `value` : le texte qui sera affiché sur le bouton.

3 PHP, un langage «côté serveur»

3.1 Un exemple en PHP

Prenons un fichier XHTML statique et changeons son nom pour qu'il finisse par l'extension `.php` plutôt que `.html`. Ceci va indiquer au serveur HTTP que le fichier doit être traité comme un script écrit en langage PHP. Si nous ne modifions rien d'autre et que nous publions ce fichier sur le serveur, la page Web s'affichera comme auparavant. Le fichier XHTML avec l'extension `.php` est du code PHP valide! Maintenant, indiquons à l'interpréteur PHP les endroits où nous voulons lui faire traiter de l'information, comme dans l'exemple ci-dessous :

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
  <title>Un premier exemple PHP</title>
</head>
<body>
  <p> Bonjour! Vous vous êtes connectés à cogito depuis l'adresse IP suivante :
  <?php print $_SERVER['REMOTE_ADDR']; ?> </p>
  <p> La requête HTTP a été traitée par PHP à <?php print date('H:i:s'); ?> </p>
</body>
</html>
```

Qui pourrait donner le résultat suivant (selon l'heure et l'endroit d'où est consultée la page Web) :

Bonjour! Vous vous êtes connectés à cogito depuis l'adresse IP suivante : 132.207.40.13
La requête HTTP a été traitée par PHP à 13:45:32

Notez les zones qui ont été remplacées par du texte inséré "dynamiquement" : l'adresse IP et l'heure. Elles proviennent des commandes PHP exécutées à l'endroit même où le résultat est inséré. Pour indiquer à l'interpréteur PHP où nous voulons insérer des commandes, il faut utiliser des balises particulières `<?php` et `?>` entre lesquelles les commandes PHP doivent être écrites.
Dans l'exemple ci-dessus, les deux blocs de commandes PHP consistent en une commande `print` qui permet d'imprimer quelque chose dans la page XHTML. Dans le premier bloc, on demande à PHP d'imprimer l'adresse IP du client Web tandis que le deuxième permet d'afficher l'heure actuelle grâce à la fonction `date()`. Il n'est pas nécessaire d'entrer plus dans les détails pour le moment...

3.2 Notions de base de PHP

Cette introduction à PHP n'est ni un cours de programmation, ni un cours de PHP mais plus une série de notions qui vous permettra de modifier du code PHP existant. Pour plus de renseignements, il est fortement conseillé de consulter le manuel PHP disponible en ligne, tant en Français qu'en Anglais. Il est extrêmement bien fait et plein d'exemples simples à comprendre.

3.2.1 Imprimer quelque chose : la commande `print`

La commande permettant d'imprimer quelque chose dans une page Web est utilisée constamment. Elle s'appelle `print` et peut s'utiliser de nombreuses manières différentes. Dans l'exemple qui suit, le résultat imprimé est toujours le même :

```
print "Bonjour!";
print 'Bonjour!';
```

```
print('Bonjour!');
print("Bonjour!");
```

Dans le cas de la commande `print` l'usage des parenthèses est facultatif (ce n'est par contre pas le cas de la plupart des fonctions). Notez que la chaîne de caractères «Bonjour!» est écrite soit entre simples guillemets `'...'` ou bien entre doubles guillemets `"..."`. Ce sont deux méthodes pour signifier à PHP que nous voulons écrire du texte. La nuance entre les deux est expliquée dans la section sur les chaînes de caractères.
Remarquez aussi que chaque commande PHP se termine par un point-virgule `;` .

3.2.2 Variables et opérations arithmétiques

Une variable commence toujours par le caractère `$`. Elle permet de stocker de l'information telle que du texte, une valeur numérique ou un tableau contenant de l'information de manière ordonnée. L'exemple suivant montre comment affecter une valeur à des variables et quelques opérateurs mathématiques simples :

```
$a = 15;
$b = 13.2;
$c = ($a + $b) * $a;
$d = (2^16 - $a) / $c;
```

Les deux premières lignes affectent simplement les valeurs 15 et 13.2 aux variables nommées respectivement `$a` et `$b`. La troixième ligne calcule l'opération arithmétique `($a + $b) * $a` (donnant le résultat 423), puis dépose le résultat dans la variable `$c`. La quatrième ligne calcul 2 exposant 16, soit 65536, puis on soustrait 15, soit 65521, puis on divise par 423, ce qui donne 154.90.

3.2.3 Les chaînes de caractères

Comme expliquée précédemment, une chaîne de caractères est créée à l'aide de caractères entourés de simples guillemets `'...'` ou bien de doubles guillemets `"..."`. Il est aussi tout à fait possible de stocker une chaîne de caractères dans une variable :

```
$chaine_a = "Une chaîne de caractères";
$chaine_b = 'une autre chaîne de caractères';
```

Il existe une différence fondamentale entre une chaîne entourée de simples guillemets et une entourée de doubles guillemets. La chaîne avec simples guillemets utilise les caractères tels qu'écrits tandis que la chaîne avec doubles guillemets permet d'insérer directement le contenu de variables ou des caractères spéciaux :

```
$a = 15;
$chaine_a = 'Il y avait $a chevaux \n et une chèvre.';
$chaine_b = "Il y avait $a chevaux \n et une chèvre.";
print $chaine_a;
print $chaine_b;
```

Si on affichait le résultat des commandes, on obtiendrait le résultat suivant :

```
Il y avait $a chevaux \n et une chèvre.
Il y avait 15 chevaux
et une chèvre.
```

La première ligne du résultat est le contenu de la chaîne `$chaine_a` tandis que les deux lignes suivantes sont le contenu de la variable `$chaine_b`. Notez que la séquence de caractères `\n` a été remplacée par un retour à la ligne tandis que `$a` a été remplacée par sa valeur.
Il est facile d'empiler (concaténer) des chaînes de caractères ensemble :

```
$a = 'AAA';
$b = 'BB';
$c = 'CCC';
$chaine_a = $a.$b;
$chaine_b = $b.$c.$b;
```

Le contenu de `$chaine_a` sera `AAABB` et celui de `$chaine_b` sera `BBCCCBB`.

3.2.4 Les tableaux associatifs

Il est possible de stocker les informations dans des structures spéciales appelées tableaux associatifs. Il existe d'autres formes de tableaux, mais celle-ci est particulièrement utile pour l'usage que nous allons faire de PHP dans ce cours. Le but d'un tableau associatif est de faire correspondre une *valeur* à une *clé*.Par exemple, à la clé "quantité" on pourrait faire correspondre la valeur 2. Voici un exemple plus complet :

```
$a = array( 'quantité'  => 2,
            'noProduit' => 3,
            'statut'    => 'livré');
```

Lorsqu'on va interroger une base de données avec PHP, chaque enregistrement va nous être retourné à l'aide d'un tableau associatif dont la clé sera le nom du champ et la valeur le résultat contenu dans ce champ pour l'enregistrement concerné.

Pour pouvoir sélectionner une entité du tableau, il suffit d'utiliser la syntaxe suivante : `$a['statut']` soit pour récupérer la valeur désignée par la clé "statut", soit pour lui affecter une nouvelle valeur :

```
print "Statut = ".$a['statut'];
$a['statut']  = 'annulé';
```

ce qui donnerait le résultat suivant :

```
Statut = livré
```

3.2.5 Opération de branchement `if` et opérateurs de comparaison

Une opération de branchement `if` permet de choisir quel bloc de code va être exécuté selon le résultat d'une expression logique booléenne :

```
if ($a > 5) {
  print '$a est strictement plus grand que 5';
} else {
  print '$a est plus petit ou égal à 5';
};
```

Les opérateurs de comparaison sont `==`, `>`, `>=`, `<` et `<=`. Notez que l'opérateur de comparaison «est égal à» s'écrit à l'aide d'un double caractère `==` pour le différentier du simple égal utilisé pour affecter une valeur à une variable. C'est une source de bug très courante dans les programmes!

3.2.6 Boucles `while`

Il existe d'autres types de boucles que celle présentée dans cette section, mais nous ne les utiliserons pas dans les exercices sur PHP/MySQL. La boucle `while` exécute le bloc de code écrit à l'intérieur d'accolades `{...}` tant qu'une certaine condition est vraie :

```
$a = 5;
while ($a > 0) {
  print '$a = '.$a;
  $a = $a - 1;
};
```

qui donnerait le résultat suivant :

```
$a = 5
$a = 4
$a = 3
$a = 2
$a = 1
```

4 Relier PHP et MySQL

Étape 1 : se connecter à un serveur de bases de données MySQL

La première étape consiste à se connecter à un serveur de bases de données MySQL. De la même manière qu'avec le client `mysql`, il est nécessaire de s'authentifier auprès du serveur MySQL. La fonction `mysqli_connect()` permet de faire cette tâche et retourne un lien vers le serveur en question qui pourra être utilisé dans la suite du programme PHP. Il faut également indiquer le nom de la base de données avec laquelle on désire travailler. Voici un exemple typique de connexion à une base de données :

```
$adresse_serveur='localhost';
$compte_mysql='nom_d_usager';
$mot_de_passe_mysql='mon_mot_de_passe';
$nom_bd='nom_de_la_base_de_données';
$lien_bd = mysqli_connect($adresse_serveur,$compte_mysql,$mot_de_passe_mysql,$nom_bd);
if ($lien_bd->connect_error) {
  // La connexion a échouée
  die("Impossible de se connecter au serveur MySQL: ".$lien_bd->connect_error);
}
```

Étape 2 : exécuter une requête

La fonction `mysqli_query()` permet d'exécuter une requête sur la base de données. Si tout se déroule comme prévu, elle retourne une variable de résultat qui pourra ensuite être traitée :

```
$requete='SELECT * FROM Nom_de_la_table';  //Pas de point virgule ans la requête
$resultat=mysqli_query($requete, $lien_bd)
  or die("Erreur de traitement :" . mysqli_error($lien_bd));  //Message d'erreur
```

Étape 3 : Utiliser le résultat de la requête

La fonction **`mysqli_num_rows()`** permet de connaître le nombre de lignes disponibles dans une variable de résultat d'une requête. Ensuite, pour extraire chaque enregistrement, un à un, il faut utiliser la fonction **`mysqli_fetch_array()`** qui retourne un tableau associatif tant que la fin des enregistrements n'a pas été atteinte. Le tableau associatif est composé de clés nommées selon les champs de la table de résultat de la requête. La valeur associée avec chaque clé est le résultat stocké pour l'enregistrement en cours. Voici un exemple qui affiche sous forme d'un tableau XHTML le résultat d'une requête :

```
if (mysqli_num_rows($resultat) == 0) {
  // Si la valeur est 0, il n'y a aucun enregistrement...
  print "<em>Aucun enregistrement...</em><br />\n";
} else {
  // ... sinon, on imprime un tableau le résultat de la requête
  print "<table>\n";
  print "  <tr>";
  print "<th>sigle</th>";
  print "<th>credits</th>";
  print "</tr>\n";
  while ($enregistrement = mysqli_fetch_array($resultat)) {
    // On imprime chaque enregistrement dans une ligne du tableau
    print "  <tr>";
    print "<td>".$enregistrement['sigle']."</td>";
    print "<td>".$enregistrement['credits']."</td>";
    print "</tr>\n";
  }
  print "</table>\n";
}
```

Quelques remarques sur le traitement du résultat :

- avant de traiter chaque ligne de résultat, nous voulons vérifier qu'il y a effectivement des enregistrements dans le résultat de la requête, d'où la nécessité d'une opération de branchement **`if`**;
- s'il y a aucun enregistrement alors on imprime un message;
- sinon on imprime un tableau XHTML comme ci-dessous;
- on imprime la balise de début du tableau et la ligne contenant les titres de chaque cellule;
- la boucle **`while`** permet d'extraire chacun des enregistrements du résultat de la requête puis d'imprimer son contenu dans une ligne du tableau;
- la condition utilisée pour la boucle **`while`** est un peu particulière. En effet, la fonction **`mysqli_fetch_array($resultat)`** retourne soit un enregistrement sous forme de tableau associatif, soit la valeur booléenne **`FALSE`** quand il n'y a plus d'enregistrements dans le résultat de la requête. Cette valeur est affectée à la variable nommée **`$enregistrement`**. La condition est alors vraie tant que la variable **`$enregistrement`** ne contient pas la valeur **`FALSE`**, ce qui correspond au comportement voulu pour cette tâche;
- la ligne du tableau XHTML contenant les résultats est imprimée;
- à la fin de la boucle **`while`**, il ne reste plus qu'à ajouter la balise de fin de tableau pour terminer la construction du tableau.

L'exécution du code PHP ci-dessus pourrait générer le code XHTML suivant :

```
<table>
  <tr><th>sigle</th><th>credits</th></tr>
  <tr><td>MEC1310</td><td>2</td></tr>
  <tr><td>MEC2510</td><td>3</td></tr>
  <tr><td>MEC3310</td><td>3</td></tr>
  <tr><td>MEC6508</td><td>3</td></tr>
</table>
```